DEVELOPMENT OF ENABLING TECHNOLOGIES FOR SINGLE CELL ANALYSIS WITH MASS SPECTROMETRY

BY

TROY J. COMI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Chemistry
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:

Professor Jonathan V. Sweedler, Chair
Professor Martha U. Gillette
Associate Professor Mary L. Kraft
Professor Yi Lu

ABSTRACT

Mass spectrometry (MS) is an effective methodology for untargeted, label-free, highly multiplexed analyses of trace compounds based on their mass-to-charge ratios. For biological applications, these properties have generated interest in determining biomarkers of diseased states, detecting drug compounds and metabolites, and observing previously unknown chemical messengers. Recent developments in instrumentation have provided exquisite sensitivity with robust performance. A growing field of single cell chemical analysis has arisen around these figures of merit. While early reports utilized manual isolation and extraction, recent developments in high-throughput sampling have enabled the examination of large populations of cells. One such method includes the analysis of dispersed single cells on a flat surface. When cells are randomly seeded onto the surface, their locations have to be determined by optical imaging to direct acquisition of isolated cells efficiently. A variety of microprobe ionization sources are suitable for such analyses, though smaller probe footprints can utilize more densely seeded samples.

This dissertation describes two technologies for performing single cell analysis with mass spectrometry. The first, synchronized desorption electrospray ionization (DESI), facilitates ambient ionization MS with high mass resolution, low duty cycle mass analyzers. The initial report utilized synchronized DESI for mass spectrometry imaging, but interrupting the desorption plume would be useful for profiling several locations on a surface in an arbitrary order for single cell analysis. The second methodology utilizes microscopy images to guide MS profiling. Specifically, image analysis software, called microMS, was developed to perform cell finding and correlate optical coordinates with the physical coordinates in a mass spectrometer. Since most of the functionality of microMS is decoupled from the mass spectrometer, the workflow can be easily extended to a variety of instruments. Using matrix-assisted laser desorption/ionization (MALDI) time of flight (TOF)-MS, rodent pancreatic islet cells were investigated and heterogeneous peptide processing was detected at the single cell level. With secondary ion mass spectrometry, disparate tissue from the mammalian nervous system was differentiated and further stratified into separate populations. A unique feature of such analyses is that only a fraction of the sample is consumed and the location of a cell is constant once the sample is dried. This property greatly simplifies sequential, follow-up analysis. As an example, MALDI-TOF-MS was utilized to rapidly screen a population of islet cells to select alpha and beta cell types. The locations of those cells were then targeted for liquid microjunction extraction in order to examine their metabolite profiles with capillary electrophoresis-MS. Finally, while microscopy-guided MS profiling is accurate enough to target single cells, the methodology is flexible enough to analyze much larger samples, including tissue sections or bacterial colonies. As an application, natural product mutant libraries were screened directly from *E. coli* colonies using microMS. The suite of technologies and protocols described increases the applicability of many mass spectrometers to characterize a range of cells, colonies and similar objects for their chemical composition.

To my family,

for their love and support

ACKNOWLEDGEMENTS

I am greatly indebted to all of my teachers and mentors, which have contributed to my education and growth as a scientist. Most recently, and importantly for this dissertation, is Prof. Jonathan Sweedler, who accepted me into his lab and has provided profound insight into each challenge. It is clear that Jonathan is a master of the craft of research and his tutelage has continually refined my ideas and refocused my work. It has been a privilege to be part of the Sweedler group for the last 3 years. I also must acknowledge Prof. Richard Perry for his advising early in my graduate work. Those formative years influenced how I conduct research and supplied an environment where I could fearlessly approach new topics and skills to tackle any project. The insight and support of my committee members have also been invaluable; I must thank Prof. Martha Gillette, Prof. Mary Kraft, and Prof. Yi Lu for their tireless service. Prof. Ryan Bailey also provided essential feedback and support on my committee for the majority of my graduate work. I also must thank Stephanie Baker, Julie Sides, and Becky Duffield for their help and support.

I have been truly lucky to work with amazing scientists within the Sweedler group. My success was possible only through close collaborations with Dr. Thanh Do, Dr. Bin Li, Dr. Tong Si, Dr. Qian Wu, Dr. Erik Jansson, Dr. Ta-Hsuan Ong, Sage Dunham, Elizabeth Neumann, Marina Philip, Monika Makurath, and Joseph Ellis. Their friendship, advice and support have been as appreciated as their scientific contributions. Dr. Stanislav Rubakhin and Dr. Elena Romanova were essential in most of the work within this dissertation and provided valuable insight and advice throughout many projects. I also recognize the wonderful collaborators I have interacted with including the Gillette and Bhargava Labs as part of the BRAIN Initiative and Dr. Maja Engelstoft. From the Perry lab, Dr. Edward Chainani, Kevin Peters, Heather Robison, Seung Ryu, Kevin Parker, Jed Veach, Yao-Min Liu, and Corryn Neumann were close friends and collaborators.

Finally I thank my family for their enduring love and support. Many nights, weekends and holidays were part of this dissertation and I am grateful for their understanding. Beryl Jones has been a wonderful friend, constant companion and loving editor. We have helped each other through graduate school and I could not have done this alone.

The support by the National Institute of Drug Abuse under Award No. P30 DA018310 to the Neuroproteomics and Neurometabolomics Center on Cell-Cell Signaling at the University of Illinois at Urbana-Champaign is gratefully acknowledged. I also acknowledge support from the Training Program at Chemistry-Interface with Biology support through NIH T32 GM070421, the Springborn endowment, and the National Science Foundation Graduate Research Fellowship Program.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION AND DISSERTATION OVERVIEW

Since Robert Hooke's description of biological cells in 1665,[1] the field of single cell analysis has developed from optical observations to chemical characterization. Cells are the smallest functional unit of life and present a challenging benchmark for analytical methods. Every cell is unique due to its ancestry and local microenvironment; populations appear only as homogenous as our inability to differentiate individuals. While not all heterogeneity is biologically relevant, new analytical methods provide views of single cells within increasing chemical information. Due to the low absolute abundance of compounds and their large diversity, no technique can currently provide a complete profile at the resolution of single cells. Genomics and transcriptomics can circumvent low copy numbers by amplifying initial sequences,[2] but for direct measurements of metabolites, peptides and proteins, instrumental detection limits are imperative. Optical and electrochemical methods are frequently applied to biological systems including single cells. Single molecule fluorescence detection and imaging is well developed,[3] while electrochemical methods can achieve nanomolar detection limits,[4] but each fail to provide highly multiplexed information. Mass spectrometry (MS) provides attomole sensitivity of several hundreds to thousands of compounds simultaneously. While many approaches are available for the analysis of single cells, this dissertation covers two direct measurement methods: MS imaging and optically-guided single cell profiling.

Chapter 2 presents a review on mass spectrometry imaging in the context of proteomics research.[5] The background encompasses many aspects commonly associated with single cell analysis, especially performing images from tissue at cell-scale resolution. Discussions also include sample preparation and matrix application, which greatly affect the final data and spatial resolution. Several ionization methods are also briefly mentioned, including the recent develop of ambient ionization techniques within the last decade.

Chapter 3 introduces the topic of single cell analysis with mass spectrometry, organized by the type of sample preparation involved.[6] While many MS-based approaches are suitable for single cell analysis, methods which utilize dissociated cells are highlighted for their high throughput and ease of preparation. One such technique is optically-guided single cell profiling of dissociated cells, which utilizes microscopy images to direct mass spectral acquisition. High throughput is achieved with optically-guided MS by selectively targeting only the cell locations. Unique aspects of optically-guided MS include its modular nature and ability to repeatedly visit the same cells. As discussed later, the choice of optical imaging and mass spectrometer do not affect basic performance. Since cells are adhered to the sample surface, their location can be revisited for follow-up analysis with the same instrument or by different platforms.

Chapter 4 discusses the implementation and performance of a modified desorption electrospray ionization (DESI) source which synchronizes ionization with low duty cycle mass analyzers.[7] DESI was among the earliest ambient ionization techniques and shows great promise in applications requiring direct analysis of samples at atmospheric pressures. A drawback of DESI when coupled to low duty cycle mass analyzers, such as Orbitrap or Fourier transform ion cyclotron resonance, is that DESI will desorb the sample surface even while the instrument is not trapping ions. This leads to decreased sensitivity from analyte losses. Since ultrahigh resolution mass analyzers are important for resolving the chemical content of complex systems, it is imperative to adapt DESI to such instruments. The approach presented is synchronized DESI, in which the desorption spray is directed away from the sample surface while the mass analyzer is not accepting ions. Originally developed by Huang, et al. for a miniature mass spectrometer,[8] synchronized DESI showed improved sensitivity on the order of the duty cycle. The source was adapted from a commercial DESI emitter and assessed for its performance with MSI. It was found that synchronized DESI is especially effective for analytes weakly bound to the sample surface, an observation supported by a model developed to simulate analyte migration during DESI-MSI. While not directly applied to single cell analysis, the integration of synchronized DESI would be important for single cell profiling in order to prevent unnecessary desorption of cell content while traveling between targets.

Chapter 5 introduces the image analysis and spatial correlation software utilized in the remaining chapters. While any optical image is suitable, the software is focused on microscopy guided MS, called microMS. microMS provides a feature-rich graphical user interface to encapsulate many processes required to utilize an optical image for MS analysis. Functions include automatic cell finding, population stratification on morphology, and distance filtering. Support for four separate instruments is described; three are utilized in specific projects in the remaining chapters. The optical and physical positions are correlated through a point-based similarity registration which requires selection of several fiducial markers. Target positions may be patterned, their analysis order optimized and finally exported in an instrument-specific format. By design of the software architecture, instrument objects interact with the graphical user interface through an abstract base class. The base class defines a limited set of functions which must be implemented for full support while including implementations of common algorithms, such as the point-based similarity registration. The design greatly simplifies the addition of new instruments and provides a unified user interface across platforms. Chapter 5 also demonstrates a powerful aspect of single cell analysis from sample substrates: sequential analysis of the same cell. Once a cell is located on the sample surface, its location is uniquely specified by its pixel position. The sample can simply be moved between mass analyzers and repeatedly analyzed. Such experiments would be difficult to perform without microMS and this feature should find additional applications in the future.

Chapter 6 presents the application of microscopy-guided MALDI-TOF MS profiling for pancreatic islets of Langerhans.[9] Islets are composed of at least four cell types, defined by the expression of peptide hormones. Alpha cells express glucagon, beta cells express insulin, gamma cells express pancreatic polypeptide, and delta cells express somatostatin. During embryonic development, the pancreas forms from two buds from the gut tube which fuse into the mature organ. The difference in origin manifests as distinct populations of cell types for islets derived from the dorsal or ventral bud. Dorsal-derived islets contain more alpha cells and ventral-derived islets have more gamma cells. Using single cell MS profiling, thousands of pancreatic cells were classified with *k*-means clustering based on the abundance of the four hormones listed above. The cell type heterogeneity between dorsal- and ventral-derived islets was successfully repeated using the methodology. As a label-free method, MALDI-TOF-MS also allowed further investigation of the chemical composition of each cell type. Focusing on dorsal/ventral heterogeneity, *m/z* values which differ between the anatomical regions were examined for each cell type. Gamma cells presented several significantly different peaks, which were identified as cleavage products of the full length pancreatic polypeptide from mass matching with LC-MS peptidomics. From full length pancreatic polypeptide (PP), cleavage at the monobasic site R17 results in PP(1-16) and PP(18-36) and dibasic cleavage at RR25-26 generates PP(1-24) and PP(27-36), which were all detected from single cells. Interestingly, the full length PP was not significantly different, indicating heterogeneity in peptide processing within dorsal- and ventral-derived gamma cells. This is also the first report of endogenous production of these cleavage products and was confirmed with MSI of pancreatic tissue.

Chapter 7 presents an adaptation of microMS for utilization with a lab-built C_{60} secondary ion mass spectrometer (SIMS).[10] While MALDI-MS provides information on intact biomolecules as large as several kDa, interference from the small molecule MALDI matrix frequently prevents analysis of metabolites. In contrast, SIMS can be performed without matrix and garners more information on small molecule metabolites. Due to limitations of the custom SIMS instrumentation, additional data analysis is required to accurately parse spectra and assign them to specific cells. Matlab scripts for performing these analyses are presented in Appendix A. While specifics slightly vary, overall the operation of SIMS single cell profiling is similar to MALDI-TOF. A difficulty found with single cell SIMS is that the sensitivity of native SIMS for intact lipid analysis was insufficient to classify cells. To improve sensitivity, samples were coated with a mixture of ionic liquid matrices which were previously shown to improve limits of detection while providing a uniform coating. With matrix enhanced SIMS, samples of dorsal root ganglia cells were easily differentiated from cerebellum cells by t-distributed stochastic neighbor embedding (t-SNE).[11] Further analysis of the two groups revealed additional subclasses which were differentiated by the relative signal intensity of phosphatidylcholine lipids PC(32:0) and PC(34:1).

Chapter 8 is an example of follow-up analysis of samples surveyed with MALDI-TOF-MS. The particular application utilized a liquid microjunction extraction system controlled by microMS which performs targeted extractions for capillary electrophoresis (CE)-MS analysis. While CE-MS can provide quantitative measurements of metabolites, its throughput is quite low with each separation taking ~40 minutes. As such, comprehensive surveys of large populations are prohibitively time consuming. Instead, MALDI-TOF-MS can provide an initial classification of cells to guide follow-up extraction. In addition to determining the accuracy and extraction efficiency, the workflow was applied to single pancreatic islet cells. MALDI-MS successfully classified populations of cells into alpha, beta and gamma cells. Six alpha and five beta cells were targeted for qualitative, CE-MS analysis. Several amino acids were putatively identified by matching exact mass and relative migration order. In both cell types, dopamine was directly detected and confirmed by matching migration time of a standard. While quantitative, comparative analysis was not performed, the addition of internal standards would facilitate such studies in the future.

Chapter 9 is an extension of microMS for high throughput screening of bacterial colonies for the production of natural products and engineered mutations. Plasmid libraries were transformed into *E. coli* strain BL21 (DE3) and grown on a porous hydrophilic membrane. Following induction, the colonies were transferred onto ITO-coated glass slides by imprinting. The colonies were found to auto-fluoresce sufficiently that no nuclear stain was required. Besides the larger size, the randomly seeded colonies resembled single cells and could be found with the existing machine vision algorithms. Each colony was sampled multiple times around its perimeter and the resulting dataset was analyzed by t-SNE and manually clustered to detect mutated peptides by mass matching. The high-throughput, label-free, direct analysis of bacterial colonies has direct implications for screening campaigns on mutant libraries.

Appendix A supplements several chapters by providing documented source code for simulations of DESI MSI, microMS, and analysis scripts for SIMS single cell profiling, CE-MS extracted ion electropherograms, and the determination of removal efficiency for the liquid microjunction system.

Appendix B presents a user manual for microMS covering basic functions and two examples of how to extend the source code to support new instrumentation.

While a diverse range of topics are presented throughout the dissertation, all chapters describe methodology with direct or indirect applications to MS for single cell analysis. In particular, the development of microMS greatly simplifies microscopy guided MS and is flexible enough to analyze diverse samples. The work includes new information on cell heterogeneity in pancreatic islets, nervous systems, and natural product libraries achieved by improvements in MS sampling approaches.

References

(1) Hooke, R. *Micrographia: or some physiological descriptions of minute bodies made by magnifying glasses, with observations and inquiries thereupon*; J. Martyn and J. Allestry: London, 1665.

(2) Shapiro, E.; Biezuner, T.; Linnarsson, S. *Nature Reviews Genetics* **2013**, *14*, 618.

(3) Weiss, S. *Science* **1999**, *283*, 1676.

(4) Jackowska, K.; Krysinski, P. *Analytical and Bioanalytical Chemistry* **2013**, *405*, 3753.

(5) Comi, T. J.; Yoon, S.; Monroe, E. B.; Sweedler, J. V. In *Reference Module in Chemistry, Molecular Sciences and Chemical Engineering*; Elsevier: 2016.

(6) Comi, T. J.; Do, T. D.; Rubakhin, S. S.; Sweedler, J. V. *Journal of the American Chemical Society* **2017**.

(7) Comi, T. J.; Ryu, S. W.; Perry, R. H. *Analytical Chemistry* **2016**, *88*, 1169.

(8) Huang, G.; Li, G.; Ducan, J.; Ouyang, Z.; Cooks, R. G. *Angewandte Chemie International Edition* **2011**, *50*, 2503.

(9) Jansson, E. T.; Comi, T. J.; Rubakhin, S. S.; Sweedler, J. V. *ACS Chemical Biology* **2016**, *11*, 2588.

(10) Do, T. D.; Comi, T. J.; Dunham, S. J. B.; Rubakhin, S. S.; Sweedler, J. V. *Analytical Chemistry* **2017**, *89*, 3078.

(11) Maaten, L. v. d.; Hinton, G. *Journal of Machine Learning Research* **2008**, *9*, 2579.

CHAPTER 2

IMAGING MASS SPECTROMETRY IN PROTEOMICS

Notes and Acknowledgements

This chapter was as update of DOI: 10.1016/B978-0-12-409547-2.11698-1, with coauthors S. Yoon, E.B. Monroe, and J.V. Sweedler, adapted and reproduced here with permission from Elsevier. The update was published as "'OMICS': PROTEOMICS | Imaging MS in Proteomics" in *Reference Module in Chemistry, Molecular Sciences and Chemical Engineering*; Elsevier: 2016. (DOI: 10.1016/B978-0-12-409547-2.11698-1). The chapter presents an overview of mass spectrometry imaging with a focus on peptidomics and recent advances, including ambient ionization imaging.

Introduction

The application of mass spectrometric imaging (MSI) as a tool to study the native distribution of molecules within biological tissues presents an intriguing method for proteomics. Within a single experiment, the distribution of hundreds of known and previously uncharacterized compounds may be studied, including peptides and proteins. MSI typically uses a microprobe ionization source to collect mass spectra from hundreds to millions of spatially defined locations across a sample and, following data acquisition, enables the creation of distribution maps of selected signals. Although relatively little sample preparation is needed for such analyses, retaining the native distribution of compounds of interest is required, and specific steps must be taken to limit analyte redistribution during tissue processing. Here we present a review of MSI methods and instrumentation, protocols for sample preparation, and experimental considerations that are required for successful MSI experiments.

Many mass spectrometric approaches to biological tissue characterization involve extraction from tissue homogenates, followed by multiple stages of liquid-phase separations, e.g., liquid chromatography (LC) or two-dimensional (2D) gel electrophoresis, before introduction to a mass analyzer to achieve information-rich detection. Separation-based mass spectrometry (MS) approaches simultaneously obtain qualitative and quantitative information on numerous distinct analytes from extremely complex samples; however, homogenization destroys spatial information and can dilute scarce analytes below the limit of detection. In contrast, by sampling in discrete areas over the sample, MSI can directly assay tissue sections without requiring interim homogenization or separations. Moreover, MSI retains important spatial information on compounds within the tissue, which is particularly useful when examining heterogeneous samples such as tumors or brain slices. Nevertheless, without a separation stage, MSI detects fewer compounds due to the inherent chemical complexity of tissues, which can also limit the performance of MS-based fragmentation. The two methodologies are therefore complementary, and one often needs to implement both approaches in order to understand both the spatial and chemical characteristics of a particular sample.

The development of several direct desorption/ionization techniques has revolutionized our ability to study proteins and peptides from tissues with MSI. The levels of sensitivity and the high information content resulting from MS measurements allow for the detection, identification, and characterization of proteins and metabolites directly from thin tissue sections. In addition, the spatial distribution of a compound of interest is revealed via the examination of a chemical image, or distribution map. Techniques that provide information about the location of a molecule within a tissue have benefited biologists for decades, yet such methods often require specific antibodies or labels to visualize molecular distributions within a sample. In contrast, MSI generates images for a wide range of analytes in a single experiment without requiring preselection or labeling.

To create an ion image using MSI, mass spectra are collected from an ordered array of locations across the sample. During data processing, the intensity of selected ions is plotted to produce a 2D distribution map across the tissue (Figure 2.1). Because a mass spectrum contains intensity values for numerous compounds, a single MSI experiment provides hundreds of ion images for both known and uncharacterized molecules.

When measuring the contents of an entire organ, individual compounds can be difficult to detect due to their low average global concentration within the tissue. Furthermore, MSI analysis of the spatial localization of compounds within a sample provides information that can suggest a biological function. For example, an analyte localizing within a specific region may indicate that it is involved in processes intrinsic to the biological structure in which it is present. Additionally, when looking for regional (cellular) differences in protein expression, high-resolution MSI provides many of the same benefits as single-cell MS. Specifically, investigating the profile of peptides and proteins at such small spatial scales benefits from the high local concentration of molecules of interest, particularly when these analytes are localized to a single cell or small cluster of cells within a tissue.

This chapter describes the MS methodologies, basic instrumentation, and sample preparation requirements that are integral to the chemical imaging of peptides and proteins in biological tissues. Current image acquisition procedures and sample preparation protocols are also examined, highlighting the latest research methods, and their successes and limitations. Throughout the review, the complementary nature of MSI and LC-MS should become more apparent.

MS Methods and Instrumentation

Proteins have major roles in many biological pathways and so the ability to localize their spatial distributions can enhance our understanding of many physiological processes. The goal of an MSI investigation is to utilize the high information content obtained via MS to map the location of both known and unknown compounds in a sample. To accomplish this, mass spectra must be collected for an array of locations across a sample by selectively interrogating each position.[1] Two traditional MS techniques are well-suited for these analyses: matrix-assisted laser desorption / ionization (MALDI)-MS and secondary ion mass spectrometry (SIMS). In these methods, a laser or ion microprobe, typically with micron or submicron dimensions, desorbs and ionizes compounds from the sample at a specific location. Similarly, recently developed ambient ionization methods with discrete desorption areas can be directly translated to MSI. Examples include desorption electrospray ionization (DESI),[2] nanospray DESI (nanoDESI),[3] and laser ablation electrospray ionization (LAESI).[4] Once ionized, compounds are separated on the basis of their mass-to-charge ratio (*m/z*) and detected. Because the ions originate near the probe beam, the spatial location of the molecules is inferred. Application-specific software enables selection of an analyte signal from the thousands that are obtained and plots the intensity for each point in the array, resulting in an ion image. Although MALDI MS, SIMS and the ambient methods discussed are all microprobe analyses, they vary with regard to their ionization mechanisms, sample handling requirements, and mass detection range.

MALDI MS

Developed in the late 1980s,[5,6] MALDI MS is one of the most powerful methods available to ionize intact biological compounds. In this approach, analytes are incorporated into a low-molecular-mass organic matrix, which strongly absorbs energy from an impinging laser beam. A wide variety of matrices have been developed for both ultraviolet[7] and infrared excitation, including glycerol and derivatives of benzoic and cinnamic acids, among others.[5] Nanosecond laser pulses resonantly excite the matrix and cause its rapid vaporization, resulting in the ejection of neutral and charged analyte molecules, matrix molecules, and matrix–analyte clusters. Owing to particle interactions, ionization occurs in the matrix crystals and the subsequent MALDI plume, producing predominantly monoprotonated $[M + H]^+$ species in positive-ion mode, and deprotonated $[M - H]^-$ species in negative-ion mode operation. Typically, MALDI MS is coupled to a time-of-flight (TOF) mass analyzer that detects ions over a wide mass range (up to hundreds of kilodaltons (kDa)), making MALDI-TOF MS suitable for peptide and protein studies.[8,9] TOF analyzers typically display mass accuracies of 20–200 ppm and a resolving power of 30,000.[10,11] High performance analyzers (e.g., Fourier transform ion cyclotron resonance (FT-ICR) or the Orbitrap) can be coupled to MALDI sources to provide higher mass accuracy and a resolving power of better than 5 ppm and 100,000,[10] respectively. These powerful mass analyzers can provide confident assignments of molecular composition from single-stage mass spectra. At higher masses, fewer molecules of a protein tend to exist in each interrogated spatial region, an issue that is compounded by lower sensitivity of the detector for larger ions. Both factors limit which proteins will be detected in an MSI experiment. At the lower mass range (< 1,000 Da), matrix compounds can form adducts with analytes, further complicating analyses.

In proteomics, two types of approaches are commonly used for protein identification. In the first, known as top-down, intact proteins are directly identified by their molecular mass and peptide fragments produced by tandem mass spectrometry (MS/MS). Alternatively, the bottom-up method utilizes on-tissue enzymatic digestion of proteins prior to mass spectral analysis.

Protein identification can be accomplished by peptide mass fingerprinting[12] and MS/MS sequencing of the digested peptides. MALDI MS is a versatile technique that can be used with either approach for proteomic studies. It also brings several benefits to imaging biological tissues. Foremost, prepared tissues are directly analyzed by applying either a thin coating of matrix[13,14] or an array of droplets[13] on a tissue section. Analyses are also remarkably salt-tolerant; physiological salts may form adducts to analyte molecules but do not inhibit or otherwise significantly perturb spectra. Additionally, several histological stains, including cresyl violet and methylene blue,[15] are compatible with MALDI MS and permit the optical identification of morphological structures. These unique advantages help explain why MALDI MSI is the most widely used method for imaging peptides, proteins, and lipids in biological tissues. Examples of the MSI of a rat brain tissue are shown in Figure 2.2.

Selection of the most appropriate matrix is a critical step to ensure the production of high quality chemical images.[16] Robust protocols for matrix application have been established and optimized for each analyte type or molecular weight. For protein analysis, α-cyano 5-hydroxycinnamic acid (CHCA), 2,5-dihydroxybenzoic acid (DHB), and 3,5-dimethoxy-4-hydroxycinnamic acid (sinapinic acid, SA) are commonly used. CHCA is generally more sensitive for detection of low-mass peptides ($<$ 2,500 Da),[17] whereas DHB provides less matrix interference during analysis of peptides or lipids in both positive- and negative-ion modes.[16] For larger peptides and proteins, SA performs better.[17,18] Other matrices offer improved ionization of specific compounds: 2,4-dinitrophenylhydrazine is used for formalin-fixed paraffin embedded samples,[19,20] 3-hydroxypicolinic acid for oligonucleotides or glycoproteins,[21] 1,5-diaminonaphtahlene for lipids,[22] and aniline as an ionic matrix for digested actin or hemoglobin proteins.[23,24] Homogeneous deposition of the matrix with small crystal sizes is vital for obtaining high spatial resolution MALDI chemical images.[25] Each matrix and application approach produces crystals of different sizes and variable analyte extraction efficiencies, which may also limit the spatial resolution obtained. Typical spatial resolutions in protein analysis range from about 100 µm to 250 µm for routine images.[26] For example, a recent MSI study using matrix sublimation of SA acquired images of chicken liver and mouse brain tissue with a spatial resolution of 10 µm.[26] Spatial resolution may also be limited by the laser spot size and pitch between ablation spots (pixels). In commercial MALDI instruments, laser spot sizes range from 200 µm down to 20 µm.[27] Recently, high spatial resolution in MALDI MSI was achieved with a transmission geometry ion source that irradiates the back side of tissue samples.[28] The back-side illumination allows closer placement of optics, resulting in a laser spot focused to a diameter of less than 1 µm. Acquisition with submicron pixel sizes is an exciting advance that provides the capability of direct imaging with sub-cellular resolution.

Traditionally, microprobe MALDI MSI has been limited in resolution to roughly the size of the laser beam profile. Image oversampling is another approach to improve spatial resolution using commercially available spectrometers; pixels are acquired with a pitch that is less than the laser spot diameter.[29] In oversampling, the laser is used to ablate the layer of matrix from one position on the sample, prior to moving to the next position on the sample. When the distance between two adjacent points is less than the size of the laser beam, any signals from the new position arise only from the region of the beam profile that is still coated with matrix. This allows for the effective imaging of samples at resolutions below the size of the laser beam.
Atmospheric pressure MALDI is a novel ionization technique developed by the Burlingame group.[30] Unlike traditional MALDI sources, ions are generated from the laser pulse at atmospheric pressure. Recently, the Spengler group[11,31] introduced atmospheric pressure

scanning microprobe MALDI using an imaging source attached to an FT-ICR mass analyzer that displays high spatial resolution and accurate mass measurement. They achieved a mass accuracy of 2 ppm and typically obtained a spatial resolution of 10 μm; however, slight oversampling with a laser ablation spot diameter of 5 μm can decrease resolution further to 3 μm.[32]

SIMS

SIMS has been widely used in materials science, including during semiconductor fabrication, for quality control and surface analysis with atomic-level chemical resolution.[33] Figure 2.3 presents a SIMS image of the lipids and other molecules, such as vitamin E, found within a single cell extracted from *Aplysia californica* using an Au^+ primary ion beam. Unlike the laser pulses used in MALDI, SIMS utilizes a tightly focused beam of primary ions that impact the sample to desorb and ionize analytes directly from the surface. Energy from the primary ions are transferred to analytes via a collision cascade, which subsequently ejects secondary ions that are analyzed by their *m/z*.[34] This process causes greater molecular fragmentation than MALDI due to the high kinetic energy of the primary ion beams (in the range of 1–30 keV), thereby limiting the upper mass range of SIMS to the low kDa range.[35] As a result, most SIMS imaging focuses on small molecules, lipids, and atomic ions. Commonly-used primary ion beams include monoatomic (Au^+, Ga^+, and Bi^+) or polyatomic (Au_n^+, Bi_n^+, and Cs_n^+) sources. The primary ion beam may be readily focused to diameters of a few hundred nanometers so that high spatial-resolution imaging is possible. Although focusing cluster ion beams is more difficult than atomic beams (e.g., the focused size of a C_{60}^+ ion beam is around 1 μm), the spatial resolution is still superior to the 10–100 μm spatial resolutions typical of MALDI MSI.

Recently developed cluster ion sources,[36,37] matrix-enhanced SIMS (ME-SIMS),[38,39] and metallization (Met-SIMS)[40,41] provide softer ionization, allowing detection of compounds as large as several kDa. The Winograd[42] research group developed ion sources for SIMS that enable the analysis of organic compounds and thereby, extend SIMS as an imaging technique for the biological sciences. Cluster ions, such as C_{60}^+ or Ar_n^+ (n = 1000–4000), fragment upon impact, distributing the large kinetic energy of the ion between analytes and many fragments from the ion beam.[43,44] As a result, SIMS with a cluster primary ion source can analyze peptides and proteins with high molecular masses in addition to small molecules. Smith *et al.*[45] reported the chemical imaging of mouse brain tissue by C_{60}^+ SIMS FT-ICR MS. Argon-cluster SIMS can expand the mass range up to 25 kDa and increase secondary ion yield by controlling the kinetic energy per atom of the Ar cluster ion.[37,46] As an alternative to cluster ion sources, surface modifications such as ME-SIMS or Met-SIMS can improve the analysis of intact biological compounds. ME-SIMS, which uses a thin layer of MALDI matrix (e.g., CHCA or DHB) applied to the sample, can measure biological compounds at higher mass ranges. Similar to MALDI, ME-SIMS produces mass spectra up to several kDa from various biological compounds. Recently, MacAlees *et al.*[47] reported identification of trypsin-digested bovine serum albumin and savinase via ME-SIMS imaging of nanoLC fractions collected on a SIMS sample plate. ME-SIMS allowed ionization of peptides up to nearly 5 kDa using a gold liquid metal ion gun. Met-SIMS is another surface modification used to improve the analysis of intact biomolecules. Coating the sample surface with a thin (1 nm to 5 nm) metal coating such as gold can extend the mass range available for SIMS analysis.[41] The metal coating is thought to reduce surface charging and analyte fragmentation by taking the brunt of the primary ion impacts.[43,44]

MS/MS is an important technique that provides more confident analyte identification. The lack of MS/MS capability with standard, commercial SIMS instruments has been a drawback to many biological studies. In efforts to address these limitations, Winograd and colleagues[36] utilized a C_{60}^+ ion source coupled with a quadrupole orthogonal TOF mass spectrometer to acquire product ion spectra. They obtained chemical images of vitamin E, cholesterol, and other lipid compounds on the surface of individual *Aplysia californica* neurons. More recently, using SIMS with MS/MS, they identified 1-hexadecyl-2-octadecenoyl-*sn*-glycero-3-phosphocholine [PC(16:0e/18:1)] as a major lipid constituent of the neural membrane.[48]

Although more limited in mass range than MALDI MSI, SIMS is capable of analyzing tissues at spatial resolutions exceeding that of MALDI MSI. Furthermore, SIMS can be used to examine tissue sections without significant sample preparation, allowing higher throughput and less analyte migration than MALDI. As discussed, the addition of a MALDI matrix or metal coating softens ionization, leading to the observation of intact, molecular ions.

Ambient Ionization

Ambient ionization MS is an emerging technique in which ions are generated at atmospheric pressure without sample preparation.[49,50] Following direct desorption and ionization under open-atmospheric conditions, generated ions are analyzed in vacuum. The ability to directly analyze biological samples under ambient conditions is a major advantage over SIMS or MALDI MSI. DESI, introduced by the Cooks group in 2004,[2] produces ions from charged microdroplets generated with an electrospray emitter. Solvent from the microdroplets pools on the sample to create a thin liquid film that extracts analytes from the surface.[51] Desorption from the film occurs by momentum transfer between incoming droplets and the film to generate secondary droplets.[52,53] Gas phase ions form by an electrospray ionization (ESI)-like mechanism[54-56] and are drawn into the mass spectrometer through an atmospheric pressure interface. As ionization is restricted to the impinging electrospray, DESI MSI is performed by rastering the spray over the sample area.[57,58] In most experiments, the mass range is limited up to *m/z* 2,000, with a lateral spatial resolution of about 100 to 200 μm. With careful optimization of operating conditions, including reducing the inner diameter of the DESI emitter capillary, the lateral resolution can improve to 35 μm.[59] DESI MSI studies typically focus on profiling lipids in various biological tissues, including brain,[60] spinal cord,[61] prostate,[62] kidney,[63] and adrenal glands,[64] among others, for the characterization and diagnosis of diseased states. DESI MS is typically less sensitive than MALDI MS for protein analysis;[65] the limits of detection were poorer with analytes having a larger mass due to less efficient desorption. Although the low sensitivity of DESI MSI currently limits its utilization in proteomic research, recent improvements in DESI MS for protein analysis have focused on detecting larger proteins. Shin *et al.*[66] measured purified proteins in a mass range from 12 kDa to 66 kDa, applied as a uniform layer on glass and Teflon via an oscillating nebulizer sample deposition system.[66,67] Though promising, more research is needed to realize ambient imaging of protein distributions with DESI.

In 2010, the Laskin group[3] introduced nanoDESI by modifying the ion source with a nanoelectrospray probe. In this approach, a liquid junction formed by two narrow capillaries was used to extract analytes on the sample surface and transfer them into the mass spectrometer via nanoelectrospray. They observed 3 pmol of cytochrome c (12 kDa) in a bovine heart sample deposited on a hydrophobic substrate. The isolated extraction and direct transfer to the mass spectrometer inlet allowed lateral resolution as low as 10 μm with high sensitivity.[68] By introducing an internal standard to the perfusion solvent, rough quantitation was possible to account for ions with multiple adducted cation variants (e.g., sodiated and potassiated lipids).[69] Some technical issues remain, including sample diffusion in the capillary and difficulties in forming and maintaining the fragile liquid junction, which currently limits the usage of nanoDESI in ambient MSI.

In contrast to the two ambient ionization methods discussed above, which desorb analytes via liquid extraction, laser ablation electrospray ionization (LAESI) utilizes a focused sampling laser beam for desorption and extractive ESI as a post-ionization process.[4] LAESI is a hybrid ambient method that combines representative characteristics of MALDI and ESI. Ionization occurs from a gaseous plume of analyte ions/neutrals desorbed by laser irradiation when electrospray droplets merge with the plume during post-ionization. By decoupling desorption and ionization, each process can be independently optimized for overall higher analytical performance. LAESI uses a mid-infrared laser to excite the endogenous water found in biological samples.[70] Kiss *et al.*[71] reported top-down identification with MSI of intact proteins by LAESI combined with a hybrid ion trap FT-ICR mass spectrometer. Hemoglobin α chain was identified directly from collision induced dissociation, which was utilized to induce fragmentations for the top-down analysis. The work demonstrates that proteins from biological tissue sections in native environments can be analyzed by LAESI MSI for proteomics.

Multimodal Imaging

An emerging interest in the field is to combine multiple imaging modalities to acquire complementary information and gain further insight on the biological system under investigation.[72] Multimodal MSI can employ a variety of optical images, ionization techniques, or MS approaches. The most common strategy is the acquisition of histological stains that are registered with mass spectral images. This routine procedure allows histology-guided classification of different regions within chemical images. Distributions of specific antigens can be targeted using immunostaining methods such as immunohistochemistry[73] or immunofluorescence.[74] Correlating chemical information with traditional medical images is another area of interest. For example, magnetic resonance imaging generates three dimensional (3D) models of tissue samples within their native context. MSI of biopsies from the same region provides specific chemical information and additional diagnostic power for the diseased state. Medical images may also present a scaffold for building 3D MSI images from serial tissue sections.[75] Vibrational microscopy is another nondestructive method to image tissue sections prior to MSI. The spectra can corroborate chemical identification obtained from MS, and images may be acquired at higher spatial resolution than with MSI.[76]

Beyond optical methods, several studies have combined multiple types of MSI to gain insight from their complementary chemical coverage or spatial resolution. Using MALDI MSI to measure protein content is enhanced further by acquiring a separate image for the metabolite and lipid content. Eberlin *et al.*[77] reported sequential imaging using DESI, followed by MALDI to measure lipid and protein content within a single tissue section. When morphologically friendly solvents are utilized for DESI MSI,[78] the tissue remains unperturbed and can be analyzed with MALDI following matrix deposition. The morphology is maintained sufficiently following MALDI to allow for histological staining. A different approach was utilized by Lanni *et al.*[79] for combined SIMS and MALDI imaging of bacterial biofilms. Instead of utilizing MALDI for protein profiling, a gold-coated sample was rapidly analyzed with laser desorption ionization (LDI) at low spatial resolution (500 μm or 1000 μm) to target regions of interest with high resolution SIMS imaging (0.6 μm). The initial LDI image was necessary as a biofilm sample is optically homogenous. In the above cases, the complimentary performances of two ionization sources were leveraged for better analyte coverage or higher throughput.

Although performing sequential MSI experiments garners additional chemical information, the need to have multiple instruments or custom hybrid ionization sources may limit widespread utilization. An alternative is to analyze the same tissue using one ionization source with multiple MS experiments. Korte and Lee[80] utilized a spiral raster to subdivide each pixel into multiple MALDI MS experiments, including high resolution-positive and -negative mode, single-stage MS on a LTQ-Orbitrap Discovery mass spectrometer. During transient acquisition in the Orbitrap, the linear ion trap performed MS/MS acquisition of selected ions to produce multiplexed data for nine distinct images. The single-stage images provided high mass accuracy profiles of a wide range of analytes and the MS/MS images displayed increased specificity and sensitivity, albeit for only a few, selected compounds. These types of imaging experiments should find application in locating drug and metabolite accumulation by selecting exogenous compounds for MS/MS images. Difficulties with the approach arise from the requirement of a single matrix to provide high sensitivity for both positive- and negative-mode MALDI, and the repeated high voltage cycling required for polarity switching. In instances where serial sections are available, it may be advantageous to correlate images between tissue sections so that matrix composition is optimized to each experiment.

A final consideration with multimodal imaging is how to handle the complex data sets. In most cases, qualitative information from one image informs conclusions on the other. For example, histology will highlight an area within a tissue section containing large concentrations of cancer cells. The spectra within this area are then examined to discover putative biomarkers specific for the diseased state. Similarly, examination of one image can dictate which regions should be subjected to further analysis. Quantitative data fusion is more difficult and requires precise registration of the two images. Examples of image fusion include SIMS images combined with scanning electron micrographs[81] or MALDI MSI fused with hematoxylin and eosin stains.[82] In these studies, high spatial resolution microscopy images were used to predict the chemical distribution acquired by MSI. The implicit assumption with these methods is that the chemical distributions match features visible in the other modality, which may not always be valid. Technologies for fusing mass spectral data for MSI are still under development. One example of fusing mass spectra in positive and negative polarity is from a report of profiling single oocytes.[83] Following principal component analysis (PCA), the predominant principal components from positive and negative mode were jointly considered for linear discriminant analysis. Quantitative data fusion represents a challenge for multimodal imaging that should see further development in the near future.

Sampling Protocols

For chemical imaging experiments to succeed, sample preparation procedures should preserve the original distribution of analytes in the tissue. Preparing a tissue section minimally requires tissue dissection, sectioning, transfer and placement on an appropriate target, followed by MS-based data acquisition, as illustrated in Figure 2.4. For SIMS and many ambient methods, the tissue may be directly imaged, whereas MALDI requires application of a matrix coating before imaging. Because each of these steps can cause analyte migration, a wide range of preparative protocols have been developed to optimize the chemical imaging of tissues with MS. SIMS has been successful for small molecule imaging and is used in many situations for cation and lipid imaging. For protein applications requiring information on higher molecular masses, MALDI MSI is often the more appropriate approach; as a result, many of the sample preparation and experimental considerations in the literature focus on MALDI MSI techniques.

Tissue Preparation

Animal sacrifice and tissue dissection protocols, like other preparative steps, must seek to maintain the native complement and distribution of analytes. Therefore, rapid decapitation of animals is the preferred method, particularly in studies of nervous tissues where pharmaceutical euthanasia agents may alter the production and processing of proteins and peptides prior to death. Additionally, rapid tissue dissection and disruption of enzymatic processes is imperative. Following death, if enzymatic activity is not quenched, significant protein degradation may result. To reduce this undesirable effect, tissue samples are typically frozen on dry ice, liquid nitrogen, or liquid nitrogen-slushed liquid propane. One common method uses liquid propane because of its relatively high thermal conductivity and ability to remain a liquid at the slightly elevated temperatures often present at the tissue–liquid interface during freezing.[84] In practice, the tissue sample is loosely wrapped in a small piece of aluminum foil and plunged into a liquid nitrogen and liquid propane mixture. The sample is removed after several minutes and typically stored at −80°C to reduce analyte migration during storage. Similar preparative strategies using dry ice and/or liquid nitrogen alone have also been shown to limit enzymatic degradation. Chemical fixation is another, yet rarely used technique, as interference caused by the fixative during MS analysis is common.

In general, the preparation of tissue sections is relatively straightforward and varies little from histological techniques. It is important, however, to limit the contact between tissues and traditional embedding media, which typically consists of a polymer matrix having a broad range of molecular masses; sections cut from embedded tissues tend to produce contaminant peaks arising from these polymers throughout the mass spectrum. As sections are cut, a thin layer of polymer may be spread over the surface of each section from the microtome blade. To reduce or eliminate the need for embedding media, sections for MSI applications are often prepared using a cryomicrotome operated at −20°C. Tissues may be affixed to the microtome stage using either frozen water (ice) or a small drop of embedding media. If embedding media is used, it is important to apply only enough to affix the tissue to the stage while preventing contact between the media and the sectioning blade. If tissues are encased in ice to help maintain morphological stability during sectioning, they should be rapidly dried to prevent analyte redistribution. Although the preparation of thin tissue sections of non-embedded tissues may be more difficult, 5–10 μm sections are readily obtained and are ideal for chemical imaging. Analysis of smaller sections can create issues, both for maintaining tissue morphology and for analyte detection, as less analyte is present in thinner sections. For irregular, fatty or porous tissues, embedding media is necessary to maintain native morphology during sectioning. Embedding is also utilized with 3D images as it can simplify image reconstruction by preventing tearing and deformation from cutting and mounting. Several types of MSI-compatible embedding media have been reported, including carboxymethylcellulose,[85] gelatin,[39] or agarose.[86] While spectral backgrounds are cleaner than with traditional polymeric media, application-specific evaluation is recommended before embedding critical tissue samples. Another alternative is to stabilize fragile sections with double-sided conductive tape.[87,88] Due to the abundance of tissue archives for a variety of disease states, significant efforts have focused on recovering proteomic information from formalin-fixed paraffin-embedded tissues. A protocol developed by Casadonte and Caprioli[89] involves sectioning, paraffin removal, antigen retrieval, and *in situ* trypsin digestion to prevent MS interference and release peptides from their crosslinks. Numerous options are available for preparing thin tissue sections from dissected samples. The methods adopted at this step can greatly affect the success of analysis and should be chosen and optimized carefully.

Following sectioning, samples must be transferred to a target suitable for the specific instrument being used, typically a metal plate or a glass slide. Transfer may be achieved by blotting the target onto the tissue, or lifting the sample onto the target using a small artist's brush, and then allowed to warm and dry. This warming and drying should be performed rapidly and may be aided by the addition of a stream of warm air or placement in a desiccator. Once on the target, the dried tissue sections may be rinsed with cold ethanol to assist in removing some physiological salts and lipids, both of which can complicate proteomic analyses and reduce instrumental sensitivity.[90] Histological staining for morphological structure identification may be performed following drying as well. Some histological stains have been shown to reduce signal intensity for families of proteins, although as stated previously, both cresyl violet and methylene blue may be used without causing interference or reduction in MS sensitivity. In many cases, histological stains may also be performed following MSI.

Another method for preparing samples for imaging is by dissection of well-defined tissue regions using microsurgery or laser capture microdissection (LCM).[91] In LCM, a laser and polymer film are used to selectively remove small portions of tissue from a larger section. This technique is often used for profiling experiments, as small, distinct regions may be selected under magnification; however, LCM may also be used to isolate larger regions for imaging experiments.[92]

Matrix Application

Central to the capabilities of both MALDI MS and SIMS to detect proteins and peptides is the application of a matrix compound to the tissue surface in order to assist in analyte desorption and ionization. Generally, this involves applying a matrix-containing solution to the tissue via droplets or spray and allowing the matrix to crystallize on the surface. During the crystallization process, analytes need to be extracted and then co-crystallized with the matrix. It is important to balance these processes. Maintaining wetness and an appropriate contact period increases analyte extraction, and thus, sensitivity, but also increases the likelihood of analyte redistribution. One option to address this issue is to apply the matrix in a dryer form, or create an environment that promotes rapid drying; although analyte migration is reduced, sensitivity suffers. Here we describe several protocols that are designed to not only resolve these concerns, but also to optimize the sensitivity and spatial resolution of a collection of mass spectra.

As an image may be thought of as an array of individual points (pixels), the application of an array of droplets on a sample can allow image acquisition when these spots are interrogated. An automated strategy aids in the formation of a regular array of small droplets to enable the creation of chemical images across a tissue. Automated application of microdroplets can be generated with an acoustic ejection system,[93] which is capable of depositing 200 μm spots at spatial resolutions of ~200 μm.

Another sample preparation method for imaging is to apply a thin layer of matrix across the tissue using either an electrospray emitter,[86] a gas nebulizer, or even an artist's airbrush. The goal is to provide adequate extraction while minimizing analyte redistribution by applying the matrix solution in a relatively dry manner. Typically, the matrix is a concentrated (10–30 mg mL^{-1}), largely organic solution (e.g., 1:2, water/acetone); the high organic composition generates smaller microdroplets via evaporation during spray formation. Using the electrospray technique, a small amount of matrix solution is placed into a pulled glass capillary and a high voltage is applied between the tip and the grounded sample, generating a fine spray. For the spray to form, the tip must be kept relatively close to the sample surface (< 1 cm) and moved over the sample to coat large areas of tissue. Electrospray application tends to produce small matrix crystals, which are beneficial for imaging applications. Perhaps the most common means to apply matrix is via aerosol spray, generated either by a gas nebulizer, such as those used to develop thin layer chromatography plates, or by an artist's airbrush. Several thin coats of matrix are applied by passing the aerosol spray across the sample multiple times and then pausing for several minutes to allow the sample to dry. Often, the last series of passes consist of solvent alone to form regularly sized and well-formed crystals. Both the nebulizer and airbrush operate in a comparable manner to produce similarly sized matrix crystals, and permit control over the amount of liquid applied with the matrix. The airbrush method also allows more control of the direction of the spray; however, more samples may be coated at one time using a nebulizer. As mentioned previously, a wet application of matrix increases analyte extraction but may result in analyte redistribution.

In contrast to spray-based methods, matrix can also be applied without solvent using sublimation.[14] In this protocol, the sample is cooled and held above a solid matrix under vacuum. As the matrix is heated, it begins to sublime and solidify upon the sample. Tissues may be thaw-mounted onto slides that are precoated with a matrix deposited by sublimation.[94] Generally, sublimation produces small crystals but results in poorer extraction than wet applications. To increase sensitivity, the coating may be re-crystalized by incubating in a heated chamber with water or organic solvent vapor.[26,95] Again, a balance has to be met between small crystals and sufficient extraction efficiency.

These matrix coating techniques, initially developed for MALDI MSI, may be readily adapted to ME-SIMS experiments without modification, keeping in mind that smaller crystal sizes are desired for ME-SIMS due to the higher spatial resolutions obtainable from SIMS instruments. As mentioned earlier, the surface metallization technique, Met-SIMS,[40,41] was developed to extend the available mass range and sensitivity of SIMS, and has also been applied to MALDI MS. A thin layer of metal (e.g., gold or silver) is applied with a sputter coater, as would be traditionally used to prepare samples for electron microscopy experiments. For MALDI MSI applications, this layer is applied on top of the matrix-coated sample.[38,39] In both MALDI MS and SIMS applications, the metal coating is believed to assist by reducing buildup of an electrical charge on the sample during analysis.[96] Unlike matrix coating, however, analytes do not appear to be directly integrated into the coating.

Chemical Imaging Experiments

Chemical imaging involves the collection of an array of mass spectra acquired at regular intervals across a sample, with some variations depending on instrumentation. In practice, the imaging process involves operating the instrument in an automated acquisition mode during which a list of locations is created. A complete mass spectrum is collected for each of those locations based upon a series of instrumental parameters, (e.g., mass range, laser power, number of acquisitions, etc.), which remain the same for each spot. This 'spot-by-spot' or microprobe process is the most commonly used means of imaging whereby each collected mass spectrum corresponds to an individual pixel in the resulting chemical image.

An alternate imaging mode developed for TOF MSI involves irradiating a large area of the sample with the laser and then separating the desorbed ions in a spatially defined manner using specialized ion optics.[97] The optics allow ions to maintain their spatial positioning throughout the TOF separation; the ions are then detected with a spatially resolved detector. A single laser pulse results in the creation of a complete ion image. In addition, this stigmatic imaging approach tends to produce higher-resolution chemical images because the size of the laser beam does not affect image resolution. However, microscope mode may require the preselection of a mass range to image ratio as opposed to the point-by-point approach, which collects entire mass spectra at each point.

Following spectra collection, software is used to convert intensity data into distribution maps of selected analytes. When utilizing most open source programs, it is necessary to convert data from proprietary software into an open source format such as imzML.[98] Open source formats typically have larger file sizes than their proprietary counterparts, but ongoing design efforts are working to decrease file sizes. Several challenges arise when analyzing MSI data, primarily resulting from the immense size of the data sets. In simple implementations, *m/z* and tolerance values are input and signals within that range are plotted into a 2D array of intensity values. It is common to up-sample the image with some interpolation, and to recolor intensities with a color map. Careful consideration is necessary to balance data sizes and accuracy of results. Common non-targeted approaches include binning *m/z* values and peak finding.[99] Both approaches can drastically reduce the overall memory requirements, but binning leads to reduced mass spectral resolving power and mass accuracy, whereas peak finding can overlook rare or less abundant species. Software packages may be commercial and proprietary, open source, or written for a particular user. Biomap (Novartis), a free software package, is popular for displaying images and superficial inspection of mass spectra, but lacks capabilities for more advanced statistical analysis. Many instrument manufacturers offer software for analyzing MSI data, e.g., FlexImaging from Bruker Daltonics and ImageQuest from Thermo Scientific. SCiLS Lab, a partner of Bruker, is another option for MSI data analysis that allows for a variety of univariate analyses, including hypothesis testing as well as supervised and unsupervised multivariate analyses.

Numerous reports have introduced alternative MSI analysis tools, typically addressing a shortcoming of commercial software or introducing statistical methods. As one example, to retain the mass resolving power of FT-ICR MSI, Smith *et al.*[100] developed Chameleon, an MSI visualization package with bin widths of 0.001 Da. In this work, hardware limitations for storing large data sets were overcome by using a mosaic data cube, a subdivided version of a continuous data cube that can be read into memory piecewise. For nanoDESI MSI, integrating image acquisition with analysis allowed for fine adjustment of sample geometry during imaging and real-time display of ion images to assess image quality.[68] New multivariate analyses have been integrated into MSI including PCA hyperspectral visualization,[101] classification by PCA,[102,103] and hierarchical clustering,[104] among others. For a more thorough review of statistical analysis in MSI, refer to the review by Jones *et al.*[105]

Several considerations must be given weight when creating an ion image. The display scale should be easy to comprehend and provide contrast across the range of intensities. Topographical features of the sample can cause peaks to shift in detected *m/z*. To correct for these shifts, ion images may be created for a larger *m/z* window or standards may be added to the matrix solution to allow the internal calibration of each individual mass spectrum in the image. In cases where semi-quantitation is desired, defects in ionization across the sample due to, for instance, sample preparation defects or topographical features, may be minimized by normalizing analyte signals using one of a variety of metrics.[106]

Conclusions

The development of MSI for localizing the distribution of peptides and proteins in biological tissues makes available a valuable new toolset that benefits a wide range of research disciplines. Contemporary mass spectrometers provide the high sensitivity necessary to detect a range of biological compounds in intact tissues while requiring relatively minimal sample preparation. The major goal of these advanced imaging approaches is to obtain information on heterogeneously distributed analytes, which is otherwise difficult to acquire using the more common technique of dissection to extract tissues for analysis with LC-MS. While MSI is a relatively new addition to the investigative options available to the mass spectrometrist, it likely will become more common in applications that require profiling of heterogeneous samples.

The sample preparation protocols described herein for MALDI MSI and SIMS imaging are similar, and both enhance analyte detection while limiting contaminants and analyte redistribution prior to analysis. Sample preparation requirements are minimized when using ambient ionization methods, which allow direct analysis of tissues that are irregularly shaped or not vacuum stable. The automated collection of thousands of mass spectra can result in a multitude of chemical images for a given experiment; these images often prove to be highly beneficial when used as discovery tools in the analyses of biological tissues. We anticipate even further refinements to these approaches to address specific analytical challenges.

Figures

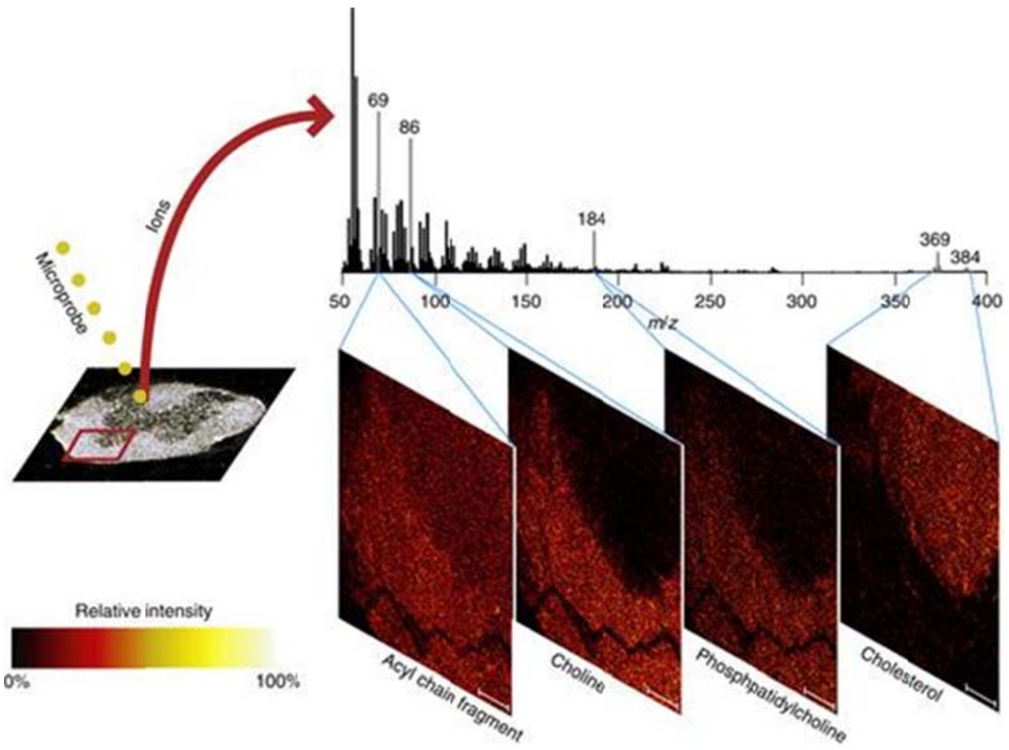


Figure 2.1. Chemical imaging of a rat spinal cord with MSI, in this case, using secondary ion mass spectrometry (SIMS). The ionizing microprobe was used to collect 65,536 individual mass spectra from a 600 µm × 600 µm region of a 10-µm-thick section of spinal cord (outlined in red at left on the optical image). The average spectrum is presented above and is used to select signals of interest from which to create ion images. The presented ion images are shown in a relative thermal scale and show the distribution of cholesterol (*m/z* 384) and several lipid fragments, including an acyl chain fragment (C_5H_9, *m/z* 69), choline (*m/z* 86), and phosphatidylcholine (*m/z* 184). Cholesterol is found largely in the myelinated white matter at the center of the spinal cord, whereas the lipid signals are present in the region where cell bodies reside.

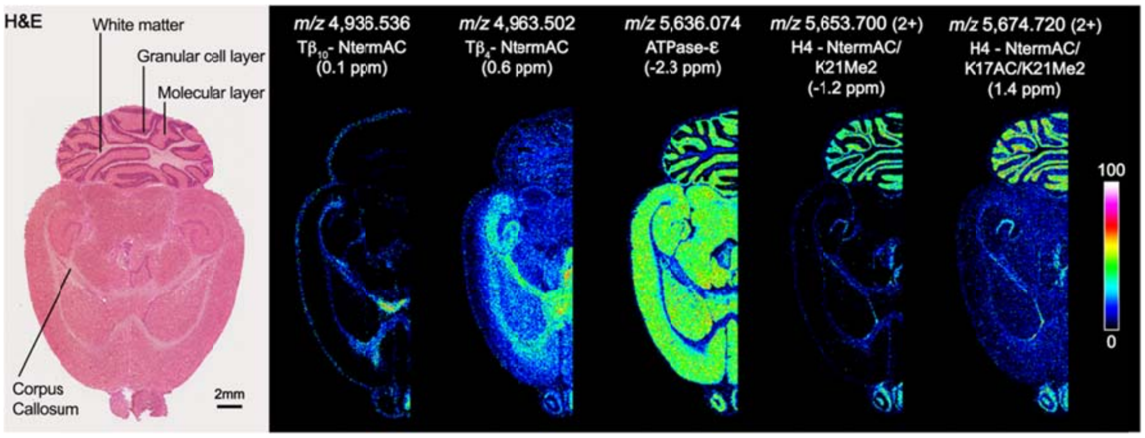


Figure 2.2. High mass resolution MALDI MSI of intact proteins within a rat brain tissue section. The high mass accuracy of FT-ICR MS was utilized to correlate single stage *m/z* values to top-down proteomics sequencing from other experiments. A hematoxylin and eosin stain was performed after MALDI analysis to compare protein distributions with histology. (Reprinted with permission from Spraggins JM *et al*. (2015) MALDI FT-ICR IMS of Intact Proteins: Using Mass Accuracy to Link Protein Images with Proteomics Data. *Journal of American Society for Mass Spectrometry* 26:974-985.)

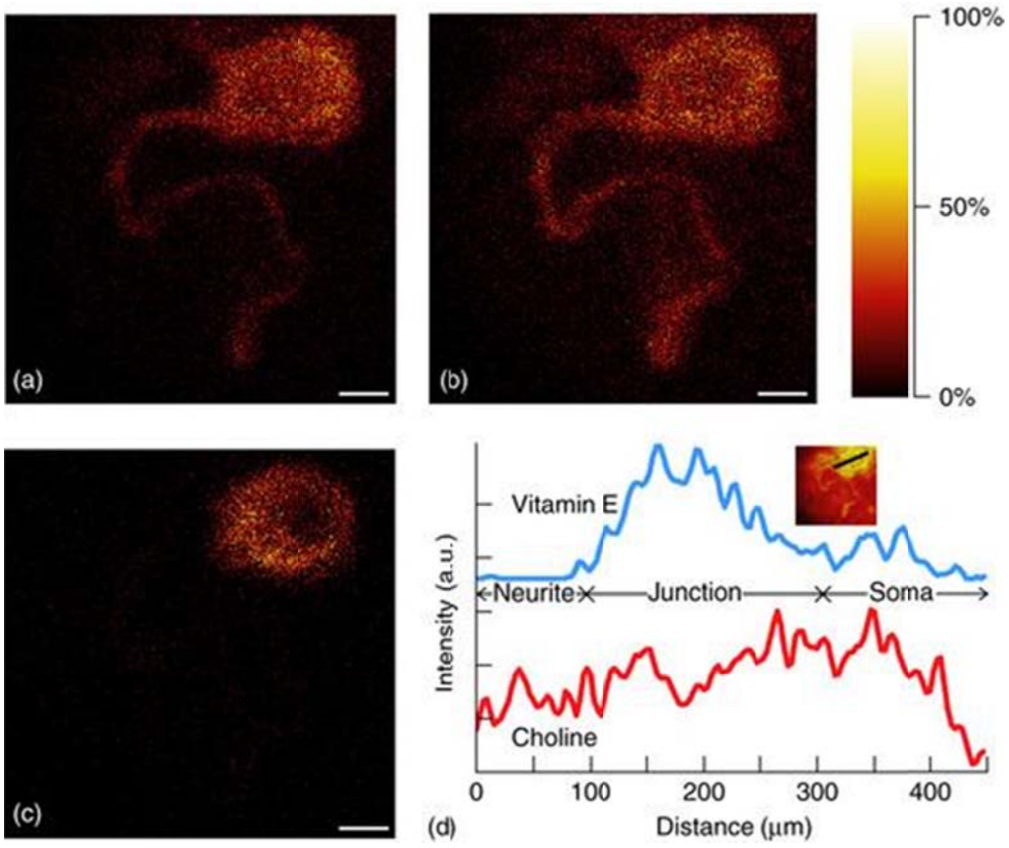


Figure 2.3. Imaging a single neuron with SIMS illustrates the homogenous distribution of cellular lipids by the (a) choline fragment (*m/z* 86) and (b) acyl-chain fragment (*m/z* 69), while (c) vitamin E (*m/z* 430) is localized at the soma-neurite junction. (d) Line scans for vitamin E (top) and choline (bottom) also illustrate this distribution. Scale bars are 100 μm. (Reprinted with permission from Monroe EB *et al*. (2005) Vitamin E imaging and localization in the neuronal membrane. *Journal of the American Chemical Society* 127: 12152–12153.)

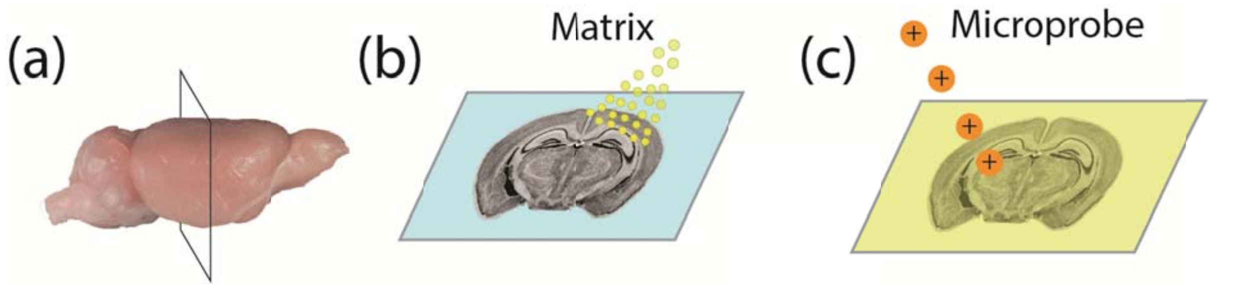


Figure 2.4. Schematic showing the tissue preparation process after animal sacrifice and tissue dissection. For MSI studies, sample preparation consists of several general steps: (a) tissue sectioning, (b) transfer to suitable surface and coating of the sample with a small layer of MALDI matrix and/or metal such as silver or gold, and (c) imaging. Images adapted from Rosen GD, Williams AG, Capra JA, Connolly MT, Cruz B, Lu L, Airey DC, Kulkarni K, Williams RW (2000) The Mouse Brain Library @ www.mbl.org. Int Mouse Genome Conference 14: 166. www.mbl.org.

References

(1) Stoeckli, M.; Chaurand, P.; Hallahan, D. E.; Caprioli, R. M. *Nature Medicine* **2001**, *7*, 493.

(2) Takats, Z.; Wiseman, J. M.; Gologan, B.; Cooks, R. G. *Science* **2004**, *306*, 471.

(3) Roach, P. J.; Laskin, J.; Laskin, A. *Analyst* **2010**, *135*, 2233.

(4) Nemes, P.; Vertes, A. *Analytical Chemistry* **2007**, *79*, 8098.

(5) Karas, M.; Bahr, U.; Giessmann, U. *Mass Spectrom Rev* **1991**, *10*, 335.

(6) Tanaka, K.; Waki, H.; Ido, Y.; Akita, S.; Yoshida, Y.; Yoshida, T.; Matsuo, T. *Rapid Communications in Mass Spectrometry* **1988**, *2*, 151.

(7) Beavis, R. C.; Chait, B. T.; Standing, K. G. *Rapid Communications in Mass Spectrometry* **1989**, *3*, 436.

(8) Remoortere, A.; van Zeijl, R. M.; van den Oever, N.; Franck, J.; Longuespée, R.; Wisztorski, M.; Salzet, M.; Deelder, A.; Fournier, I.; McDonnell, L. *J Am Soc Mass Spectrom* **2010**, *21*, 1922.

(9) Mainini, V.; Bovo, G.; Chinello, C.; Gianazza, E.; Grasso, M.; Cattoretti, G.; Magni, F. *Molecular BioSystems* **2013**, *9*, 1101.

(10) Aichler, M.; Walch, A. *Laboratory Investigation* **2015**, *95*, 422.

(11) Römpp, A.; Spengler, B. *Histochemistry and Cell Biology* **2013**, *139*, 759.

(12) Clauser, K. R.; Baker, P.; Burlingame, A. L. *Analytical Chemistry* **1999**, *71*, 2871.

(13) Kaletaş, B. K.; van der Wiel, I. M.; Stauber, J.; Lennard, J. D.; Güzel, C.; Kros, J. M.; Luider, T. M.; Heeren, R. M. A. *PROTEOMICS* **2009**, *9*, 2622.

(14) Hankin, J. A.; Barkley, R. M.; Murphy, R. C. *J Am Soc Mass Spectrom* **2007**, *18*, 1646.

(15) Chaurand, P.; Schwartz, S. A.; Billheimer, D.; Xu, B. J.; Crecelius, A.; Caprioli, R. M. *Analytical Chemistry* **2004**, *76*, 1145.

(16) Chughtai, K.; Heeren, R. M. A. *Chemical Reviews* **2010**, *110*, 3237.

(17) Cillero-Pastor, B.; Heeren, R. M. A. *Journal of Proteome Research* **2014**, *13*, 325.

(18) Franck, J.; Arafah, K.; Barnes, A.; Wisztorski, M.; Salzet, M.; Fournier, I. *Analytical Chemistry* **2009**, *81*, 8193.

(19) Fenaille, F.; Tabet, J.-C.; Guy, P. A. *Analytical Chemistry* **2004**, *76*, 867.

(20) Lemaire, R.; Desmons, A.; Tabet, J. C.; Day, R.; Salzet, M.; Fournier, I. *Journal of Proteome Research* **2007**, *6*, 1295.

(21) Wu, K. J.; Steding, A.; Becker, C. H. *Rapid Communications in Mass Spectrometry* **1993**, *7*, 142.

(22) Yang, J.; Caprioli, R. M. *Analytical Chemistry* **2013**, *85*, 2907.

(23) Lemaire, R.; Tabet, J. C.; Ducoroy, P.; Hendra, J. B.; Salzet, M.; Fournier, I. *Analytical Chemistry* **2006**, *78*, 809.

(24) Franck, J.; El Ayed, M.; Wisztorski, M.; Salzet, M.; Fournier, I. *Analytical Chemistry* **2009**, *81*, 8305.

(25) MacAleese, L.; Stauber, J.; Heeren, R. M. A. *PROTEOMICS* **2009**, *9*, 819.

(26) Yang, J.; Caprioli, R. M. *Analytical Chemistry* **2011**, *83*, 5728.

(27) Norris, J. L.; Caprioli, R. M. *Chemical Reviews* **2013**, *113*, 2309.

(28) Zavalin, A.; Todd, E. M.; Rawhouser, P. D.; Yang, J.; Norris, J. L.; Caprioli, R. M. *Journal of Mass Spectrometry* **2012**, *47*, 1473.

(29) Jurchen, J. C.; Rubakhin, S. S.; Sweedler, J. V. *J Am Soc Mass Spectrom* **2005**, *16*, 1654.

(30) Laiko, V. V.; Baldwin, M. A.; Burlingame, A. L. *Analytical Chemistry* **2000**, *72*, 652.

(31) Koestler, M.; Kirsch, D.; Hester, A.; Leisner, A.; Guenther, S.; Spengler, B. *Rapid Communications in Mass Spectrometry* **2008**, *22*, 3275.

(32) Guenther, S.; Römpp, A.; Kummer, W.; Spengler, B. *International Journal of Mass Spectrometry* **2011**, *305*, 228.

(33) Heeren, R. M. A.; McDonnell, L. A.; Amstalden, E.; Luxembourg, S. L.; Altelaar, A. F. M.; Piersma, S. R. *Applied Surface Science* **2006**, *252*, 6827.

(34) Vickerman, J. C. In *Surface Analysis – the Principal Techniques*; John Wiley & Sons, Ltd: 2009, p 113.

(35) Pól, J.; Strohalm, M.; Havlíček, V.; Volný, M. *Histochemistry and Cell Biology* **2010**, *134*, 423.

(36) Carado, A.; Passarelli, M. K.; Kozole, J.; Wingate, J. E.; Winograd, N.; Loboda, A. V. *Analytical Chemistry* **2008**, *80*, 7921.

(37) Rabbani, S.; Barber, A. M.; Fletcher, J. S.; Lockyer, N. P.; Vickerman, J. C. *Analytical Chemistry* **2011**, *83*, 3793.

(38) Wu, K. J.; Odom, R. W. *Analytical Chemistry* **1996**, *68*, 873.

(39) Altelaar, A. F. M.; van Minnen, J.; Jiménez, C. R.; Heeren, R. M. A.; Piersma, S. R. *Analytical Chemistry* **2005**, *77*, 735.

(40) Altelaar, A. F. M.; Piersma, S. In *Mass Spectrometry Imaging*; Rubakhin, S. S., Sweedler, J. V., Eds.; Humana Press: 2010; Vol. 656, p 197.

(41) Altelaar, A. F. M.; Klinkert, I.; Jalink, K.; de Lange, R. P. J.; Adan, R. A. H.; Heeren, R. M. A.; Piersma, S. R. *Analytical Chemistry* **2006**, *78*, 734.

(42) Pacholski, M. L.; Winograd, N. *Chemical Reviews* **1999**, *99*, 2977.

(43) Wucher, A.; Tian, H.; Winograd, N. *Rapid Communications in Mass Spectrometry* **2014**, *28*, 396.

(44) Mochiji, K.; Hashinokuchi, M.; Moritani, K.; Toyoda, N. *Rapid Communications in Mass Spectrometry* **2009**, *23*, 648.

(45) Smith, D. F.; Robinson, E. W.; Tolmachev, A. V.; Heeren, R. M. A.; Paša-Tolić, L. *Analytical Chemistry* **2011**, *83*, 9552.

(46) Kozo, M. *J. Anal. Bioanal. Techniques* **2011**, S2:001.

(47) MacAleese, L.; Duursma, M. C.; Klerk, L. A.; Fisher, G.; Heeren, R. M. A. *J Proteomics* **2011**, *74*, 993.

(48) Passarelli, M. K.; Ewing, A. G.; Winograd, N. *Analytical Chemistry* **2013**, *85*, 2231.

(49) Harris, G. A.; Galhena, A. S.; Fernandez, F. M. *Analytical Chemistry* **2011**, *83*, 4508.

(50) Alberici, R. M.; Simas, R. C.; Sanvido, G. B.; Romao, W.; Lalli, P. M.; Benassi, M.; Cunha, I. B.; Eberlin, M. N. *Anal Bioanal Chem* **2010**, *398*, 265.

(51) Bereman, M. S.; Muddiman, D. C. *J Am Soc Mass Spectrom* **2007**, *18*, 1093.

(52) Costa, A. B.; Cooks, R. G. *Chemical Communications* **2007**, 3915.

(53) Costa, A. B.; Cooks, R. G. *Chemical Physics Letters* **2008**, *464*, 1.

(54) Takats, Z.; Wiseman, J. M.; Cooks, R. G. *Journal of Mass Spectrometry* **2005**, *40*, 1261.

(55) Kebarle, P. *Journal of Mass Spectrometry* **2000**, *35*, 804.

(56) Myung, S.; Wiseman, J. M.; Valentine, S. J.; Takáts, Z.; Cooks, R. G.; Clemmer, D. E. *Journal of Physical Chemistry B* **2006**, *110*, 5045.

(57) Wiseman, J. M.; Ifa, D. R.; Song, Q.; Cooks, R. G. *Angew Chem Int Ed Engl* **2006**, *45*, 7188.

(58) Ifa, D. R.; Gumaelius, L. M.; Eberlin, L. S.; Manicke, N. E.; Cooks, R. G. *Analyst* **2007**, *132*, 461.

(59) Campbell, D. I.; Ferreira, C. R.; Eberlin, L. S.; Cooks, R. G. *Analytical and Bioanalytical Chemistry* **2012**, *404*, 389.

(60) Eberlin, L. S.; Dill, A. L.; Golby, A. J.; Ligon, K. L.; Wiseman, J. M.; Cooks, R. G.; Agar, N. Y. R. *Angewandte Chemie International Edition* **2010**, *49*, 5953.

(61) Girod, M.; Shi, Y.; Cheng, J.-X.; Cooks, R. G. *Analytical Chemistry* **2011**, *83*, 207.

(62) Eberlin, L. S.; Dill, A. L.; Costa, A. B.; Ifa, D. R.; Cheng, L.; Masterson, T.; Koch, M.; Ratliff, T. L.; Cooks, R. G. *Analytical Chemistry* **2010**, *82*, 3430.

(63) Dill, A. L.; Eberlin, L. S.; Zheng, C.; Costa, A. B.; Ifa, D. R.; Cheng, L.; Masterson, T. A.; Koch, M. O.; Vitek, O.; Cooks, R. G. *Analytical and Bioanalytical Chemistry* **2010**, *398*, 2969.

(64) Wu, C.; Ifa, D. R.; Manicke, N. E.; Cooks, R. G. *Analyst* **2010**, *135*, 28.

(65) Douglass, K. A.; Venter, A. R. *Journal of Mass Spectrometry* **2013**, *48*, 553.

(66) Shin, Y.-S.; Drolet, B.; Mayer, R.; Dolence, K.; Basile, F. *Analytical Chemistry* **2007**, *79*, 3514.

(67) Basile, F.; Kassalainen, G. E.; Ratanathanawongs Williams, S. K. *Analytical Chemistry* **2005**, *77*, 3008.

(68) Lanekoff, I.; Heath, B. S.; Liyu, A.; Thomas, M.; Carson, J. P.; Laskin, J. *Analytical Chemistry* **2012**, *84*, 8351.

(69) Lanekoff, I.; Thomas, M.; Laskin, J. *Analytical Chemistry* **2014**, *86*, 1872.

(70) Nemes, P.; Woods, A. S.; Vertes, A. *Analytical Chemistry* **2010**, *82*, 982.

(71) Kiss, A.; Smith, D. F.; Reschke, B. R.; Powel, M. J.; Heeren, R. M. A. *Proteomics* **2013**, *14*, 1283.

(72) Masyuko, R.; Lanni, E. J.; Sweedler, J. V.; Bohn, P. W. *Analyst* **2013**, *138*, 1924.

(73) Rauser, S.; Marquardt, C.; Balluff, B.; Deininger, S.-O.; Albers, C.; Belau, E.; Hartmer, R.; Suckau, D.; Specht, K.; Ebert, M. P.; Schmitt, M.; Aubele, M.; Höfler, H.; Walch, A. *Journal of Proteome Research* **2010**, *9*, 1854.

(74) Solé-Domènech, S.; Sjövall, P.; Vukojević, V.; Fernando, R.; Codita, A.; Salve, S.; Bogdanović, N.; Mohammed, A.; Hammarström, P.; Nilsson, K. P.; LaFerla, F.; Jacob, S.; Berggren, P.-O.; Giménez-Llort, L.; Schalling, M.; Terenius, L.; Johansson, B. *Acta Neuropathologica* **2013**, *125*, 145.

(75) Sinha, T. K.; Khatib-Shahidi, S.; Yankeelov, T. E.; Mapara, K.; Ehtesham, M.; Cornett, D. S.; Dawant, B. M.; Caprioli, R. M.; Gore, J. C. *Nat Methods* **2008**, *5*, 57.

(76) Li, Z.; Chu, L.-Q.; Sweedler, J. V.; Bohn, P. W. *Analytical Chemistry* **2010**, *82*, 2608.

(77) Eberlin, L. S.; Liu, X.; Ferreira, C. R.; Santagata, S.; Agar, N. Y. R.; Cooks, R. G. *Analytical Chemistry* **2011**, *83*, 8366.

(78) Eberlin, L. S.; Ferreira, C. R.; Dill, A. L.; Ifa, D. R.; Cheng, L.; Cooks, R. G. *Chembiochem* **2011**, *12*, 2129.

(79) Lanni, E. J.; Masyuko, R. N.; Driscoll, C. M.; Aerts, J. T.; Shrout, J. D.; Bohn, P. W.; Sweedler, J. V. *Analytical Chemistry* **2014**, *86*, 9139.

(80) Korte, A. R.; Lee, Y. J. *J Am Soc Mass Spectrom* **2013**, *24*, 949.

(81) Tarolli, J. G.; Jackson, L. M.; Winograd, N. *Journal of the American Society for Mass Spectrometry* **2014**, *25*, 2154.

(82) Van de Plas, R.; Yang, J.; Spraggins, J.; Caprioli, R. M. *Nature Methods* **2015**, *12*, 366.

(83) González-Serrano, A. F.; Pirro, V.; Ferreira, C. R.; Oliveri, P.; Eberlin, L. S.; Heinzmann, J.; Lucas-Hahn, A.; Niemann, H.; Cooks, R. G. *Plos One* **2013**, *8*, e74981.

(84) Ryan, K. P.; Bald, W. B.; Neumann, K.; Simonsberger, P.; Purse, D. H.; Nicholson, D. N. *Journal of Microscopy* **1990**, *158*, 365.

(85) Stoeckli, M.; Staab, D.; Schweitzer, A. *International Journal of Mass Spectrometry* **2007**, *260*, 195.

(86) Kruse, R.; Sweedler, J. *J Am Soc Mass Spectrom* **2003**, *14*, 752.

(87) Chaurand, P.; Stoeckli, M.; Caprioli, R. M. *Analytical Chemistry* **1999**, *71*, 5263.

(88) Goodwin, R. J.; Nilsson, A.; Borg, D.; Langridge-Smith, P. R.; Harrison, D. J.; Mackay, C. L.; Iverson, S. L.; Andren, P. E. *Journal of Proteomics* **2012**, *75*, 4912.

(89) Casadonte, R.; Caprioli, R. M. *Nature Protocols* **2011**, *6*, 1695.

(90) Todd, P. J.; Schaaff, T. G.; Chaurand, P.; Caprioli, R. M. *Journal of Mass Spectrometry* **2001**, *36*, 355.

(91) Xu, B.; Caprioli, R.; Sanders, M.; Jensen, R. *J Am Soc Mass Spectrom* **2002**, *13*, 1292.

(92) Chaurand, P.; Caprioli, R. M. *Electrophoresis* **2002**, *23*, 3125.

(93) Shimma, S.; Furuta, M.; Ichimura, K.; Yoshida, Y.; Setou, M. *Surface and Interface Analysis* **2006**, *38*, 1712.

(94) Grove, K.; Frappier, S.; Caprioli, R. *J Am Soc Mass Spectrom* **2011**, *22*, 192.

(95) Bouschen, W.; Schulz, O.; Eikel, D.; Spengler, B. *Rapid Communications in Mass Spectrometry* **2010**, *24*, 355.

(96) Delcorte, A.; Bour, J.; Aubriet, F.; Muller, J. F.; Bertrand, P. *Analytical Chemistry* **2003**, *75*, 6875.

(97) Luxembourg, S. L.; Mize, T. H.; McDonnell, L. A.; Heeren, R. M. A. *Analytical Chemistry* **2004**, *76*, 5339.

(98) Schramm, T.; Hester, A.; Klinkert, I.; Both, J. P.; Heeren, R. M.; Brunelle, A.; Laprevote, O.; Desbenoit, N.; Robbe, M. F.; Stoeckli, M.; Spengler, B.; Rompp, A. *Journal of Proteomics* **2012**, *75*, 5106.

(99) Yang, C.; He, Z.; Yu, W. *BMC Bioinformatics* **2009**, *10*, 4.

(100) Smith, D. F.; Kharchenko, A.; Konijnenburg, M.; Klinkert, I.; Pasa-Tolic, L.; Heeren, R. M. *Journal of the American Society for Mass Spectrometry* **2012**, *23*, 1865.

(101) Fonville, J. M.; Carter, C. L.; Pizarro, L.; Steven, R. T.; Palmer, A. D.; Griffiths, R. L.; Lalor, P. F.; Lindon, J. C.; Nicholson, J. K.; Holmes, E.; Bunch, J. *Analytical Chemistry* **2013**, *85*, 1415.

(102) Pirro, V.; Eberlin, L. S.; Oliveri, P.; Cooks, R. G. *Analyst* **2012**, *137*, 2374.

(103) Veselkov, K. A.; Mirnezami, R.; Strittmatter, N.; Goldin, R. D.; Kinross, J.; Speller, A. V. M.; Abramov, T.; Jones, E. A.; Darzi, A.; Holmes, E.; Nicholson, J. K.; Takats, Z. *Proceedings of the National Academy of Sciences of the United States of America* **2014**, *111*, 1216.

(104) Deininger, S. O.; Ebert, M. P.; Futterer, A.; Gerhard, M.; Rocken, C. *Journal of Proteome Research* **2008**, *7*, 5230.

(105) Jones, E. A.; Deininger, S. O.; Hogendoorn, P. C.; Deelder, A. M.; McDonnell, L. A. *Journal of Proteomics* **2012**, *75*, 4962.

(106) Deininger, S.-O.; Cornett, D.; Paape, R.; Becker, M.; Pineau, C.; Rauser, S.; Walch, A.; Wolski, E. *Anal Bioanal Chem* **2011**, *401*, 167.

CHAPTER 3

CATEGORIZING CELLS ON THE BASIS OF THEIR CHEMICAL PROFILES: PROGRESS IN SINGLE-CELL MASS SPECTROMETRY

Notes and Acknowledgements

This chapter was published as an invited perspective in *J. Am. Chem. Soc.* **2017**, DOI: 10.1021/jacs.6b12822, coauthored by T. D. Do, S. S. Rubakhin and J. V. Sweedler. The article is adapted and reprinted here with permission from the American Chemical Society, copyright 2017. We gratefully acknowledge support from the National Institutes of Health, Award Number P30 DA018310 from the National Institute on Drug Abuse, U01 MH109062 from the National Institute of Mental Health, and the National Science Foundation, Award No. CHE 16-067915. T. J. C. acknowledges support from the NSF Graduate Research Fellowship Program, the Springborn Fellowship, and the Training Program at Chemistry-Interface with Biology (T32 GM070421).

The chemical differences between individual cells within large cellular populations provide unique information on organisms' homeostasis and the development of diseased states. Even genetically identical cell lineages diverge due to local microenvironments and stochastic processes. The minute sample volumes and low abundance of some constituents in cells hinder our understanding of cellular heterogeneity. Although amplification methods facilitate single-cell genomics and transcriptomics, the characterization of metabolites and proteins remains challenging both because of the lack of effective amplification approaches and the wide diversity in cellular constituents. Mass spectrometry has become an enabling technology for the investigation of individual cellular metabolite profiles with its exquisite sensitivity, large dynamic range, and ability to characterize hundreds to thousands of compounds. While advances in instrumentation have improved figures of merit, acquiring measurements at high throughput and sampling from large populations of cells are still not routine. This chapter highlights the current trends and progress in mass-spectrometry-based analysis of single cells, with a focus on the technologies that will enable the next generation of single-cell measurements.

Introduction

Cells are the "atomic unit" of life. Inspired by Robert Hooke's discovery of biological cells in 1665,[1] scientists, evoking the philosophical musings of Marcus Aurelius,[2] began to ponder: "The thing, what is it, fundamentally? What is its nature and substance, its reason for being?" These central questions set the framework for defining cell biology. Much of the early single-cell work relied on observations of cells with optical microscopy; current research has extended these investigations to the chemical and molecular regimes. Studies examining complex chemical questions about cells have detailed, extended, and even challenged established dogma as new measurements are made.[3–7] Much of the research emphasis has shifted from the characterization of bulk cell populations to that of individual cells, from cell types to subtypes, and from directly observing macroscopic traits to measuring single-cell genomes, proteomes, and metabolomes. While all cells share a core set of biochemical compounds, they also display an astonishing chemical diversity that allows the formation of unicellular communities and complex multicellular species. With improved analytical capabilities, morphologically homogeneous populations of cells emerge as unique, with individual characteristics and properties.[3]

Early successes of single-cell electrophoresis were reported from the 1950s to 1970s. In 1956, Edström[8] successfully determined the relative composition of ribose nucleic acids within large, mammalian neuronal cells by microphoresis with a cellulose fiber. Separation of hemoglobin from individual erythrocytes using polyacrylamide fiber electrophoresis followed in

1965.[9] Two-dimensional gel electrophroesis of proteins from single *Aplysia californica* neurons was reported in 1977,[10] around the time single-cell mass spectrometry (MS) began to develop. In their pioneering work in the 1970s, Hillenkamp and co-workers[11] used laser ablation mass analysis to generate mass spectra from tissue sections and cultured cells. They ablated several <5-µm-diameter regions on an inner-ear tissue section with a laser to obtain mass spectra containing low-molecular-weight ions at each associated laser spot.[12] As another example from the 1970s, Iliffe et al.[13] demonstrated single-cell gas chromatography−mass spectrometry of amino acids in an *Aplysia* neuron. This period also witnessed the introduction of flow cytometry and fluorescence-activated cell sorting.[14] However, it was not until 1992, when James Eberwine's group[15] demonstrated that the molecular profile of a single, potentiated CA1 neuron depends on the abundance of multiple RNAs, that the field of comprehensive single-cell chemical analysis began to take shape.

After these early seminal reports, single-cell chemical characterization approaches became more robust and provided greater information, enabling astounding advances in bioanalytical techniques that have progressively revealed single-cell heterogeneity. Interdisciplinary developments include single-cell genomics and transcriptomics,[16−19] electrochemistry,[20−22] single-molecule microscopy and spectroscopy,[23−26] nuclear magnetic resonance,[27,28] capillary electrophoresis (CE),[29−32] MS,[6,33−37] and microfluidics,[38,39] to name a few. Clearly, single-cell "omics" comprises a number of rapidly growing interdisciplinary fields. We view MS as the major analytical platform for single-cell metabolomics and proteomics (SCMP) due to its versatility, multiplexed capabilities, and relatively high throughput. Modern MS instruments provide limits of detection and analyte coverages that are suitable for non-targeted SCMP. However, effective, high-throughput single-cell sampling remains a major challenge. In fact, details related to sampling often dictate the selection of the most appropriate MS instrument and experimental protocols to use for a specific investigation.

This chapter describes recent progress in the development of MS-based analytical techniques and the attendant cell isolation approaches used for SCMP investigations. These diverse MS-based methodologies are ideally suited for the characterization of heterogeneous cellular populations through qualitative and quantitative chemical profiling of individual cells.

Setting the Stage: Mass Spectrometry Instrumentation in Single-Cell Research

MS has evolved from a gas-phase, one-dimensional analytical technique into a versatile approach that provides high mass resolution, analyte coverage, and sensitivity. Several key advances in instrumentation, combined with innovative methodologies, have set performance benchmarks for an eclectic range of MS applications (for comprehensive reviews, see refs 40 and 41). Here, we focus on the aspects of MS that make it uniquely suited to single-cell analysis.

The major challenges to single-cell chemical measurements lie in the relatively small quantity of analytes, the low volume of material, and the chemical diversity of cellular constituents. SCMP measurements are made possible by improving the sensitivity and analyte coverage of analytical techniques capable of handling the small-volume (femto-scale) samples extracted from single cells (e.g., eukaryotic cells are 5−100 μm in diameter; bacterial cells range from 0.2 to 2 μm). Small molecules, such as metabolites and lipids, are often concentrated within cells, whereas peptides, proteins, and genetic material may exist at only a few copies. Ionizing intact biomolecules requires soft MS probes that minimize molecular fragmentation.

A variety of MS methods are suitable for single-cell studies. Matrix-assisted laser desorption/ionization (MALDI) and electrospray ionization (ESI) are two robust approaches for the ionization of intact peptides and proteins from single cells. Secondary ion mass spectrometry (SIMS) utilizes a focused, accelerated primary ion beam to sputter sample surfaces and has been used for sampling from cells for several decades. While traditional primary ion beams induce molecular fragmentation, newly developed cluster ion sources can desorb and ionize intact metabolites, lipids, and small peptides. Furthermore, SIMS ionization, when performed below the static limit, causes negligible damage to sample surfaces, which permits subsequent analyses of the same samples. Lastly, the speed, sensitivity, and precision of inductively coupled plasma (ICP) MS is the foundation for mass cytometry, a prominent technique for targeted single-cell analysis.

The detection limit of an MS-based platform depends on the performance of the mass analyzer. Many modern instruments offer sufficiently high ion transmission efficiency, a wide mass range, and high mass accuracy to measure cellular content, with several commercially available MS platforms that are appropriate for SCMP measurements.[4,6,42,43] Among them, the time-of-flight (TOF) mass analyzer has been widely used in single-cell research because of its relatively low cost, large *m/z* detection window, and satisfactory performance for most MS profiling and imaging experiments, especially when fast scan rates are required. Limits of detection for TOF-MS can be below an attomole of a peptide while maintaining a mass resolution above 20 000. Spectra are acquired in tens of microseconds, though several hundred TOF spectra are frequently summed for a better signal-to-noise ratio (S/N). In "omics" work requiring high mass accuracy and mass resolution, ion cyclotron resonance (ICR)[44,45] and Orbitrap mass analyzers[46] offer superior performance. Based on the duration of the transient acquired for Fourier transformation, resolution in excess of 100 000 is routine, with an acquisition frequency of about 1 Hz. In hybrid instruments, high-resolution mass analyzers are coupled to collision cells, enabling selection of precursor ions and exact mass measurements on their fragments. Multistage fragmentation of ions (MS^n) and analysis of fragments are essential for characterization of unknowns.

Herein, we focus on the strengths, weaknesses, and future prospects of MS-based SCMP methods. From among a myriad of techniques, these were chosen to provide an overview of the field because they offer great promise for advancing single-cell research. As stated earlier, sample properties and preparation strategies oftentimes determine the appropriate MS instrument to use for a specific application. Thus, while this discussion focuses on the MS technologies, it is organized by the sampling approaches. In the first method (Figure 3.1A), intact tissue slices can be directly analyzed using imaging technologies that provide subcellular spatial resolution. Alternatively, targeted cells can be isolated from tissues (Figure 3.1B) prior to MS measurements. The success of this approach depends on prior classification of cell types and subtypes, and on the dexterity of the researcher performing the cell isolation. Finally, single-cell samples can be prepared by digesting tissues into thousands to millions of single cells (Figure 3.1C). Dissociation alleviates the stringent requirements of the first two methods and creates additional opportunities for cells to stabilize prior to analysis.

Direct Tissue Analysis: Placing Single Cells into Context

Mass spectrometry imaging (MSI), an information-rich approach for direct tissue analysis, provides unprecedented details on the chemical composition of tissue and cell specimens. Typically, an MS image is acquired by sampling a regularly spaced grid on a thin tissue section or dispersed cell population, collecting a mass spectrum at each spot. MSI is an attractive option when determining the spatial context of individual cells within tissues is important, or when single-cell isolation is not feasible. Different MS ionization methods facilitate the successful analysis of numerous biochemical classes, including proteins, small peptides, lipids, and metabolites (Figure 3.2). MALDI-MSI (Figure 3.2A) is the most common technique used in tissue imaging. A recent review by Römpp and Spengler[47] highlights several successful studies in which MALDI-MSI provided detailed histological information on phospholipids, drug molecules, neuropeptides, and tryptic peptides at (or close to) the single-cell level. While 10−35-μm pixel widths are common, MALDI-MSI at 3 μm spatial resolution was performed on the lateral ventricle region of a coronal mouse brain section to image phospholipids.[47]

MSI at nanometer resolution can be achieved by SIMS imaging, which employs a tightly focused, accelerated primary ion beam for desorption and ionization (Figure 3.2B). SIMS is suitable for mapping elements, metabolites, small molecules, lipids, and peptide fragments at subcellular resolution (for a review on the fundamentals of SIMS, see Boxer et al.[48]). Several primary ion beams are suitable for biological analyses. High-energy and reactive sources may provide sufficient ion current to afford submicron spatial resolution but tend to fragment the chemical bonds of larger molecules.[48] Ostrowski et al.[49] utilized an indium liquid metal ion beam focused to 200 nm to examine the plasma membrane of *Tetrahymena*. The images revealed a decrease in abundance of phosphatidylcholine and an increase in aminoethylphosphonolipid at highly curved fusion pores, which are utilized during cell mating. Subsequent to this report, a variety of cell types have been analyzed by SIMS imaging,[50–54] providing subcellular distributions of lipids, metabolites, and small molecules.

Elemental secondary ions can be characterized by a magnetic sector analyzer equipped with up to seven detectors set to particular *m/z* values - a technique referred to as nanoSIMS. State-of-the-art nanoSIMS is quantitative, can achieve spatial resolution <50 nm, and allows 3D chemical mapping. NanoSIMS has been applied for subcellular-resolution imaging of metabolic pathways, interacting microorganisms, and microbial communities.[34,55–59] The main drawbacks are relatively low sample throughput and the high cost of isotope-labeled substrates. Nevertheless, the clever use of isotopes allows nanoSIMS to interrogate the 3D composition of representative cell subtypes.

Recent developments with polyatomic and cluster ion sources have expanded the biochemical coverage of SIMS by allowing direct measurement of intact molecules below *m/z* 2000. The cluster ion sources achieve primary ion beam diameters approaching 1 μm, equivalent to high-resolution MALDI sources.[60] Complementary MS imaging, non-MS analyses,[61] and matrix-enhanced reagents[62–64] have been incorporated to improve molecular coverage and quantitation of SIMS imaging. Aspects of the sample preparation pipeline contribute significantly to the spatial integrity of measured molecular distributions. SIMS is especially sensitive to minute amounts of environmental contamination, as analysis is restricted to the topmost layer of the surface. While primary ion beams may be focused to tens of nanometers, obtaining such high spatial resolution is still extremely challenging.

Most MSI experiments are non-targeted and label free, but at the pixel widths required for subcellular imaging, only abundant compounds will be detectible. Imaging mass cytometry (Figure 3.2C), can improve the limits of detection for specific compounds by using affinity-based probes to selectively localize target antigens. As a direct analog to immuno-gold staining used with electron microscopy, imaging mass cytometry couples metal-conjugated antibodies developed for mass cytometry with a laser or ion beam, allowing antigen localization in tissue sections and individual cells. Giesen et al.[65] used imaging mass cytometry with a high spatial resolution laser ablation system to localize 32 proteins and posttranslational modifications (PTMs) at 1 μm resolution to delineate cell heterogeneity in human breast cancer tissue sections.

Angelo et al.[66] adapted the mass cytometry pipeline to SIMS imaging, effectively improving the spatial resolution of the method to 50 nm. The chelated metal isotope adducts generated secondary ions, which were analyzed via a magnetic sector mass spectrometer equipped with multiple detectors. The technique, referred to as multiplexed ion beam imaging (MIBI), was successfully applied to human breast cancer samples to reveal tumor immunophenotypes. The current acquisition rate for MIBI is 2 h for a 0.250 mm^2 field-of-view for 10 distinct targets.[66]

Rastering the desorption probe over large areas, as in MSI, effectively analyzes each cell, but does so at the expense of throughput and considerable cost in instrument time and assay sensitivity. At the Nyquist frequency to resolve individual cells, each cell should be sampled at least four times; this divides the cellular analytes among each pixel and may cause some compounds to fall below the limit of detection. Still, the drive to acquire higher resolution MS images has spurred the development of improved ion beam optics, sensitive mass analyzers, and optimized sample preparation protocols. We expect instrument capabilities will continue to progress and cellular resolution will become standard in commercial MALDI-MS instrumentation over the next few decades. A limitation to the continued development of smaller pixel sizes is the absolute abundance of compounds within a given region. A 1-µm pixel contains just over 1% of the area of a 10-µm diameter cell, requiring analyte concentrations 2 orders of magnitude higher to be observable in a single pixel as opposed to the entire cell. Compounding this effect for MALDI-MS is the compromise between analyte extraction and delocalization during matrix application.

Imaging mass cytometry circumvents these issues with the application of rare-earth-labeled antibodies. Each antibody holds several hundred isotope atoms, which amplifies the signal from a single binding event. The shortcomings of mass cytometry imaging are inherited from affinity labels: the a priori selection of antigens, cost of generating antibodies, and limited plexity (though not as severe as fluorescence probes). We envision mass cytometry imaging experiments will be performed on a tissue section following non-targeted MSI acquisition, similar to work performed with immunohistochemistry. Such an experiment could place the non-targeted data into the context of more traditional cell subtyping to improve biomarker identification. As subcellular MSI resolution becomes more widespread, the distinction between imaging and single-cell analysis will be less pronounced. The capability to examine each cell within its native environment would revolutionize medical, pharmaceutical, and fundamental research.

Specific Cell-Type Targeting: Meeting the Needs for Separation and Quantitation

When molecular characterization is the paramount experimental objective, measurements that do not provide spatial information can be undertaken. Additional analytical dimensions, such as separation and quantitation, can be coupled with MS to enable information-rich single-cell measurements. CE is a qualitative and quantitative technique used in analyses of single cells and subcellular compartments. It features rapid analyte separations based on the electrophoretic mobility of molecules, including those with the same molecular weights (e.g., diastereomers), with high resolving power and low sample consumption (a microliter or less).[31,32] Many aspects of CE have greatly progressed in recent decades, and include the development of advanced separation modes and nanoscale sampling, and the interface of CE with different detection methods.[30,67] While CE is powerful on its own, it is even more productive when coupled with optical, electrochemical, or MS-based detection. For example, CE-MS provides a label-free and unique characterization method for investigation of endogenous biomolecules in complex cellular mixtures (Figure 3.3). Hyphenating CE with other detection modalities, such as laser-induced fluorescence, allows targeted cell analysis based on chemical signatures, but those approaches are limited to molecules with native fluorescence and those that can be tagged with a fluorophore via derivatization chemistry.[67] Single-cell metabolomics studies using CE-ESI-MS have demonstrated detection limits for molecules in the low nanomolar range, high-efficiency separations, and increased analyte coverage. The injection of only 0.1% of the total content from a single *Aplysia californica* metacerebral cell (150 μm in diameter) yielded unambiguous detection of more than 100 compounds.[68] Preconcentration methods further improve analyte coverage, especially when initial concentrations of extracted analytes are below the detection limits of MS systems.[30,42] Improvements in sheathless CE-MS interfaces have allowed investigation of complex bioanalytical problems, as in the characterization of protein isoforms and combinatorial PTMs reported by Yates and co-workers.[69,70] Recent examples from Dovichi[71] and Nemes[72,73] of the developing *Xenopus laevis* embryo demonstrate the great promise for CE-MS-based single-cell proteomics.

Though capable of sensitive, quantitative analysis, a limitation of CE is its low throughput. Even a state-of-the-art CE platform operates at a rate of less than one cell per minute.[74,75] Typical separations, performed in longer capillaries, can last between 5 and 60 min to achieve optimal resolution; however, chip-based CE devices do increase throughput. Moreover, the duration of a set of experiments may be constrained by the endurance of intact cells within a physiological solution prior to analysis (a few hours), which ultimately limits the number of cells that can be assayed from one population.[74] Further constraining throughput, each sample and target analyte requires an optimal set of CE conditions, including background electrolyte, chiral selectors, pH, separation voltage, and temperature, among others.

To increase throughput, researchers have focused on the development of automated cell-handling modules that are compatible with a wide range of background electrolytes and analyte classes. CE columns can be embedded in, or coupled with, microfluidic devices that permit fluorescence-activated cell sorting (FACS) and automatic cell trapping, culturing, sorting, and lysis prior to CE separation. Higher peak capacities are achieved by combining multiple capillary columns in series to provide complementary separation dimensions. Examples include the velocity gap mode, which manipulates the electrical fields on connected capillaries with conductivity detection at the joint,[76] and 2D-CE, which employs orthogonal separation conditions in connected capillaries.[77]

Recent advances in CE have overcome technical hurdles for the detection and separation of chiral molecules, such as D/L-amino acids and peptide diastereomers,[78] at a resolution and sensitivity that is currently inaccessible by other label-free, MS-coupled mobility spectrometry or spectroscopy approaches.[79] Furthermore, these molecules are separated non-destructively with minimal loss, which is another advantage of CE over MS-based separation methods. In addition, performing the separation post-ionization can introduce additional complexity due to the formation of protomers (molecular isomers that differ only in the site of protonation).[80] However, many conditions used in chiral separations have yet to be made compatible with ESI-MS, awaiting future optimization.

Owing to its superb sensitivity and prospects for high throughput, CE-MS has become a method of choice for separation-based, quantitative analyses of single cells. Compared to other single-cell techniques, CE-MS applications that directly introduce cells into the capillary for lysis and separation reduce the time between cell rupture and analyte characterization. Such rapid analyses limit unwanted side reactions and degradation that lead to non-specific profile variations. The future of high-throughput CE-MS offers a unique approach to classify cell types and identify new subtypes, which will provide complementary profiles to other methods.

Dissociated and Cultured Cell Samples: Searching for Cell Subtypes and Rare Cells

In the final approach discussed here, cells are either separated from tissue sections by dissociation or cultured. Once in solution, cells may be labeled for mass cytometry, or deposited onto a surface for single-cell profiling. The native connections between cells in the tissue are dismantled and extraction is more limited than with specific cell isolation, but dissociated cell measurement approaches can have a higher per-cell throughput than the MS methods described above.

Mass cytometry is one of the most versatile MS-based techniques for multiplexing single-cell measurements on an "omics" scale. As briefly mentioned when discussing MSI, mass cytometry operates much like flow cytometry, in which fluorescently labeled markers, including antibodies, are used to characterize the presence of a panel of antigens in large populations of individual cells. However, instead of fluorescence labels, mass cytometry uses rare earth metal isotope tags with high plexity (Figure 3.4A). The binding of the conjugates to molecular targets is quantified with an inductively coupled plasma (ICP)-MS instrument. The ICP torch completely consumes the cells while atomizing sample droplets, which provides low background and elimination of matrix effects.[81] The throughput of mass cytometry is currently limited by the lifetime of analytes in the ion cloud (~300 μs),[82] which allows measurement of up to 1000 cells per second.[83] This throughput is several-fold higher than that offered by imaging mass cytometry but comes at the expense of information on tissue organization. Most mass cytometers are coupled to TOF mass analyzers (e.g., the commercialized CyTOF) as they are capable of rapid acquisition times (13 μs per scan) and allow 20–30 scans per cell.[84] Additional DNA stains are used to discriminate cellular events from debris and distinguish single cells from doublets or aggregates of cells. Metal calibration beads are also spiked into each sample to serve as internal standards.[85] Cell-based multiplexing methods, such as mass-tag cell barcoding,[86,87] can be utilized to reduce antibody consumption, acquisition time, and eliminate cell-to-antibody ratio-dependent effects.[81,82] For example, a binary barcoding can utilize n rare metal isotopes to uniquely label 2^n individual cell samples before they are mixed, stained, and analyzed in one batch.[88] Mass cytometry has assisted the discovery of complex aspects of single-cell chemistry, including different stages of the cell cycle, phenotypes and signaling responses, cytokine expression, and cell viability.[82,88–92]

Cell surface markers, the degree of expression, and PTM events can be used to identify cellular phenotypes and distinguish cell populations. For example, a single-cell mass cytometry study using 31 distinct transition and rare earth metal isotopes to label two antibody staining panels revealed 24 distinct immune cell populations in bone marrow during hematopoiesis.[89] Currently, mass cytometry surpasses other MS-based single-cell techniques in the total number of analyzed cells per experiment. Newell et al.[93] combined mass cytometry with combinatorial peptide-major histocompatibility complex staining to analyze samples of 84 million T-cells for distinct phenotypes and their ability to recognize viral epitopes.

A technical inefficiency of mass cytometry lies in the nebulization of single cells, which stochastically loses approximately 70% of the cells in the process of forming droplets.[81] Although this loss does not inherently introduce a significant sampling bias, improvements in cell introduction efficiency would reduce cell consumption. The sensitivity of mass cytometry is greatly affected by the loading of metal atoms on each antibody. The metal chelating chemistry facilitates a maximum of ~100 metal reporter ions per antibody molecule.[81] Mass cytometry can seamlessly measure 58 or more different parameters simultaneously, though this requires a priori knowledge about the cells and well-defined molecular targets with specific antibodies. The limited number of commercially available rare metal isotopes also limits the number of antigens that can be measured simultaneously. Currently, 37 stable lanthanide isotopes that are compatible with metal chelating chemistry are available at sufficient purity.[82] While antibodies can recognize a wide range of antigens, mass cytometry is less effective for smaller molecules, such as metabolites and peptides, which may not be accessible to antibodies or cross-linked by fixation. These molecules can be specific biomarkers for disease-transformed cells.[94] Therefore, the complexity of multidimensional single-cell analysis is another area worth improving,[95] including new affinity agents that can bind small-molecule metabolites.

Mass cytometry is poised to extend the capabilities of many immunofluorescence methods beyond the limitation of fluorescence spectral overlap. In a clinical setting, the rapid and accurate quantification of numerous biomarkers can facilitate deeper subtyping of tissue sections or biopsy samples. Though mass cytometry requires preselection of antigens, it should continue to find application in targeted cell population profiling. While mass cytometry can profile cellular states at given points in time with high throughput and plexity, an important caveat is that cells are destroyed by the ICP torch, preventing follow-up characterization of selected cellular subtypes.

A distinct non-targeted approach involves dispersing cells onto sample surfaces where they are analyzed with an MS microprobe. In contrast to MSI, the contents of one cell are completely sampled during a single analysis. Manual placement of cells is a low-throughput implementation of this type of handling.[43] A higher throughput method is to disperse cells sufficiently such that no neighbors are within the microprobe region. With the correct choice of seeding density, separated cells greatly relax instrumental sampling requirements and allow more stringent extraction procedures, further increasing analyte sensitivity. As described below, two methods of dispersed cell sampling have been developed recently for MALDI-MS analysis of single cells, one based on constrained cell positions and the other on randomly seeding the cells.

The first cell-dispersed approach involves constraining the cell positions. A variety of microfluidic constructs are available for trapping single cells for subsequent high-throughput analysis. Microdroplet arrays can systematically trap single cells in microwells, allowing subsequent profiling by ESI-MS.[96] The sensitivity of the trapping depends on the ratio of the diameters of the cell and the microwell, limiting the sizes of analyzed cells. The current implementation also requires manual sampling of each well. For high throughput sampling, Zenobi and co-workers[97] developed an omniphobic, patterned surface specifically for constraining microdroplets of MALDI matrix solution, called microarrays for MS (MAMS) (Figure 3.4B). By depositing cells into these microwells, their contents remain isolated due to the omniphobic microarray walls. This isolation allows the application of more rigorous extraction methods, such as shock freezing,[98] as analytes neither severely dilute nor become contaminated by nearby cells. Cell deposition in MAMS is achieved by a variety of methods, including piezoelectric printing of cellular solutions[99] or submerging the surface in a cell solution.[97,100] Each well contains a variable number of cells described by a Poisson distribution.[98] As such, with a cell concentration generating the maximum probability of wells containing one cell (average, $\lambda = 1$), approximately 37% of wells are occupied by one cell. Another 37% of the wells are empty, with the remaining 26% containing two or more cells. Orthogonal methods, such as optical microscopy, can enumerate the cell counts in each well. Once cell number and positions are determined, cellular analytes are extracted and samples are coated with MALDI matrix. The contents are analyzed by simply collecting spectra at each predetermined point in a regular array. Unlike subcellular MSI, the required positional accuracy and laser spot size are easily achieved by most commercial instruments.

Using this methodology, the metabolic profiles of several single-celled microorganisms were investigated, showing quantities of nucleoside di- and triphosphates, as well as lipids unique for each species, with concentrations proportional to the number of cells within a given well; Raman spectra were also obtained and correlated with a given microwell.[97] Further experiments correlated fluorescence and Raman microspectroscopy acquired from the freshwater algae *Haematococcus pluvialis* and combined the images with MS measurements to discriminate between encystment stages.[100] In addition, using *Saccharomyces cerevisiae* as a model organism, Zenobi and coworkers[98] investigated the metabolic consequences of environmental and genetic perturbations on several metabolites, recapitulating population-level changes and discriminating genotypic differences.

Advantages of MAMS include the capabilities to thoroughly extract analytes from deposited cells and ensure each sample is isolated from nearby cells, limiting cross contamination. However, the efficiency for random seeding is low (only 37% of wells contain single cells) and the spatial constraints of the microwells limit investigations of long-range cellular outgrowth and changes related to cell-to-cell signaling. Theoretically, MAMS could facilitate studies of interactions between small cell populations. With conventional random seeding or printing, the likelihood of two cells from each of two populations occupying the same well is 0.372 = 14%; however, the cases when a well is occupied by more than one cell of each type are also interesting. This would allow investigations into the competition between malignant and immune cells for small populations of each, generating a large, random assortment of populations on a single device. FACS could also be used as an enabling, selective cell deposition technology coupled to MAMS. Precise seeding of specific, preselected phenotypes could construct complex cell distributions to allow full utilization of each MAMS device.

An alternative method for high throughput analysis of isolated, individual cells involves randomly dispersing them on a surface, and using optical microscopy to precisely locate the dispersed cells on a transparent indium tin oxide-coated glass slide.[101] Suspensions of cells are deposited onto conductive surfaces and the cells allowed to attach to the substrate. High-contrast, fluorescence images of a nuclear stain deliver a simple data set to locate individual cells. Registration of the microscopy image with the mass spectrometer coordinate system provides the location of each selected cell. Once MALDI matrix is applied, the laser is positioned over each cell in turn and a spectrum acquired (Figure 3.4C). In the initial report, microscopy-guided single-cell MALDI-MS was coupled to principal component analysis-based outlier detection to perform an unsupervised analysis in a population of dispersed pituitary cells. Several peptides were detected at high S/N from individual pituitary cells, including arginine vasopressin, oxytocin, and α-melanocyte-stimulating hormone. Additional MS profiling of cells from pancreatic islets of Langerhans demonstrated single-cell sensitivity to canonical peptide hormones, including intact insulin, glucagon, pancreatic polypeptide, and somatostatin. In a follow-up study on single islet cells,[102] the levels of peptide hormones were used to classify cells into traditional histological classes, showing good agreement with previous reports. Furthermore, cell-type-specific peptide heterogeneity was compared between the dorsal- and ventral-derived islets, with results indicating an increased abundance of processed pancreatic polypeptide within ventral-derived γ-cells. The peptides were not previously observed endogenously, and the anatomical heterogeneity in peptide processing would be difficult to detect with bulk measurements.

Successful analyte profiling using microscopy-guided MALDI-MS largely depends on accurate cell positioning under the laser probe, requiring the ability to locate a 10-µm cell over a ~20 cm^2 microscope slide. Assuming a random seeding, the probability of individual cells being sufficiently far apart is determined by a spatial Poisson point process, which has the same form as a Poisson distribution. Again, at ideal conditions, only 37% of the seeded cells will be sufficiently spaced for analysis, but there is a relatively large area available for seeding. As such, the total number of cells analyzed in a given footprint will be larger than with reported examples of MAMS. Furthermore, long-range interactions should be easier to observe, as there is no physical barrier between cells. Coupling with FACS may be more difficult, as the cells in droplets impacting the surface could migrate without being confined in omniphobic wells.

While both high throughput studies described above used MALDI-MS, these methodologies could be adapted to work with other microprobe-based MS analyses such as DESI, SIMS, or liquid microjunction probes.[103]

An exciting aspect of dispersed-cell methods is the ease with which they can be coupled with complementary analytical methods, e.g., combining with optical microscopy to count the number of cells in each MAMS well or locate cell bodies. A clear extension of the methodology is the use of exogenous or endogenous probes or reporters to provide pre-MS subtyping of cells. For example, transfection of cells with fluorescent probes could simplify rare cell detection within a population. Any spatially localized analytical technique capable of sampling from a surface is readily adapted to provide additional information on analyzed cells. Vibrational microscopy, a nondestructive profiling method, could be used to generate further information on cellular contents. Additional MS experiments are also possible, if performed in the correct order. Unlike MSI, the data sets are easily combined based on the unique cell location, greatly simplifying data fusion. For sample preparation, we expect to see FACS utilized in more powerful and efficient seeding setups. Precise deflection of cell-containing droplets would allow placement of suitable numbers of cells at evenly spaced intervals. Combined with appropriate molecular biology and pharmacology tools, interactions between different cell types could be assayed, as described earlier.

Finally, an intriguing aspect of MALDI-MS is that only a small fraction of the cell is consumed for analysis.[104] Material remaining on the substrate is available for subsequent, follow up analysis by tandem MS or other methods on the same cell. The prospects are especially exciting for the integration of MALDI-MS-based profiling with orthogonal analytical and biochemical approaches. High-throughput MALDI-MS could provide a non-targeted, label-free profile of thousands of cells within a population. Utilizing multidimensional analysis on such a data set would facilitate the selection of individual cells that are representative of a given subclass. Focusing subsequent assays on the characteristic cells would reduce the number of analyses required to practically characterize an entire population. For instance, preselecting cells with MALDI-MS would greatly enhance the effective throughput of CE or single-cell transcriptomics by targeting cells that provide the most information on the population composition.

Outlook and Concluding Remarks

Mass spectrometry is an information-rich analytical technology, positioned at the forefront of single-cell metabolomics, peptidomics, and proteomics. Progress thus far has been impressive. Current-generation instruments display exquisite sensitivity for the multiplexed, label-free measurement of hundreds of biomolecules from cellular samples. With careful sample preparation, analyte separation, and/or labeling, relative and absolute quantitative MS analysis of single cells becomes feasible. Issues with single-cell investigations arise from sampling, during the transition from organism to the instrument. Manual sample manipulation is suitable for detailed analysis of a small subset of cells;[4,105] however, this sampling approach is less applicable for the characterization of large-scale cellular heterogeneity in complex structures. Automatic profiling of an entire tissue section by MSI can collect spectra from thousands of cells, but has not solved issues related to matrix effects and subdividing cell contents. In contrast, representative populations of dispersed cells may be seeded on surfaces for microprobe-based MS analysis. By physically separating cells, MALDI matrix application can be optimized to improve analyte extraction and limit matrix effects from nearby cells, allowing the identification of rare individuals within a population. Sample throughput is enhanced over MSI, albeit at the cost of locational context within the native tissue. Each method offers a unique set of performance characteristics that are suitable to approach a given biological question.

Beyond more advanced instrumentation, a key shortcoming to the methods discussed herein is their limited utilization outside of MS research groups. Mass cytometry is gaining momentum as an alternative to flow cytometry by providing rapid, quantitative assessments of hundreds of antigens at a rate of thousands of cells per hour. These targeted methods, together with label-free MS analyses, greatly enhance the capabilities of SCMP-MS for discovery and hypothesis-driven investigations. Wider acceptance of single-cell MS technologies as practical analytical methods will broaden the breadth of questions addressed by SCMP-MS and facilitate its further integration with more routine genomics and transcriptomics approaches. Streamlining the workflows and simplifying data interpretation will encourage further acceptance by a wider multidisciplinary user base.

Willard Quine once said, “Physics investigates the essential nature of the world, and biology describes a local bump.”[106] The advent of single-cell MS created an opportunity to explore changes in “local bumps” at a finer resolution than ever before. Through interdisciplinary investigations, we are beginning to discover the low-abundance cellular minorities in homogeneously bulk populations of cells that may cause drastic phenotypic changes. Sampling techniques that provide high throughput, high spatial and/or temporal resolution, and broad molecular coverage enable the determination of individual cellular properties while discriminating between unusual cell profiles and statistical noise. The body of work produced in SCMP, aligned with results gathered by transcriptomics and genomics, allows detailed understanding of changes occurring in individual cells during normal and pathological states, with promising applications in medicine.

Figures

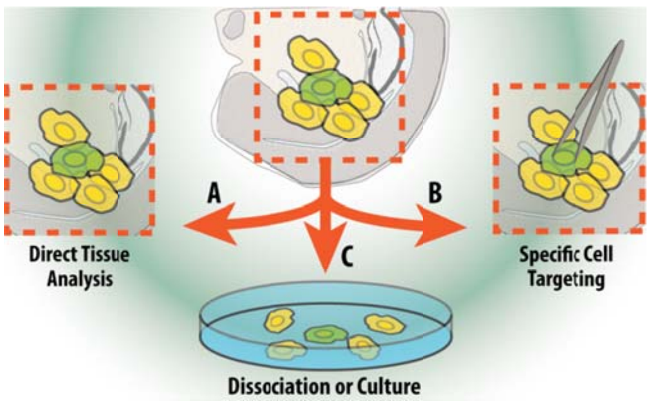


Figure 3.1. Overview of the single-cell sampling methods covered in this chapter. (A) Tissue may be sectioned and mounted on a suitable surface for imaging native distributions of analytes. (B) Specific large cells can be isolated from tissue for subsequent analysis. (C) Cells from tissue may be dissociated or cultured in growth medium.

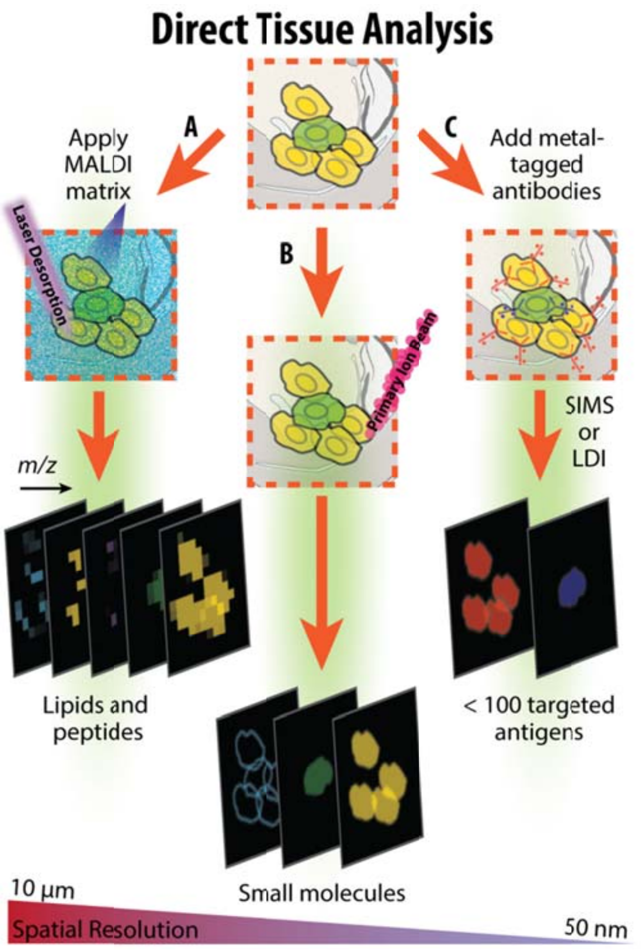


Figure 3.2. Several MSI methods obtain single-cell resolution. (A) Application of a matrix is required for MALDI-MSI and must be optimized to maintain native spatial distributions. While spatial resolution is poorer than with SIMS, MALDI ionization is much softer, such that intact lipids and peptides are detectable. (B) SIMS provides the highest spatial resolution with focused primary ion beams but is limited in analyte coverage, typically detecting fragment ions and small compounds. (C) Imaging mass cytometry is capable of targeted localization of protein antigens with resolutions similar to SIMS.

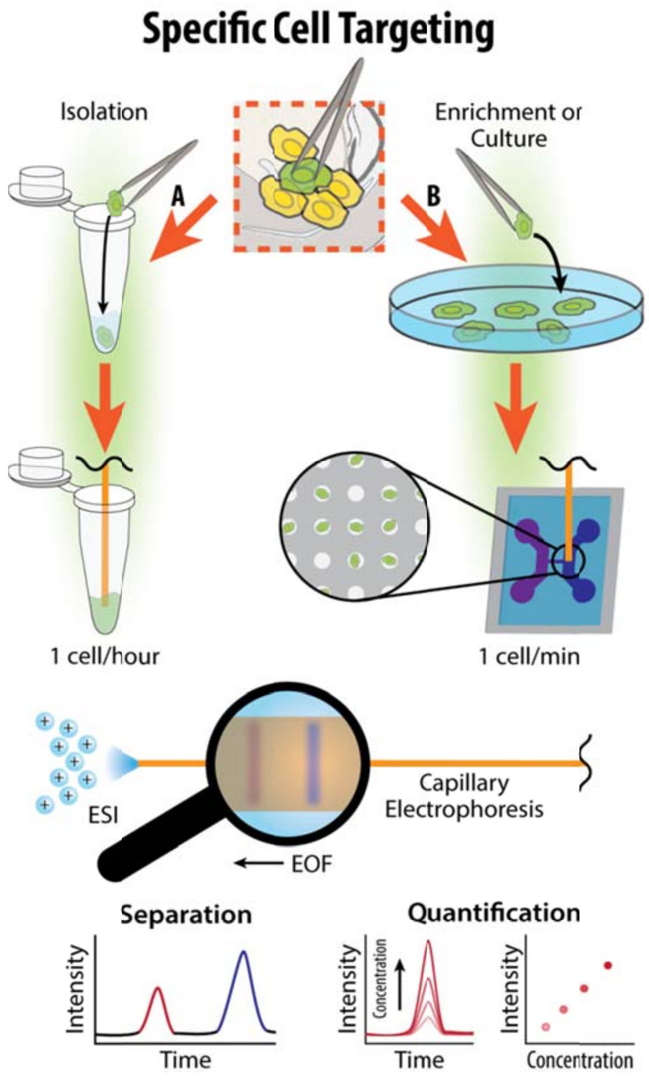


Figure 3.3. Illustration of an experimental workflow utilizing CE-MS to separate and quantify endogenous molecules in single cells. Specific cell types are either (A) isolated from tissue manually or (B) chemically labeled and sorted by microfluidic devices. Each cell is homogenized or lysed, and its content is subjected to CE-MS separation and quantitation.

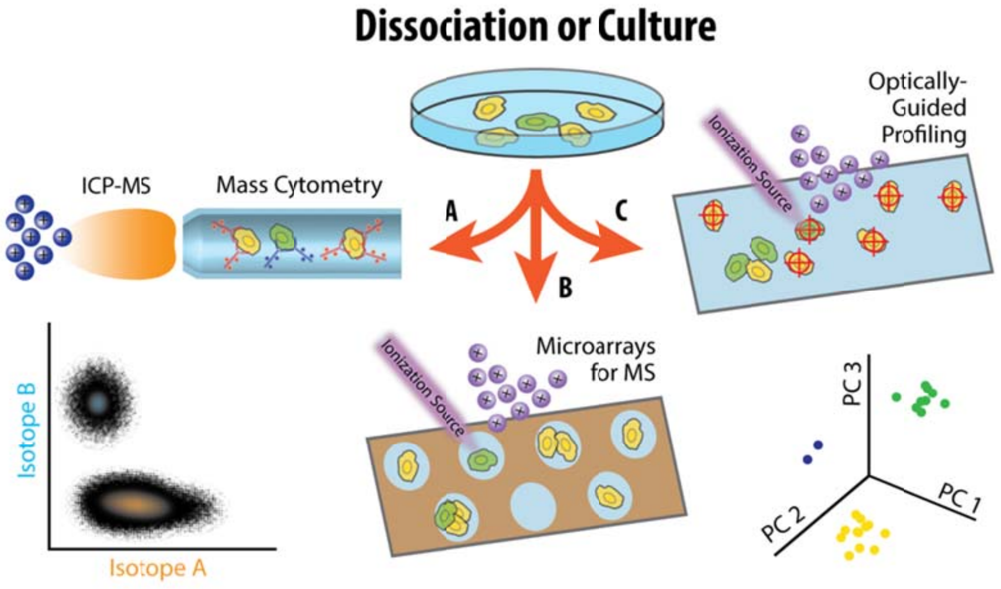


Figure 3.4. Analysis of dissociated or cultured cells provides the highest throughput of any SCMP-MS method. (A) Mass cytometry uses rare earth metal-labeled affinity tags to quantitatively measure up to hundreds of preselected antigens. The current throughput is ~1 kHz and data can be visualized with traditional cytometry plots or multivariate analysis. Dissociated cells can also be attached to surfaces for MALDI-MS profiling within (B) microarrays for MS or randomly seeded and targeted by (C) optically guided profiling.

References

(1) Hooke, R. *Micrographia: or, Some physiological descriptions of minute bodies made by magnifying glasses. With observations and inquiries thereupon*; J. Martyn and J. Allestry: London, 1665.

(2) Aurelius, M. *Meditations: A new translation*; Modern Library: New York, 2002.

(3) Ackermann, M. *Nat. Rev. Microbiol.* **2015**, *13*, 497−508.

(4) Nemes, P.; Rubakhin, S. S.; Aerts, J. T.; Sweedler, J. V. *Nat. Protoc.* **2013**, *8*, 783−799.

(5) Zenobi, R. Science **2013**, *342*, 1243259.

(6) Rubakhin, S. S.; Romanova, E. V.; Nemes, P.; Sweedler, J. V. *Nat. Methods* **2011**, *8*, S20−29.

(7) Chen, X.; Love, J. C.; Navin, N. E.; Pachter, L.; Stubbington, M. J.; Svensson, V.; Sweedler, J. V.; Teichmann, S. A. *Nat. Biotechnol*. **2016**, *34*, 1111−1118.

(8) Edström, J.-E. *Biochim. Biophys. Acta* **1956**, *22*, 378−388.

(9) Matioli, G. T.; Niewisch, H. B. *Science* **1965**, *150*, 1824−1826.

(10) Ruchel, R. *J. Chromatogr*. **1977**, *132*, 451−468.

(11) Kupka, K. D.; Schropp, W. W.; Schiller, C.; Hillenkamp, F. *Scan. Electron Microsc.* **1980**, 635−640.

(12) Meyer zum Gottesberge-Orsulakova, A.; Kaufmann, R. *Scan. Electron Microsc.* **1985**, 393−405.

(13) Iliffe, T. M.; McAdoo, D. J.; Beyer, C. B.; Haber, B. *J. Neurochem.* **1977**, *28*, 1037−1042.

(14) Herzenberg, L. A.; Parks, D.; Sahaf, B.; Perez, O.; Roederer, M.; Herzenberg, L. A. *Clin. Chem.* **2002**, *48*, 1819−1827.

(15) Mackler, S. A.; Brooks, B. P.; Eberwine, J. H. *Neuron* **1992**, *9*, 539−548.

(16) Shapiro, E.; Biezuner, T.; Linnarsson, S. *Nat. Rev. Genet*. **2013**, *14*, 618−630.

(17) Wang, Y.; Navin, N. E. *Mol. Cell* **2015**, *58*, 598−609.

(18) Saliba, A. E.; Westermann, A. J.; Gorski, S. A.; Vogel, J. *Nucleic Acids Res*. **2014**, *42*, 8845 −8860.

(19) Eberwine, J.; Sul, J. Y.; Bartfai, T.; Kim, J. *Nat. Methods* **2014**, *11*, 25−27.

(20) Clausmeyer, J.; Schuhmann, W. *TrAC, Trends Anal. Chem.* **2016**, *79*, 46−59.

(21) Li, X.; Dunevall, J.; Ewing, A. G. *Acc. Chem. Res.* **2016**, *49*, 2347−2354.

(22) Kang, M.; Momotenko, D.; Page, A.; Perry, D.; Unwin, P. R. *Langmuir* **2016**, *32*, 7993−8008.

(23) Lord, S. J.; Lee, H. L.; Moerner, W. E. *Anal. Chem.* **2010**, *82*, 2192−2203.

(24) Xia, T.; Li, N.; Fang, X. *Annu. Rev. Phys. Chem.* **2013**, *64*, 459−480.

(25) Konig, I.; Zarrine-Afsar, A.; Aznauryan, M.; Soranno, A.; Wunderlich, B.; Dingfelder, F.; Stuber, J. C.; Pluckthun, A.; Nettels, D.; Schuler, B. *Nat. Methods* **2015**, *12*, 773−779.

(26) Wang, X.; Li, X.; Deng, X.; Luu, D. T.; Maurel, C.; Lin, J. *Nat. Protoc.* **2015**, *10*, 2054−2063.

(27) Rubakhin, S. S.; Lanni, E. J.; Sweedler, J. V. *Curr. Opin. Biotechnol.* **2013**, *24*, 95−104.

(28) Webb, A. *Anal. Chem.* **2012**, *84*, 9−16.

(29) Huang, W.-H.; Ai, F.; Wang, Z.-L.; Cheng, J.-K. *J. Chromatogr. B: Anal. Technol. Biomed. Life Sci.* **2008**, *866*, 104−122.

(30) Frost, N. W.; Jing, M.; Bowser, M. T. *Anal. Chem.* **2010**, *82*, 4682−4698.

(31) Kleparnik, K. *Electrophoresis* **2013**, *34*, 70−85.

(32) Zhong, X.; Zhang, Z.; Jiang, S.; Li, L. *Electrophoresis* **2014**, *35*, 1214−1225.

(33) Passarelli, M. K.; Ewing, A. G. *Curr. Opin. Chem. Biol.* **2013**, *17*, 854−859.

(34) Watrous, J. D.; Dorrestein, P. C. *Nat. Rev. Microbiol.* **2011**, *9*, 683−694.

(35) Musat, N.; Foster, R.; Vagner, T.; Adam, B.; Kuypers, M. M. FEMS *Microbiol. Rev.* **2012**, *36*, 486−511.

(36) Ong, T. H.; Tillmaand, E. G.; Makurath, M.; Rubakhin, S. S.; Sweedler, J. V. *Biochim. Biophys. Acta, Proteins Proteomics* 2015, 1854, 732−740. (37) Spengler, B. Anal. Chem. **2015**, *87*, 64−82.

(38) Sanchez-Freire, V.; Ebert, A. D.; Kalisky, T.; Quake, S. R.; Wu, J. C. *Nat. Protoc.* **2012**, *7*, 829−838.

(39) Yin, H.; Marshall, D. *Curr. Opin. Biotechnol.* **2012**, *23*, 110−119.

(40) Glish, G. L.; Vachet, R. W. *Nat. Rev. Drug Discovery* **2003**, *2*, 140−150.

(41) Maher, S.; Jjunju, F. P. M.; Taylor, S. *Rev. Mod. Phys.* **2015**, *87*, 113−135.

(42) Liu, J. X.; Aerts, J. T.; Rubakhin, S. S.; Zhang, X. X.; Sweedler, J. V. *Analyst* **2014**, *139*, 5835−5842.

(43) Rubakhin, S. S.; Garden, R. W.; Fuller, R. R.; Sweedler, J. V. *Nat. Biotechnol.* **2000**, *18*, 172−175.

(44) Marshall, A. G.; Hendrickson, C. L.; Jackson, G. S. *Mass Spectrom. Rev.* **1998**, *17*, 1−35.

(45) Nikolaev, E. N.; Kostyukevich, Y. I.; Vladimirov, G. N. *Mass Spectrom. Rev.* **2016**, *35*, 219−258.

(46) Eliuk, S.; Makarov, A. *Annu. Rev. Anal. Chem.* **2015**, *8*, 61−80.

(47) Römpp, A.; Spengler, B. *Histochem. Cell Biol.* **2013**, *139*, 759−783.

(48) Boxer, S. G.; Kraft, M. L.; Weber, P. K. *Annu. Rev. Biophys.* **2009**, *38*, 53−74.

(49) Ostrowski, S. G.; Van Bell, C. T.; Winograd, N.; Ewing, A. G. *Science* **2004**, *305*, 71−73.

(50) Fletcher, J. S.; Rabbani, S.; Henderson, A.; Blenkinsopp, P.; Thompson, S. P.; Lockyer, N. P.; Vickerman, J. C. *Anal. Chem.* **2008**, *80*, 9058−9064.

(51) Passarelli, M. K.; Ewing, A. G.; Winograd, N. *Anal. Chem.* **2013**, *85*, 2231−2238.

(52) Passarelli, M. K.; Winograd, N. *Surf. Interface Anal.* **2011**, *43*, 269−271.

(53) Tucker, K. R.; Li, Z.; Rubakhin, S. S.; Sweedler, J. V. *J. Am. Soc. Mass Spectrom.* **2012**, *23*, 1931−1938.

(54) Kraft, M. L.; Klitzing, H. A. *Biochim. Biophys. Acta, Mol. Cell Biol. Lipids* **2014**, *1841*, 1108−1119.

(55) Popa, R.; Weber, P. K.; Pett-Ridge, J.; Finzi, J. A.; Fallon, S. J.; Hutcheon, I. D.; Nealson, K. H.; Capone, D. G. *ISME J.* **2007**, *1*, 354−360.

(56) Musat, N.; Halm, H.; Winterholler, B.; Hoppe, P.; Peduzzi, S.; Hillion, F.; Horreard, F.; Amann, R.; Jorgensen, B. B.; Kuypers, M. M. *Proc. Natl. Acad. Sci. U. S. A.* **2008**, *105*, 17861−17866.

(57) Ghosal, S.; Fallon, S. J.; Leighton, T. J.; Wheeler, K. E.; Kristo, M. J.; Hutcheon, I. D.; Weber, P. K. *Anal. Chem.* **2008**, *80*, 5986−5992.

(58) Finzi-Hart, J. A.; Pett-Ridge, J.; Weber, P. K.; Popa, R.; Fallon, S. J.; Gunderson, T.; Hutcheon, I. D.; Nealson, K. H.; Capone, D. G. *Proc. Natl. Acad. Sci. U. S. A.* **2009**, *106*, 9931−9931.

(59) Liu, W. T.; Yang, Y. L.; Xu, Y. Q.; Lamsa, A.; Haste, N. M.; Yang, J. Y.; Ng, J.; Gonzalez, D.; Ellermeier, C. D.; Straight, P. D.; Pevzner, P. A.; Pogliano, J.; Nizet, V.; Pogliano, K.; Dorrestein, P. C. *Proc. Natl. Acad. Sci. U. S. A.* **2010**, *107*, 16286−16290.

(60) Angerer, T. B.; Blenkinsopp, P.; Fletcher, J. S. *Int. J. Mass Spectrom.* **2015**, *377*, 591−598.

(61) Lanni, E. J.; Masyuko, R. N.; Driscoll, C. M.; Dunham, S. J.; Shrout, J. D.; Bohn, P. W.; Sweedler, J. V. *Anal. Chem.* **2014**, *86*, 10885−10891.

(62) Dertinger, J. J.; Walker, A. V. *J. Am. Soc. Mass Spectrom.* **2013**, *24*, 348−355.

(63) Fitzgerald, J. J.; Kunnath, P.; Walker, A. V. *Anal. Chem.* **2010**, *82*, 4413−4419.

(64) Svara, F. N.; Kiss, A.; Jaskolla, T. W.; Karas, M.; Heeren, R. M. *Anal. Chem.* **2011**, *83*, 8308−8313.

(65) Giesen, C.; Wang, H. A.; Schapiro, D.; Zivanovic, N.; Jacobs, A.; Hattendorf, B.; Schuffler, P. J.; Grolimund, D.; Buhmann, J. M.; Brandt, S.; Varga, Z.; Wild, P. J.; Gunther, D.; Bodenmiller, B. *Nat. Methods* **2014**, *11*, 417−422.

(66) Angelo, M.; Bendall, S. C.; Finck, R.; Hale, M. B.; Hitzman, C.; Borowsky, A. D.; Levenson, R. M.; Lowe, J. B.; Liu, S. D.; Zhao, S.; Natkunam, Y.; Nolan, G. P. *Nat. Med.* **2014**, *20*, 436−442.

(67) Lin, Y.; Trouillon, R.; Safina, G.; Ewing, A. G. *Anal. Chem.* **2011**, *83*, 4369−4392.

(68) Lapainis, T.; Rubakhin, S. S.; Sweedler, J. V. *Anal. Chem.* **2009**, *81*, 5858−5864.

(69) Han, X.; Wang, Y.; Aslanian, A.; Bern, M.; Lavallee-Adam, M.; Yates, J. R., 3rd *Anal. Chem.* **2014**, *86*, 11006−11012.

(70) Wang, Y.; Fonslow, B. R.; Wong, C. C.; Nakorchevsky, A.; Yates, J. R., 3rd *Anal. Chem.* **2012**, *84*, 8505−8513.

(71) Sun, L.; Dubiak, K. M.; Peuchen, E. H.; Zhang, Z.; Zhu, G.; Huber, P. W.; Dovichi, N. J. *Anal. Chem.* **2016**, *88*, 6653−6657.

(72) Lombard-Banek, C.; Moody, S. A.; Nemes, P. *Angew. Chem., Int. Ed.* **2016**, *55*, 2454−2458.

(73) Lombard-Banek, C.; Reddy, S.; Moody, S. A.; Nemes, P. *Mol. Cell. Proteomics* **2016**, *15*, 2756−2768.

(74) Chen, S.; Lillard, S. J. *Anal. Chem.* **2001**, *73*, 111−118.

(75) Marc, P. J.; Sims, C. E.; Allbritton, N. L. *Anal. Chem.* **2007**, *79*, 9054−9059.

(76) Li, X.; Li, Y.; Zhao, L.; Shen, J.; Zhang, Y.; Bao, J. J. *Electrophoresis* **2014**, *35*, 2778−2784.

(77) Kohl, F. J.; Sanchez-Hernandez, L.; Neususs, C. *Electrophoresis* **2015**, *36*, 144−158.

(78) Bai, L.; Sheeley, S.; Sweedler, J. V. *Bioanal. Rev.* **2009**, *1*, 7−24.

(79) Domalain, V.; Hubert-Roux, M.; Tognetti, V.; Joubert, L.; Lange, C. M.; Rouden, J.; Afonso, C. *Chem. Sci.* **2014**, *5*, 3234−3239.

(80) Warnke, S.; Seo, J.; Boschmans, J.; Sobott, F.; Scrivens, J. H.; Bleiholder, C.; Bowers, M. T.; Gewinner, S.; Schollkopf, W.; Pagel, K.; von Helden, G. *J. Am. Chem. Soc.* **2015**, *137*, 4236−4242.

(81) Bendall, S. C.; Nolan, G. P.; Roederer, M.; Chattopadhyay, P. K. *Trends Immunol.* **2012**, *33*, 323−332.

(82) Bjornson, Z. B.; Nolan, G. P.; Fantl, W. J. *Curr. Opin. Immunol.* **2013**, *25*, 484−494.

(83) Bandura, D. R.; Baranov, V. I.; Ornatsky, O. I.; Antonov, A.; Kinach, R.; Lou, X.; Pavlov, S.; Vorobiev, S.; Dick, J. E.; Tanner, S. D. *Anal. Chem.* **2009**, *81*, 6813−6822.

(84) Di Palma, S.; Bodenmiller, B. *Curr. Opin. Biotechnol.* **2015**, *31*, 122−129.

(85) Atkuri, K. R.; Stevens, J. C.; Neubert, H. *Drug Metab. Dispos.* **2015**, *43*, 227−233.

(86) Krutzik, P. O.; Crane, J. M.; Clutter, M. R.; Nolan, G. P. *Nat. Chem. Biol.* **2008**, *4*, 132−142.

(87) Krutzik, P. O.; Nolan, G. P. *Nat. Methods* **2006**, *3*, 361−368.

(88) Bodenmiller, B.; Zunder, E. R.; Finck, R.; Chen, T. J.; Savig, E. S.; Bruggner, R. V.; Simonds, E. F.; Bendall, S. C.; Sachs, K.; Krutzik, P. O.; Nolan, G. P. *Nat. Biotechnol.* **2012**, *30*, 858−867.

(89) Bendall, S. C.; Simonds, E. F.; Qiu, P.; Amir, E.-a. D.; Krutzik, P. O.; Finck, R.; Bruggner, R. V.; Melamed, R.; Trejo, A.; Ornatsky, O. I.; Balderas, R. S.; Plevritis, S. K.; Sachs, K.; Pe'er, D.; Tanner, S. D.; Nolan, G. P. *Science* **2011**, *332*, 687−696.

(90) Chattopadhyay, P. K.; Gierahn, T. M.; Roederer, M.; Love, J. C. *Nat. Immunol.* **2014**, *15*, 128−135.

(91) Newell, E. W.; Sigal, N.; Bendall, S. C.; Nolan, G. P.; Davis, M. M. *Immunity* **2012**, *36*, 142 −152.

(92) Behbehani, G. K.; Bendall, S. C.; Clutter, M. R.; Fantl, W. J.; Nolan, G. P. *Cytometry, Part A* **2012**, *81*, 552−566.

(93) Newell, E. W.; Sigal, N.; Nair, N.; Kidd, B. A.; Greenberg, H. B.; Davis, M. M. *Nat. Biotechnol.* **2013**, *31*, 623−629.

(94) Tomita, M.; Kami, K. *Science* **2012**, *336*, 990−991.

(95) Amir, E.-a. D.; Davis, K. L.; Tadmor, M. D.; Simonds, E. F.; Levine, J. H.; Bendall, S. C.; Shenfeld, D. K.; Krishnaswamy, S.; Nolan, G. P.; Pe'er, D. Nat. Biotechnol. **2013**, *31*, 545−552.

(96) Fujita, H.; Esaki, T.; Masujima, T.; Hotta, A.; Kim, S. H.; Noji, H.; Watanabe, T. M. *RSC Adv.* **2015**, *5*, 16968−16971.

(97) Urban, P. L.; Jefimovs, K.; Amantonico, A.; Fagerer, S. R.; Schmid, T.; Madler, S.; Puigmarti-Luis, J.; Goedecke, N.; Zenobi, R. *Lab Chip* **2010**, *10*, 3206−3209.

(98) Ibanez, A. J.; Fagerer, S. R.; Schmidt, A. M.; Urban, P. L.; Jefimovs, K.; Geiger, P.; Dechant, R.; Heinemann, M.; Zenobi, R. Proc. Natl. Acad. Sci. U. S. A. **2013**, *110*, 8790−8794.

(99) Krismer, J.; Sobek, J.; Steinhoff, R. F.; Fagerer, S. R.; Pabst, M.; Zenobi, R. *Appl. Environ. Microbiol.* **2015**, *81*, 5546−5551.

(100) Fagerer, S. R.; Schmid, T.; Ibanez, A. J.; Pabst, M.; Steinhoff, R.; Jefimovs, K.; Urban, P. L.; Zenobi, R. *Analyst* **2013**, *138*, 6732−6736.

(101) Ong, T. H.; Kissick, D. J.; Jansson, E. T.; Comi, T. J.; Romanova, E. V.; Rubakhin, S. S.; Sweedler, J. V. *Anal. Chem.* **2015**, *87*, 7036−7042.

(102) Jansson, E. T.; Comi, T. J.; Rubakhin, S. S.; Sweedler, J. V. *ACS Chem. Biol.* **2016**, *11*, 2588−2595.

(103) Pan, N.; Rao, W.; Kothapalli, N. R.; Liu, R.; Burgett, A. W.; Yang, Z. *Anal. Chem.* **2014**, *86*, 9376−9380.

(104) Page, J. S.; Sweedler, J. V. *Anal. Chem.* **2002**, *74*, 6200−6204.

(105) Rubakhin, S. S.; Sweedler, J. V. *Anal. Chem.* **2008**, *80*, 7128−7136.

(106) Quine, W. V. *Theories and things*; Harvard University Press: Cambridge, MA, 1981.

CHAPTER 4

SYNCHRONIZED DESORPTION ELECTROSPRAY IONIZATION MASS SPECTROMETRY IMAGING

Notes and Acknowledgements

This chapter was published as an original research article in *Analytical Chemistry* **2016**, DOI: 10.1021/acs.analchem.5b03010, coauthored by S. W. Ryu and R. H. Perry. The article is adapted and reprinted here with permission from the American Chemical Society, copyright 2015. T. J. Comi performed most mass spectrometry analyses, data analysis, simulations and authoring of the manuscript. S. W. Ryu performed confirmatory experiments on mass spectrometry imaging of Rhodamine 6G spots and assisted in manuscript editing. Funding for this work was provided by the University of Illinois at Urbana−Champaign, NIH Training Program at Chemistry-Interface with Biology (NIH T32 GM070421), National Science Foundation Graduate Research Fellowship Program, and the Springborn Fellowship. Dr. Edward T. Chainani is gratefully acknowledged for help in designing initial circuitry, assisting with valve latency measurements, and many useful conversations.

Introduction

Fingerprint evidence is universally recognized as a reliable method for biometric identification of criminal suspects.[1] Latent fingermarks, impressions of fingerprint ridge patterns on surfaces, consist of endo-, semi-exo-, and exogenous compounds related to suspect physiology, diet, and fingertip contact with external chemical compounds including illicit drugs and explosives.[2,3] Typically, fingermark visualization at crime scenes is achieved by imaging photoluminescent agents, including ninhydrin and nanoparticles that target endogenous chemical compounds such as amino acids and glycerides, respectively.[1,2,4,5] Mass spectrometry imaging (MSI) techniques such as secondary ion MS (SIMS),[6-9] matrix-assisted laser desorption/ionization (MALDI),[10-17] and desorption electrospray ionization (DESI),[9,18-21] provide high selectivity and the capability to identify unlabeled chemical components, which can significantly improve the accuracy of suspect identification and provide evidence of recent activities.

One of the advantages of DESI is that it enables direct fingermark imaging at atmospheric pressure without the need for pretreatment,[22-31] thereby preserving evidence integrity. It is often beneficial to perform MSI experiments at high spatial resolutions (step size ≤ 75 µm) on mass analyzers capable of high *m/z*-resolution (> 60 000) and accurate mass measurements (< 2 ppm). When DESI-MSI is performed on a pulsed mass analyzer such as the Orbitrap,[32-34] the percentage of time that the desorbed plume is effectively sampled by the instrument is given by the ratio of the injection time (viz. ion accumulation period (IT)) to the total time required to acquire each spectrum (t = IT + transient acquisition time). For a typical IT = 500 ms and a resolution setting of 100 000 at *m/z* 400 (transient acquisition = 1.8 s for an LTQ-Orbitrap XL), the desorbed species are sampled for only ~0.2t while the remaining desorbed material is discarded.[35,36] In addition, the redistribution and spreading of analytes on the surface, known as the "washing effect",[37] negatively impacts spatial resolution throughout the entire scan, a problem that is exacerbated for analytes weakly attached to smooth surfaces such as fingermarks at crime scenes.

Cooks and coworkers developed a DESI source that synchronizes the nebulizing gas and DESI voltage (*s*DESI-MS) with the IT of a discontinuous atmospheric pressure interface on a rectilinear ion trap miniature mass spectrometer. This modification increased sensitivity, desorbed material only during the IT, and reduced the amount of solvent deposited on the surface in DESI.[38,39] In this report, we develop an *s*DESI-MS imaging source coupled to a high-resolution MS (HRMS; Figure 4.1). The *s*DESI-MSI source utilizes two solenoid valves (Figure 4.1). One valve synchronizes the nebulizing gas flow with the IT of a LTQ-Orbitrap XL HRMS. During transient acquisition, the DESI nebulizing gas is turned off and the second valve delivers a perpendicular stream of nitrogen gas (N_2) that prevents solvent accumulation at the emitter tip. Thus, solvent is deposited only during IT, minimizing analyte redistribution by the "washing effect". In addition to improving sensitivity and decreasing the amount of sample desorbed per HRMS scan, our results show that synchronization improves spatial resolution by a factor of ~4−6 for analytes (e.g. Rhodamine 6 G spots) weakly attached to smooth surfaces (e.g. photographic paper). In addition, under specific experimental conditions, synchronization was essential to obtain distinct MS images of low-intensity endogenous fatty acids (FA) in fingermarks on glass. For example, using a step size of 25 μm and a microdroplet spray composition of $CH_3OH:H_2O$ (9:1), *s*DESI-MSI images of the fingermark ridge patterns were generated using ion signals for lignocercic acid and cerotic acid. However, for the same set of conditions, continuous desorption did not yield distinct MS images. Simulations modeling analyte movement during desorption and the "washing effect" replicate these experimental results by varying the washing parameter. All these results demonstrate that synchronization improves spatial resolution and sensitivity by decreasing the time analytes are redistributed by the "washing effect". Generally, *s*DESI expands the scope of analytes, surfaces, and experimental conditions available for study such as the high-resolution imaging of analytes that are weakly attached to smooth surfaces.

Methods

Materials

ACS grade chloroform ($CHCl_3$), HPLC-grade water (H_2O; Macron Fine Chemicals, Center Valley, PA), HPLC-grade methanol (CH_3OH; Fisher Scientific, Pittsburgh, PA), Rhodamine 6G, (R6G; Sigma-Aldrich, St. Louis, MO), 1,2-dipalmitoyl-*sn*-glycero-3-phosphocholine (DPPC; Avanti Polar Lipids, Alabaster, AL, USA), 1,2-dipalmitoyl-*sn*-glycero-3-phospho-(1'-rac-glycerol) (DPPG; Avanti Polar Lipids), bovine brain total lipid extract (BBE; Avanti Polar Lipids), Ampicillin (AMP; Sigma-Aldrich), Bradykinin (BK; Sigma-Aldrich), and ultra-high purity nitrogen (N_2; S.J. Smith Co., Decatur, IL) were used as received.

Synchronized Desorption Electrospray Ionization Mass Spectrometry Imaging Source

The *s*DESI-MSI source (Figure 4.1) involves modifying a commercial Prosolia 2D Omnispray DESI-MSI source (Indianapolis, IN, USA) to include two three-way valves (ASCO, Florham Park, NJ, USA). One of the valves modulates the nebulizing gas (180 psi) and the second regulates a perpendicular N_2 gas stream (shutter gas; 20 psi). When the nebulizing gas is off the shutter gas removes solvent delivered to the emitter tip in a direction that is parallel to the sample surface. The three-way valve that regulates the nebulizing gas (V1) is normally closed, while the shutter gas is normally open (V2) (Figure 4.1c). Gas is emitted from the three-way valve common ports so that N_2 backpressure rapidly vents to atmosphere when switching between ion source ON (Figure 4.1b) and OFF states (Figure 4.1c). Valve modulation and timing is controlled with a home-built electronic circuit (Figure 4.2). Direct measurement of gas flow impinging on a microphone shows a valve latency of 16.6 ± 2.9 ms and a 10.5 ± 2.7 ms for switching the shutter gas to the OFF and ON states, respectively, and 494.2 ± 4.6 ms desorption for 500 ms IT (Figure 4.3). In a typical *s*DESI-MSI experiment, valve latency is ~3% of IT.

Analyte Depletion Rate and Sensitivity

DESI-MS and *s*DESI-MS extracted ion chromatograms (XIC) were used to estimate the rates of analyte depletion for R6G (100 pg), DPPC (10 ng), DPPG (20 ng), AMP (20 ng), BK (200 μg) or BBE (200 μg) deposited on polytetrafluoroethylene (PTFE) Omni Slides (Prosolia, Inc.). Five XICs were acquired per analyte when the *s*DESI-MS source was stationary (static mode) and then averaged to generate decay curves. The exponential regression constant reflects the rate of sample depletion from the surface.

Sensitivity was characterized by rastering the ionization source across deposited spots of various analytes in seven evenly distributed rows. The source was continuously moved to investigate sensitivity independent of desorption rates at the spray impact site, maintain good reproducibility without internal standards,[40] and simulate imaging conditions. A custom MATLAB (Mathworks, Natick, Massachusetts, USA) script was used to integrate analyte signal intensity across each spot. Differences in sensitivity (i.e. slope of calibration curves) were tested for significance using analysis of covariance (ANCOVA; R software environment).

Latent Fingermark Imaging

Sebum-enhanced fingermark impressions on glass microscope slides were produced from a male donor using previously reported methods[41] and analyzed immediately (optical images of fingerprints were manually distorted to protect donor privacy). Briefly, after thorough hand-washing, the finger was rubbed against the side of the nose and then pressed against a glass slide. Selected regions of the fingermarks were imaged using either $CHCl_3$:CH_3OH (1:1; 2 μL/min) or CH_3OH:H_2O (9:1; 1 μL/min) with pixel widths of 150 μm, 75 μm, or 25 μm (emitter voltage = -5 kV; capillary temperature = 275 ^oC; IT = 500 ms; resolution setting = 100,000; *m/z* range = 100 – 400). At the smallest pixel width of 25 μm, a 1 mm^2 area is analyzed in approximately one hour; *s*DESI does not affect image acquisition time. Optical images of fingerprints on regular office paper were acquired using a flatbed scanner (Epson Perfection 2400 Photo). The fingerprints were deposited on paper using an inkless fingerprint pad (Lee Products Company, Bloomington, MN).

We designed a software package (C# programming language) to process and visualize MSI data. Thermo Fisher Scientific RAW data files are converted to the mzXML file format using ProteoWizard.[42] The software loads the mzXML files into memory without data reduction or binning of *m/z* values, and then generates chemical images using false colors to represent signal intensities (Figure 4.4). The software package also provides the capability to average spectra within user-drawn regions of interest (ROI), subtract spectra between ROIs, export spectra as comma-separated value text, and perform hyperspectral visualization[43] with *m/z* binning as described by Xiong et. al.[44]

Spatial Resolution

The spatial resolution of DESI-MSI and *s*DESI-MSI were compared using patterns of R6G dots on photographic paper (Epson, Long Beach, CA). The patterns were generated from a red Sharpie marker (Sanford Corp., Oak Brook, IL) and an unpolished stainless steel mesh template (Small Parts Inc., Miramar, FL). The R6G patterned surfaces were fabricated by pressing the SS mesh on top of the photographic paper immediately after it was drawn on with Sharpie, which produces an array of R6G dots (~100 µm diameter) spaced by ~500 µm. Then, a second array was superimposed on the first, generating a R6G dots with variable spacing. Optical images of the R6G dots were acquired using an EVOS *fl* Inverted Fluorescence Microscope (Advanced Microscopy Group, Life Technologies, Thermo Fisher Scientific). Pattern dimensions were estimated from the fluorescence images using ImageJ (http://imagej.nih.gov/ij/).

Simulations of Desorption and the "Washing Effect"

Simulations estimating the relative impact of the desorption/ionization and washing (W) efficiencies on sensitivity, decay rate, and spatial resolution, were performed using the finite difference method[45] (implemented in MATLAB). For simplicity, the model estimates a circular spray profile at the surface (Figure 4.5), instantaneous analyte dissolution and immediate thin film formation upon spray impact. The rate of desorption is modeled as a two-dimensional Gaussian distribution that accounts for concentric regions of high (H; 100 µm diameter) and low (L; 500 µm diameter) ionization efficiency within the spray profile at the surface.[46] The model for desorption is combined with a cosine-distributed "washing effect" parameter to describe analyte movement on the surface relative to the center of the spray profile ($\vec{c}$; Figure 4.5). The magnitudes of the washing and desorption/ionization efficiencies were modified by adjusting their corresponding rate constants (washing: R_W; desorption/ionization: $R_L + R_H$). At each time step Δt (10 ms), the amount of analyte (I) at a given input pixel ($\vec{p}$) changes due to low efficiency desorption from the outer spray plume, high efficiency desorption/ionization from the inner spray plume, washing from $\vec{p}$ to neighboring pixels (W_{out}), and washing from neighboring pixels to $\vec{p}$ (W_{in}). I for a given pixel is updated for the next time step by the following relationship:

$$I(t+\Delta t) = \big(W_{in} - (L + H + W_{out})I(t)\big)\Delta t + I(t)$$

where W_{out}, is distributed to neighboring pixels at position i ($\vec{N_i}$) based on the magnitude of their projections on the radial vector $\vec{r} = \vec{p} - \vec{c}$. The fraction of analyte redistributed to pixel i is:

$$W_{in\ to\ i\ from\ j} = W_{out\ from\ j\ to\ i} = \frac{\langle \vec{r}, \vec{N_i} \rangle}{\sum_{\langle \vec{r}, \vec{N_i} \rangle > 0} \langle \vec{r}, \vec{N_i} \rangle} W_{out,j}$$

where, $\langle \vec{r}, \vec{N_i} \rangle$ denotes an inner product. After each Δt, the center of the spray profile at the surface was moved a distance equal to the *xy*-translation stage velocity × Δt = 1.3 µm. At the end of each row of the output image, the spray profile is moved to the beginning of the next row, simulating the fly-back motion of a *xy*-translation stage used for DESI-MSI. Simulations of *s*DESI-MSI were performed identically but the input analyte distribution was only updated during IT. Source code for simulations is provided in Appendix A.

Results and Discussion

The impact of synchronization on sensitivity was evaluated by comparing the DESI and *s*DESI calibration curves for R6G spots on Omni Slides. The calibration curves were acquired at various velocities of the *xy*-translation stage (100 µm/s, 50 µm/s, 25 µm/s, and 0 µm/s) to simulate MSI conditions. Our results show that synchronization of DESI with IT improved sensitivity by factors of 1.77 ± 0.13, 2.02 ± 0.15, 2.96 ± 0.43, and 3.51 ± 0.55, respectively. Further experiments showed that the magnitudes of these improvements in sensitivity depend on various experimental conditions such as the composition of the microdroplet spray, the sample surface, and the nature of the analyte (Figure 4.6 and Table 4.1).[38,39] The sensitivity improvement with slower raster speeds is particularly important for MSI, as it suggests that synchronization will largely benefit images acquired at high spatial resolution.

The depletion rate was estimated by recording the intensity of specific lipid signals from BBE samples as function of time, and then calculating the decay constant from exponential regression curves (Figure 4.7). With synchronization, signals for [36:1 PS – H]$^-$ (*m/z* 788.536), [40:6 PS – H]$^-$ (*m/z* 834.520), [38:4 PI – H]$^-$ (*m/z* 885.541), and [42:2 sulfatide (ST) – H]$^-$ (*m/z* 888.614) decayed slower by an average factor of 5.90 ± 0.71 (Figure 4.7b – 4.7e), which is in agreement with a desorption period of ~0.2*t* for *s*DESI. The decay curves also show that the

DESI and *s*DESI variances are relatively similar (Figure 4.6), indicating that synchronization does not significantly degrade reproducibility between technical replicates.

After performing DESI and *s*DESI MSI of R6G spots on photographic paper, optical images showed higher amounts of residual R6G when desorption/ionization was synchronized with IT (Figure 4.8a,b and 4.9), which agrees with the trends observed for the DESI and *s*DESI decay rates (Figure 4.7). In addition, the MS images and XICs of the R6G spots on photographic paper show baseline-resolution for *s*DESI (Figure 4.8a), while the DESI XICs contain valleys between spots with intensities as high as ~20-25% (Figure 4.8b). By considering two spots as resolved when the valley intensity is less than 10%, these results demonstrate that synchronization improves spatial resolution by a factor of ~4. Although the microdroplet spray profiles at the surface of water-sensitive paper have relatively similar dimensions, *s*DESI deposits less liquid on the surface, as indicated by the lower intensity of the blue color (Figure 4.8c). As a result, the R6G spots rapidly dissolve and diffuse within the liquid pool, degrading spatial resolution via the "washing effect".

To demonstrate the utility of *s*DESI-MSI, we performed high-resolution imaging of low-abundance chemical compounds in latent fingermarks, which are typically located on smooth surfaces such as glass at crime scenes. Using a step size of 150 μm and microdroplet spray composition of $CHCl_3$:CH_3OH (1:1), endogenous FAs with ion abundance less than 5% relative intensity, such as $[24:0\ FA - H]^-$ (*m/z* 367.390) and $[26:0\ FA - H]^-$ (*m/z* 395.424) produce distinct ridge patterns with and without synchronization (Figures 4.10 and 4.11; see Table 4.2 for the identification of other endogenous FAs). However, chemical images of these FAs in fingermarks were only observed at step sizes of 75 μm ($CHCl_3$:CH_3OH (1:1)) and 25 μm (CH_3OH:H_2O (9:1)) with *s*DESI-MSI (Figure 4.10). These results show that *s*DESI-MSI is a sensitive method for mapping the spatial distribution of low-abundance chemicals in fingermarks, which has potential applications in forensics.

To gain preliminary insights about the mechanisms of *s*DESI, we simulated analyte motion as a function of parameters that describe desorption (R_H, R_L; the magnitude is the combined effect of R_H and R_L) and the "washing effect" (R_W; Figure 4.12). Using fluorescent images of the R6G spots (Figure 4.8) as input distributions for the simulations, the magnitude of the desorption parameter had a proportional relationship to the analyte signal intensity while the washing parameter was inversely proportional. In addition, the simulated ratio of the signal intensities for *s*DESI and DESI is always greater than unity, supporting the experimental observation that synchronization improves sensitivity (Figure 4.12a and 4.13).

Simulated XICs for *s*DESI-MSI contain peaks that do not broaden or distort at high magnitudes of R_W (Figure 4.12b). This result is reflected in the higher spatial resolution obtained for simulated *s*DESI-MS images compared to a continuous microdroplet spray ($R_W = 1$ and $R_H = 0.01$,$R_L = 0.005$; Figure 4.12c). Interestingly, when the magnitudes of both R_W and R_H, R_L were increased, the model only generated MS images for *s*DESI-MSI ($R_W = 1$ and $R_H = 0.025$,$R_L = 0.01$; Figure 4.12c), replicating the results shown in Figure 4.10. It appears, for a continuous microdroplet spray, analytes are washed away by the outer region of the spray profile before they can interact with the central region of higher ionization efficiency (Figure 4.5). When the magnitude of R_W was reduced to 0.1, the spatial resolution is relatively similar for DESI and *s*DESI MS images, which is agrees with experimental MS images of lipids embedded in the extracellular matrix of rat brain tissue sections (Figure 4.14). Since R_W is related to the magnitude of the analyte-surface interaction, synchronization produces the greatest improvements for analytes that have a weak interaction with the surface. The simulations also suggest that synchronization of desorption/ionization with IT improves sensitivity and spatial resolution by reducing the time that the “washing effect” is operating during each MS scan.

Conclusions

In this report we describe the development and characterization of a DESI-MS imaging source that synchronizes desorption/ionization with the ion accumulation period (IT) of an LTQ-Orbitrap XL mass spectrometer. Our results show that synchronization improves sensitivity, increases spatial resolution, and reduces the amount of sample consumed per MS scan by factors as high as ~4-6. These improvements were necessary to obtain informative, high-resolution MS images (step sizes ≤75 μm) of low-intensity fatty acids in latent fingermarks on glass, highlighting the utility of *s*DESI for mapping the spatial distribution of weakly-bound analytes on smooth surfaces. MSI simulations support that the benefits of *s*DESI are the result of depositing a lower volume of solvent on the surface per MS scan, thereby minimizing the redistribution and spreading of analytes on the surface ("washing effect"). Overall *s*DESI expands the range of analytes, surfaces, and experimental conditions accessible to DESI-MSI.

Figures and Tables

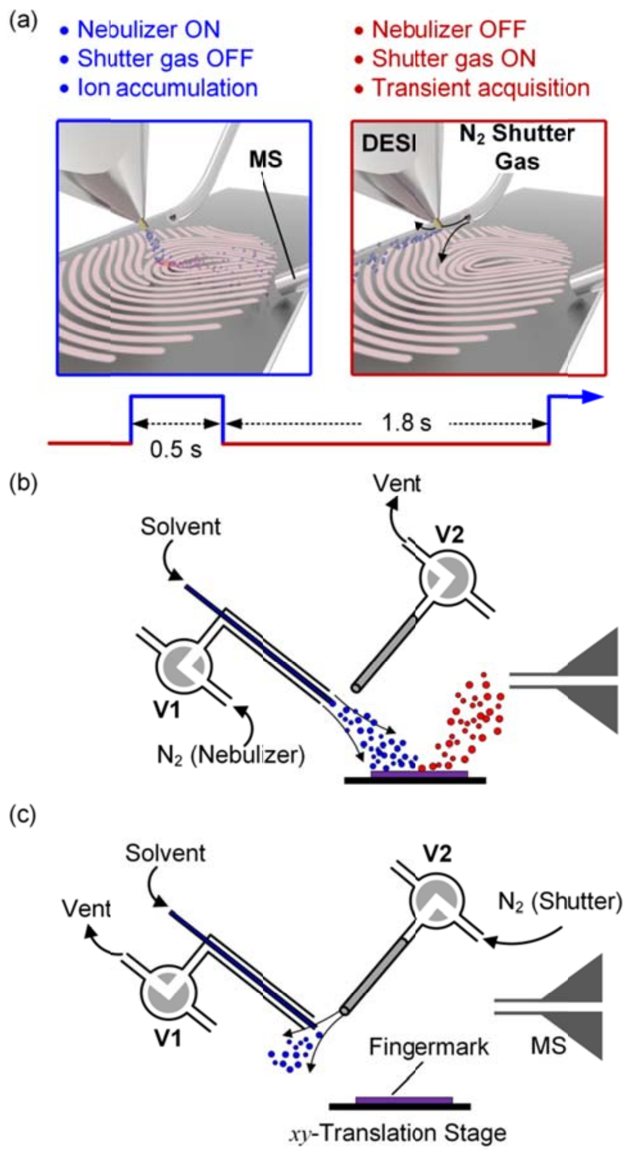


Figure 4.1. (a) Three-dimensional representation of the sDESI-MSI source. The timing diagram shows synchronization of the nebulizing gas ON and OFF states with ion accumulation and transient acquisition, respectively. (b) Valve positions of sDESI-MSI during ion accumulation. (c) Valve positions of sDESI-MSI during transient acquisition.

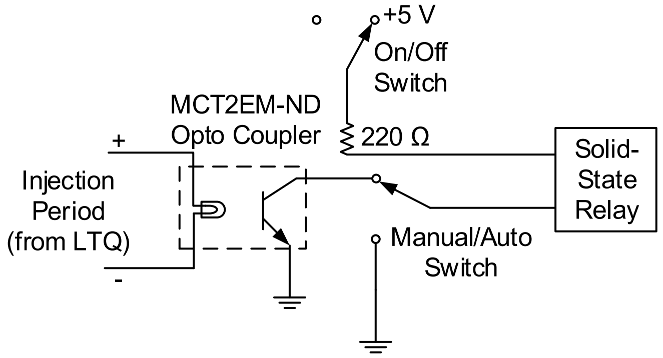


Figure 4.2. Circuit diagram of LTQ-Orbitrap XL MALDI to transistor-transistor logic (TTL) converter. Valve modulation and timing is controlled by the TTL signals which are synchronized with ion accumulation. The optoelectronic coupler isolates instrument electronics and removes noise from the input signal. The valves are powered with 120 V AC that is switched ON and OFF with a solid-state relay.

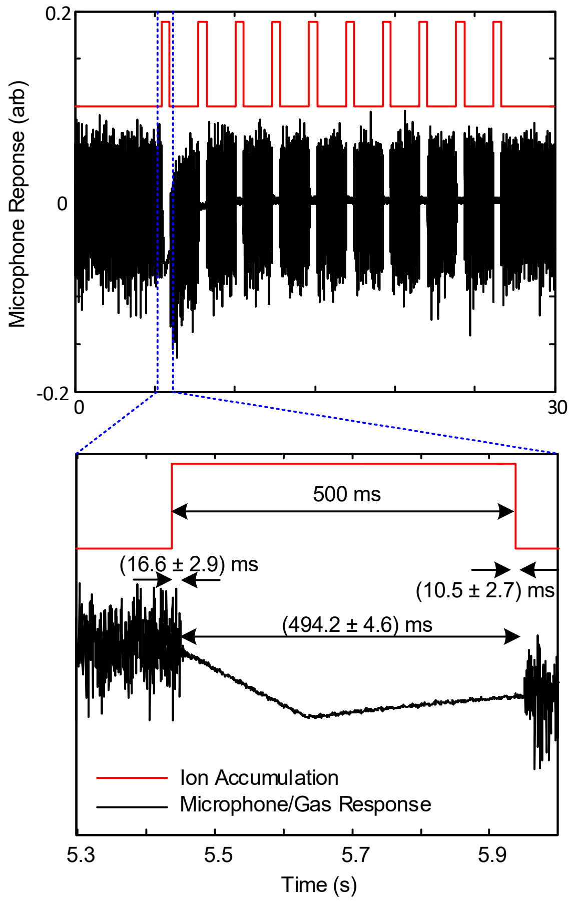
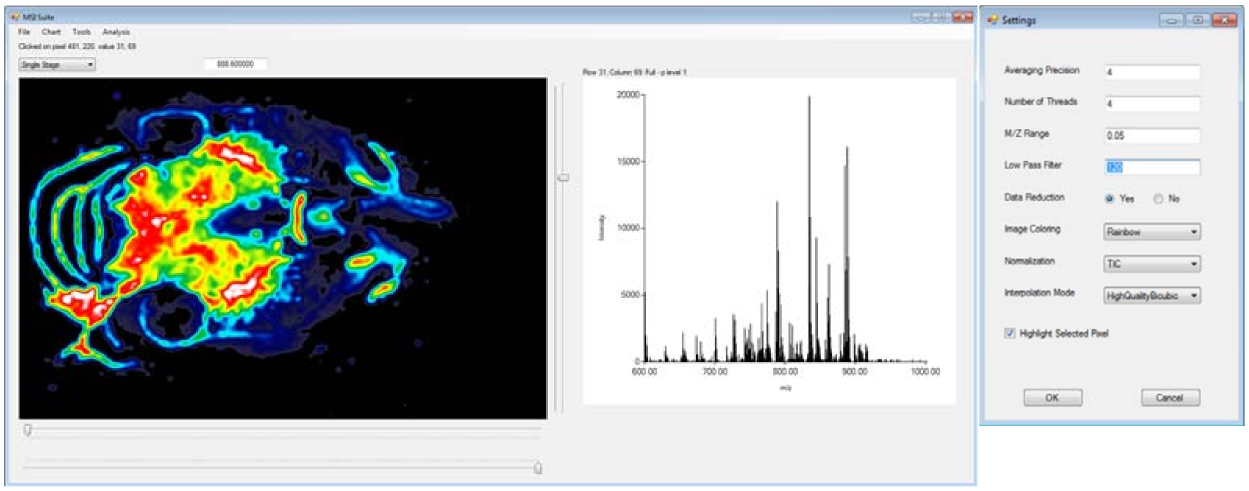


Figure 4.3. Characterization of V2 latency (bottom panel shows zoomed-in view of 5.3 – 6.0 s). A microphone was placed in close proximity to the shutter gas outlet to record ON and OFF times of the N_2 gas relative to an input trigger generated from a data acquisition board. Gas impinging on the microphone diaphragm causes large amplitude noise while the valve is open. Initial capacitance charging in the microphone electronic circuit caused a rise in baseline voltage unrelated to valve performance.



Clicked on pixel 481, 220, value 31, 69

Averaging Precision: 4

Number of Threads: 4

M/Z Range: 0.05

Low Pass Filter: 120

Data Reduction: [✓ Yes] No

Image Coloring: Rainbow

Normalization: TIC

Interpolation Mode: HighQualityBicubic

☑ Highlight Selected Pixel

Figure 4.4. Screen capture of in-house developed MSI analysis software. Image displayed is DESI MSI of a coronal mouse brain section acquired with 3 μL/min of 1:1 $CHCl_3$:CH_3OH, 180 psi N_2, 150 μm pixel width.

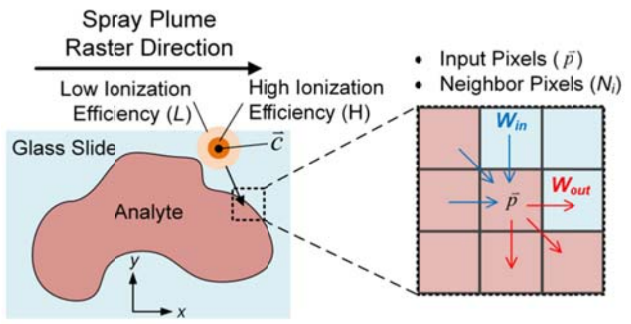

Spray Plume
Raster Direction
Low Ionization Efficiency (L)
High Ionization Efficiency (H)
$\vec{c}$
Glass Slide
Analyte
y
x
• Input Pixels ($\vec{p}$)
• Neighbor Pixels (N_i)
W_{in}
W_{out}
$\vec{p}$

Figure 4.5. Scheme showing the variables used in simulating analyte removal and redistribution during DESI and *s*DESI-MSI.

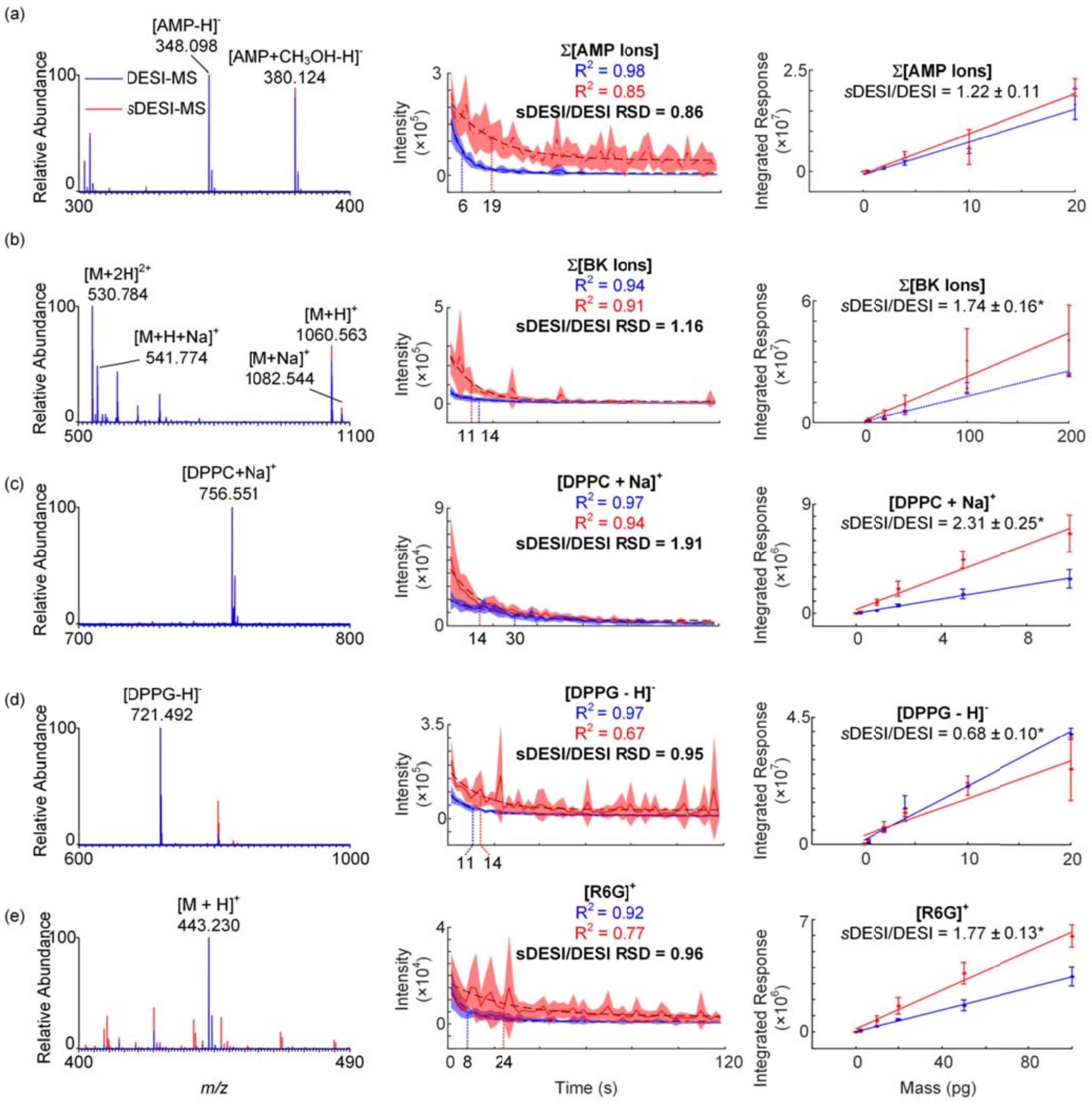


Figure 4.6. Mass spectra, decay curves, and calibration curves for (a) Ampicillin (AMP), (b) Bradykinin (BK), (c) 1,2-dipalmitoyl-sn-glycero-3-phosphocholine (DPPC), (d) 1,2-dipalmitoyl-sn-glycero-3-phospho-(1'-rac-glycerol) (DPPG), and (e) Rhodamine 6G (R6G) using DESI (blue) and *s*DESI (red). Dotted lines represent the time at which signal intensity decreases by 30%. Decays shown with shading for ± 1 S.D., *n*=5, calibration curves shown with error bars ± 1 S.D. *n* = 3. *s*DESI/DESI relative standard deviation estimated from the first 10 s of acquisition. Asterisks (*) represent statistically significant differences determined by ANCOVA ($p < 0.05$).

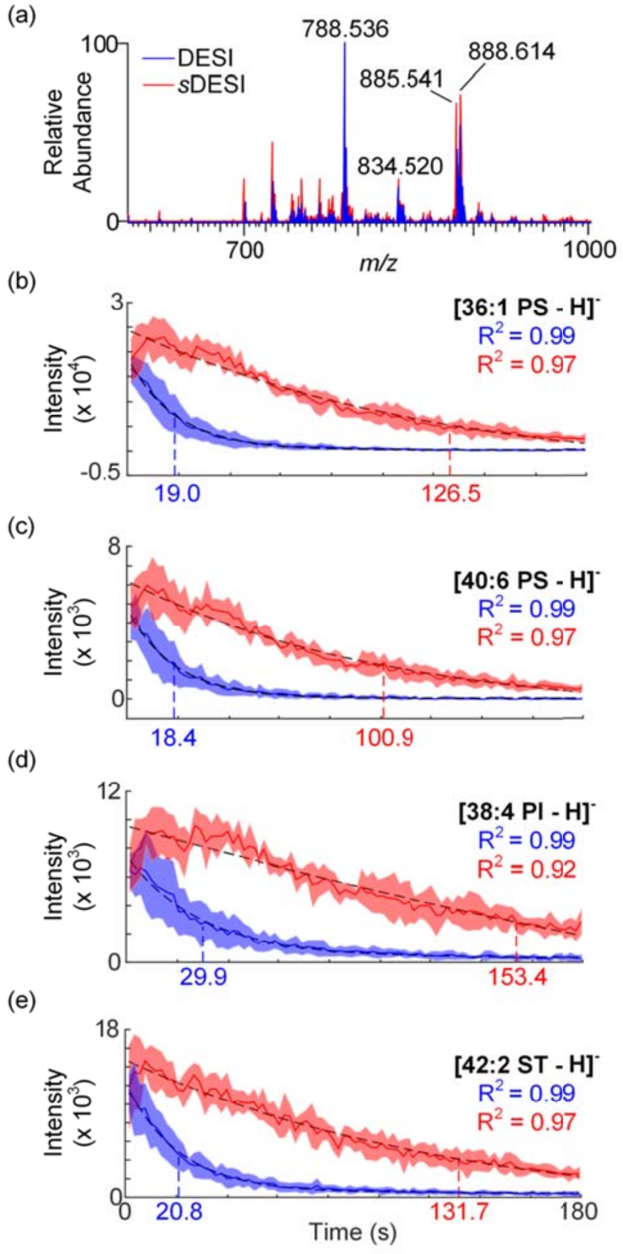


Figure 4.7. (a) Mass spectra of BBE for DESI (blue) and *s*DESI (red). Signal decay of species at (b) *m/z* 788.537, (c) *m/z* 834.520, (d) *m/z* 885.541, and (e) *m/z* 888.614 at a single point. Shaded areas represent ±1 S.D. of five measurements and dotted lines show the time at which signal decreases to 30% of the highest intensity. The time point at which signal decayed to 30% for [38:4 PI –H]$^-$ was determined directly from the graph, owing to a poor exponential fit.

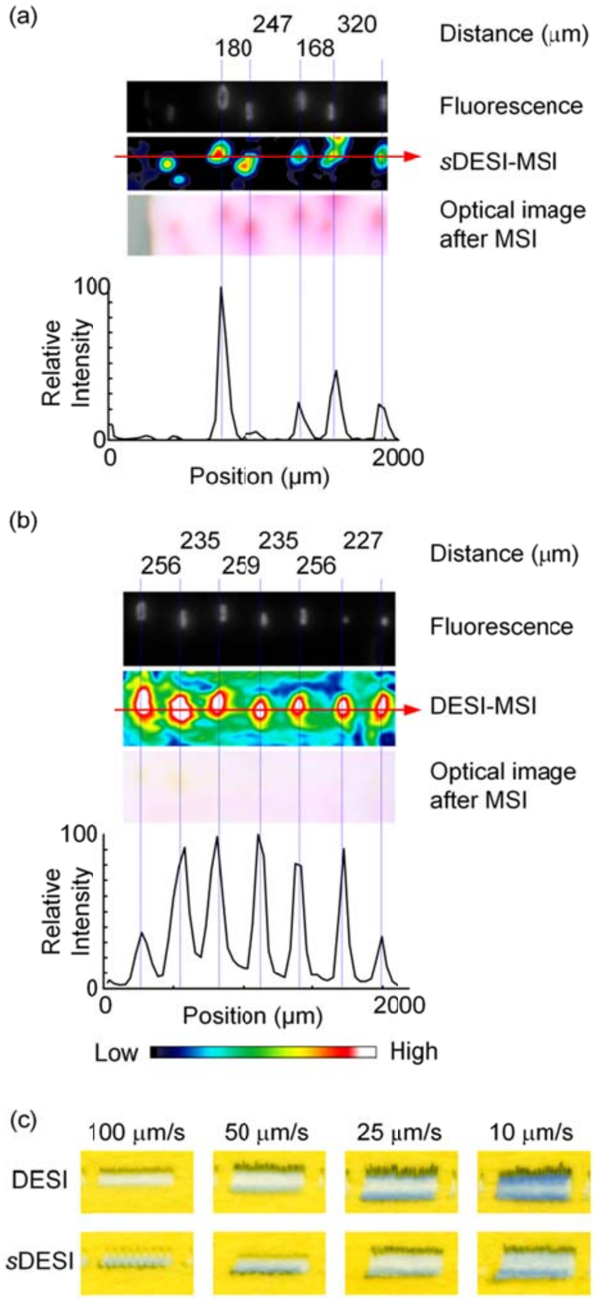


Figure 4.8. (a) *s*DESI-MS and (b) DESI-MS analysis of R6G dots deposited on photographic paper. Images of the R6G dots were acquired using fluorescence microscopy, MSI (1:1 $CHCl_3$:CH_3OH spray solvent and 40 μm pixel width), and a flatbed scanner in the listed order. XICs for one row (red arrows) are also shown for each MS image. Note that while the R6G dots are centered on the XIC for 4.8b, several dots are partially sampled in 4.8a leading to varied relative intensity. (c) Profiles of the DESI and *s*DESI microdroplet sprays at the surface of water-sensitive paper with *xy*-translation stage velocities of 100, 50, 25, and 10 μm/s. The water-sensitive paper changes color from yellow to blue upon interaction with water in the solvent.

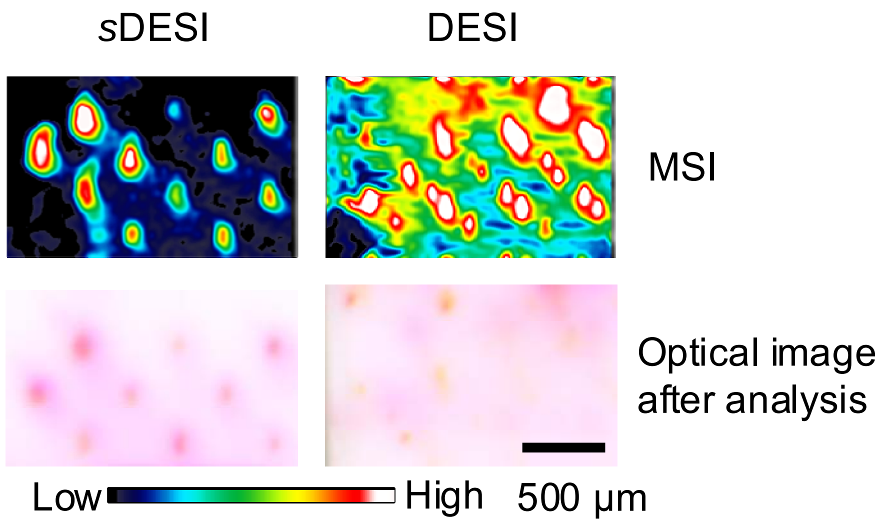


Figure 4.9. Mass spectrometry and optical images of R6G spots deposited on photographic paper using 1:1 $CHCl_3$:CH_3OH spray solvent and a resolution setting of 60,000. The optical images were acquired after DESI and *s*DESI analysis. In contrast to Figure 4, these images were acquired with a scan rate 50% faster but with the same pixel width. The faster scanning appears to more severely smear DESI-MSI.

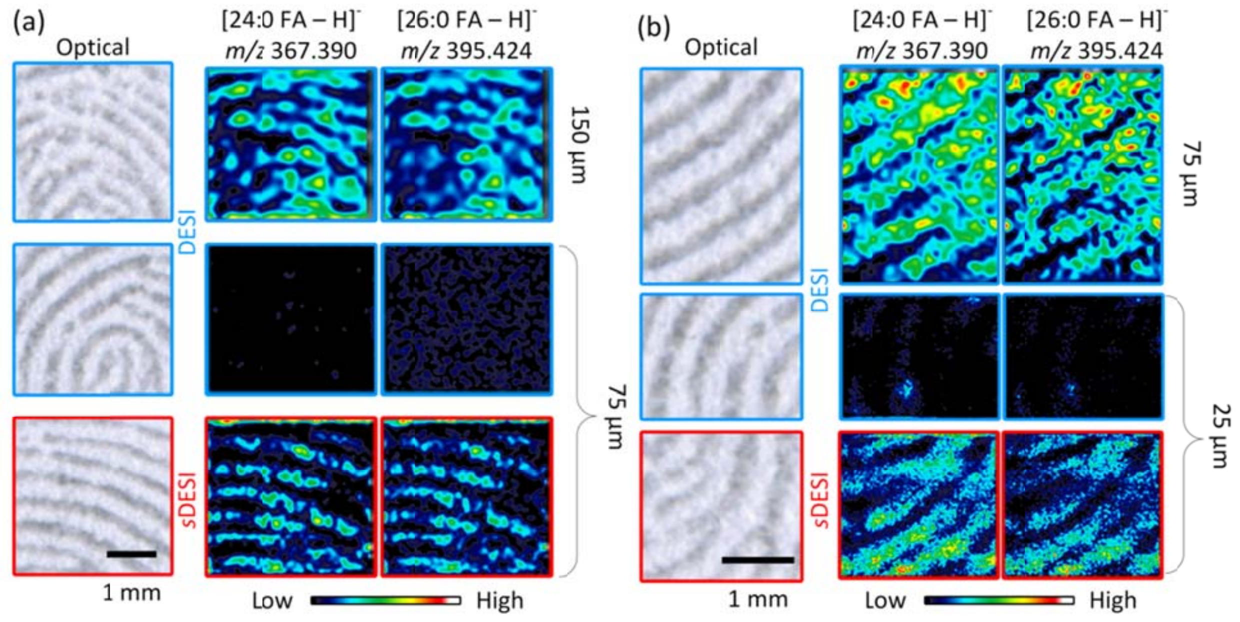


Figure 4.10. Optical scans of a fingerprint and MS images showing the ridge patterns of a latent fingermark on glass analyzed with $CHCl_3$:CH_3OH (1:1) (a) and CH_3OH:H_2O (9:1) spray solvent (b). MS images were generated by plotting the spatial distribution of the fatty acids (FA) lignoceric acid (24:0 FA) and cerotic acid (26:0 FA). MS images grouped by braces have the same intensity scale.

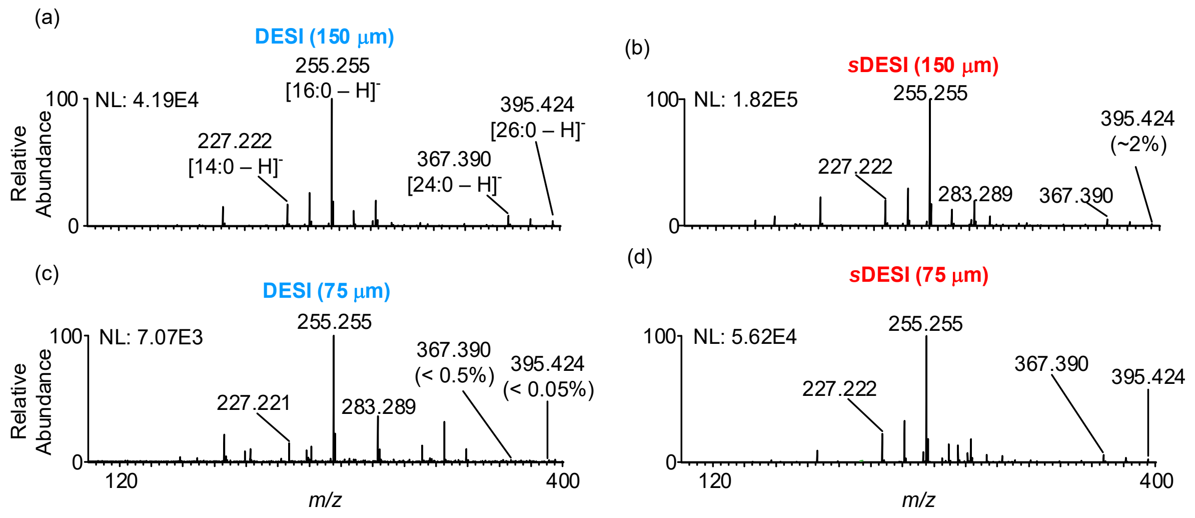


Figure 4.11. (a) DESI and (b) *s*DESI mass spectra using a pixel width of 150 µm. (c) DESI and (d) *s*DESI mass spectra using a pixel width of 75 µm. Mass spectra represent an average across one row of the corresponding images shown in Figure 4.10a.

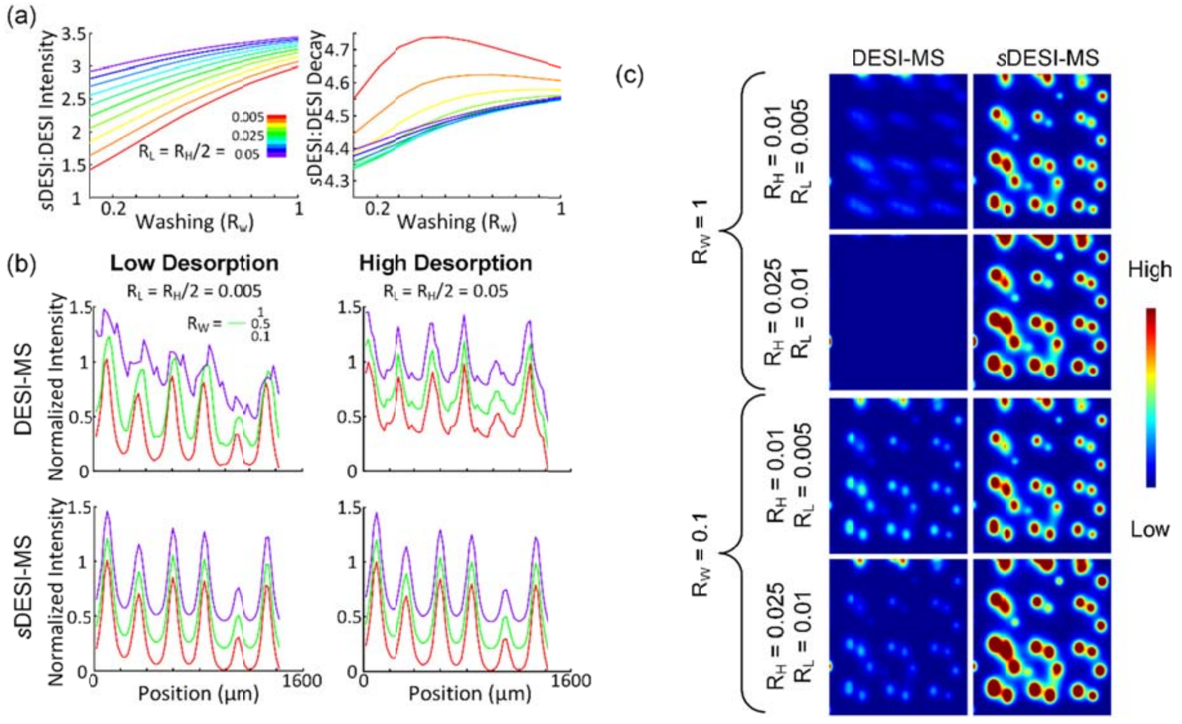


Figure 4.12. Simulated DESI and *s*DESI analysis. (a) Comparison of sensitivity and decay rate for static profiling. (b) Line scans from an image corresponding to the XIC in Figure 4b. (c) Simulated images for DESI and *s*DESI. The parameters R_W, R_L, and R_H represent the combined effect of various physical and chemical properties on the "washing effect", desorption, and ionization. R_L and R_H were varied while keeping R_W constant and noting the effect on simulated XICs and MS images. The values of the parameters in the figure represent one set of simulated conditions that replicate experimental observations.

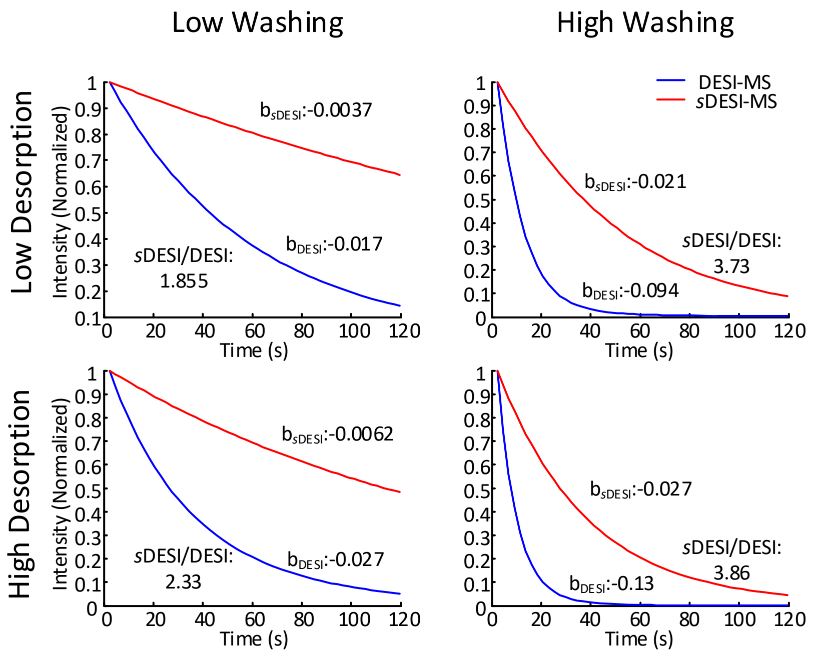


Figure 4.13. Example decay curves for simulated DESI and *s*DESI with the plume held at a single position. *s*DESI/DESI corresponds to the ratio of integrated intensity over the two minute simulation time.

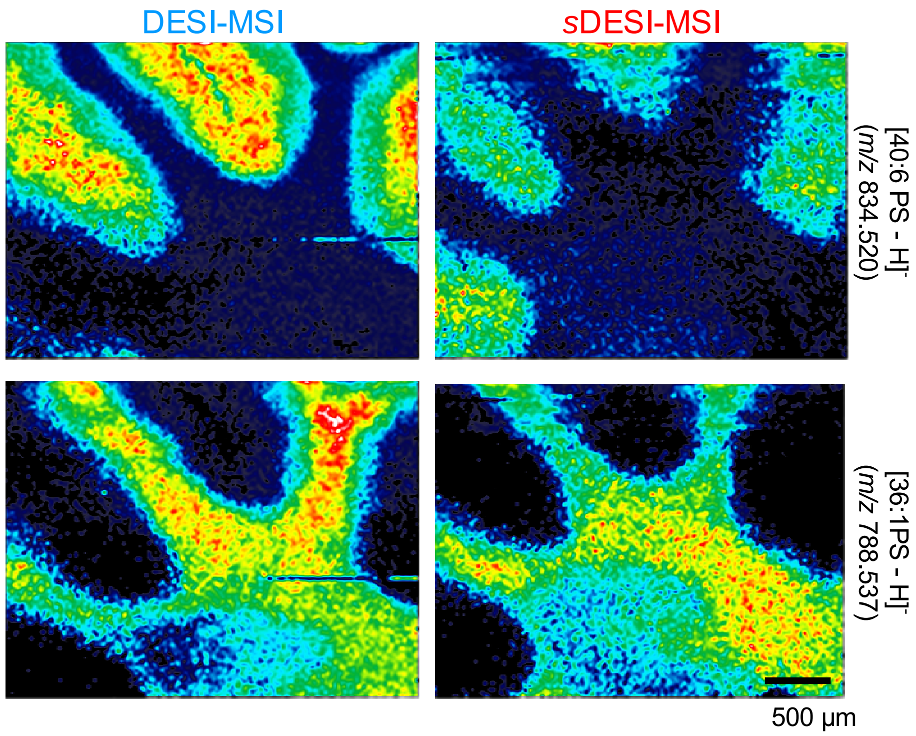


Figure 4.14. DESI and *s*DESI MS images of rat brain (25 µm sampling period; 1:1 $CHCl_3$:CH_3OH spray solvent at a flow rate of 3 µL/min). Sagittal tissue sections of unstripped rat brain (Pel-Freez, Rogers, AR, USA) were cut without embedding media using a Leica CM3050 S cryostat microtome.

Table 4.1. Ratio of *s*DESI/DESI sensitivity to Rhodamine 6G (R6G) for various solvent and surface compositions. For all experiments, R6G was desorbed using a raster velocity of 100 µm/s, mass resolution setting of 100,000, and an ion accumulation time of 500 ms. Asterisks (*) represent statistically significant differences determined by ANCOVA ($p < 0.05$).

Solvent	**Surface**	***s*DESI/DESI**
1:1 $CHCl_3$:CH_3OH	Omni-Slide	1.77 ± 0.13*
CH_3OH		0.953 ± 0.071
1:1 CH_3OH:H_2O		1.130 ± 0.069*
1:1 $CHCl_3$:CH_3OH	Paper	1.25 ± 0.19
	Dollar Bill	2.01 ± 0.18*
	Stainless Steel	1.73 ± 0.51*
	TLC Plate	1.14 ± 0.18
	Glass	1.40 ± 0.33

Table 4.2. Some of the fatty acids (FA) identified in latent fingermarks.

FA	**$[M - H]^-$ (Da)**
14:0	227.202
15:0	241.217
16:1	253.217
16:0	255.232
17:0	269.248
18:1	281.248
18:0	283.264
21:4	317.250
22:0	339.326
24:0	367.357
25:0	381.373
26:0	395.424

References

(1) Green, F. M.; Salter, T. L.; Stokes, P.; Gilmore, I. S.; O'Connor, G. *Surface and Interface Analysis* **2010**, *42*, 347.

(2) Francese, S.; Bradshaw, R.; Ferguson, L. S.; Wolstenholme, R.; Clench, M. R.; Bleay, S. *Analyst* **2013**, *138*, 4215.

(3) Weyermann, C.; Roux, C.; Champod, C. *J. Forensic Sci.* **2011**, *56*, 102.

(4) Jelly, R.; Patton, E. L. T.; Lennard, C.; Lewis, S. W.; Lim, K. F. *Anal. Chim. Acta* **2009**, *652*, 128.

(5) Choi, M. J.; McDonagh, A. M.; Maynard, P.; Roux, C. *Forensic Sci. Int.*, *179*, 87.

(6) Attard-Montalto, N.; Ojeda, J. J.; Reynolds, A.; Ismail, M.; Bailey, M.; Doodkorte, L.; de Puit, M.; Jones, B. J. *Analyst* **2014**, *139*, 4641.

(7) Bailey, M. J.; Jones, B. N.; Hinder, S.; Watts, J.; Bleay, S.; Webb, R. P. *Nucl. Instrum. Methods Pys. Res., Sect. B* **2010**, *268*, 1929.

(8) Bright, N. J.; Webb, R. P.; Bleay, S.; Hinder, S.; Ward, N. I.; Watts, J. F.; Kirkby, K. J.; Bailey, M. J. *Anal. Chem.* **2012**, *84*, 4083.

(9) Bailey, M. J.; Ismail, M.; Bleay, S.; Bright, N.; Levin Elad, M.; Cohen, Y.; Geller, B.; Everson, D.; Costa, C.; Webb, R. P.; Watts, J. F.; de Puit, M. *Analyst* **2013**, *138*, 6246.

(10) Tang, H.-W.; Lu, W.; Che, C.-M.; Ng, K.-M. *Anal. Chem.* **2010**, *82*, 1589.

(11) Wolstenholme, R.; Bradshaw, R.; Clench, M. R.; Francese, S. *Rapid Commun. Mass Spectrom.* **2009**, *23*, 3031.

(12) Ferguson, L. S.; Creasey, S.; Wolstenholme, R.; Clench, M. R.; Francese, S. *J. Mass Spectrom.* **2013**, *48*, 677.

(13) Bradshaw, R.; Wolstenholme, R.; Blackledge, R. D.; Clench, M. R.; Ferguson, L. S.; Francese, S. *Rapid Commun. Mass Spectrom.* **2011**, *25*, 415.

(14) Ferguson, L.; Bradshaw, R.; Wolstenholme, R.; Clench, M.; Francese, S. *Anal. Chem.* **2011**, *83*, 5585.

(15) Bradshaw, R.; Rao, W.; Wolstenholme, R.; Clench, M. R.; Bleay, S.; Francese, S. *Forensic Sci. Int.* **2012**, *222*, 318.

(16) Ferguson, L. S.; Wulfert, F.; Wolstenholme, R.; Fonville, J. M.; Clench, M. R.; Carolan, V. A.; Francese, S. *Analyst* **2012**, *137*, 4686.

(17) Bradshaw, R.; Bleay, S.; Clench, M. R.; Francese, S. *Sci. Justice* **2014**, *54*, 110.

(18) Ifa, D. R.; Wiseman, J. M.; Song, Q. Y.; Cooks, R. G. *International Journal of Mass Spectrometry* **2007**, *259*, 8.

(19) Takats, Z.; Wiseman, J. M.; Cooks, R. G. *Journal of mass spectrometry : JMS* **2005**, *40*, 1261.

(20) Takats, Z.; Wiseman, J. M.; Gologan, B.; Cooks, R. G. *Science* **2004**, *306*, 471.

(21) Bailey, M.; Bradshaw, R.; Francese, S.; Salter, T. L. R.; Costa, C.; Ismail, M.; Webb, R.; Bosman, I.; Wolff, K.; de Puit, M. *Analyst* **2015**.

(22) Cooks, R. G.; Ouyang, Z.; Takats, Z.; Wiseman, J. M. *Science* **2006**, *311*, 1566.

(23) Albert, A.; Shelley, J. T.; Engelhard, C. *Anal. Bioanal. Chem.* **2014**, *406*, 6111.

(24) Awad, H.; Khamis, M. M.; El-Aneed, A. *Appl. Spectrosc. Rev.* **2015**, *50*, 158.

(25) Harris, G. A.; Galhena, A. S.; Fernandez, F. M. *Analytical Chemistry* **2011**, *83*, 4508.

(26) Ifa, D. R.; Wu, C. P.; Ouyang, Z.; Cooks, R. G. *Analyst* **2010**, *135*, 669.

(27) Li, L. P.; Feng, B. S.; Yang, J. W.; Chang, C. L.; Bai, Y.; Liu, H. W. *Analyst* **2013**, *138*, 3097.

(28) Monge, M. E.; Harris, G. A.; Dwivedi, P.; Fernandez, F. M. *Chem. Rev.* **2013**, *113*, 2269.

(29) Venter, A.; Nefliu, M.; Cooks, R. G. *TrAC, Trends Anal. Chem.* **2008**, *27*, 284.

(30) Weaver, E. M.; Hummon, A. B. *Adv. Drug Delivery Rev.* **2013**, *65*, 1039.

(31) Wu, C. P.; Dill, A. L.; Eberlin, L. S.; Cooks, R. G.; Ifa, D. R. *Mass Spectrom. Rev.* **2013**, *32*, 218.

(32) Hu, Q. Z.; Noll, R. J.; Li, H. Y.; Makarov, A.; Hardman, M.; Cooks, R. G. *J. Mass Spectrom.* **2005**, *40*, 430.

(33) Perdian, D. C.; Lee, Y. J. *Anal. Chem.* **2010**, *82*, 9393.

(34) Perry, R. H.; Cooks, R. G.; Noll, R. J. *Mass Spectrom. Rev.* **2008**, *27*, 661.

(35) Makarov, A.; Denisov, E.; Kholomeev, A.; Baischun, W.; Lange, O.; Strupat, K.; Horning, S. *Analytical Chemistry* **2006**, *78*, 2113.

(36) Gustafsson, J. O.; Eddes, J. S.; Meding, S.; Koudelka, T.; Oehler, M. K.; McColl, S. R.; Hoffmann, P. *Journal of proteomics* **2012**, *75*, 5093.

(37) Pasilis, S. P.; Kertesz, V.; Van Berkel, G. J. *Analytical Chemistry* **2007**, *79*, 5956.

(38) Huang, G.; Li, G.; Ducan, J.; Ouyang, Z.; Cooks, R. G. *Angewandte Chemie* **2011**, *123*, 2551.

(39) Cooks, R. G. D., J. ; Huang, G. ; Li, G. ; Yan, X. ; Sokol, E. ; Li, X.; US. Patent US 20130280819 A1: 2013.

(40) Wiseman, J. M.; Evans, C. A.; Bowen, C. L.; Kennedy, J. H. *Analyst* **2010**, *135*, 720.

(41) Ifa, D. R.; Manicke, N. E.; Dill, A. L.; Cooks, G. *Science* **2008**, *321*, 805.

(42) Chambers, M. C.; Maclean, B.; Burke, R.; Amodei, D.; Ruderman, D. L.; Neumann, S.; Gatto, L.; Fischer, B.; Pratt, B.; Egertson, J.; Hoff, K.; Kessner, D.; Tasman, N.; Shulman, N.; Frewen, B.; Baker, T. A.; Brusniak, M.-Y.; Paulse, C.; Creasy, D.; Flashner, L.; Kani, K.; Moulding, C.; Seymour, S. L.; Nuwaysir, L. M.; Lefebvre, B.; Kuhlmann, F.; Roark, J.; Rainer, P.; Detlev, S.; Hemenway, T.; Huhmer, A.; Langridge, J.; Connolly, B.; Chadick, T.; Holly, K.; Eckels, J.; Deutsch, E. W.; Moritz, R. L.; Katz, J. E.; Agus, D. B.; MacCoss, M.; Tabb, D. L.; Mallick, P. *Nat Biotech* **2012**, *30*, 918.

(43) Fonville, J. M.; Carter, C. L.; Pizarro, L.; Steven, R. T.; Palmer, A. D.; Griffiths, R. L.; Lalor, P. F.; Lindon, J. C.; Nicholson, J. K.; Holmes, E.; Bunch, J. *Anal. Chem.* **2013**, *85*, 1415.

(44) Xiong, X.; Xu, W.; Eberlin, L.; Wiseman, J.; Fang, X.; Jiang, Y.; Huang, Z.; Zhang, Y.; Cooks, R. G.; Ouyang, Z. *J. Am. Soc. Mass. Spectrom.* **2012**, *23*, 1147.

(45) Grossmann, C.; Roos, H. G.; Stynes, M. *Numerical Treatment of Partial Differential Equations*; Springer Berlin Heidelberg, 2007.

(46) Kertesz, V.; Van Berkel, G. J. *Rapid Commun. Mass Spectrom.* **2008**, *22*, 2639.

CHAPTER 5

microMS: A Python Platform for Image-Guided Mass Spectrometry Profiling

Notes and Acknowledgements

This chapter is adapted from a completed manuscript written with coauthors by E. K. Neumann, T. D. Do, and J. V. Sweedler, submitted for publication on March 28, 2017. T. J. Comi prepared the majority of the manuscript and wrote the described source code. E. K. Neumann and T. D. Do performed acquisitions for sequential MS analyses and assisted with code design and debugging. Stanislav Rubakhin is acknowledged for assistance with sample preparation, and Tong Si for useful discussions. Support from the National Institutes of Health, Award Number P30 DA018310 from the National Institute on Drug Abuse and from the National Institute of Mental Health Award Number 1U01 MH109062 are also acknowledged. T. J. Comi and E. K. Neumann acknowledge support from the National Science Foundation Graduate Research Fellowship Program and the Springborn Fellowship. T. J. Comi received additional support through the Training Program at Chemistry-Interface with Biology (T32 GM070421).

Introduction

Image-guided mass spectrometry (MS) provides a link between the spatial dimensions in a digital image and the physical location of a sample within a microprobe system. MS imaging (MSI) is a subset of image guided chemical sampling that frequently utilizes regularly spaced acquisition positions overlaid on an optical scan to recreate the spatial distribution of analytes within a sample. However, traditional MSI is low throughput and less sensitive than targeted profiling when the target objects (e.g. biological cells and bacterial colonies) are widely dispersed or smaller than the microprobe size. In the past decades, single cell analysis with MS has attracted great interest due to its sensitivity and ability to handle volume-limited samples.[1-6]

Many classes of biomolecules within individual cells are detectible with a variety of MS probes, facilitating new discoveries of single cell heterogeneity and a better understanding of the relationship between chemical contents and cellular functions. When MSI is applied to tissue sections,[7-10] the resolution to differentiate neighboring cells requires sampling each cell multiple times, effectively splitting available analytes among pixels. Due to difficulties in sample preparation and stringent instrument requirements, MSI at or below single cell resolution is far from routine in most laboratories. In the case of dispersed cells,[11,12] traditional MS imaging is not an optimal approach as most of the measurement time is spent characterizing the space between the cells. The limitations of MSI for high throughput analysis of single cells have led to the development of new methods to locate or deposit cells.

Recently, high throughput approaches to single cell MS have driven analyses of dissociated single cells which are either chemically labeled[13] or coordinate registered. MS profiling of adhered cells provides advantages in data fusion by simplifying data processing and allowing sequential analysis of the same cell. Microarrays for mass spectrometry[14-16] (MAMS) have demonstrated such capabilities by combining Raman microspectroscopy with MS.[17] As an alternative to MAMS, cells may be randomly seeded on a substrate, greatly relaxing fabrication requirements at the expense of a necessarily gentle sample extraction. Ong et al. presented such an approach, by locating single cells on an indium tin oxide (ITO)-coated glass slide based on their position in a whole-slide fluorescence microscopy image.[18] A challenge with this initial report was the complex scheme for generating custom geometry files, which required manual interaction through several disjointed pieces of software. To facilitate broader adoption of optically-guided single cell profiling, we sought to streamline the process of directing MS acquisition with whole-slide microscopy images. As reported by Jansson et al., the first iteration utilized a point-based similarity registration scheme, which improved target localization accuracy over the previously reported piece-wise linear transform.[19] User interaction was also simplified, allowing fluid interaction with microscope images through a graphical user interface (GUI). All functions required to begin acquiring single cell mass spectra on a Bruker ultrafleXtreme instrument were contained in the single piece of software.

Here we present the first version of microMS to support microscopy-guided MS for a variety of image files and mass spectrometers. The software architecture permits new microprobe instruments to be supported with minor modifications to the source code. Virtually any spatially restricted sampling probe capable of precisely recording and moving to a given location can perform such profiling.

First, the unique features of microMS are described along with the necessary modifications to expand device support for both commercial and customized instruments. We then illustrate an example of using microMS on three MS systems for off-line, targeted profiling of single cells from the mammalian nervous system.

Materials and Methods

Software

microMS is written in python v3.5. In addition to base components, microMS requires the matplotlib, PyQt5, numpy, scipy, openslide, skimage, pyserial packages. Installation instructions, usage details and most recent source code may be found at http://neuroproteomics.scs.illinois.edu/microMS.htm.

The program structure is modeled in Figure 5.1. The main GUI class is composed of two widgets in the GUICanvas package for displaying a microscope image or population-level statistics as a histogram. Each widget interacts with a microMSModel object, which represents a single microscopy experiment (as a blobList and slideWrapper) and mass spectrometer (as a coordinateMapper).

Targets in microMS are represented as objects called “blobs”, to generalize a biological cell as any object formed by a group of high intensity pixels. In Figure 5.2, there are three blobs; each with a unique Cartesian (*x, y*) location on the image, a corresponding effective radius, and circularity. The circularity is a ratio of the blob area to its perimeter squared, scaled between 0 and 1, with 1 being a perfect circle, i.e. blobs 1 and 3 are single cells whereas blob 2 is not. A collection of blobs is stored in a blobList object, which also implements methods to query and filter a population of targets.

A slideWrapper provides an object for interacting with a set of microscopy images representing brightfield and multiple fluorescence channel images. The current field of view is maintained to simplify controller interaction with the image. The ImageUtilities package also contains modules for cell finding, patterning target positions, and optimizing travel paths.

Object models for MS instruments are contained in the coordinateMappers package, as shown in the model in Figure 5.1. The coordinateMapper is an abstract base class providing an interface which the GUI software utilizes to interact with different instrument systems. The core functionality of the mapper is to align pixel positions with physical coordinates and provide a means to translate target positions on an image to instrument-specific directions. The design of the software architecture simplifies the addition of new instruments. Integration of ambient ionization methods, including the single-probe[20,21] or nanoDESI[22] are enticing candidates as they have demonstrated single cell sensitivities in imaging and profiling applications. Currently, four concrete implementations are supplied in the CoordianteMappers package: a Bruker UltrafleXtreme, a Bruker SolariX, the AB Sciex oMALDI sample stage attached to a custom hybrid MALDI/C_{60}^{+}-SIMS, and a lab-built 3-axis liquid microjunction probe. Details about the implementations will be discussed in the next section. Demonstrations for the addition of new instruments to microMS may be found in the user manual packaged with the source code and in Appendix B.

Chemicals

All chemicals were purchased from Sigma Aldrich (St. Louis, MO) and used without further purification.

Single Cell Dissociation

Two, 2-2.5 month old male Sprague Dawley outbred rats (*Rattus norvegicus*) (www.envigo.com) were housed on a 12-h light cycle and fed ad libitum. Animal euthanasia was performed in accordance with the appropriate institutional animal care guidelines (the Illinois Institutional Animal Care and Use Committee), and in full compliance with federal guidelines for the humane care and treatment of animals. Dissected cerebellum and suprachiasmatic nucleus (SCN) tissues were incubated in a solution of 1% Hoechst 33342 in oxygenated modified Gey's balanced salt solution (mGBSS) for 30 minutes at 37°C. The mGBSS solution was removed and the tissues were incubated in an oxygenated solution of 6 units of papain, 1 mM L-cysteine, and 0.5 mM ethylenediaminetetraacetic acid for 80 minutes at 37°C. Tissue was then mechanically dissociated in mGBSS with 0.04% paraformaldehyde. A solution of 80% glycerol in mGBSS was added to a final concentration of 40% glycerol. The cell suspension was then transferred onto ITO-coated glass slides (Delta Technologies, Loveland, CO) with at least 12 fiducial marks etched by a diamond-tipped pen.

Microscopy Imaging

Brightfield and fluorescence images were acquired on a Zeiss Axio Imager M2 (Zeiss, Jena, Germany) equipped with an Ab cam Icc5 camera, X-CITE Series 120 Q mercury lamp (Lumen Dynamics, Mississauga, Canada), and a HAL 100 halogen illuminator (Zeiss, Jena, Germany). The 31000v2 DAPI filter set was used for fluorescence excitation. The images were acquired in mosaic mode with a 10x objective and 13% overlap. Images were processed and exported as tiff files using ZEN software version 2 blue edition (Zeiss, Jena, Germany).

Sample Preparation

Slides were coated with 50 mg/mL 2,5-dihydroxybenzoic acid (DHB) dissolved in 1:1 (v/v) LC-grade ethanol:water with 0.1% trifluoroacetic acid with an automatic sprayer described previously.[19,23] The matrix solution was supplied at 10 mL/hr and nebulized with N_2 gas at 50 psi over 100 passes. Samples were affixed to a rotating plate with the nebulizer positioned 1.5 cm above the samples, resulting in a MALDI matrix thickness of ~0.1-0.2 mg/cm^2.

Instrument Parameters

Single cell analysis was performed on three instruments. The UltrafleXtreme mass spectrometer (Bruker Daltonics, Billerica, MA) was set with a mass window of *m/z* 400-3000. The "Ultra" (~100 µm footprint) laser setting was used with 300 laser shots at 1000 Hz for each cell to generate a MALDI-TOF mass spectrum at each cell. The second instrument is a 7 T SolariX FT-ICR mass spectrometer (Bruker Daltonics, Billerica, MA), operated with a mass window of *m/z* 150-3000, yielding a 4 Mword time-domain transient. Spectra were calibrated to the phosphatidylcholine headgroup at *m/z* 184.07332. Adsorption mode was used to effectively double the mass resolving power. Each MALDI spectrum was acquired with 20 laser shots at 1000 Hz and 60% laser energy. The laser setting produced a ~100 µm footprint. The last instrument is a custom hybrid MALDI/C_{60}^+ Q-TOF mass spectrometer, described in detail elsewhere.[11] The C_{60}^+ ion beam was utilized for secondary ion mass spectrometry (SIMS), with the mass analyzer operated in positive mode with a mass range of *m/z* 60-850. Correctly parsing the spectra requires additional instrument modifications and data analysis routines, described elsewhere.[24]

Results and Discussion

Instrument support in microMS

Bruker instruments are discussed first as their MALDI sample stages and coordinate mappers are similar. The commonality is exploited by the brukerMapper abstract base class, which is a derived class of coordinateMapper. BrukerMapper implements methods for reading and writing xeo geometry files, which are required in Bruker software for automatic acquisition. The brukerMapper class also defines an intermediate coordinate system between physical, motor coordinates and the fractional distances used in xeo files. The classes derived from brukerMapper require a limited set of concrete method implementations to be fully functional as many features are supported in the bases class. The simplest case is ultraflexMapper, for the Bruker ultrafleXtreme instrument, which defines the required methods to parse user input.

The solarixMapper class for a Bruker SolariX instrument is similar to the ultraflexMapper class with three minor modifications: 1) the xeo files are limited to 400 positions, 2) an xlsx Excel file is also required for automatic acquisition, and 3) input coordinates are read directly from the system clipboard. The flexImagingSolarix object extends solarixMapper and overrides the saved file format for import in flexImaging software. These two instruments provide examples of supporting microscopy-guided MS on Bruker MALDI sources.

Instruments from other vendors inherit directly from the coordinateMapper. One example is the oMALDIMapper for interfacing with an AB Sciex oMALDI server. With this instrument, the sample positions are encoded in ptn pattern files which contain an x,y coordinate relative to the starting position with calibrated motor steps. Hence, in comparison to brukerMapper, oMALDIMapper transforms motor coordinates to ptn coordinates instead of fractional distances. To further simplify correlation of mass spectra to image coordinates, a corresponding text file is also exported with pixel positions of each target. As a final consideration, the sample stage was found to have significant motor slop upon changing direction. The motor slop is corrected before exporting the ptn file to ensure accurate targeting.

In the preceding examples, microscopy images are correlated with physical positions on a mass spectrometer sample stage to generate instrument-specific target coordinates. This off-line workflow is ideal for instruments lacking support for external control of the sample stage, as is usually the case. To demonstrate capabilities with on-line analysis and instrument control, additional interfaces were developed for controlling Zaber linear actuators. The zaberMapper class contains a simple implementation of the abstract base class coordinateMapper. It also has an instance variable connectedInstrument, which is used by microMS to interact with the sample stage. Another abstract base class, connectedInstrument specifies the method signatures necessary for a connected instrument. The concrete implementation provided is a system with three linear actuator stages, zaber3axis. This module inherits from zaberInterface, containing serial wrappers to simplify interaction with each stage, and implements the connectedInstrument interface. In addition to reading the current, physical position for coordinate registration, the user directs stage movement on the optical image or with key strokes.

microMS Functionalities

General features of microMS include locating targets, filtering the target population, patterning each target, and coordinating the image with the physical location of a mass spectrometer stage. Only the last step is instrument specific. Users should refer to the User Guide in Appendix B for a comprehensive illustration of these functionalities.

(a) Locating targets. Targets may be specified on the microscope image either by manually selecting locations or by performing automatic blob finding. In the former approach, targets are added via holding the "shift" key and left mouse clicking on the center of the feature of interest. This generates a blob of default radius and circularity of 1. A custom radius is specified by clicking and dragging from the circumference to the center of the feature.

In automatic blob finding, the search takes place over the entire image area unless a region of interest (ROI) is specified. ROIs are defined by clicking and dragging a rectangular area or drawing the region a vertex at a time. The blob finding algorithm thresholds the specified image color and then groups together pixels above that threshold, as shown in Figure 5.2B. Each group is then evaluated for its size and circularity. Putative blobs falling outside the user-specified parameters are discarded. Several features are available to assist with selection of suitable blob finding parameters. Different blob finding parameters are interactively tested on the current image field of view. Additionally, microMS reports pixel intensities, object size and circularity of positions selected with a middle mouse button click. Judicious selection of these parameters will find most cells while excluding imaging and background artifacts.

After the targets are located, their properties are stored as lists within microMS. Up to ten separate target lists are maintained and each list is displayed as a different color. New target sets generated by filtering or patterning are automatically stored in an empty list with the original target set left intact.

(b) Filtering targets. Frequently, it is beneficial to filter the target list, either to refine putative blobs or to stratify targets based on morphology. Basic filtering methods provided by microMS are selected through the menu bar which include ROI filtering and distance filtering. Within a specific ROI, the blobs can be selectively removed or exclusively retained in a new target list. ROI filtering is especially useful for removing targets which are near fiducials or potentially contaminated by substrate background. Distance filtering helps to ensure each MS target position will correspond to a unique object (e.g. a single cell). An appropriate value for distance filtering is chosen based on the microprobe size, target accuracy, and the desired number of samples per blob. For example, a 100 μm diameter probe on a system with 50 μm target accuracy would require distance filtering of at least 100 μm to minimize the chance of sampling a nearby blob. During distance filtering, any target with a neighbor closer than the specified value is removed from the blob list.

In addition to common filtering functions, microMS supports interactive examination of population-level statistics through the histogram window to partition the target list (Figure 5.3). Metrics include blob size, circularity, nearest neighbor distance, and fluorescence intensity. Note that there is some redundancy between histogram filtering and blob finding. The overlap permits the selection of lenient blob finding parameters to exhaustively identify all putative cells which are then refined to the final target set. Such a scheme allows distance filtering to identify all possible contaminating objects and sub-classification based on size.

High and low pass filters of the targets may be defined on the histogram window. Targets falling within a filter range are dynamically displayed on the microscope image in the corresponding color (Figure 5.3B-C). Selecting a blob in the microscope image highlights its value on the histogram to assist with defining filter limits. High and low pass filters define new target lists that may be further refined by additional histogram operations. This function allows operations such as filtering a population based on size followed by selection of targets with a particular fluorescent stain. More routinely, the histogram provides a simple method to identify and remove artifacts from blob finding. Figure 5.3C shows an example of isolating unresolved cells, which helps ensure data quality.

(c) Patterning targets. By default, microMS generates one acquisition target per blob. Single target sampling is sufficient when the microprobe size is similar to or larger than the target object. However, when the object is larger than the probe, a single acquisition is insufficient to robustly sample heterogeneous objects. Alternatively, MSI of each blob can be acquired at each target location. To address advanced sampling requirements, microMS provides three sample patterning schemes, shown in Figure 5.4.

The first option is a rectangular packed array of points centered on the target (Figure 5.4A). Users select a raster spacing and number of layers to define the overall size of the image. Alternatively, the size is dynamically adjusted to the target radius to ensure complete sampling of heterogeneously sized populations. The resulting data is directly interpretable as an MS image with common MSI software.

Similar to rectangular packing is the hexagonal close packing pattern (Figure 5.4B). With a circular desorption probe, the hexagonal packing provides denser sampling of the target. Users define the target separation, number of layers, and specify dynamic layering. While hexagonally packed data are more difficult to reconstruct into an image, averaging the spectra yields a representative spectrum for the blob.

Finally, the circular pattern generates targets around the circumference of each blob (Figure 5.4C). In some cases, analyzing the center of a blob produces low sensitivity due to the morphology of the target or the biological nature of the samples. Instead, targets are placed immediately outside of a blob to acquire representative spectra. For circular patterning, the user defines a minimum target-to-target distance, maximum number of targets, and offset from the circumference. The actual number of targets around a blob is determined by the blob size and the specified offset while maintaining the target-to-target distance above the minimum limit. Targets are then equally spaced around the blob. Averaging the resulting data provides a characteristic spectrum of the area directly surrounding each blob.

(d) Coordinate transformation to instrument systems. Once all targets are determined, the pixel positions must be translated into the physical coordinates of a mass spectrometer or similar platform. Image correlation in microMS is accomplished through a point-based similarity registration. In point-based registration, the target localization error scales inversely with the square root of the number of fiducial points. As such, while microMS supports arbitrary numbers of fiducials, at least 12 fiducials are recommended for robust coordinate training. Similarity transformations do not correct for shearing of images, so the field of view must remain normal to the sample surface during image acquisition and MS analysis.

Generally, a fiducial is located in the microscope image and the instrument system with the assistance of an integrated video camera. When the microprobe is positioned over the center of a fiducial mark, the same location is selected on the image in microMS with a right mouse click. This opens a popup window requesting the physical *x, y* position. A default *x, y* position is displayed in the popup which directly reads the stage position, pastes text from the computer clipboard, or predicts the closest location, depending on the selected instrument.

Fiducials are displayed on the microscope image as blue circles with labels corresponding to the nearest set position on the instrument, as shown in the schematic of Figure 5.5. A few feedback features are included to help the user assess the quality of the current registration. If applicable, labels display set points of the instrument coordinate system. The labels shown in Figure 5.5B correspond to a Bruker MTP slide II adapter. For some instruments, a set of preprogrammed positions are displayed, showing the predicted location of those points on the microscope image. A large deviation, typically due to an inaccurate input will be detected by a discrepancy in the expected and displayed label. Finally, the fiducial with the worst fiducial localization error is highlighted in red, indicating that specific position assignment should be reconsidered. Correcting the problematic fiducial will cause the next worst fiducial to be highlighted. Once the same fiducial stays highlighted, the registration is close to optimal. Adjusting the worst fiducial is good practice to produce accurate targets.

With a full set of fiducials, the target positions may be saved in instrument-specific format for offline analysis. Alternatively, microMS can communicate directly with an instrument to instruct it to move and perform an analysis. Due to limited vendor support, direct instrument control is demonstrated on a lab-built stage.

The current microMS distribution supports the Bruker ultrafleXtreme, Bruker solariX, AB Sciex oMALDI server, and a lab-built liquid microjunction extraction stage. For the supported MS systems, the user registers fiducials and saves instrument positions without modifications. Furthermore, microMS has ample room for customization due to the abstract base class construction of the CoordinateMappers package. The general framework remains unchanged, but there are opportunities to tune each function to a specific application. Examples include the ability to directly read fiducial positions from an instrument, grab the contents of the computer clipboard, and perform stage movement slop correction.

Accuracy of point based similarity registration. A vital metric for optically guided profiling is the target localization error. An accurate transformation between optical image and physical location ensures that each sample corresponds to the position of interest. microMS allows training sets of arbitrary sizes. Including more fiducials reduces target localization error, effectively distributing uncertainty of a given fiducial over the entire transformation. Several factors influence accuracy including the precision of stage movement, fiducial localization accuracy (in both image and physical coordinate systems), number of fiducials, whole-slide image stitching, and proper sample positioning during image and MS acquisition. Users should carefully consider these factors to establish adequate probe size and distance cutoffs prior to data acquisition.

To assess the accuracy of an MS system with microMS, an image based method was developed to link the requested and actual target positions, as shown in Figure 5.6. The target localization error is defined as the Euclidean distance between a requested position and the actual, transformed position during coordinate registration and is synonymous with accuracy. To assess this value, a thin layer of DHB matrix was coated on an ITO glass slide to act as a tracer for the probe position. A standard sample was prepared with 16-24 fiducials along the exterior of a target. An additional set of fiducials were included within this region to mimic the location of samples in a profiling experiment. The interior marks were not used for coordinate registration, but rather to assist with overlaying the pre- and post-analysis images. Several target locations were manually placed around each interior fiducial. Dividing the targets between multiple trials is useful for designing experiments to test the effect of possible confounding variables.

Next, fiducial training was performed with the MS system and the set of targets was desorbed with sufficient time to noticeably remove the DHB matrix. After desorption, the target area was optically imaged again to reveal the actual position of sampling events. Desorption locations and sizes are marked as blobs in microMS and saved for further analysis. To overlay the two images, subsets of the pre- and post-desorption image were cropped and roughly positioned prior to intensity-based registration with custom scripts in MATLAB (R2015b). The resulting transformation was used to map the target pixel positions onto the post-extraction image. The distance between the requested and actual desorption positions is a direct measurement of the target localization error. With this method, the target localization of the Bruker ultrafleXtreme was found to be 38.3 ± 3.9 µm (mean ± S.E.M, *n* = 71, Figure 5.6) over an area of approximately half a microscope slide, an error of about one part per thousand. As previously mentioned, the probe radius should be as large as the target localization error and the distance filter applied should be larger than the sum of this error and the probe radius.

In experiments to assess the effect of various confounding factors on target accuracy, desorption was repeated multiple times with the same sample and slide image. Different laser spot sizes, users, target locations and fiducial training sets were examined. The only significant factor found was the fiducial training set (Figure 5.6D). Overall accuracy is not dependent on the target location (Figure 5.6E), laser spot size or user (data not shown). Within an experiment, the accuracy is fairly constant, independent of the user or location on the sample. However, repeating an experiment with the same sample could produce significantly different accuracy. This result confirms the profound effect of quality fiducial training sets on the target accuracy. Extreme care is required when training fiducials to ensure the image target locations correspond to the expected mass spectra.

A demonstration: Sequential analysis of the same target

For single cell profiling experiments, the physical location of a cell on the slide effectively isolates it from neighbors and prevents mixing, which greatly simplifies data fusion. microMS provides a utility for performing sequential analysis of the same target on different instruments with ease. Using the optical image as a map to record each target address, the image position can be transformed into any supported instrument coordinate system. A careful selection of the order of experiments facilitates the repeated analysis of a sample to provide complementary chemical information.

Figure 5.7 shows two examples of sequential single cell profiling using MS instruments with different capabilities. In panel A, dispersed, rat cerebellum cells were initially profiled with a Bruker ultrafleXtreme to rapidly assess the lipid content with moderate resolution and mass accuracy. From the initial mass spectral dataset, cells without significant lipid signals are discarded from further consideration as they likely represent artifacts from optical imaging or sample preparation such as dust particles. The resulting population is then selected for follow-up, high resolution, high mass accuracy analysis with a Bruker solariX FT-ICR. Due to the increased sample acquisition time, exhaustive analysis of large populations is cost-prohibitive. Performing a preliminary filtering maximizes the efficiency of subsequent data analysis, without consuming the entire cellular content.

While the overall lipid profiles are similar, there are some discrepancies between lipid ratios of the two methods. This could represent changes in the sample layers that each technique is analyzing. Nonetheless, the advantage of single cell FT-ICR is immediately apparent with the ppm mass accuracy and over an order of magnitude higher mass resolution, shown in each inset for putative $[PC(32:0)+H]^+$. Once the MALDI-TOF has identified cells with abundant lipid signal, they are filtered to locate individuals requiring exact mass measurement for elemental composition analysis. Such a workflow facilitates exhaustive cell population analysis while efficiently utilizing the FT-ICR as needed.

As a second example, Figure 5.7B displays a C_{60}^+-SIMS mass spectrum and the corresponding MALDI-TOF mass spectrum of a single cell derived from the rat suprachiasmatic nucleus. Here, the low sample consumption of SIMS was leveraged by follow-up MALDI-TOF to provide more, complementary information than would be possible with either technique alone. In the low mass range, peaks corresponding to phosphatidylcholine headgroup and a cholesterol fragment are apparent in the SIMS spectrum at *m/z* 184.09 and 369.31 respectively. MALDI-TOF demonstrates better sensitivity to intact lipids and detects lipid dimers and peptides. Comparing the identity of lipids over the same range, SIMS appears to favor sodiated adducts ($[PC(32:0)+Na]^+$ and $[PC(34:1)+Na]^+$ at *m/z* 756.47 and 782.55 respectively) more than the protonated forms seen in MALDI-TOF ($[PC(32:0)+H]^+$ and $[PC(34:1)+H]^+$ at *m/z* 734.54 and 760.55 respectively). These relative intensities likely reflect the different ionization processes occurring in each instrument. Together, a wide mass range is covered to provide a more complete profile of the sample.

These examples may represent the first demonstration of multiple MS platforms measuring the same individual cells with high throughput. In each example, the ability to repeatedly analyze the same cell was leveraged to acquire complementary information from multiple instruments. Such an experiment would be difficult to perform at high throughput without linking the target locations by the optical image of each sample. With microMS, sequential analysis is facile, enabling each cell to be exhaustively characterized by multiple techniques.

Conclusions and Future Directions

microMS is the first generation of an open source python package for robust image analysis and coordinate registration, which are essential for optically-guided MS profiling. microMS provides a rich feature set for image analysis suited for optically-guided MS profiling. Targets may be automatically located, filtered, stratified and patterned prior to MS analysis. These functions provide access to single cell profiling with multichannel fluorescence image analysis. The unique aspect of microMS is how mass spectrometers are represented for MS profiling. The implementation of specific MS systems through an abstract base class and software architecture provides a straightforward means for adapting microMS to arbitrary microprobe instruments. While this simplifies connecting microMS to new systems, it also facilitates sequential analysis of the same target by uniquely addressing each cell coordinate. We believe the rich feature set and ease of extending microMS to a variety of mass spectrometers and other instruments will facilitate the growth of single cell profiling.

Figures

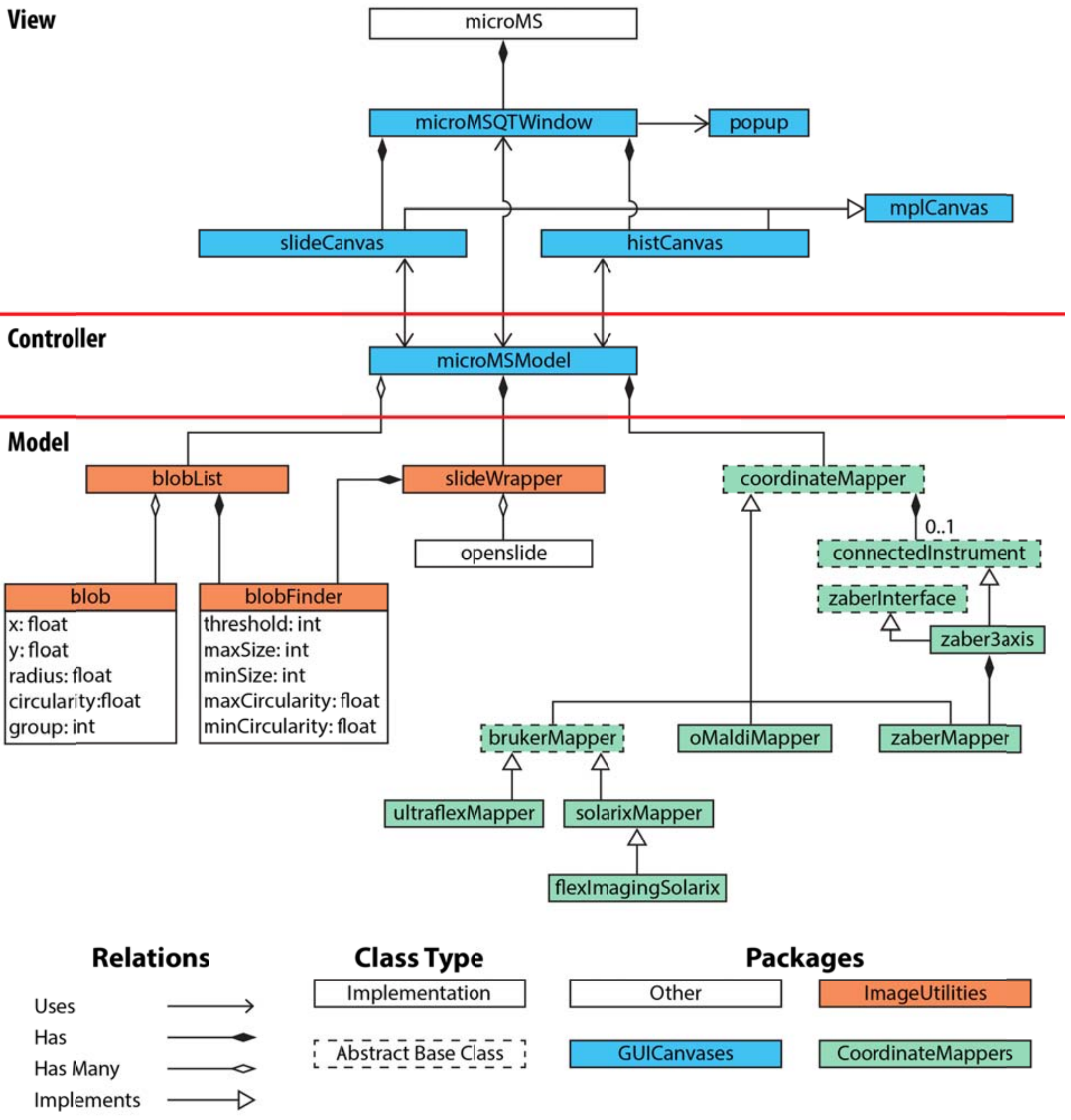
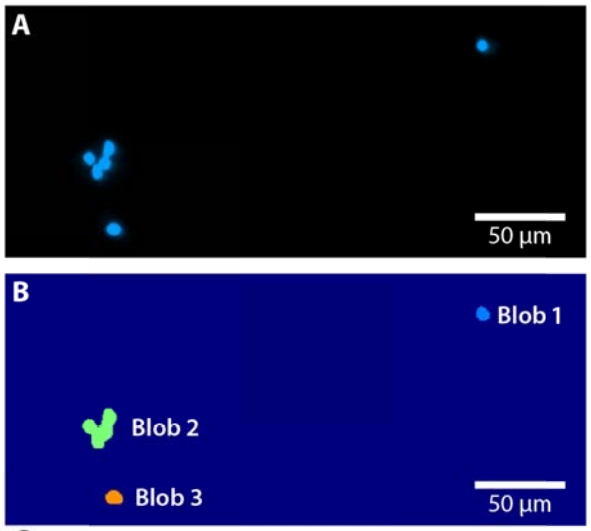


Figure 5.1. Unified Modeling Language (UML) class diagram of microMS showing the organization and relation between each file. The overall architecture follows a model-view-controller design pattern. The view and controller are contained in the GUICanvases package which holds the GUI components and the controller (microMSModel). The model is split between ImageUtilities and CoordinateMappers packages. ImageUtilities contains modules for interacting with microscopy images and targets. CoordinateMappers provides an interface to another coordinate system for on-line or off-line analysis of targets.



C

Blob	X(pix)	Y(pix)	Radius† (pix)	Circularity‡
1	44619	27077	6.79	0.99
2	44241	27192	15.4	0.50
3	44251	27257	7.94	0.98

Figure 5.2. Overview of blob finding with microMS. From an input image (A), the pixel intensity is filtered by a threshold intensity. (B) Pixels above the threshold are grouped with neighboring pixels to generate putative blobs. Each group of pixels is evaluated for its size (in pixels) and circularity. The area (*A*) and perimeter (*P*) are directly measured from the threshold image. †The effective radius is calculated as $\sqrt{A/\pi}$. ‡The circularity is calculated as $\frac{4\pi A}{P^2}$.

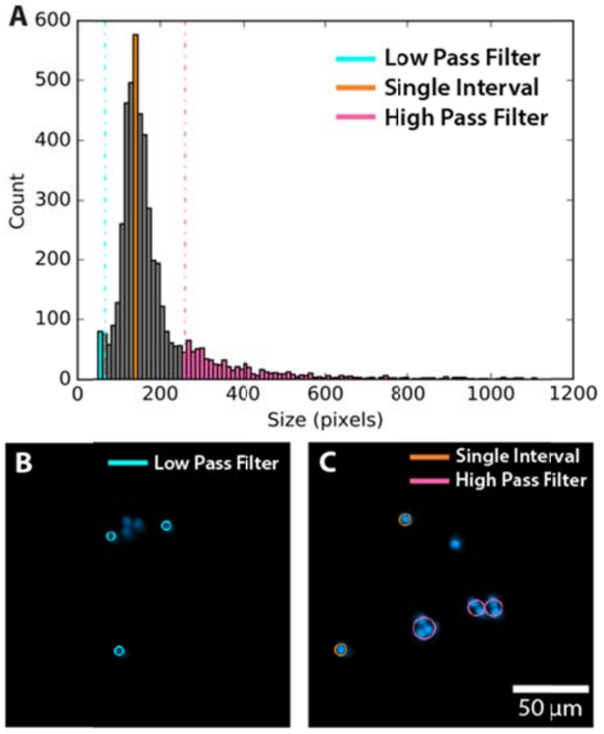


Figure 5.3. Population-level filtering through the histogram window. (A) A population of found blobs may be filtered by size, circularity, minimum pairwise distance, or fluorescence intensity. The histogram can be divided into a low pass, high pass, or single interval with the appropriate blobs dynamically colored in the microscope image (B and C).

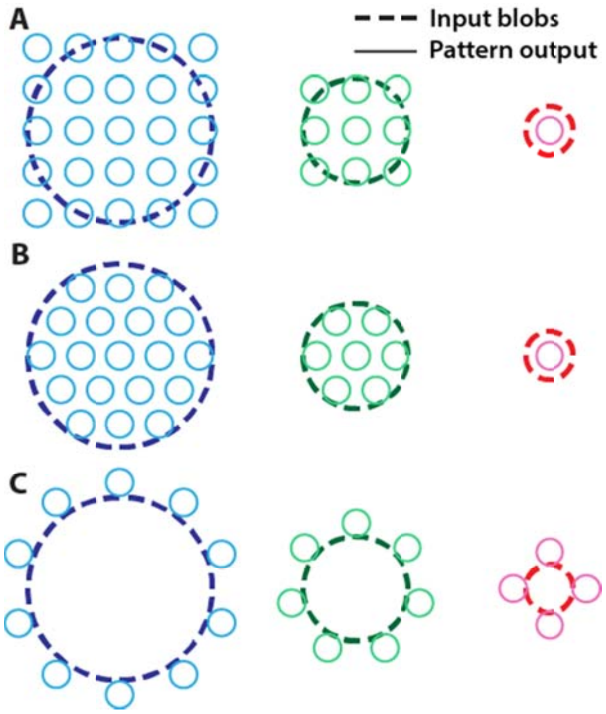


Figure 5.4. Schematic illustration of the three patterning methods provided in microMS. Each example shows dynamic pattern sizes, though static patterning is also available. (A) Rectangular packing produces even *x,y* spacing around the center point of each blob. (B) Hexagonal packing provides more efficient sampling of the entire blob. (C) Circular packing positions targets around the circumference of each blob.

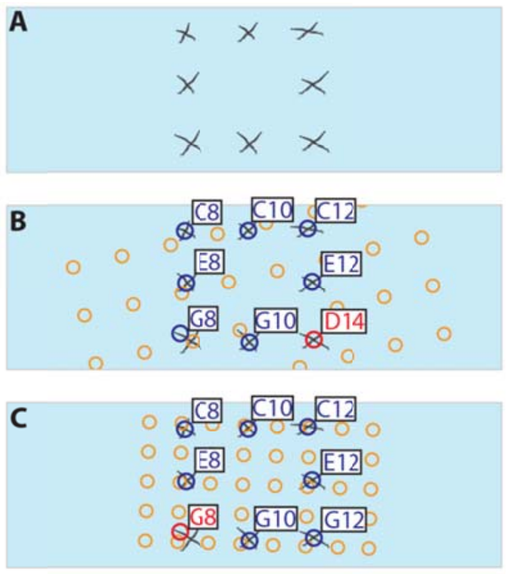


Figure 5.5. Schematic of fiducial training. (A) The input image includes several fiducial points, such as etched x marks on the glass slide. (B) An initial attempt at registration with labels of the nearest named coordinate for each fiducial. Fiducials are shown in blue, except the point with the worst fiducial localization error, which is in red. A set of predicted locations in yellow are also be displayed. (C) After removing and retraining the worst fiducial the next worst fiducial is dynamically highlighted.

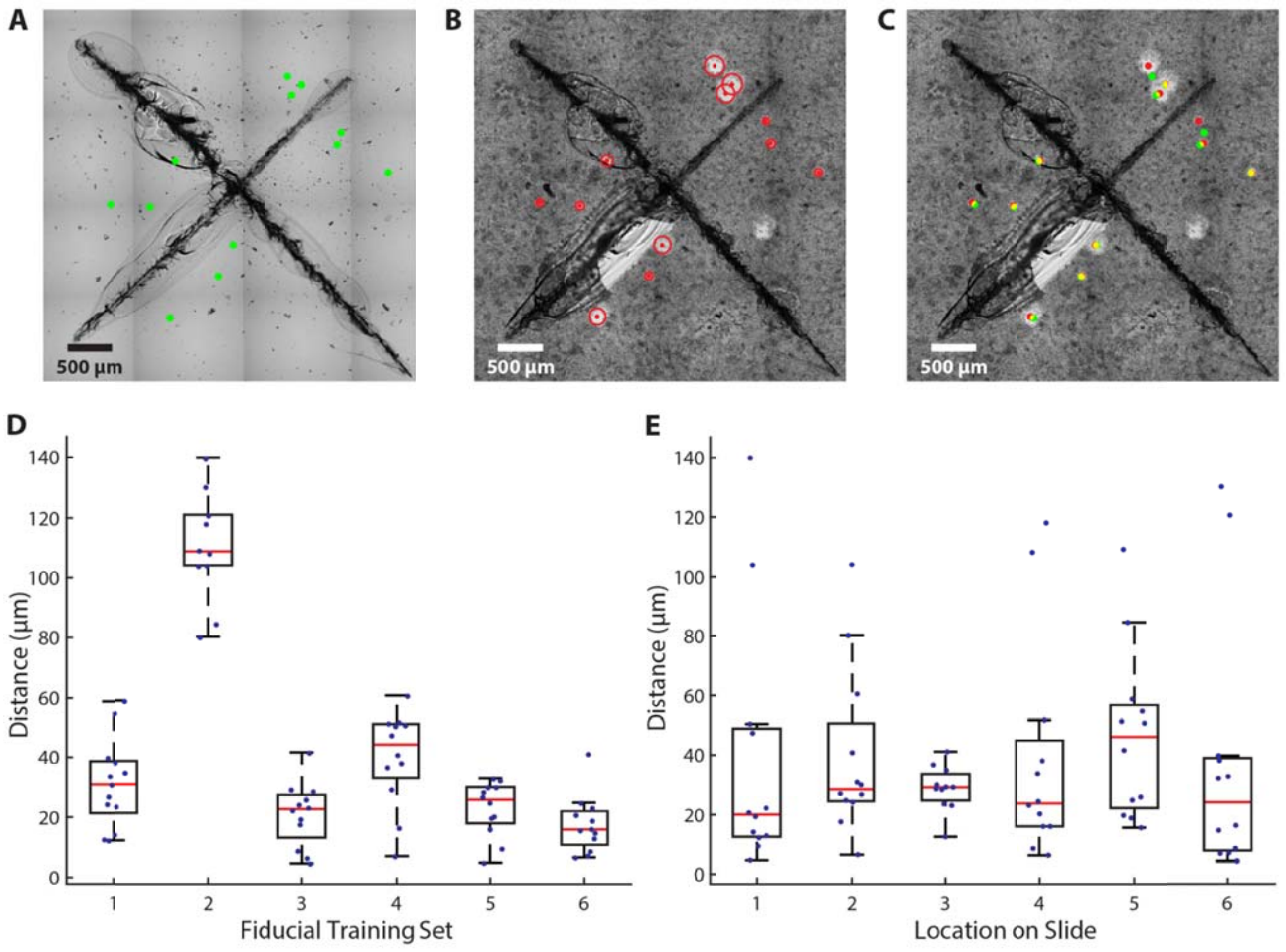


Figure 5.6. Determination of target localization error. (A) Target locations (green) are marked around an image of an etched x mark. The sample is then coated with a thin layer of MALDI matrix and analyzed by optically-guided MS to generate desorption craters in the matrix. (B) The location of resulting desorption events (red) are determined by optical microscopy. (C) Image registration of panels A and B allows the direct mapping of requested target locations onto the desorption marks. Overlap is shown in yellow. The distance between these positions is the target localization error of the registration set. The effect of various parameters may be assessed simultaneously including multiple training sets, location on slide, or size of microprobe, as shown here. A three-way linear ANOVA demonstrated that while the specific fiducial training set significantly affected accuracy ($p << 0.05$), the location on the slide ($p = 0.6$) and spot size ($p = 0.3$) did not.

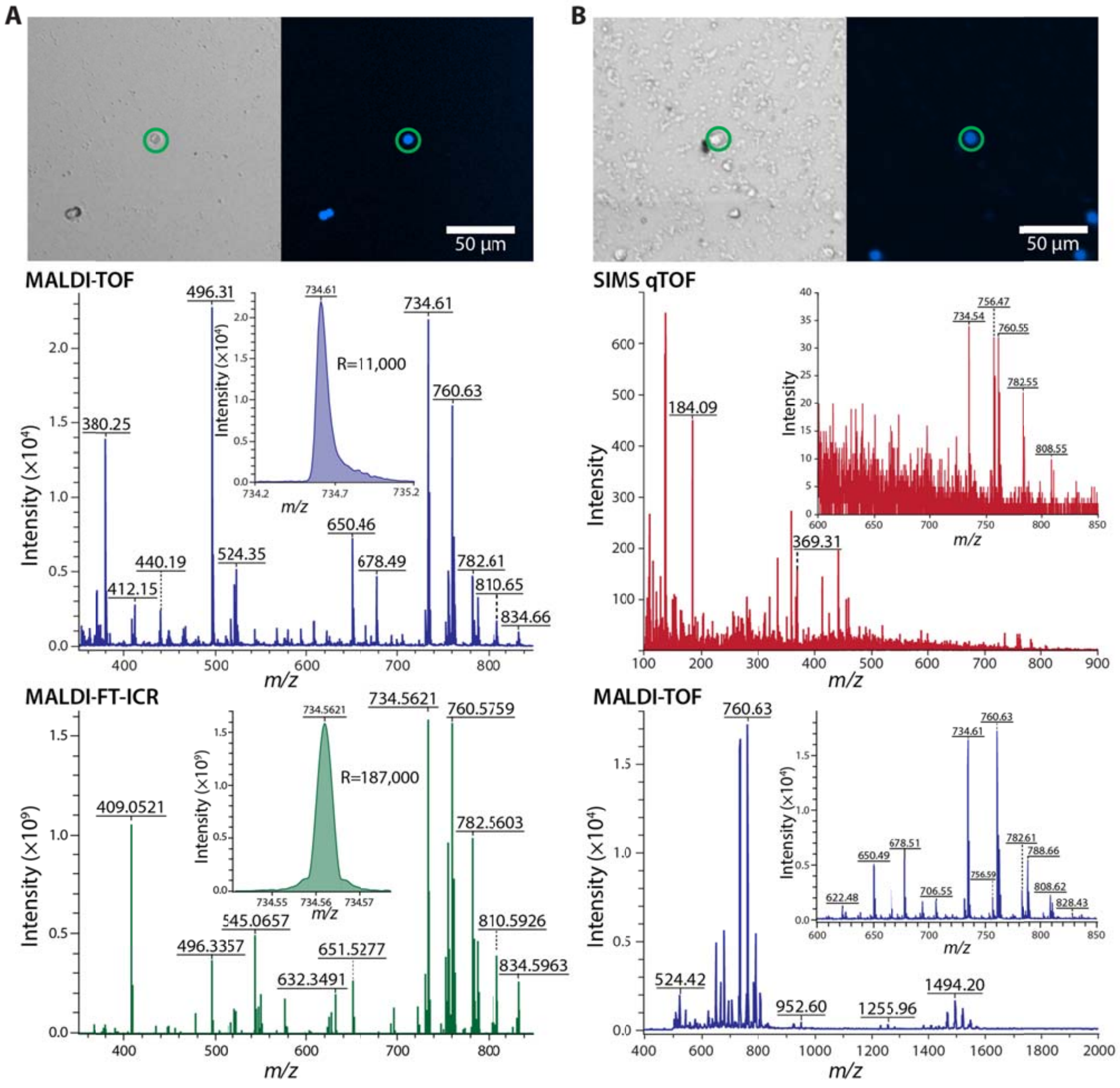


Figure 5.7. Sequential analysis of the same cell with two separate MS systems. Once a cell has been located in the optical image (top), its location remains fixed through multiple analyses allowing two instruments to probe the same set of selected cells. (A) MALDI-TOF MS (middle) of a cerebellum-derived cell followed by MALDI-FT-ICR MS (bottom). MALDI-TOF provides high throughput screening of thousands of cells to highlight rare or representative individuals. Here, FT-ICR provides high mass resolution and high mass accuracy for unequivocal elemental composition of selected cellular contents. (B) SIMS profiling (middle) followed by MALDI-TOF MS (bottom) with a DHB-coated, suprachiasmatic nucleus derived cell. SIMS provides information on small molecule compounds while MALDI-TOF MS effectively detects larger species, such as lipid dimers and peptides. The inset demonstrates some overlap of intact lipid coverage from each modality.

References

(1) Lanni, E. J.; Rubakhin, S. S.; Sweedler, J. V. *J Proteomics* **2012**, *75*, 5036.

(2) Rubakhin, S. S.; Lanni, E. J.; Sweedler, J. V. *Current Opinion in Biotechnology* **2013**, *24*, 95.

(3) Comi, T. J.; Do, T. D.; Rubakhin, S. S.; Sweedler, J. V. *Journal of the American Chemical Society* **2017**.

(4) Armbrecht, L.; Dittrich, P. S. *Analytical Chemistry* **2017**, *89*, 2.

(5) Chen, X.; Love, J. C.; Navin, N. E.; Pachter, L.; Stubbington, M. J. T.; Svensson, V.; Sweedler, J. V.; Teichmann, S. A. *Nat Biotech* **2016**, *34*, 1111.

(6) Zenobi, R. *Science* **2013**, *342*, 1243259.

(7) Korte, A. R.; Yandeau-Nelson, M. D.; Nikolau, B. J.; Lee, Y. J. *Anal Bioanal Chem* **2015**, *407*, 2301.

(8) Kompauer, M.; Heiles, S.; Spengler, B. *Nat Meth* **2016**, *advance online publication*.

(9) Lee, J. K.; Jansson, E. T.; Nam, H. G.; Zare, R. N. *Analytical Chemistry* **2016**, *88*, 5453.

(10) Passarelli, M. K.; Ewing, A. G. *Current opinion in chemical biology* **2013**, *17*, 854.

(11) Lanni, E. J.; Dunham, S. J.; Nemes, P.; Rubakhin, S. S.; Sweedler, J. V. *Journal of The American Society for Mass Spectrometry* **2014**, *25*, 1897.

(12) Yeager, A. N.; Weber, P. K.; Kraft, M. L. *Biointerphases* **2016**, *11*, 02A309.

(13) Spitzer, Matthew H.; Nolan, Garry P. *Cell* **2016**, *165*, 780.

(14) Pabst, M.; Fagerer, S. R.; Kohling, R.; Kuster, S. K.; Steinhoff, R.; Badertscher, M.; Wahl, F.; Dittrich, P. S.; Jefimovs, K.; Zenobi, R. *Anal Chem* **2013**, *85*, 9771.

(15) Ibáñez, A. J.; Fagerer, S. R.; Schmidt, A. M.; Urban, P. L.; Jefimovs, K.; Geiger, P.; Dechant, R.; Heinemann, M.; Zenobi, R. *Proceedings of the National Academy of Sciences* **2013**, *110*, 8790.

(16) Krismer, J.; Sobek, J.; Steinhoff, R. F.; Fagerer, S. R.; Pabst, M.; Zenobi, R. *Applied and environmental microbiology* **2015**, *81*, 5546.

(17) Fagerer, S.; Schmid, T.; Ibanez, A.; Pabst, M.; Steinhoff, R.; Jefimovs, K.; Urban, P.; Zenobi, R. *Analyst* **2013**.

(18) Ong, T. H.; Kissick, D. J.; Jansson, E. T.; Comi, T. J.; Romanova, E. V.; Rubakhin, S. S.; Sweedler, J. V. *Anal Chem* **2015**.

(19) Jansson, E. T.; Comi, T. J.; Rubakhin, S. S.; Sweedler, J. V. *ACS Chemical Biology* **2016**, *11*, 2588.

(20) Pan, N.; Rao, W.; Liu, R.; Kothapalli, N.; Burgett, A.; Yang, Z. *Planta Medica* **2015**, *81*, IL55.

(21) Pan, N.; Rao, W.; Kothapalli, N. R.; Liu, R.; Burgett, A. W.; Yang, Z. *Analytical Chemistry* **2014**, *86*, 9376.

(22) Laskin, J.; Heath, B. S.; Roach, P. J.; Cazares, L.; Semmes, O. J. *Analytical Chemistry* **2012**, *84*, 141.

(23) Li, B.; Comi, T. J.; Si, T.; Dunham, S. J.; Sweedler, J. V. *Journal of Mass Spectrometry* **2016**, *51*, 1030.

(24) Do, T. D.; Comi, T. J.; Dunham, S. J. B.; Rubakhin, S. S.; Sweedler, J. V. *Analytical Chemistry* **2017**.

CHAPTER 6

SINGLE CELL PEPTIDE HETEROGENEITY OF RAT ISLETS OF LANGERHANS

Notes and Acknowledgements

This chapter was published as an original research article in *ACS Chemical Biology* **2016**, DOI: 10.1021/acschembio.6b00602, coauthored by E.T. Jansson, S.S. Rubakhin, and J.V. Sweedler. The article is adapted and reprinted here with permission from the American Chemical Society, copyright 2016. E.T. Jansson performed islet isolation, LC-MS analysis, molecular dynamics simulations and authoring of the manuscript. T.J. Comi performed MALDI MS analyses, multivariate statistics, and authoring of the manuscript. The project described was supported by the National Institutes of Health, Award Number P30 DA018310 from the National Institute on Drug Abuse, and Award Number U01 MH109062 from the National Institute of Mental Health. The Core Facilities at the Carl R. Woese Institute for Genomic Biology, UIUC, provided access to the digital pathology scanner. We thank Ning Yang, UIUC, for technical assistance with the FT-ICR MS.

Introduction

The heterogeneity and variability of individual cells is critical for the survival and propagation of life. Cellular heterogeneity is thought to be necessary for higher-level systems, suggesting that "variation is function", where nuanced variation across single cells provides a means of graded response.[1] Greater phenotypic heterogeneity is also vital; for example, the cellular heterogeneity of islets of Langerhans, which are composed of four major cell types. Each type of endocrine cell is functionally interconnected, yet chemically distinct, providing plastic responses of an organism to fluctuating environments. Cellular heterogeneity facilitates the endocrine functions of islets of Langerhans that are essential for glucose homeostatis.[2,3]

Due to the importance of islets of Langerhans in normal and pathological glucose utilization, the cellular heterogeneity and variability of this microorgan are well studied using approaches such as immunohistochemical profiling,[4-7] single cell transcriptomics analysis,[1,8,9] and metabolomic and proteomic profiling.[10,11] Mass spectrometry (MS) has become an important tool for the investigation of peptide content in organelles, cells, and tissues, such as the endocrine pancreas. MS has been efficiently used to probe the secretome, metabolome, peptidome, and proteome of whole islets.[12-16] MS investigations of the peptide content of islets have revealed unknown cell-to-cell signaling molecules and indicated novel prohormone processing. However, most prior studies utilized individual or pooled islets, consisting of thousands of cells. Hence, these ensemble measurements conceal the biological variability and functional heterogeneity of individual cells. Matrix-assisted laser desorption/ionization (MALDI) MS is uniquely suitable for the detection and characterization of peptides in small samples, including single cells, due to its high sensitivity and low sample consumption. Beginning with the MALDI MS analysis of single neurons from *Lymnaea stagnalis* and *Aplysia californica* more than 20 years ago,[17,18] the technique has evolved into a powerful tool for measuring peptide content in minute samples such as cells and even individual organelles.[19-24]

Microscopy-guided, single cell MALDI MS greatly increases sample throughput, facilitating the measurement of peptide and metabolite content for thousands of cells.[10,11] Such rapid analysis of cell populations is key to successfully assessing cellular heterogeneity. The technique is amenable to peptide detection from virtually any type of cell. Non-targeted, label-free analysis of large populations coupled with multivariate statistical analysis allows detection of rare cellular phenotypes and enables the study of differential peptide expression in cellular subtypes. Here, using microscopy-guided, single cell MALDI MS we examined the cellular heterogeneity of islets of Langerhans located in the dorsal and ventral regions of the rat pancreas. By classifying each spectrum based on canonical and cell-specific peptide expression, the cellular composition of individual islets was directly measured and the proportion of α-, β-, γ-, and δ-cells was found to differ between the pancreatic regions. Furthermore, the dataset revealed novel prohormone processing products as well as their location-specific abundance.

Methods

Chemicals

Collagenase P (from *Clostridium histolyticum*) used in the enzymatic isolation of islets of Langerhans was purchased from Roche Diagnostics (Indianapolis, IN). Mass spectrometer calibration was performed using a Peptide Calibration Standard Kit II (angiotensin II, angiotensin I, substance P, bombesin, ACTH clip 1–17, ACTH clip 18–39, somatostatin 28, bradykinin fragment 1–7, renin substrate tetradecapeptide porcine with added bovine insulin) obtained from Bruker Daltonics (Billerica, MA). All other chemicals were purchased from Sigma-Aldrich (St. Louis, MO).

Isolation of Islets of Langerhans and Single Cell Preparation

Male, four-month old Sprague-Dawley rats from Harlan Laboratories (Indianapolis, IN) were euthanized by decapitation. The vertebrate animal use protocol was approved by the Institutional Animal Care and Use Committee at the University of Illinois at Urbana–Champaign. Islet isolation was performed as described elsewhere,[52] with minor modifications. Modified Gey's balanced salt solution (mGBSS) was prepared, containing 1.5 mM $CaCl_2$, 4.9 mM KCl, 0.2 mM KH_2PO_4, 11 mM $MgCl_2$, 0.3 mM $MgSO_4$, 138 mM NaCl, 27.7 mM $NaHCO_3$, 0.8 mM NaH_2PO_4, and 25 mM HEPES dissolved in Milli-Q water (Millipore, Billerica, MA), with the pH adjusted to 7.2 using NaOH in Milli-Q water. For islet isolation, the mGBSS was supplemented to a final concentration of 5 mM d-glucose and 1% (w/v) bovine serum albumin (buffer 1). Each pancreas was injected through the bile duct with 2 mL of 1.4 mg/mL collagenase P solution dissolved in buffer 1. Next, the ventral and dorsal regions were surgically dissected following morphological landmarks along the lower duodenum as described elsewhere.[5] The two resulting tissues were placed in separate glass vials, each containing 1 mL of the collagenase P solution. The pancreatic tissues were incubated in a recirculating water bath for 20–30 min at 37 °C to digest exocrine tissue while leaving islets primarily intact. The resulting suspension was washed twice with buffer 1 and centrifuged for 3 min at 300 × g. The resulting pellet was resuspended in 10 mL buffer 1 and islets were manually isolated with a micropipette under visual control using an inverted microscope.

To stabilize the cells and label their nuclei for fluorescent targeting, each islet was transferred to 20 µL of a staining and cell stabilization solution consisting of 40% (v/v) glycerol in buffer 1 with 0.1 mg/mL Hoechst 33342.[53] Islets were incubated for 2 h at 15 °C before trituration of individual islets into single cells onto indium tin oxide (ITO)-coated glass microscopy slides (Delta Technologies, Loveland, CO). Studies of intact islets were performed as above but without trituration. To decrease bias, single cell dispersions from individual islets were randomly placed onto subdivided areas of the slides, such that each slide contained cells from at least two different animals. These efforts ensured that significant differences in abundance were not due to batch effects between slides.

For the LC–MS experiments, about 100 islets were collected and transferred to an Eppendorf tube containing 200 µL acidified methanol (90% MeOH, 9% formic acid (FA), 1% H_2O) for peptide extraction, sonicated for 5 min, and incubated on ice for 1 h. The sample was centrifuged for 20 min (20 000 × g, 4 °C), the supernatant was then dried down in a Savant

SpeedVac vacuum concentrator (Thermo Scientific, Waltham, MA) and reconstituted in 50 µL 5% MeOH, 0.1% FA. Sample clean-up was performed with a C18 spin-column (Thermo Scientific) pre-equilibrated with 5% MeOH, 0.1% FA. After sample loading and analyte retention, the column was washed twice with 1 mL 5% MeOH, 0.1% FA. The retained peptides were eluted twice using 50 µL 70% MeOH, 0.1% FA. The final sample used for LC–MS analysis was prepared by lyophilizing the eluent and reconstituting it in 10 µL 5% MeOH, 0.1% FA.

Optical Imaging for Registration of Fiducial Marks and Single Cell Locations

To create a system of spatial coordinates, a set of fiducial marks were made on conductive ITO slides. Cross marks were made prior to single cell suspension deposition with a diamond pen on 10–15 locations spread across each slide. Glycerol-stabilized cells or individual islets were deposited onto the slides. After overnight incubation in a minimal volume of glycerol-containing solution at ambient conditions, the slides were quickly rinsed with 150 mM ammonium acetate, pH 10, and dried with a gentle stream of nitrogen gas.[54] Cells and fiducial markers were located using a Nanozoomer digital slide-scanner system (Hamamatsu, Middlesex, NJ). Silver paint applied with a marking pen surrounding the dispersed cells was targeted for autofocusing to acquire fluorescent and bright-field images of the suspension area. The images were processed and analyzed to determine the relative coordinates of cells, islets, and fiducial markers.

Single Cell MALDI MS Profiling

To provide a higher throughput and more reproducible coating than via a typical airbrush, an automatic sprayer system was developed using low-cost electric motors and linear actuators to coat up to four slides simultaneously (Figure 6.1). Pumping matrix solution through a fused silica capillary inserted into a stainless steel tube, similar to prototype desorption electrospray ionization sources, generated the nebulizing spray. The slides were affixed to a rotating plate with the nebulizer oscillating radially over them. By rapidly rotating the samples and performing numerous oscillating passes, uncertainty in sprayer position was averaged and an even MALDI matrix coating was achieved. A solution of 50 mg/mL 2,5-dihydroxybenzoic acid (DHB) in an acetone/H_2O mixture, 1/1 (v/v) with 0.05% trifluoroacetic acid (TFA), delivered at 0.5 mL/min, was used in sample preparation for MALDI MS. For single cell profiling, the nebulizer was placed 1 cm above the surface and oscillated over the samples 25 times with a nitrogen gas pressure of 50 psi, resulting in a DHB coating of ~0.2 mg/cm^2. Intact islets required a thicker, dryer coating, with the sprayer operated at a 7 cm distance from the sample with nebulizing gas pressure at 100 psi and 100 passes.

A point-based similarity registration algorithm[55] was utilized to align the relative coordinates of cells, islets, and fiducial points on the ITO slides with the stage positions of a Bruker ultrafleXtreme MALDI-TOF/TOF mass spectrometer (Bruker Daltonics). Scored crosshairs, acting as fiducial markers on the slides, were located in the bright-field image and in the ultrafleXtreme camera system. Python scripts written in-house were utilized to mark fiducial locations and input corresponding stage locations to the registration model (Figure 6.2). Implementing a simple threshold and group algorithm, the same software was then used to find cells in the fluorescent and bright-field images (Figure 6.3), allowing selective recognition of the biological structure by user-defined levels of fluorescence signal intensity, cell size, and cell circularity. Pixel positions in the microscopy image were transformed to fractional distances, which are required to generate custom geometry files with the registration parameters. In addition to saving information on the registration points and cell-finding parameters, the software generated a custom geometry file for direct import to the flexControl software (Bruker

Daltonics) that operated the MALDI MS automatic acquisition. Spectra were acquired with a Bruker ultrafleXtreme MALDI-TOF/TOF mass spectrometer equipped with a frequency tripled Nd-YAG solid state laser. The mass scan window was set to *m/z* 400–6000 and the laser set to the "Ultra" footprint setting at an ~100-µm footprint diameter. The Bruker ultrafleXtreme AutoXecute feature was utilized with the custom geometry file as previously reported.[24] Each spectrum represents the summed signals acquired during 1000 laser shots fired at 1000 Hz. From the 48 dispersed islets analyzed, approximately 32,000 spots were profiled based on fluorescence microscopy identification of the locations of single cells. A 100-µm distance filter removed half of the spectra to ensure each profile corresponded to a single cell (Figure 6.4). The dataset was then imported into ClinProTools software (Bruker Daltonics) with a 16-fold data reduction to perform null spectra exclusion, baseline subtraction, and total ion current normalization. Initial examination of the principal component analysis (PCA) loading plots suggested the major contributors to sample variance were classical biochemical markers for each cell type (glucagon, insulin, somatostatin, and PP). The mass accuracy for peptides observed with MALDI MS are listed in Table 6.1. To simplify comparison with previous histological reports, the intensities of somatostatin, glucagon, PP, and insulin were exported from ClinProTools for further analysis in MATLAB (Mathworks, Natick, MA). After parsing the xml files, empty spectra were excluded by removing samples with peptide intensities of less than 3.5 times the median value of each peptide. Cell phenotyping was performed by *k*-means clustering using a cosine distance, and then further validated with the same threshold based on the median intensity. To visualize the classes on a single plot, the dimension of the data was reduced with PCA as shown in Figure 6.6. The cell counts from each cluster were then matched to their corresponding cellular populations, shown in Table 6.2.

Next, the inter-anatomical differences between pancreatic islet and cellular subtypes were evaluated. Mass spectra were classified based on their peptide content and the dorsal and ventral islet-derived cells were considered as two separate classes in the ClinProTools analyses. The 50 most intense signals between *m/z* 1000–6000 were evaluated for statistically significant differences for each of the four cell types. The dataset was tested for univariate normality with the Anderson–Darling test, which showed the data not to be normally distributed. Hence, statistical tests of differences were performed with the Wilcoxon test applying the Benjamini–Hochberg procedure for false discovery rate correction, with $p < 0.05$ considered to be statistically significant.

Improved Precision and Throughput of Microscopy-guided Single Cell MALDI MS

To facilitate the accurate targeting of small (<10 µm) features located at distances of more than 1 cm, as well as to enable high throughput MALDI MS profiling, multiple enhancements have been made over our previous efforts. After samples were deposited on ITO glass slides, cell finding and registration were performed via a custom Python graphical user interface, automating many manual steps from the original protocol (see "Details of the Python Script for Cell Finding and Generating Custom XEO Geometry Files" section below, Figure 6.3, and Chapter 5). The geometric transformation for generating mass spectral spatial coordinates developed in our laboratory was replaced with a probabilistic transformation using a point-based similarity algorithm. Point-based registration is more robust in preventing fiducial localization errors and decreases the target localization error by fourfold. When Gaussian noise, $N(\mu = 0$ µm, $\sigma = 100$ µm) was added to the fiducial coordinates, simulating fiducial localization uncertainty, the target localization error decreased from 164 ± 72 µm to 38 ± 15 µm (Figure 6.2), with the similarity transformation compared with the geometric transformation. Fluorescence images of cell suspensions deposited on ITO glass slides were acquired with a histological slide scanner, operating in batch mode with automatic image stitching. Finally, a custom MALDI matrix application system was designed to generate more consistent matrix coatings compared to manual artistic airbrush application (Figure 6.1); four slides can be coated simultaneously in less than 10 min.

MALDI MSI of regions surrounding individual cells suggests that analyte spreading is restricted to the first 50 µm, at which point the signal intensity drops to 10% of its peak value (Figure 6.4). The analyte spreading metric required a minimum 100 µm cell-to-cell distance to obtain single cell MALDI MS profiling data. Overall, we increased the robustness of the small cell localization process, and decreased the sample preparation time prior to MALDI MS acquisition from >2 h per sample to 45 min per sample.

MALDI MSI

MSI was performed on sections of rat pancreas. The tissue was fast frozen after dissection and sectioned in 5-µm thick slices without chemical fixation at −25 ℃ using a Leica CM 3050 S cryostat (Leica Microsystems, Bannockburn, IL). Pancreas sections were deposited on ITO-coated glass slides at room temperature (23–25 ℃). Specimens for MSI were spray coated with 5 mg/mL 2-(4-hydroxyphenylazo) benzoic acid in a methanol/H_2O mixture (20/80 v/v) containing 0.1% FA and 0.01% TFA with an artist's airbrush. MSI was performed with an utrafleXtreme MALDI-TOF/TOF mass spectrometer operating in reflector mode at positive polarity. The laser beam was set to "ultra", corresponding to an ~100 µm footprint. The mass spectra were acquired in 25- or 50-µm-spaced arrays, with some expected oversampling. The MS calibration standards were deposited onto slide locations nearby the tissue sections. The peptide ions, and in some cases known endogenous ions, such as several of the most common lipids, as well as MALDI matrix ions, were used in the post-processing step for data recalibration. MSI data acquisition and processing was performed with flexControl, flexImaging, and ClinProTools software (all Bruker Daltonics).

NanoLC–Fourier Transform (FT)-Ion Cyclotron Resonance (ICR) MS

Peptide identification was performed via an FT-ICR mass spectrometer (LTQ-FT Ultra, Thermo Scientific) coupled with a nanoLC system (Eksigent 1D Plus, Eksigent, Dublin, CA). Peptide extract (2 µL) from islets was mixed with 8 µL of loading solvent (5% acetonitrile (ACN), 0.2% FA), injected onto a peptide trap column (150 µm inner diameter (i.d.) × 2 cm length, 5 µm Magic AQ particles, 100 Å pore size, New Objective, Inc., Woburn, MA) and desalted with the loading solvent. The column was flushed with loading solvent. The trap column was then placed in line with the analytical column (PicoFrit column, 75 µm i.d. × 15 cm length, 5 µm Magic AQ particles, 100 Å pore size, New Objective). Mixtures of ACN/water with 0.2% FA were used as chromatographic solvents A (5/95 v/v) and B (95/5 v/v). The analytes were separated with a flow rate of 300 nL/min over a gradient with a solvent A and B mixture as follows: 0–10 min, 0–20% solvent B; 10–65 min, 20–55% solvent B. For MS acquisition, the mass scan window was set to *m/z* 300–2000, data-dependent precursor selection was restricted to the top five most intense ions, dynamic exclusion was enabled with a repeat count of 2, and an exclusion duration of 180s.

Peptide Sequencing

Native Thermo LC–MS data in raw file format were processed with PEAKS 7 (Bioinformatics Solutions Inc., ON, Canada) for peptide sequencing. The data were searched against a UniprotKB/SwissProt rat database of canonical sequences (June 2015 release, *Rattus norvegicus*, 7923 entries). The parent mass error tolerance was set to 50 ppm, the fragment mass error tolerance was set to 0.01 Da. Enzymatic digestion was set to “none”. Acetylation (N-term and

K), amidation, oxidation (M), Pyro-glu (E and Q), half of a disulfide bridge, and phosphorylation were allowed as variable modifications. The filtering conditions used (peptide $-10 \log P \geq 15$, protein $-10 \log P \geq 20$, proteins unique peptide ≥ 0) resulted in a false discovery rate of 1.5% for peptide spectrum matches.

Bioinformatic Analysis

Probabilities for processing of dibasic sites in the pancreatic prohormone (PAHO_RAT) were calculated using the NeuroPred tool (available online at http://neuroproteomics.scs.illinois.edu), with mammalian as the selected model option.

Ab initio calculations of PP, PP(1–24), and PP(27–36) folding were performed using the PEP-FOLD tool (available online at http://mobyle.rpbs.univ-paris-diderot.fr), with the sequence for full-length PP obtained from Uniprot KB (PAHO_RAT).

Results and Discussion

Intra- and Inter-islet Cellular Heterogeneity

High-throughput single cell MALDI MS profiling was used for phenotyping cells from islets of Langerhans to uncover the chemical cellular heterogeneity of islets in anatomically and developmentally distinct parts of the pancreas. Isolated from 48 individual islets (6 dorsal and 6 ventral islets each from a total of 4 animals), cells were deposited on ITO-coated glass slides and examined with single cell MALDI MS. Single cell dispersions of individual islets were deposited on separated areas of the ITO slides to keep track of the source islet. Acquired mass spectra were classified and counted. Each successful measurement revealed the presence of chemically distinct hormone profiles in individual cells (Figures 6.5 and 6.6). A number of peptides characteristic for islet cell types, including glucagon (α-cells), insulin (β-cells), PP (γ-cells), and somatostatin-14 (δ-cells) were observed (Figure 6.5). The cellular composition of individual islets from the same pancreatic lobe were similar. In contrast, significant differences were found between cell populations of the islets of the dorsal and ventral pancreas. Our findings show that α-cells are more abundant in dorsal pancreas islets and γ-cells are more prevalent in ventral pancreas islets (Figure 6.6, Table 6.2, and Figure 6.7). A χ^2-test for independence on the contingency table showed significant differences between islet cell populations in the two regions (n_{Dtot} = 1768, n_{Vtot} = 1738, $p < 0.00001$, n = 24 dorsal islets, n = 24 ventral islets). Importantly, these findings are in agreement with previously published histological reports focused only on major biomarkers due to the use of affinity probes.[4-7] Here, our non-targeted and multiplex analytical technology generated a dataset to test for differential processing of islet prohormones between anatomical regions of the rat pancreas.

Discovery of Endogenous Pancreatic Prohormone-Originated Peptides and their Different Abundances in γ-Cells of Dorsal and Ventral Islets

A statistically significant increase in pancreatic prohormone-related peptide signal intensities was observed in γ-cells from ventral islets when compared to dorsal islets collected from four animals (Figures 6.8 and 6.9). These include PP(1–24) +34%, PP(27–36) +32%, PP(1–16) +7%, and PP(18–36) +44%. In addition, correlation plots of signal intensities for PP against signal intensities for the other peptide products within single γ-cells indicate heterogeneity of the chemical content in cells from the ventral lobe, whereas cells from the dorsal lobe appear more clustered together (Figure 6.10). Parent ion *m/z* values of peptides detected with single cell MALDI MS were matched to *m/z* values of pancreatic prohormone peptides identified with LC–MS sequencing (Tables 6.1 and 6.3). We hypothesize that the peptides are formed by endogenous, enzymatic cleavage of the pancreatic prohormone at the dibasic or monobasic sites of full-length PP (Figures 6.8 and 6.9). This hypothesis is supported by MALDI mass spectrometry imaging (MSI) of fast-frozen pancreas sections from which we detected molecular signals corresponding to the PP(27–36) peptide, concomitant with full-length PP (Figure 6.11), adding confidence to the notion that PP(27-36) is synthesized endogenously. Interestingly, the relative signal intensity of the pancreatic prohormone C-terminal peptide (aa69–98) was 25% higher in cells of the ventral pancreas islets compared to those from the dorsal pancreas. No significant differences in the relative intensities of full-length PP were found in the same cellular populations.

Although the internal dibasic site of PP has been recognized as a possible processing site,[47] the resulting products have only been reported in CA-77 cells transfected with full-length pancreatic prohormone,[26] and have not been described in previous studies using MS.[12-16] Bioinformatic analysis of the pancreatic prohormone using the NeuroPred tool[48] predicts that the dibasic RR site in the PP sequence is a potential processing site with a probability of 0.53 ± 0.10 and a 95% confidence interval (CI). The probability for processing at the C-terminal site resulting in full-length PP is 0.91 ± 0.04 (95% CI). This modeling suggests that the prohormone convertase processing of full-length PP at its dibasic site may form PP(27–36).

Molecular modeling using the PEP-FOLD tool[49] indicates that PP(1–24) and PP(27–36) retain the α-helix secondary structure.[40,41,49] Hence, the dibasic cleavage does not disrupt the shape of the C-terminus of PP. The region has an exact match to the C-terminal of the TRPRY-NH_2 motif, suggesting that PP(27–36) may possess some affinity to NPY receptors. Other fragments of NPY, which exhibit the C-terminal motif TRPRY-NH_2, display some bioactivity and binding affinity to NPY receptors, albeit more weakly than the full-length peptide.[38-45] Previous studies of N-terminal truncation of PP demonstrated the binding affinity of PP(27–36) to be 3 orders of magnitude weaker than PP. In functional assays, the maximum inhibition of cAMP accumulation of PP(27–36) was 50% of the full-length PP with an EC50 of 3.5 μM for PP(27–36) compared to 0.09 nM of PP.[36,37] This suggests that PP(27–36) acts as a partial agonist to the Y4 receptor. The studies discussed above utilized simplified models that cannot be used to uncover all of the spatiotemporal activities in which PP(27–36) may participate *in vivo*.

We observed that the C-terminal peptide of pancreatic prohormone, as well as PP(1–24), PP(27–36), PP(1–18), and PP(20–36), displayed higher signal intensities in ventral islet-derived γ-cells. This finding, in conjunction with equal signal intensities of full-length PP, could be explained by heterogeneous expression or regulation of prohormone convertases for the shortened PP products, whereas the convertases yielding full-length PP remained the same in each set of islets. MALDI MSI and LC–MS also detected products of PP at its monobasic site, resulting in the formation of PP(1–16) and PP(18–36). Processing of the pancreatic prohormone at a single arginyl residue has previously been reported to occur,[50] however, the exact processing pathway remains unclear. Our findings indicate these peptides are generated endogenously from the pancreatic prohormone and display a specific, heterogeneous distribution. Though the peptides display weaker binding affinity and less efficacy to G protein-coupled receptors compared to full-length PP, these molecules may operate within the feedback mechanism of Y4 receptors as partial agonists. It has been reported that the pancreatic prohormone is less amenable to processing with prohormone convertases than other NPY-family peptides.[26] Although less common, prohormone processing may be tissue-specific, such that the final peptide products vary between anatomical regions, in agreement with our findings.[26-35]

Other Neuropeptides in Islets of Langerhans

WE-14 peptide (*m/z* 1677.8) from the chromogranin A prohormone was detected with LC–tandem MS analysis of extracts of homogenized islets of Langerhans (Table 6.3); however, due to homogenization, its cellular origin was uncertain. Further leveraging detection of rare phenotypes, single cell MALDI MS analysis of cellular populations obtained by islet dissociation revealed that WE-14 is selectively localized to δ-cells. WE-14 was accounted for among the top 50 most-intense analyte signals in δ-cells, but in no other cell types. Furthermore, WE-14 had a similar abundance between dorsal- and ventral-derived islet cells (2.50 ± 0.64, $n_{D\delta} = 16$; 3.2 ± 2.7, $n_{V\delta} = 15$; $p = 0.94$, n.s.). Rat islets of Langerhans have a complex spherical microarchitecture consisting of a core occupied by β-cells, with the other secretory cell types lining the periphery of the islet.[6] A previous peptidomic study using MS revealed the presence of WE-14 in analyte extracts of homogenized islets of Langerhans,[15] and an immunohistochemical study localized WE-14 to the outer edge of rat islets of Langerhans where δ-cells are located.[25] The results presented here are in good agreement with these previous findings but provide additional important details, including the localization of WE-14 in δ-cells within the islets of Langerhans. By isolating the rare δ-cells from the thousands of neighboring endocrine cells, their peptidome could be selectively investigated. Other neuropeptides were also detected in analyte extracts from islets of Langerhans using LC–MS, including aa513–532 (N- and C-terminal dibasic site processing) from the secretogranin-1 prohormone, and secretoneurin from the secretogranin-2 prohormone (Table 6.3). These peptides were not observed in our single cell data obtained with MALDI MS, which may reflect differences in analyte extraction due to sample preparation, ionization efficiency, and/or their lower abundance.

Conclusions

Determining the mechanisms involved in modulating the synthesis and processing of endogenous peptides, and regulation mediated through ligand–receptor interactions, is integral to understanding normal physiology as well as diseased states. From pancreatic islets, we have detected cell type-specific expression of known peptide hormones displaying anatomical heterogeneity. Peptides resulting from the processing of PP were detected endogenously and found to be enriched in γ-cells from ventral islets. Furthermore, we located the neuropeptide WE-14 with high abundance in δ-cells. The physiological roles and mechanisms of action of both PP and WE-14 are less understood compared to many other peptide hormones such as insulin. The detection of such molecules and determination of their specific localization are initial steps towards determining signaling mechanisms and physiological effects. A more complete understanding of pancreatic cell-to-cell signaling hormones may help reveal the mechanisms of metabolic diseases, in particular, the development of type 2 diabetes mellitus — a disease affecting a growing number of individuals worldwide.

MALDI MS of individual islet cells allowed detection and colocalization of previously unreported peptides with well-studied pancreatic hormones. The non-targeted, single cell MALDI MS profiling facilitated measurements of the peptide content of rare cells (γ-cells in dorsal islets) and tests for significant differences in abundance for the same cell type derived from a more abundant source of cells (γ-cells in ventral islets). The method is label-free and capable of detecting hundreds of compounds in thousands of cells within an hour, enabling experiments that would not be feasible with traditional flow cytometry. Moreover, the sampling procedure for comparative analysis of endocrine cells is suitable for many other types of peptidergic system studies. Examples include investigating temporal changes of tissue microarchitecture and pathological or drug-induced changes in peptide production.

Figures and Tables

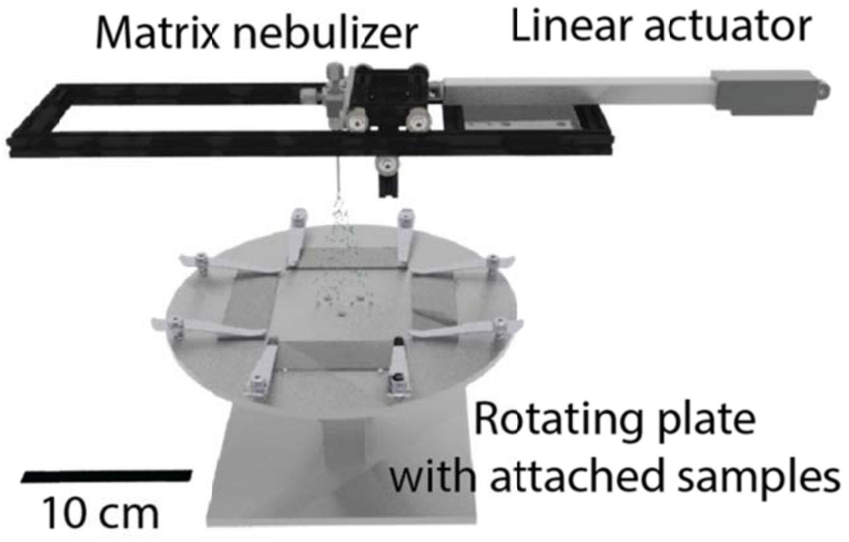


Figure 6.1. Three-dimensional depiction of the automated MALDI matrix coating system. Slides are affixed to the surface of a rotating disk with slide clamps. As the disk rotates (~ 7 Hz) the linear actuator oscillates the matrix nebulizer over the sample surface. As the center-to-nebulizer distance increases, the linear speed of the actuator decreases to compensate for the change in surface area covered by the spray.

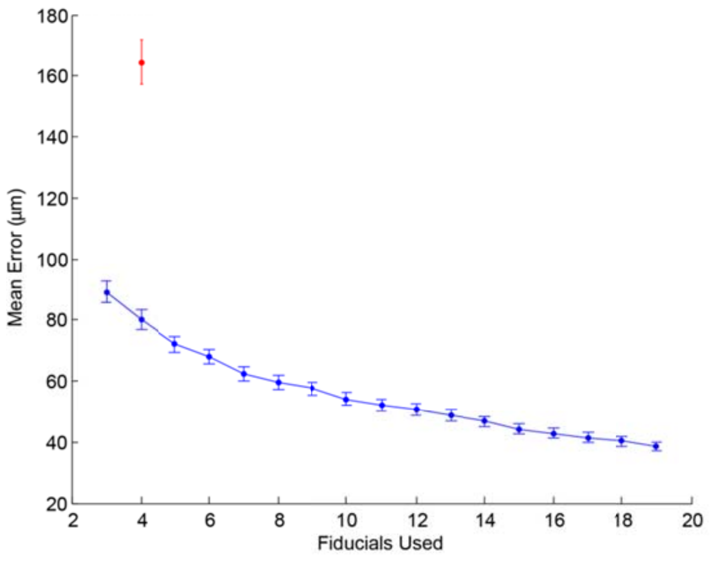


Figure 6.2. Comparison of the accuracy and robustness of geometric (red) and point-based (blue) registration transformation. To simulate uncertainty in fiducial location, Gaussian noise, *N*(µ = 0 µm, σ = 100 µm), was added to the x- and y-coordinates of the fiducial marks. Each point represents the mean ± SEM for 100 simulations of noise. As more fiducials are included in the model, the target localization error decreases below the added noise. To ensure accuracy, 10–15 fiducials were utilized for every slide.

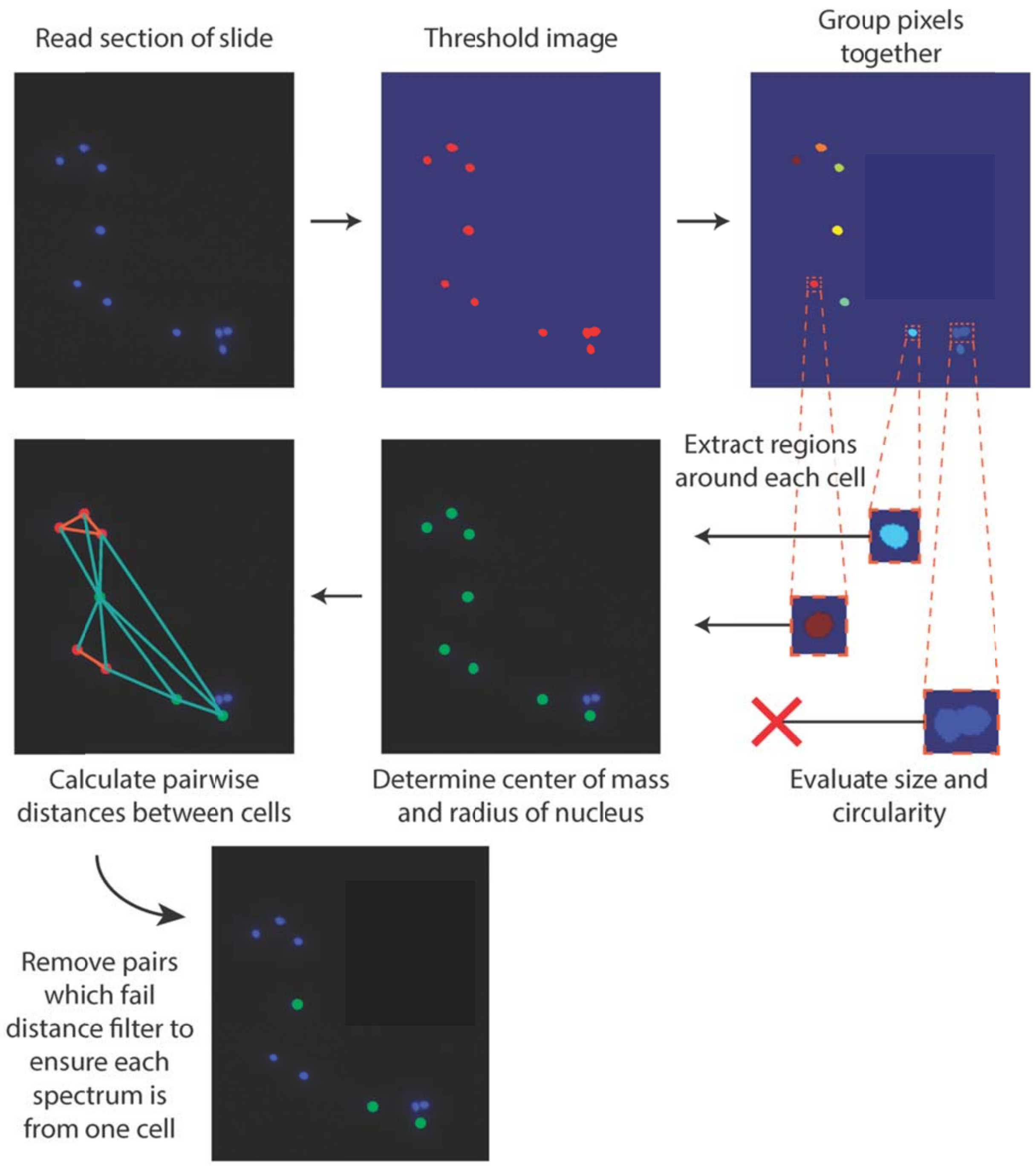

Read section of slide
Threshold image
Group pixels together
Extract regions around each cell
Evaluate size and circularity
Determine center of mass and radius of nucleus
Calculate pairwise distances between cells
Remove pairs which fail distance filter to ensure each spectrum is from one cell

Figure 6.3. Overview schematic of the cell-finding algorithm. The high-contrast fluorescence images allow a simple threshold and group method to locate the center of each cell. Neighboring pixels are grouped together to generate a putative cell. Each possible cell is evaluated to determine if the intensity and circularity are within range, which eliminates background artifacts. Single cell locations are further evaluated by calculating the pairwise distance between each cell, eliminating those within 100 µm of each other.

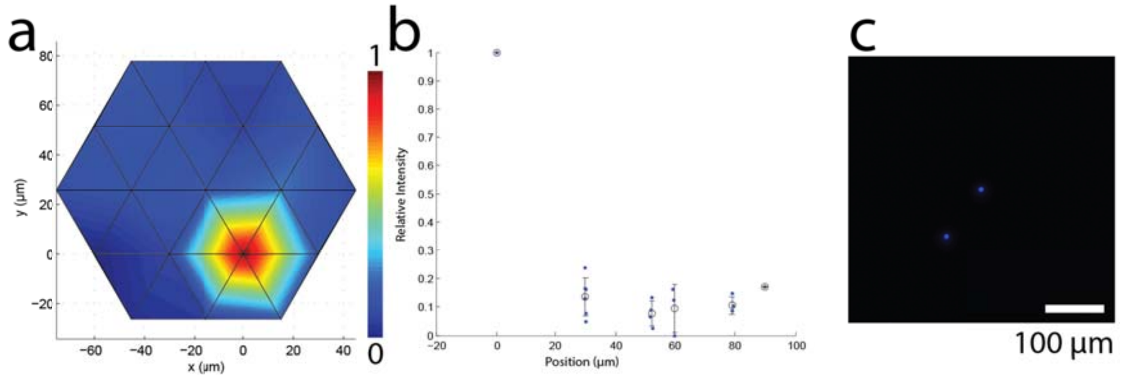


Figure 6.4. Insulin C-peptide redistribution from single cells after MALDI matrix application measured with MALDI MSI. (**a**) Representative figure of a hexagonally packed raster grid centered around single cells (point-to-point distance, 30 μm), analyzed for 18 β-cells on multiple ITO glass slides coated with matrix. (**b**) Defining the center point as the most intense position, the relative intensity as a function of distance from center showed a rapid drop in intensity after even a 30 μm step. Analysis of multiple spots indicated peptide spreading is limited to 50 μm from the center of the cell, where the intensity had dropped by ~90%. (**c**) All data utilized 100 μm as a cell-to-cell distance cut-off to exclude cells which were too closely located, and interfered with neighboring cells.

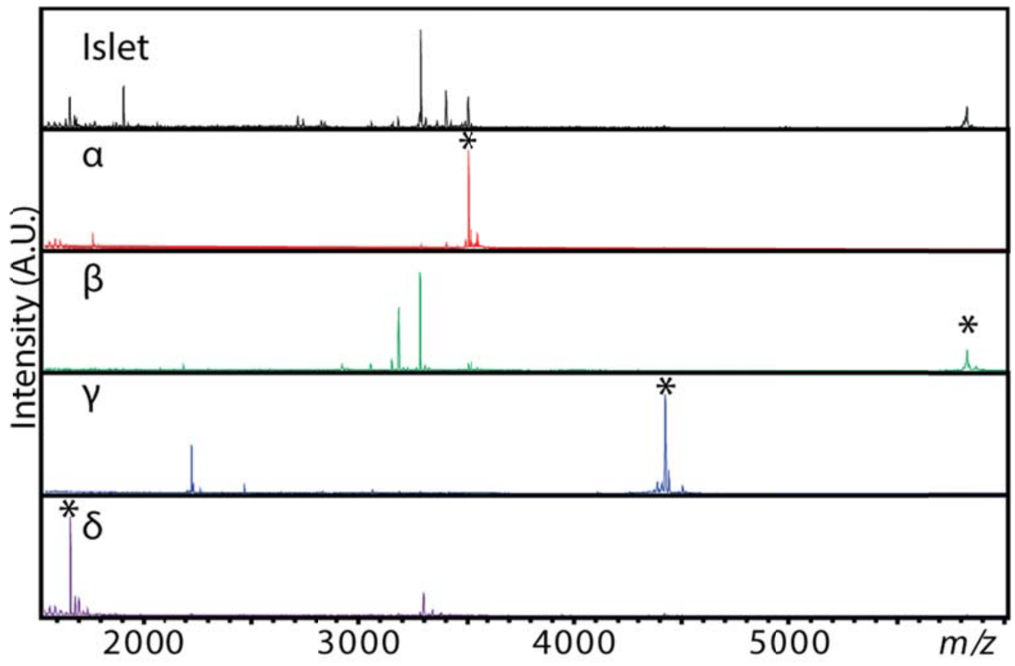


Figure 6.5. Representative mass spectra acquired from individual α-, β-, γ-, and δ-cells, as well as an intact islet of Langerhans collected from the dorsal pancreas. Peptide cell-type qualifiers/biomarkers: glucagon *m/z* 3481.5 (α-cells), insulin *m/z* 5799.9 (β-cells), PP *m/z* 4397.2 (γ-cells), and somatostatin-14 *m/z* 1637.7 (δ-cells). These signals are highlighted with asterisks. Many other characteristic signals were observed in the cells as described herein.

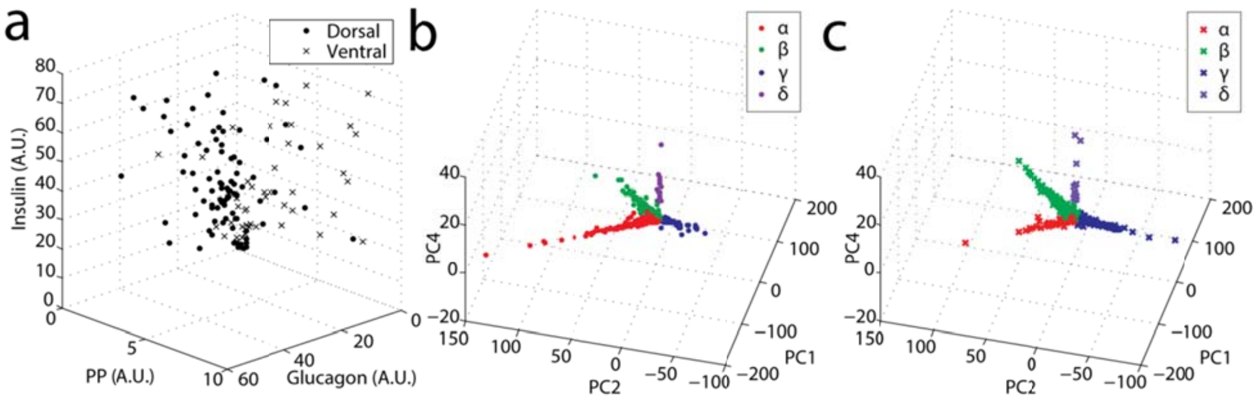


Figure 6.6. MALDI MS characterization of intact islets of Langerhans and corresponding single cell populations obtained from individual islets. (a) Intact islets have different peptide profiles in the dorsal (filled circles) and ventral (crosses) pancreas. Distributions of total ion current (TIC)-normalized intensities of insulin, glucagon, and PP signals acquired from intact islets. (b and c) Results of *k*-means clustering of single cell data shown in color: α-cells (red), β-cells (green), γ-cells (blue), δ-cells (purple). *k*-means clustering was used on the full data set to classify and count the different cell types observed in single cell preparations from islets of Langerhans based on TIC-normalized peptide signal intensities. The classification highlights differences in cell composition between islets from dorsal and ventral pancreata, reflected in the relative abundance of α- (red) and γ- (blue) cells. The data were dimension-reduced with principal component analysis for visualization purposes only, here shown with principal components (PC) 1, 2, and 4. (b) Dorsal pancreas islet cells (n = 1768). (c) Ventral pancreas islet cells (n = 1738).

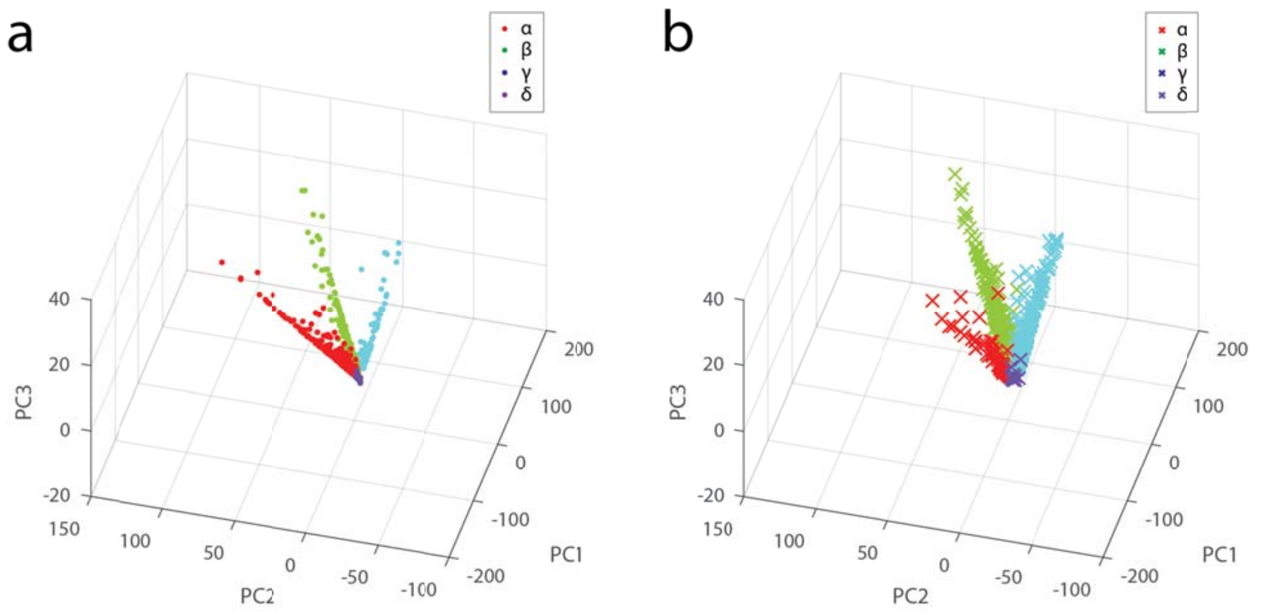


Figure 6.7. PCA allows visualization of the four canonical cell types observed with MALDI MS using principal components 1–3. Figure 6.6 contains PC 1,2, and 4 and is displayed at a different angle.

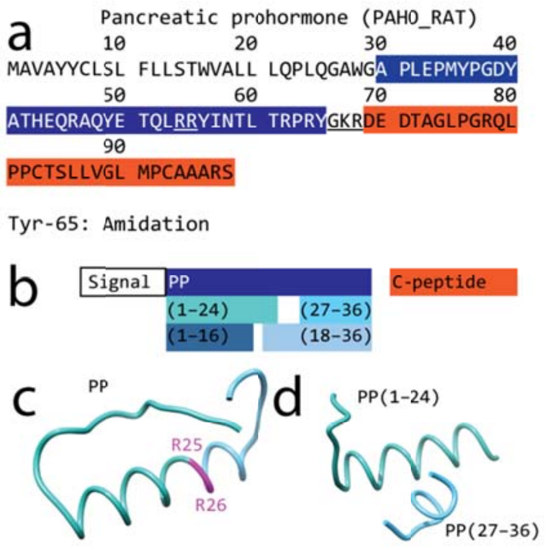


Figure 6.8. Sequence and predicted structure of peptides originating from the rat pancreatic prohormone. (a) Sequence of rat pancreatic prohormone (UNIPROT ID: P06303, PAHO_RAT); the dibasic sites are underlined. (b) Observed peptide products resulting from processing of the pancreatic prohormone (filled color boxes). (c) *Ab initio*-calculated structure of PP; the dibasic site is outlined in magenta. (d) Molecular modeling shows that the N- and C-terminal products (sea green and sky blue, respectively) resulting from cleavage at the dibasic site retain the α-helix folding of the parent peptide.

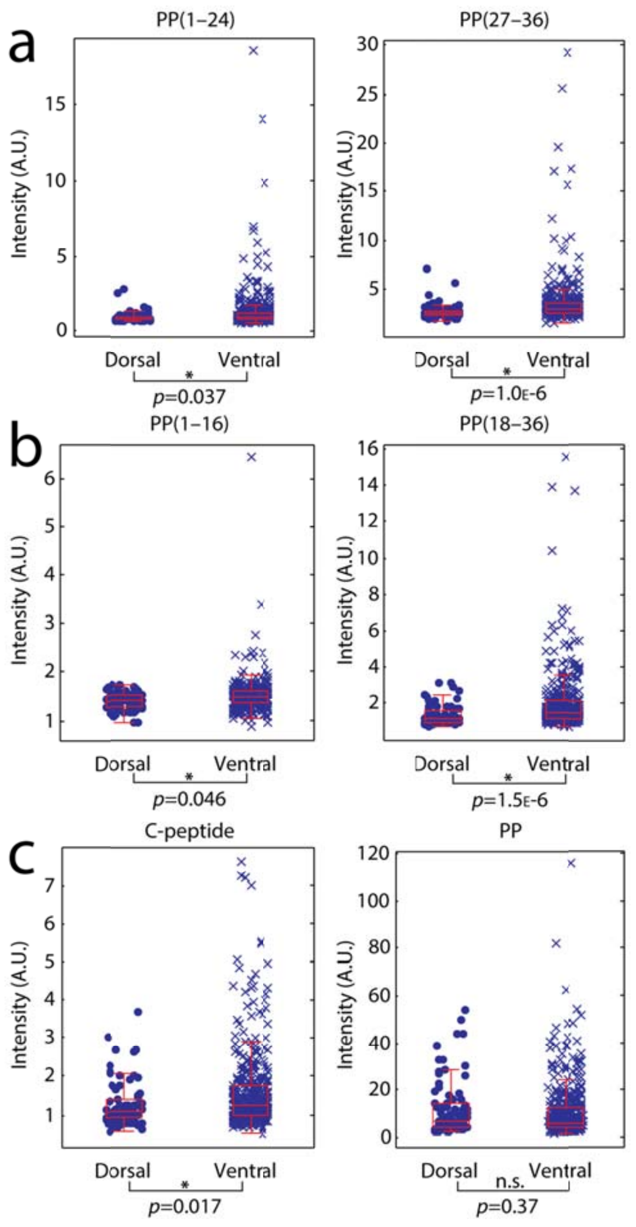


Figure 6.9. Comparison of pancreatic prohormone peptide signals acquired from γ-cells of the dorsal and ventral islets ($n_{D\gamma}$ = 79 cells, $n_{V\gamma}$ = 418 cells). Box and whisker plots of TIC-normalized signal intensities are shown for (a) peptide products from internal dibasic cleavage of PP, left: PP(1–24), *m/z* 2808.2; right: PP(27–36), *m/z* 1295.7. (b) Peptide products from internal monobasic cleavage of PP, left: PP(1–16), *m/z* 1818.8; right: PP(18–36), *m/z* 2441.3. (c) Canonical peptide products from the pancreatic prohormone, left: C-terminal peptide, *m/z* 3037.4; right: PP, *m/z* 4397.2.

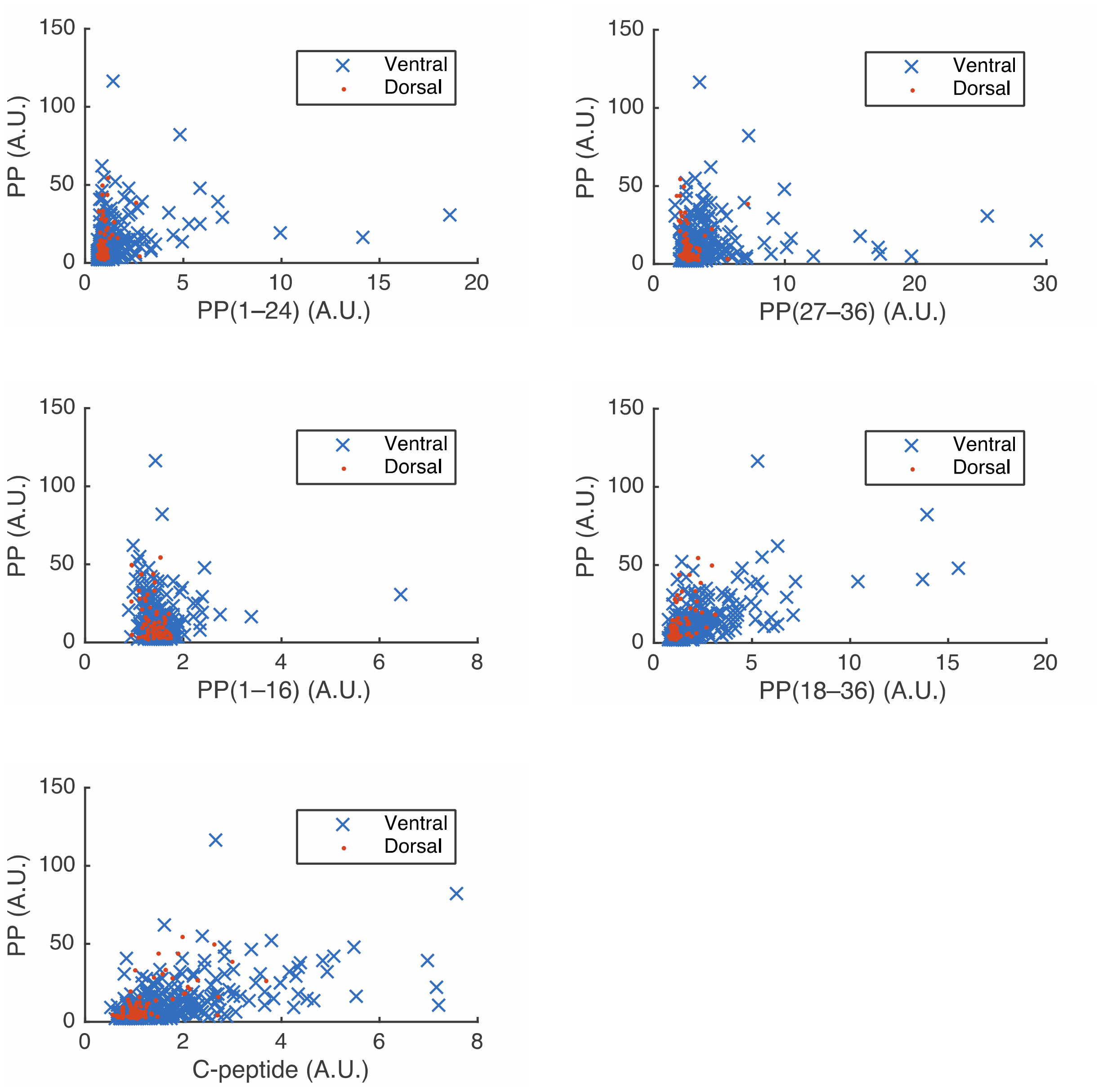


Figure 6.10. MALDI MS signal intensity of PP plotted against the signal intensities of PP(1–24), PP(27–36), PP(1–16), PP(18–36), and the C-terminal peptide from the pancreatic prohormone within single γ-cells ($n_{D\gamma}$ = 79 cells, $n_{V\gamma}$ = 418 cells) from ventral and dorsal islets are compared.

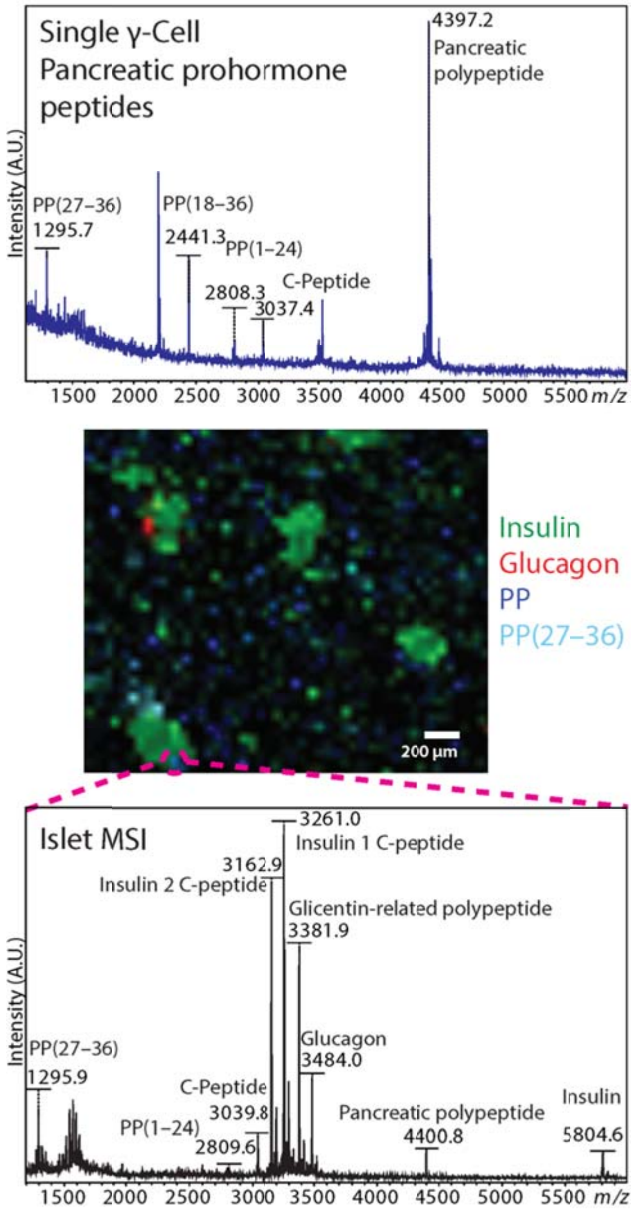


Figure 6.11. Comparison of mass spectra acquired from a single γ-cell (top) and a pixel on a pancreatic islet cross-section (bottom). Color-coded molecular ion distributions for four peptides are shown on the MALDI MS image (center).

Table 6.1. Canonical and processed PP-peptides from islets of Langerhans detected with MALDI MS.

Peptide	PTM	Elemental composition*	Monoisotopic mass* (Da)	Observed mass (Da)	Mass error (ppm)
Insulin 1 (a+b chain)	3 internal disulfides	$C_{259}H_{388}N_{65}O_{75}S_6$	5800.687	5799.906	134.6
Insulin 2 (a+b chain)	3 internal disulfides	$C_{256}H_{383}N_{64}O_{76}S_7$	5793.612	5792.84	133.3
Glucagon		$C_{153}H_{226}N_{43}O_{49}S$	3481.624	3481.472	43.66
Somatostatin-14	1 internal disulfide	$C_{76}H_{105}N_{18}O_{19}S_2$	1637.724	1637.704	12.21
PP	C-terminal amidation	$C_{195}H_{299}N_{58}O_{57}S$	4397.2	4397.172	6.368
PP C-peptide	1 internal disulfide	$C_{128}H_{216}N_{37}O_{42}S_3$	3037.491	3037.415	25.02
PP(27-36)	C-terminal amidation	$C_{59}H_{95}N_{18}O_{15}$	1295.722	1295.71	9.261
PP(1-24)		$C_{124}H_{183}N_{32}O_{41}S$	2808.294	2808.231	22.43
PP(1-16)		$C_{81}H_{116}N_{19}O_{27}S$	1818.801	1818.76	22.54
PP(18-36)	C-terminal amidation	$C_{108}H_{174}N_{35}O_{30}$	2441.317	2441.257	24.58

*Masses are given as $[M+H]^+$

Table 6.2. Population fractions of cell types for dorsal (*n* = 24) and ventral (*n* = 24) rat pancreatic islets of Langerhans measured with MALDI MS analysis of single cell populations.

Cell type	Dorsal	Ventral
α	0.43	0.081
β	0.51	0.67
γ	0.045	0.24
δ	0.0090	0.0086
n_{cells}	1768	1738

Table 6.3. Peptides detected with LC-MS from rat islets of Langerhans. All sequences were uniquely found within the database.

Protein Accession	Entry name	Peptide	Score (%)	-10lgP	Mass	ppm	*m/z*	RT
P01323	INS2_RAT	R.EVEDPQVAQLELGGGPGAGDLQTLALEVARQ.K	99.8	53.88	3159.6047	-0.5	1054.2084	33.63
P01323	INS2_RAT	E.LGGGPGAGDLQTLALEVARQ.K	99.8	49.88	1922.0221	-1.1	962.0173	32.45
P01323	INS2_RAT	R.EVEDPQVAQLELGGGPGAGDLQTLAL.E	99.7	48.23	2576.2969	-2.6	1289.1523	34.02
P01323	INS2_RAT	Q.LELGGGPGAGDLQTLALEVARQ.K	99.6	45.55	2164.1487	-2.2	1083.0792	33.48
P01323	INS2_RAT	R.E(-18.01)VEDPQVAQLELGGGPGAGDLQTLALEVARQ.K	99.6	40.35	3141.5942	-1	1048.2043	35.87
P01323	INS2_RAT	R.EVEDPQVAQLELGGGPGAGDLQTL.A	99.6	40.05	2392.1758	-2.3	1197.0924	32.38
P01323	INS2_RAT	R.EVEDPQVAQLELGGGPGAGDLQTLALEVA.R	99.5	38.85	2875.4451	-0.6	959.4883	35.53
P01323	INS2_RAT	R.EVEDPQVAQLELGGGPGAGDLQTLA.L	99.1	37.39	2463.2129	-1.6	1232.6117	32.17
P01323	INS2_RAT	G.GPGAGDLQTLALEVARQ.K	98.6	36.25	1694.8951	-2.2	848.4529	31.86
P01323	INS2_RAT	L.ELGGGPGAGDLQTLALEVARQ.K	98.5	34.62	2051.0647	-0.8	1026.5388	32.79
P01323	INS2_RAT	R.EVEDPQVAQLELGGGPGAGDL.Q	93.9	29.03	2049.9854	-1.6	1025.9983	31.59
P01323	INS2_RAT	F.VK(+42.01)QHLC(-1.01)GSHLVEALYLVC(-1.01) GERGFFYTPMSRREVEDPQVAQLELGGGPGAGDLQTL.A	81.3	24.94	6009.9604	-6.7	1202.9913	32.28
P01323	INS2_RAT	D.PQVAQLELGGGPGAGDLQTLALEVARQ.K	57.1	22.24	2687.4241	-1.3	896.8141	33.61
P01322	INS1_RAT	R.EVEDPQVPQLELGGGPEAGDLQTLALEVARQ.K	99.9	57.75	3257.6414	-2.1	1086.8855	34.43
P01322	INS1_RAT	E.LGGGPEAGDLQTLALEVARQ.K	99.8	51.34	1994.0432	-1.4	998.0275	32.6
P01322	INS1_RAT	R.EVEDPQVPQLELGGGPEAGDLQTL.A	99.6	41.34	2490.2124	-2.3	1246.1106	32.86
P01322	INS1_RAT	R.E(-18.01)VEDPQVPQLELGGGPEAGDLQTLALEVARQ.K	99.6	41.21	3239.6309	-0.4	1080.8838	36.02
P01322	INS1_RAT	Q.LELGGGPEAGDLQTLALEVARQ.K	99.6	40.6	2236.1699	-0.4	1119.0918	33.55
P01322	INS1_RAT	R.EVEDPQVPQLELGGGPEAGDLQTLAL.E	99.5	39.44	2674.3337	-1.6	892.4504	34.47
P01322	INS1_RAT	R.EVEDPQVPQLELGGGPEAGDLQTLA.L	99.5	38.65	2561.2495	-0.3	1281.6316	32.57
P01322	INS1_RAT	R.EVEDPQVPQLELGGGPEAGDLQ.T	98.5	33.7	2276.0808	-1.4	1139.0461	31.19
P01322	INS1_RAT	D.PQVPQLELGGGPEAGDLQTLA.L	97.6	32.25	2089.0691	-2.2	1045.5396	32.32
P01322	INS1_RAT	R.EVEDPQVPQLELGGGPEAGDLQTLALEVA.R	97	31.58	2973.4817	-2.3	992.1656	35.72
P06883	GLUC_RAT	G.SWQHAPQDTEENARSFPASQTEPLEDPDQINE(-.98).D	99.6	45.95	3664.6301	-1.5	1222.5488	27.91
P06883	GLUC_RAT	G.SWQHAPQDTEENARSFPASQTEPLEDPDQINED(-.98).K	99.6	45.12	3779.657	-4.4	1260.8875	27.93

Table 6.3. (cont.)

Protein Accession	Entry name	Peptide	Score (%)	-10lgP	Mass	ppm	*m/z*	RT
P06883	GLUC_RAT	Q.HAPQDTEENARSFPASQTEPLEDPDQINE(-.98).D	99.6	44.08	3263.4602	-0.4	1088.8269	26.85
P06883	GLUC_RAT	Q.HAPQDTEENARSFPASQTEPLEDPDQINE(-.98).D	99.6	44.08	3263.4602	-0.4	1088.8269	26.85
P06883	GLUC_RAT	R.HSQGTFTSDYSKYLDSRRAQDFVQWLMNT.K	99.6	42.88	3480.6157	-2	871.1595	33.46
P06883	GLUC_RAT	R.HSQGTFTSDYSKYLDS.R	99.1	36.93	1834.8009	-1.7	918.4061	28.4
P06883	GLUC_RAT	Q.HAPQDTEENARSFPASQTEPLEDPDQINED(-.98).K	98.6	35.79	3378.4871	-1	1127.1685	26.9
P06883	GLUC_RAT	Q.HAPQDTEENARSFPASQTEPLEDPDQIN(-.98).E	97.6	31.99	3134.4177	-2.3	1045.8108	26.88
P06883	GLUC_RAT	R.SFPASQTEPLEDPDQINED(-.98).K	93.7	28.55	2129.939	-2.4	1065.9742	28.68
P06883	GLUC_RAT	R.SFPASQTEPLEDPDQINE(-.98).D	90.5	26.9	2014.9119	-0.3	1008.4629	28.75
P06883	GLUC_RAT	Q.HAPQDTEENARSFPASQTEPLEDPDQINEDK(+42.01) RHS(+79.97)QGTFTSDYSKYLDSRRAQDFVQW(-.98).L	81	24.79	6788.0503	-7.7	1132.3403	27.66
P06883	GLUC_RAT	G.SWQHAPQDTEENA.R	57.9	22.77	1511.6277	-0.6	756.8207	19.98
P06303	PAHO_RAT	G.APLEPMYPGDYATHEQRAQYETQLRRYINTLTRPRY(-.98).G	99.7	46.64	4396.1924	-1.3	733.705	29.92
P06303	PAHO_RAT	G.APLEPMYPGDYATHEQRAQYETQL.R	99.7	46.19	2807.2861	-1.7	936.7678	28.96
P06303	PAHO_RAT	G.APLEPMYPGDYATHEQ.R	99.6	44.67	1817.793	-0.3	909.9034	28.55
P06303	PAHO_RAT	G.APLEPM(+15.99)YPGDYATHEQRAQYETQL.R	99.5	39.83	2823.281	-1	942.1	28.22
P06303	PAHO_RAT	G.APLEPMYPGDYATHE.Q	98.5	33.62	1689.7344	0.8	845.8751	28.82
P06303	PAHO_RAT	G.APLEPMYPGDYATHEQRAQYET.Q	98.1	33.31	2566.1433	-0.7	856.3878	27.99
P06303	PAHO_RAT	R.AQYETQLRRYINTLTRPRY(-.98).G	95.3	30.54	2440.3086	0.4	611.0847	29.34
P06303	PAHO_RAT	R.YINTLTRPRY(-.98).G	81	24.76	1294.7146	-0.9	432.5784	25.61
P06303	PAHO_RAT	G.APLEPMYPGDYATHEQRAQYE.T	34.6	20.42	2465.0957	-1.1	822.705	27.99
P06303	PAHO_RAT	G.APLEPMYPGDYATHEQRAQYETQLRRYINTLTRPRY.G	9.6	16.13	4397.1763	-2.1	733.8685	28.71
P62329	TYB4_RAT	M.SDK(+42.01)PDM(+15.99)AEIEKFD KSKLKKTETQEKNPLPSKETIEQEKQAGES	99.8	53.42	4976.481	-1.1	830.4199	25.53
P62329	TYB4_RAT	M.SDK(+42.01)PDMAEIEKFDKSKLKKT ETQEKNPLPSKETIEQEKQAGES	99.8	51.12	4960.4863	-2	827.7534	25.94

Table 6.3. (cont.)

Protein Accession	Entry name	Peptide	Score (%)	-10lgP	Mass	ppm	*m/z*	RT
P62329	TYB4_RAT	K.LKKTETQEKNPLPSKETIEQEKQAGES	94.4	30.06	3069.583	1.1	768.4039	17.54
P62329	TYB4_RAT	M.SDK(+42.01)PDMAEIEKFDKSKLKKT(+79.97) ETQEKNPLPSKETIEQEKQAGES	90.5	26.98	5040.4526	-2	841.081	26.26
P62329	TYB4_RAT	M.S(+42.01)DKPDMAEIEKFDKSKLKKTETQEKN PLPSKETIEQEKQAG.E	89.8	26.3	4744.4116	-0.1	791.7425	26.07
P62329	TYB4_RAT	F.DKSKLKKTETQEKNPLPSKETIEQEKQAGES	56.7	22	3527.8318	-2.6	706.5718	15.17
P62329	TYB4_RAT	K.KTETQEKNPLPSKETIEQEKQAGES	10.4	18.06	2828.4038	-0.3	708.108	22.89
P62329	TYB4_RAT	M.S(+42.01)DKPDMAEIEKFD.K	9.5	15.8	1565.6919	-2.4	783.8514	29.28
P63312	TYB10_RAT	M.A(+42.01)DKPDMGEIASFDKAKLKKTETQEKN TLPTKETIEQEKRSEIS	99.9	64.29	4933.5229	-1.9	823.2595	26.47
P63312	TYB10_RAT	M.ADK(+42.01)PDMGEIASFDKAKLKKTETQEKN TLPTKETIEQEKRSE.I	95.7	30.99	4733.4067	-2.5	677.2065	26.36
P63312	TYB10_RAT	M.ADK(+42.01)PDM(+15.99)GEIASFDKAKLKKT ETQEKNTLPTKETIEQEKRSEIS	93.8	28.59	4949.5176	-1.6	825.9255	25.89
P63312	TYB10_RAT	M.ADK(+42.01)PDMGEIASFDKAKLKKT(+79.97) ETQEKNTLPTKETIEQEKRSEIS	57.3	22.35	5013.4893	-1.2	836.5878	26.76
P25886	RL29_RAT	K.APAQAPKGAQAPVKAP	99.5	39.09	1500.8412	-0.3	501.2875	14.8
P25886	RL29_RAT	K.VQTKAEAKAPAKAQAKAPAQAPKGAQAPVKAP	97.7	32.61	3121.7722	-0.9	521.3022	10.77
P25886	RL29_RAT	K.AEAKAPAKAQAKAPAQAPKGAQAPVKAP	90.8	27.46	2665.5027	-4.7	534.1053	11.23
O35314	SCG1_RAT	R.LGALFNPYFDPLQWKNSDFE.K	99.6	44.88	2400.1426	-4	1201.0737	36.55
P10362	SCG2_RAT	R.IPAGSLKNEDTPNRQYLDEDMLLKVLEYLNQEQA EQGREHLA(-.98).K	97.6	32.17	4866.4243	-8.6	974.2838	33.3
P10362	SCG2_RAT	R.TNEIVEEQYTPQSLATLESVFQELGKLTGPSNQ.K	55.3	21.08	3649.7998	1.3	1217.6088	38.06
P13668	STMN1_RAT	M.A(+42.01)SSDIQVKELEKRASGQAFEL.I	99.1	37.02	2347.2019	-0.7	783.4073	30.14
P50878	RL4_RAT	K.KLKKPAGKKVVTKKPAEKKPTTEEKKSAA	98.5	33.94	3147.907	-2	525.6574	8.72
P10354	CMGA_RAT	R.WSRMDQLAKELTAE.K	93.9	29.02	1676.8192	-1.8	839.4154	30.06
B2RZ37	REEP5_RAT	K.EVKKATVNLLGDEKKST	90.1	26.53	1859.0364	-1	465.7659	17.65

Table 6.3. (cont.)

Protein Accession	Entry name	Peptide	Score (%)	-10lgP	Mass	ppm	*m/z*	RT
Q4QRB4	TBB3_RAT	F.SVVPSPKVSDTVVEPY.N	89.9	26.38	1701.8824	-1.1	851.9476	28.56
P69897	TBB5_RAT	F.SVVPSPKVSDTVVEPY.N	89.9	26.38	1701.8824	-1.1	851.9476	28.56
Q6P9T8	TBB4B_RAT	F.SVVPSPKVSDTVVEPY.N	89.9	26.38	1701.8824	-1.1	851.9476	28.56
P12969	IAPP_RAT	R.NVAEDPNRESLDFLLL	57.1	22.25	1843.9315	-1.3	922.9718	34.64
P62630	EF1A1_RAT	Y.KIGGIGTVPVGRVETGVLKPGMVVT.F	55.4	21.1	2463.4246	0.8	822.1494	28.98

References

(1) Dueck, H.; Eberwine, J.; Kim, J. *Bioessays* **2015***, 38*, 172-180.

(2) Wills, Q. F.; Boothe, T.; Asadi, A.; Ao, Z.; Warnock, G. L.; Kieffer, T. J.; Johnson, J. D. *Islets* **2016***, 8*, 48-56.

(3) Benninger, R. P.; Hutchens, T.; Head, W. .; McCaughey, M.; Zhang, M.; Le Marchand, S.; Satin, L.; Piston, D. *Biophys. J.* **2014***, 107*, 2723-2733.

(4) Baetens, D.; Malaisse-Lagae, F.; Perrelet, A.; Orci, L. *Science* **1979***, 206*, 1323-1325.

(5) Elayat, A. A.; el-Naggar, M. M.; Tahir, M. *J. Anat.* **1995***, 186*, 629-637.

(6) Suckale, J.; Solimena, M. *Front. Biosci.* **2008***, 13*, 7156-7171.

(7) Merkwitz, C.; Blaschuk, O. W.; Schulz, A.; Lochhead, P.; Meister, J.; Ehrlich, A.; Ricken, A. M. *Prog. Histochem. Cytochem.* **2013***, 48*, 103-140.

(8) Battich, N.; Stoeger, T.; Pelkmans, L. *Cell* **2015***, 163*, 1596-1610.

(9) Li, J.; Klughammer, J.; Farlik, M.; Penz, T.; Spittler, A.; Barbieux, C.; Berishvili, E.; Bock, C.; Kubicek, S. *EMBO Rep.* **2015***, 17*, 178-187.

(10) Rubakhin, S. S.; Romanova, E. V.; Nemes, P.; Sweedler, J. V. *Nat. Meth.* **2011***, 8*, S20-S29.

(11) Zenobi, R. *Science* **2013***, 342,* DOI: 10.1126/science.1243259.

(12) Edwards, J. L.; Kennedy, R. T. *Anal. Chem.* **2005***, 77*, 2201-2209.

(13) Schmudlach, A.; Felton, J.; Cipolla, C.; Sun, L.; Kennedy, R. T.; Dovichi, N. J. *Analyst* **2016***, 141*, 1700-1706.

(14) Waanders, L. F.; Chwalek, K.; Monetti, M.; Kumar, C.; Lammert, E.; Mann, M. *Proc. Natl. Acad. Sci. U.S.A.* **2009***, 106*, 18902-18907.

(15) Boonen, K.; Baggerman, G.; D'Hertog, W.; Husson, S. J.; Overbergh, L.; Mathieu, C.; Schoofs, L. *Gen. Comp. Endocrinol.* **2007***, 152*, 231-241.

(16) Stewart, K. W.; Phillips, A. R. J.; Whiting, L.; Jüllig, M.; Middleditch, M. J.; Cooper, G. J. S. *Rapid Commun. Mass Spectrom.* **2011***, 25*, 3387-3395.

(17) Jimenez, C. R.; van Veelen, P. A.; Li, K. W.; Wildering, W. C.; Geraerts, W. P. M.; Tjaden, U. R.; van der Greef, J. *J. Neurochem.* **1994***, 62*, 404-407.

(18) Garden, R. W.; Moroz, L. L.; Moroz, T. P.; Shippy, S. A.; Sweedler, J. V. *J. Mass Spectrom.* **1996***, 31*, 1126-1130.

(19) Rubakhin, S. S.; Greenough, W. T.; Sweedler, J. V. *Anal. Chem.* **2003***, 75*, 5374-5380.

(20) Neupert, S.; Predel, R. *Biochem. Biophys. Res. Commun.* **2005***, 327*, 640-645.

(21) Altelaar, A. F. M.; Taban, I. M.; McDonnell, L. A.; Verhaert, P. D. E. M.; de Lange, R. P. J.; Adan, R. A. H.; Mooi, W. J.; Heeren, R. M. A.; Piersma, S. R. *Int. J. Mass Spectrom.* **2007***, 260*, 203-211.

(22) Jarecki, J. L.; Andersen, K.; Konop, C. J.; Knickelbine, J. J.; Vestling, M. M.; Stretton, A. O. *ACS Chem. Neurosci.* **2010***, 1*, 505-519.

(23) Chen, R.; Ouyang, C.; Xiao, M.; Li, L. In situ identification and mapping of neuropeptides from the stomatogastric nervous system of Cancer borealis. *Rapid Commun. Mass Spectrom.* **2014***, 28*, 2437-2444.

(24) Ong, T.; Kissick, D. J.; Jansson, E. T.; Comi, T. J.; Romanova, E. V.; Rubakhin, S. S.; Sweedler, J. V. *Anal. Chem.* **2015***, 87*, 7036-7042.

(25) Barkatullah, S. C.; Curry, W. J.; Johnston, C. F.; Hutton, J. C.; Buchanan, K. D. *Histochem. Cell Biol.* **1997***, 107*, 251-257.

(26) Wulff, B. S.; Johansen, T. E.; Dalbøge, H., H.; O'Hare, M. M. T.; Schwartz, T. W. *J. Biol. Chem.* **1993***, 268*, 13327-13335.

(27) Brakch, N.; Galanopoulou, A. S.; Patel, Y. C.; Boileau, G.; Seidah, N. G. *FEBS Lett.* **1995***, 362*, 143-146.

(28) Itoh, Y.; Tanaka, S.; Takekoshi, S.; Itoh, J.; Osamura, R. Y. *Pathol. Int.* **1996***, 46*, 726-737.

(29) Tanaka, S.; Kurabuchi, S.; Mochida, H.; Kato, T.; Takahashi, S.; Watanabe, T.; Nakayama, K. *Arch. Histol. Cytol.* **1996***, 59*, 261-271.

(30) Rawdon, B. B.; Larsson, L. I. *Histochem. Cell Biol.* **2000***, 114*, 105-112.

(31) Portela-Gomes, G. M.; Stridsberg, M. *J. Histochem. Cytochem.* **2001***, 49*, 483-490.

(32) Webb, G. C.; Dey, A.; Wang, J.; Stein, J.; Milewski, M.; Steiner, D. F. *J. Biol. Chem.* **2004***, 279*, 31068-31075.

(33) Rholam, M.; Fahy, C. *Cell Mol. Life Sci.* **2009***, 66*, 2075-2091.

(34) Ozawa, S.; Katsuta, H.; Suzuki, K.; Takahashi, K.; Tanaka, T.; Sumitani, Y.; Nishida, S.; Yoshimoto, K.; Ishida, H. *Endocr. J.* **2014***, 61*, 607-614.

(35) Katsuta, H.; Ozawa, S.; Suzuki, K.; Takahashi, K.; Tanaka, T.; Sumitani, Y.; Nishida, S.; Kondo, T.; Hosaka, T.; Inukai, K.; Ishida, H. *Endocr. J.* **2015***, 62*, 485-492.

(36) Walker, M. W.; Smith, K. E.; Bard, J.; Vaysse, P. J. J.; Gerald, C.; Daouti, S.; Weinshank, R. L.; Branchek, T. A. *Peptides* **1997***, 18*, 609-612.

(37) Bard, J. A.; Walker, M. W.; Branchek, T.; Weinshank, R. L. U.S. Patent 5,976,814, Nov 2, 1999.

(38) Gehlert, D. R.; Schober, D. A.; Beavers, L.; Gadski, R.; Hoffman, J. A.; Smiley, D. L.; Chance, R. E.; Lundell, I.; Larhammar, D. *Mol. Pharmacol.* **1996***, 50*, 112-118.

(39) Cerdá-Reverter, J. M.; Larhammar, D. *Biochem. Cell Biol.* **2000***, 78*, 371-392.

(40) Berglund, M. M.; Lundell, I.; Eriksson, H.; Söll, R.; Beck-Sickinger, A. G.; Larhammar, D. *Peptides* **2001***, 22*, 351-356.

(41) Keire, D. A.; Bowers, C. W.; Solomon, T. E.; Reeve, J.R.,Jr. *Peptides* **2002***, 23*, 305-321.

(42) Balasubramaniam, A.; Mullins, D. E.; Lin, S.; Zhai, W.; Tao, Z.; Dhawan, V. C.; Guzzi, M.; Knittel, J. J.; Slack, K.; Herzog, H.; Parker, E. M. *J. Med. Chem.* **2006***, 49*, 2661-2665.

(43) Parker, M. S.; Sah, R.; Sheriff, S.; Balasubramaniam, A.; Parker, S. L. *Regul. Pept.* **2005***, 132*, 91-101.

(44) Walther, C.; Mörl, K.; Beck-Sickinger, A. G. *J. Pept. Sci.* **2011***, 17*, 233-246.

(45) Domin, H.; Pięketa, E.; Piergies, N.; Święch, D.; Kim, Y.; Proniewicz, L. M.; Proniewicz, E. *J. Colloid Interface Sci.* **2015***, 437*, 111-118.

(46) Li, J.; Tian, Y.; Wu, A. *Regen. Biomater.* **2015***, 2*, 215-219.

(47) Boel, E.; Schwartz, T. W.; Norris, K. E.; Fiil, N. P. *EMBO J.* **1984***, 3*, 909-912.

(48) Southey, B. R.; Amare, A.; Zimmerman, T. A.; Rodriguez-Zas, S. L.; Sweedler, J. V. *Nucleic Acids Res.* **2006***, 34*, W267-W272.

(49) Shen, Y.; Maupetit, J.; Derreumaux, P.; Tufféry, P. *J. Chem. Theory Comput.* **2014***, 10*, 4745-4758.

(50) Schwartz, T. W. *J. Biol. Chem.* **1987***, 262*, 5093-5099.

(51) Atkins Jr., N.; Mitchell, J. W.; Romanova, E. V.; Morgan, D. J.; Cominski, T. P.; Ecker, J. L.; Pintar, J. E.; Sweedler, J. V.; Gillette, M. U. *PLoS ONE* **2010***, 5*, 1-13.

(52) Carter, J. D.; Dula, S. B.; Corbin, K. L.; Wu, R.; Nunemaker, C. S. *Biol. Proced. Online* **2009***, 11*, 3-31.

(53) Tucker, K. R.; Li, Z.; Rubakhin, S. S.; Sweedler, J. V. *J. Am. Soc. Mass Spectrom.* **2012***, 23*, 1931-1938.

(54) Berman, E. S. F.; Fortson, S. L.; Checchi, K. D.; Wu, L.; Felton, J. S.; Wu, K. J. J.; Kulp, K. S. *J. Am. Soc. Mass Spectrom.* **2008***, 19*, 1230-1236.

(55) Fitzpatrick, J. M.; Hill, D. L. G.; Maurer Jr., C. R. Handbook of Medical Imaging. In *Handbook of medical imaging, volume 2. Medical image processing and analysis.*; Fitzpatrick, J. M., Sonka, M., Eds.; SPIE Press, Bellingham WA, USA: 2009; Vol. 2, pp 447-513.

CHAPTER 7

SINGLE CELL PROFILING USING IONIC LIQUID MATRIX-ENHANCED SECONDARY ION MASS SPECTROMETRY FOR NEURONAL CELL TYPE DIFFERENTIATION

Notes and Acknowledgements

This chapter was published as an original research article in *Analytical Chemistry* **2017**, DOI: 10.1021/acs.analchem.6b04819, coauthored by T.D. Do, S.J.B. Dunham, S.S. Rubakhin and J.V. Sweedler. The article is adapted and reprinted here with permission from the American Chemical Society, copyright 2017. T.D. Do and T.J. Comi authored the manuscript, designed the hardware modifications and programmed the Arduino microcontroller. T.J. Comi performed data analysis and T.D. Do acquired data. The project was supported by the National Institutes of Health, Award Number P30 DA018310 from the National Institute on Drug Abuse, and Award Number 1U01 MH109062 from the National Institute of Mental Health. The authors also acknowledge Xiying Wang for help with sample preparation, and Joseph Ellis and Elizabeth Neumann for assistance with instrumentation and useful discussions.

Introduction

Single cell heterogeneity appears in seemingly homogeneous cell populations, even when derived from identical genetic blueprints. Adjacent cells within tissues have distinct identities and chemical contents; probing these differences aids in our understanding of the interplay between chemistry, cell activity, and function in complex tissues. As a single cell divides and differentiates into distinct subpopulations or into a malignant tumor, fluctuations in chemical composition and changes in cellular state manifest as diverging cell lineages, confounded with decisions related to cell fate from environmental cues.[1] Cell populations appear only as homogeneous as our ability to detect differences in their chemical composition. Newly developed techniques measure heterogeneity in genetic materials, proteins, peptides, lipids, and metabolites.[2-11] Recent successes in single cell studies help address confounding questions in cell biology and shape the next generation of drug discovery and development efforts.[1,12] Even so, there remains a need for single cell techniques that are capable of simultaneously detecting many classes of biological molecules in populations of cells.[13] The search for rare cells, which for decades was akin to finding a needle in a haystack, has become tractable with the emergence of high-throughput and sensitive measurement techniques.

Typical mammalian cells contain a few picoliters of volume, with analyte concentrations ranging from picomolar to millimolar. Thus, a successful single cell analytical technique should provide a low absolute detection limit, a high dynamic range, and multiplexed coverage of analyte classes.[5] Mass spectrometry (MS) has become a versatile and robust method for performing volume-limited biological measurements. Mass spectrometry imaging (MSI) is at the forefront of MS-based, label-free platforms for analyzing single cells,[6,14-18] demonstrating cellular and subcellular spatial resolution[19-21] and untargeted detection of biological molecules.[5,8] If cellular analytes are efficiently desorbed and ionized, the gas phase ions can be further interrogated with hybrid MS instrumentation for structural fragmentation,[22,23] ion sizes and shapes,[23-26] secondary structures,[27,28] and thermodynamic properties.[29,30]

Secondary ion mass spectrometry (SIMS) and matrix-assisted laser desorption/ionization (MALDI) are two common MS ionization microprobes that are suitable for spatially-resolved surface analysis of single cells.[31-34] MALDI uses focused laser light to desorb and ionize sample analytes incorporated into a suitable matrix. In contrast, SIMS utilizes a beam of accelerated primary ions or larger clusters that bombards the sample surface to sputter and generate secondary ions in the gas phase for mass-to-charge (*m/z*) analysis. Since the primary ion beam can be tightly focused, SIMS can achieve subcellular spatial resolution.[31]

Previous SIMS imaging investigations have established sample preparation methods that improve the limits of detection and molecular coverage of biological samples, including metal assisted[35-37] and ionic liquid (IL) matrix-enhanced SIMS.[38-40] Many IL mixtures have unique physical and chemical properties that can be optimized for SIMS- and MALDI-based detection.[40,41]

Recently, we demonstrated the capability of a high-throughput, microscopy-guided MALDI MS profiling method to classify dissociated rat pituitary cells, including rare cells, as well as elucidate the cellular heterogeneity of rat islets of Langerhans.[32,33] The approach circumvents the need for MS raster imaging[42,43] of a large region of interest, which is time consuming and often splits cell signals over multiple pixels. However, most single cell SIMS studies also utilize raster imaging[22,34,35,44-50] to fully leverage the subcellular spatial resolution of the method and localize analytes within single cells, albeit at low throughput and reduced sensitivity.

Establishing optically-guided single cell SIMS profiling should facilitate lipidomics and metabolomics studies on large populations of cells. Here we report a combination of matrix-enhanced SIMS (ME-SIMS) and multivariate statistical analysis to profile single cells from the *Aplysia californica* central nervous system, the rat dorsal root ganglion (DRG), and the rat cerebellum. These neuronal cell types were chosen because they represent well-characterized large (>75 μm in diameter), medium (10–50 μm), and small (5–10 μm) cells, respectively. The *A. californica* samples included large neurons with well-studied metabolite and lipid contents,[22,35,42,46,49] and are therefore suitable for our method validation experiments. The DRG contains the cell bodies of sensory neurons actively participating in neuropathic pain.[51] DRGs are traditionally classified based their on size,[52] electrophysiological properties,[53] and peptide content.[54] Cellular heterogeneity within the DRG was previously shown to affect opioid peptide sensitivity[55] and produce differential responses to neuropathic pain.[56] The cerebellar cells are critical to cognitive function and motor control,[57] and were chosen as small cell targets for this study.

Here we performed ME-SIMS utilizing three different IL matrixes to determine their ability to enhance the sputtering/ionization efficiency and chemical signals for single cell SIMS profiling. In addition, ME-SIMS tandem MS was performed to identify and characterize metabolites, including lipids, from single cells. Data sets acquired from populations of DRG and cerebellar cells were classified by *t*-distributed stochastic neighbor embedding (t-SNE). Each cellular population was further sub-divided by the same method, revealing their heterogeneity based on lipid content.

Methods

Chemicals

All chemicals were purchased from Sigma Aldrich (St. Louis, MO) and used without further purification.

Matrix Preparation

Three IL matrix solutions were evaluated. The first, MI-CHCA, was prepared by dissolving 250 mg of α-cyano-4-hydroxycinnamic acid (CHCA; > 98% purity) in 10 mL LC-grade methanol, followed by an addition of 105 μL of 1-methylimidazole (MI; Reagent Plus, 99%), with 10 mL of LC-grade acetonitrile added to the total volume of 20 mL. The second, TRIP-CHCA, was similarly prepared using 252 μL of tripropylamine (TRIP; > 98% purity). The third, Mix-CHCA, was prepared by mixing equal volumes of the MI-CHCA and TRIP-CHCA solutions. DHB matrix was prepared by dissolving DHB (99% purity) to 50 mg/mL in 1:1 (v/v) LC-grade ethanol:water and 0.1% trifluoroacetic acid solvent.

Sample Preparation

Aplysia californica. Two *Aplysia californica* (100–250 g body weight) were purchased from the National Resource for *Aplysia* (Rosenstiel School of Marine and Atmospheric Science University of Miami, FL). The mollusks were kept in aerated, circulated, filtered and chilled to 20 °C sea water prepared from Instant Ocean Sea Salt (Instant Ocean, Aquarium Systems Inc., Mentor, OH) dissolved in purified water. Animals were anesthetized by injection of isotonic $MgCl_2$ (~30% to 50% of body weight) into the body cavity. Central nervous system ganglia were dissected and placed in artificial sea water (ASW) containing 460 mM NaCl, 10 mM KCl, 10 mM $CaCl_2$, 22 mM $MgCl_2$, 26 mM $MgSO_4$ and 10 mM HEPES in Milli-Q water (Millipore, Billerica, MA), with the pH adjusted to 7.8 using 1 M NaOH in Milli-Q water. Ganglia were treated with enzyme solution consisting of 1% (wt/vol) protease type IX (Sigma Aldrich, St. Louis, MO) in ASW supplemented with 100 units/mL penicillin G, 100 µg/mL streptomycin, and 100 µg/mL gentamicin for 45 min at 34.4 °C. The connective tissue surrounding neurons and neuropil was surgically removed and multiple individual neurons mechanically isolated. Cells were deposited on indium-titanium oxide (ITO)-coated glass slides, 70–100 ohms, 25 × 75 × 1.1 mm (Delta Technologies, Loveland, CO), which were washed with Milli-Q water and ethanol before use. ITO-coated glass slides were placed in ASW and cells were allowed to adhere for 30 min. Next, ASW was replaced with a 33% glycerol 67% ASW (v/v) solution that was decanted after a 5 min incubation. Cells were left to dry overnight.

Rattus norvegicus. Seven 2.5–3 month old male Sprague-Dawley outbred rats (*Rattus norvegicus*) (www.envigo.com) were housed on a 12-h light cycle and fed ad libitum. Animal euthanasia was performed in accordance with the appropriate institutional animal care guidelines (the Illinois Institutional Animal Care and Use Committee), and in full compliance with federal guidelines for the humane care and treatment of animals. The studies were planned in accordance with the ARRIVE guidelines.[58]

Rats were killed by quick decapitation using a sharp guillotine. Rat trunks were placed on ice, where all surgical procedures were performed. Dorsal root ganglia (DRG) were surgically isolated during the ~10-min dissection procedure and placed into ~5 mL of cold Modified Gey's balanced salt solution (mGBSS) containing 1.5 mM $CaCl_2$, 4.9 mM KCl, 0.2 mM KH_2PO_4, 11 mM $MgCl_2$, 0.3 mM $MgSO_4$, 138 mM NaCl, 27.7 mM $NaHCO_3$, 0.8 mM NaH_2PO_4, and 25 mM HEPES dissolved in Milli-Q water, with the pH adjusted to 7.2 using 1 M NaOH in Milli-Q water.

To remove the surrounding connective tissue and isolate individual neurons, the DRG were incubated in 2.5% collagenase in oxygenated mGBSS for 25 min. All steps in the protocol were carried out at 37 °C, and oxygenated mGBSS was used in all cases. The DRG were then washed with 1% bovine serum albumin (BSA) in mGBSS for 7 min. The BSA solution was replaced with mGBSS and the DRG were incubated for an additional 20 min. Next, the DRG were treated with 0.65% trypsin in mGBSS for 20 min, followed by a 1% BSA solution wash for 7 min. The BSA solution was replaced with mGBSS containing Hoechst 33342 nuclear stain (Thermo Fisher Scientific, Waltham, MA) (1 mg/mL Hoechst 33342 stock solution, stored at 14 °C, diluted 1:500 in mGBSS) in which the DRG were incubated for 10 min. The Hoechst nuclear stain solution was then replaced with mGBSS and the DRG were mechanically dissociated by trituration. Cells were stabilized using a 40% glycerol, 60% mGBSS (v/v) mixture and after 5–10 min, plated on ITO-coated glass slides. The samples were then stored in the dark overnight to allow the cells to adhere onto the glass surface. On the next day, excess glycerol-containing media was removed from the preparations. At this stage, the cells were either subjected to immediate SIMS or MALDI analysis, or left to adhere to the ITO-coated glass surface in a nitrogen-purged dry box for 24 h before the MS measurements. Before analysis, the sample slides were rinsed with 2 mL of 150 mM ammonium acetate buffer (pH 10). This step helped remove the excess glycerol and did not induce observable damage to the cells.[32,33,59]

Instrumentation

The single cell profiling experiments were performed on two instruments. The first, a customized hybrid MALDI/C_{60}-SIMS Q-TOF mass spectrometer, described in detail elsewhere,[46] was operated in positive ion mode for all SIMS measurements. Negative ion mode on the custom instrument did not provide sufficient ion current during single cell experiments. The 40-μm diameter, 20 kV C_{60}^+ ion beam (Ionoptika, Ltd., Hampshire, UK) was operated in continuous mode with 500 pA sample current to yield an ion dose of 2.5×10^{14} ions/cm^2. Positive secondary ions were collected from *m/z* 60–850 using a Q1 bias of 5%, 5%, 20%, 35% and 35% at *m/z* 100, 180, 300, 500 and 700, respectively. The signal accumulation time was set to 2 s. The time-bins-to-sum was set to 10. Tandem MS spectra were collected in product-ion mode with the argon collision gas and collision-induced dissociation energy set at 35 eV. The C_{60}-SIMS instrument required minor hardware modifications to utilize the previously-reported cell finding software[32,33] for single cell measurements.

Several modifications were made to the cell coordinate registration protocol and hardware of the previously-reported C_{60} SIMS instrument.[46] For the registration protocol, three types of coordinate systems were utilized to interface with the instrument and optical images. The first system was in pixel coordinates in the microscopy image such that each dispersed cell had a unique (X, Y) pixel location. The second coordinate system was the stepper motor location (X_{motor}, Y_{motor}) of the instrument *x,y*-translation stage. When the ITO-coated glass slide was loaded into the instrument stage, the fiducial markers were used to calculate the transformation from (X, Y) to (X_{motor}, Y_{motor}) for each cell. These steps were identical to the previously reported coordinate registration protocol.[32,33] Unfortunately, the oMALDI Server software (AB Sciex, Framingham, MA) does not accept motor coordinates as direct inputs. The "Search Pattern" feature directed the stage to dwell at a series of points relative to a pre-set origin dictated by a separate coordinate system, which we refer to as the "Pattern" (PTN) coordinate system. One PTN unit is equivalent to 0.5 mm. The origin was set at the point $X_{motor,}$ Y_{motor} = (8,000, 17,600) which was equal to X_{PTN}, Y_{PTN} = (5, 5) in the PTN coordinate system. Through two coordinate transformations, each cell from the optical image was mapped to a PTN coordinate readable by oMALDI Server. Cells were then analyzed with the 20kV C_{60}^{+} ion beam as the stage traveled through the locations in the "Search Pattern" inputs.

An Arduino Atmega 2560 board (https://www.arduino.cc/) was programmed by the Arduino Software, v1.6.9, to do the described tasks (Figure 7.1A). Since each stepper motor was tracked by a linear encoder, its signals from the quadrature channels A/B were tapped out and connected to two interrupt pins on the Arduino board, which provide fast response times. A 4x decoding method was used to determine when the stage was moving (i.e., changes in signals of either encoder) and when the stage stopped (i.e., no changes in both encoders). As the data acquisition on the mass spectrometer was initiated by Analyst (AB Sciex), a 5 V signal from the mass spectrometer control board was received by interrupt pin 19 on the Arduino board to activate stage monitoring and synchronize the start times of the stage and mass spectrometer. When the stage stopped at a cell, the Arduino board would delay for 3 s and then send a 5 V signal through pin 13 to a relay controlling the C_{60}^+ beam, initiating desorption for 1 s (see Figure 7.1B). Longer beam "On" times cause damage on the sample surface and generate more chemical noise. The beam "Off" delays ensured no contamination occurred between cells of interest and surrounding neighbors. Data from the Arduino, including clock time, elapsed time, encoder position and ion beam status, were recorded with the PLX-DAQ add-on for Excel 2010 (https://www.parallax.com/downloads/plx-daq).

Single cell SIMS profiling was accomplished by registering the *x,y* translation stage of the oMALDI server (AB Sciex, Framingham, MA) with whole-slide, bright-field, and fluorescence images. Cells deposited onto indium-titanium oxide (ITO)-coated glass slides were placed into a custom sample holder that can accommodate slides as large as 40×25 mm^2 (about one-half of a standard microscope slide). Several mechanically-etched fiducial markers facilitated point-based similarity registration to map cell locations back to their stage coordinates. Because of the limitations with the SIMS instrument control, mass spectral data were continuously acquired as the stage moved. Because the sample stage repeatedly travels and stops at cell locations, it is critical to synchronize its movements with the primary C_{60}^+ ion beam activity and record when a cell is reached. The ion beam should only be "On" when the cell location is reached, and "Off" during sample stage movement, to ensure that only targeted cells are bombarded with primary ions. The dwell time of the translation stage was set to 6 s per cell. For each cell, the C_{60}^+ ion beam was signaled to turn on for 1 s after a 3 s delay. The 1 s sputtering time was found to be optimal, as shorter times yielded inadequate signals whereas longer beam exposure caused IL matrix depletion and complication of the mass spectra with additional background ion signals. The acquisition rate on the mass spectrometer was set to 2 s to improve the likelihood that the entire analyte sputtering event was captured in a single scan. In ~20% of scans, the analyte sputtering event was split between two acquisition windows, resulting in a separation of low mass and high mass ions between two mass spectra (see Figure 7.1C).

The second instrument used was a Bruker ultrafleXtreme MALDI TOF/TOF mass spectrometer with a frequency tripled Nd:YAG solid state laser. Single cell MALDI MS analysis was performed as previously reported.[32,33] The molecular mass scan window was set to *m/z* 400−8000 and the laser was operated in the "Ultra" mode, producing a ~100-µm diameter footprint. The ultrafleXtreme AutoXecute feature (Bruker Daltonics, Billerica, MA) was utilized with the custom geometry file as previously reported.[33] Each spectrum represents the summed signals acquired during 1,000 laser shots fired at 1 kHz.

Optical Imaging and Determination of Pixel Coordinates for Individual Cells

Each dispersed cell population on an ITO-coated glass slide was imaged using an Axio Imager M2 (Carl Zeiss, Oberkochen, Germany) in fluorescence and bright-field modes. An X-CITE 120 mercury lamp (Lumen Dynamics, Mississauga, Canada) and a 31000v2 DAPI filter set (Chroma Technology, Irvine, CA) were employed for fluorescence imaging. Because the ITO glass slides are transparent and conductive, they are compatible for both MALDI-MS and SIMS single cell profiling experiments. A 10× objective was used to obtain a mosaic image of the targeted surface with 13% overlap between neighboring images. Images were taken using an AxioCam 503 Mono camera (Carl Zeiss) with a resolution of 1936 × 1460 pixels. All mosaic optical images were stitched with a minimum overlap of 5% and maximum shift of 10%. The stitched image was loaded into microMS[32] for either manual or automatic cell finding. The fiducial marks were used to register the image coordinates to the *x,y* translation stage coordinate of the mass spectrometers. On the basis of the registration, the cell coordinates were saved in either a pattern coordinate file-format readable by the oMALDI server for SIMS experiments or a custom geometry file for the MALDI-MS FlexControl software for mass spectral acquisition.

Matrix Application

ITO-coated glass slides were affixed onto a rotating plate for automatic matrix application, as described elsewhere.[32] Spraying conditions were optimized for each MS system. The distance between the spray tip and the rotating plate was 5 cm for SIMS and 2 cm for MALDI MS, with a nitrogen gas pressure of 50 psi. The solution flow rate was set to 30 mL/h for SIMS and 10 mL/h for MALDI MS, resulting in a matrix coating of 6 mg/cm^2 and 15 mg/cm^2, respectively. DHB matrix was also employed for MALDI MS for comparison with the IL matrixes. The same spraying conditions were used for DHB as in IL MALDI-MS.

Multivariate Statistical Analysis

Data analysis was performed with custom scripts written in MATLAB (R2015b). MALDI MS data were read directly with the readbrukermaldi function (https://github.com/AlexHenderson/readbrukermaldi), resampled to 10,000 *m/z* values in the range *m/z* 500–1000, background corrected, smoothed, and normalized by standardizing the area under each spectrum to the median of the data set. SIMS data was first converted from the native wiff format into mzXML with msconvert[60] for import into MATLAB. The cell coordinates, diameters, and the corresponding log file from the instrument microcontroller were also utilized to parse continuous SIMS acquisition files. As the Arduino monitors when an acquisition begins, the start time for mass spectral acquisition and stage movement were synchronized. In MATLAB, the *x,y* translational stage stop events were recorded and used to delineate mass spectra corresponding to target cell positions. Stage dwell events shorter than 3 s or longer than 7 s were discarded as noise in the stage or encoders. As the ion beam is only "On" for 1 s within the 6 s dwell time, the spectrum with the highest intensity phosphocholine head group (*m/z* 184.07) and PC(34:1) (*m/z* 760.56) signal was selected as the single cell spectrum. Dwell events of single cell signals that occurred in two adjacent acquisition windows were discarded as "split cells". Finally, all mass spectra acquired from DRG and cerebellar cells were filtered for an intensity of the *m/z* 184.07 signal greater than 250 counts.

Statistical significance was established with a Wilcoxon rank sum test as intensity distributions were non-normal by a Kolmogrov-Smirnov test. Initial mass spectral data visualization was performed with principal component analysis (PCA) to evaluate the effects of different ionic liquid formulations. Unsupervised cell classification for DRG and cerebellum samples was performed with *t*-SNE[61] to reduce and group the data in two dimensions, followed by *k*-means clustering with a Euclidean distance metric. The number of clusters was determined by the mean silhouette value as a function of *k*.

For PCA, each cell spectrum was considered as an independent sample with the different *m/z* channels as the measured variables. The intensities of each *m/z* value for each cell produced a two dimensional matrix which was decomposed to score and loading values with the built-in pca function in MALTAB. Principal component (PC) scores for 1 and 2 were displayed on a scatter plot to assess data grouping based on ionic liquid formulation. The loading plot of PC1 was also displayed to determine the cause of data spread in PC1 axis. In this plot, negative loading for a specific *m/z* indicates the specie is present in high abundance in cells found in the negative PC1 range and similar for positive values. Based on this interpretation and the assistance of average spectra, it was determined negative PC scores correspond largely to cells with high chemical noise while positive values contained biologically relevant lipid and phosphocholine signals.

Results and Discussion

IL Matrix Enables Detection of Lipids in Single Cell SIMS Profiling Experiments

SIMS ionization allows the characterization of small molecules with high spatial resolution. Although SIMS does not require a matrix,[31] analysis of small-volume samples and low-concentration analytes can benefit from such treatments.[36,38,39,62] *A. californica* pedal ganglia neurons (~75 µm or larger in diameter) were used to validate cell targeting and examine the effects of the IL-matrix coating. Several room temperature ILs have been reported to enhance lipid, cholesterol, and peptide signals in SIMS measurements.[38-40] The enhancements are somewhat predictable, as ILs used in ME-SIMS are typically mixtures of an organic base and a traditional MALDI matrix, such as CHCA, acting as an acid.

Unlike ME-SIMS with a traditional MALDI matrix, the components of ILs are positively and negatively charged species that favorably assist proton transfer to, or capture from, analytes while preventing matrix crystallization. A total of 47 pedal neurons from two *A. californica* were examined with SIMS, producing one mass spectrum per cell. The averaged mass spectrum acquired with native conditions is shown in the left two panels of Figure 7.2A. The same sample was then coated with the MI-CHCA matrix and the same 47 cells were profiled to assess the effect of the IL matrix in SIMS analysis. The averaged mass spectrum acquired with the MI-CHCA coating is shown on the right side of Figure 7.2A. In both cases, the characteristic signals from α-tocopherol were detected at *m/z* 430.35 for the intact molecular ion as well as *m/z* 165.05 and *m/z* 205 for the fragments.[22,35,49] However, many lipids were detected in the *m/z* 600–850 mass range exclusively in the presence of MI-CHCA. Although the relative intensity of vitamin

E (*m/z* 430.35) was not significantly different after IL application (p = 0.73), the relative intensity of the phosphocholine head group (*m/z* 184.07) increased significantly, by two-fold (p <0.005) (Figure 7.2B). Finally, signals corresponding to fragments, sodiated and potassiated adducts of known glycerophospholipids (*m/z* 709.5, *m/z* 782.5, and *m/z* 808.6), and diacylglycerophosphocholines (*m/z* 746.5 and *m/z* 768.5) increased significantly (Figure 7.3), consistent with previously published SIMS imaging data.[49] Profiling of metabolites in the *A. californica* neurons demonstrates the utility of an IL matrix for SIMS analysis of individual cells.

IL Matrix Optimization for SIMS Analysis of DRGs

While the profiling of pedal ganglion neurons demonstrates the capabilities of MI-CHCA for SIMS single cell lipid detection, other matrix compositions were investigated to improve the figures of merit.

Matrix Composition

A matrix capable of proton transfer will likely assist in the ionization of lipid compounds during SIMS ionization.[38] Furthermore, for single cell profiling experiments it is imperative that the matrix uniformly covers the sample. Uniform coverage partially depends on interactions between the IL and substrate surface. Micrographs of an ITO-coated glass slide spray-coated with the MI-CHCA matrix (Figure 7.4A) show the matrix deposition non-uniformity when MI-CHCA solution was sprayed coated on a clean ITO glass slide. Similar to many imidazolium-based ILs, the high surface tension of MI-CHCA may lead to generation of non-uniform “puddles” on the surface.[41] Poor matrix coverage introduces experimental cell-to-cell variability and redistributes analytes and background contaminants from uncoated regions of the substrate. Therefore, while MI-CHCA is a suitable matrix for general-purpose SIMS analyses, single cell SIMS measurements require improvements in matrix properties. Advantages of using IL matrixes include the flexibility to tailor their physical properties by modifying IL components, or by utilizing different matrix mixtures.

Previous studies on IL structures suggest that increasing the alkyl chain lengths produces ILs with lower surface tensions, leading to more uniform sample coverage.[41] As such, the TRIP-CHCA matrix was considered, as well as an equal-volume mixture of MI-CHCA and TRIP-CHCA, referred to here as Mix-CHCA. Both TRIP-CHCA and Mix-CHCA showed more uniform sample coverage with the same coating conditions (see Figure 7.4A). Although previous reports demonstrated that MI-CHCA provides higher signal enhancement for SIMS than TRIP-CHCA,[38,39] the uniformity of the matrix coating was not considered in those studies; rather, the matrix and analyte solutions were mixed and spotted on the substrate.[38,39]

The three IL matrixes—MI-CHCA, TRIP-CHCA, Mix-CHCA—were evaluated for lipid analysis of rat DRG cells. The DRG contains a variety of physiologically important lipids, making it a viable model for method development in the study of biologically significant lipid contents.[63,64] Lipids have been shown to influence the activities of DRG neurons, and changes in lipid metabolism have been implicated in chronic neuropathic and inflammatory pain.[64-67] For these comparisons between IL matrixes, glycerol-stabilized DRG cell samples were washed with ammonium acetate buffer and stored in a nitrogen-purged dry box for 24 h before IL matrix application and MS analysis.

The observed lipid profiles obtained from the same animal using the three different IL matrixes, and using Mix-CHCA across different animals, were compared. At least 300 DRG cells in each sample were profiled in each set of measurements. To prevent measurement bias from inadequate lipid signals occurring due to: (a) systematic errors in cell coordinate registration; (b) random motor slop; (c) IL matrix application quality; or (d) inadequate ionization enhancement provided by the investigated matrix,[38,39] single cell spectra with an *m/z* 184.07 (phosphocholine head group) signal intensity of less than 250 counts were removed from subsequent analyses. The fraction of removed cells provides a measure of matrix quality, assuming systematic errors did not vary significantly among different batches. As shown in Figure 7.4B, TRIP-CHCA had the highest fraction of removed mass spectra, likely reflecting lower repeatability of analyte extraction or matrix enhancement with this IL matrix. The fraction of removed single cell mass spectra using Mix-CHCA was lower than even MI-CHCA when comparing samples from the same animal.

Figure 7.5A shows the principal component analysis (PCA) score and loading plots of the filtered data sets. The data acquired using the three IL matrixes are well separated (no overlap of the 95% confidence ellipses), whereas the three data sets obtained using three animals and Mix-CHCA overlap significantly. The score plots suggest that the matrix-related differences in data sets are larger than day-to-day and animal-to-animal variability. Mass spectra of individual cells investigated with MI-CHCA and characterized by negative PC1 scores generally had lower lipid signals and strong chemical noise in the *m/z* 200–400 range, as shown by the averaged mass spectra and the loading plot of PC1 (Figure 7.5A). The use of Mix-CHCA improved the number of cells with abundant lipid signals while reducing chemical noise. The relative signal intensities and number of analytes observed with Mix-CHCA are comparable to that of MI-CHCA, and better than TRIP-CHCA (see Figure 7.5B), suggesting that the presence of TRIP in the mixture does not adversely affect lipid signal intensities. Taking into account the uniform sample coverage and matrix enhancement, Mix-CHCA was chosen for further single-cell profiling with ME-SIMS.

Sample Preparation

Previous single cell studies suggested that cells analyzed shortly after isolation and sample preparation produce higher analyte signal intensities than those with prolonged storage, as sample degradation significantly reduces endogenous analyte concentrations.[35] In agreement with these findings, Figure 7.6 shows that sample storage profoundly impaired data quality. DRG cell samples prepared and analyzed on the same day produced lipid signals three-fold more intense than signals acquired from similar samples that were stored for one day in a dry nitrogen atmosphere prior to IL matrix application. The improved sensitivity allows the detection of many minor lipid species. Most lipids detected in the *m/z* 600–850 range are phosphatidylcholines (PCs), as confirmed by tandem MS (Figure 7.7 and Table 7.1). A number of the observed PCs exhibit alkyl chains possessing between 30 and 36 carbons. The alkyl chains have at least two unsaturated forms as well as the fully saturated form (inset in Figure 7.6B). The improved sensitivity from analysis of fresh biological samples is in agreement with prior MALDI-MSI data from rat brain tissue.[68]

ME-SIMS Single Cell Profiling Provides Complementary Data to MALDI MS

From individual DRG cells, PC(34:1) at *m/z* 760.56 and PC(32:0) at *m/z* 734.54 displayed the highest relative intensity, with *m/z* 760.56 as the base peak with SIMS. To verify that the single cell SIMS measurements produce semi-quantitative information on endogenous lipid levels similar to MALDI MS, DRG cell profiling was also conducted with the Bruker ultrafleXtreme MALDI TOF/TOF mass spectrometer. A comparative analysis of freshly prepared samples coated with either DHB or MI-CHCA matrix was conducted. The Mix-CHCA matrix performed poorly for MALDI MS (lower total ion counts with a strong bias toward low-mass matrix peaks; data not shown) although the lipid distributions obtained with Mix-CHCA are consistent with those obtained with MI-CHCA and DHB (see Figure 7.8 and Figure 7.9). The analyte signal profiles in the averaged mass spectra in the *m/z* 500–850 region agree qualitatively.

Quantitatively, the *m/z* 760.56 peak was typically the base peak in either analysis but the relative intensity of *m/z* 734.54 increased in the ME-SIMS experiments. Whereas SIMS investigates only the top few molecular layers of a sample, the inherent analyte extraction from sample volumes that occurs during MALDI matrix application leads to acquisition of mass spectra representative of the whole-cell content. Therefore, SIMS mass spectra likely possess signals more specific for the surface of cell membranes, producing the observed difference in lipid intensity ratios between the two approaches (Figure 7.9B). However, since the relative abundance of the two PC lipids is not significantly different between MALDI MS using DHB and MI-CHCA (Figure 7.9), the choice of the IL matrix in MALDI MS measurements may not lead to a change in the relative sensitivity during lipid detection.

The mass spectral peaks at *m/z* 478.3, 496.3, 522.3, and 550.3 (data not shown) are likely fragments of glycerophospholipids. This conclusion is made based on similarity in their *m/z* values to those of reported molecules and the presence of a peak at *m/z* 184.07 in the tandem MS product ion mass spectra of PC(32:0) at *m/z* 734.54 and PC(34:1) at *m/z* 760.56, and other lipids (Figure 7.7). Therefore, variations in the degree of analyte fragmentation between SIMS and MALDI MS may also account for the offset in detected ratios of intact lipids (Figure 7.9B). In summary, the relative lipid composition observed with SIMS appears to reflect endogenous cell content and provides complementary chemical information to MALDI MS.

Single Cell Profiling with SIMS Enables Differentiation of Cell Types with Similar Lipid Compositions

As discussed in the previous section, the relative PC lipid ratio from single cell ME-SIMS profiling appears to be an intrinsic property of a given cellular origin, at least for the cell types assayed here. However, incorporating more intact lipids into a multivariate dimension reduction, such as t-SNE, may improve identification of additional subpopulations based on variations in minor lipid species. To determine the capability of SIMS to distinguish single cells with similar lipid profiles, cell populations from the DRG and cerebellum were individually profiled. For each cell type, two technical replicates were performed with a total of 548 DRG cells and 995 cerebellum cells. The same threshold limit of 250 counts for *m/z* 184.07 was applied as before. The set of lipid compounds detected from cerebellum neurons is similar to DRG cells, and the two most dominant PC lipids, PC(32:0) and PC(34:1), are the same in both cell types. The mass spectra acquired from DRG and cerebellar cells have a number of signals within the *m/z* 700–850 range, corresponding to intact lipids. The spectra were normalized to intensities of the PC(34:1) *m/z* 760.56 signal.

The resulting data set was analyzed with t-SNE to provide a visual representation of the sample heterogeneity. Figure 7.10A shows the separation of DRG and cerebellum neurons determined by t-SNE. These cells from different origins are well-separated, with minimal overlap between the corresponding data sets, depicted by 95% confidence ellipses. The identity of the base peaks is distinct between the two averaged mass spectra acquired from each cell type. For DRGs, the unsaturated PC(34:1) at *m/z* 760.56 is the most intense signal, whereas for cerebellum neurons, the saturated PC(32:0) is the most intense peak. The high intensities of PC(32:0) and PC(34:1) signals may reflect the specific roles of these compounds in membrane integrity and fluidity. The cerebellar cell membranes, with relatively more PC(32:0), are expected to be less rigid than the membranes of larger DRG cells in which unsaturated PC(34:1) produces the strongest signal. This simple test illustrates the facile differentiation of nervous system cell types corresponding to their origin, based on lipid profiles.

Multivariate Statistical Analysis Reveals Subpopulations of DRG and Cerebellar Cells

Figure 7.11A shows the two clusters obtained from *k*-means clustering of the t-SNE distributions. *k*-means classifies spectra according to their distance on the t-SNE plot and groups them into a specified number of clusters. Qualitatively, the two clusters differ based on the relative intensities of PC(34:1) and PC(32:0) signals, as well as the relative abundance of minor lipids such as PC(36:1) at *m/z* 788.60, PC(38:5) at *m/z* 808.60, and PC(40:5) at *m/z* 836.60. One can argue that the difference observed here arose due to experimental factors such as target accuracy and cell finding errors. However, the average spectra contain similar signal to noise ratios (S/N) (see Figure 7.11B), indicating the cells from each subpopulation were analyzed with similar target accuracy. Additionally, if target accuracy was poor, the mass spectra would be expected to cluster based on cell diameters, since larger cells could tolerate higher uncertainty in target positioning. Therefore, our data suggest the presence of at least two chemically distinct DRG cell populations. As seen in Figure 7.11B, in the first subpopulation (red), the relative signal intensities of other lipids to PC(34:1) are higher than in the second population (blue). The two cell clusters appear to distribute similarly along the cell size axis (Figure 7.11C). A previous study suggested that an increase in the relative lipid content due to peripheral inflammation occurs mainly in small-diameter DRG cells, but not in larger-diameter cells.[64] Thus, the method described here can be instrumental in further characterization of such changes on a single cell level within these populations.[64 6060]

A similar multivariate analysis was performed on the cerebellar cell data set (Figure 7.12). Here, k-means clustering was optimal with four subpopulations. Unlike the outcome of the DRG data set analysis, the clusters are more distinct in the t-SNE space (Figure 7.12A). Qualitatively, the most intense PC(32:0) signal changed noticeably between clusters, increasing in clockwise order from the green to red cluster in the t-SNE space. The intensity of the saturated PC(34:0) at *m/z* 762.56 also followed the same trend (Figure 7.12B). Taking into account cell diameter and the size of the primary ion beam footprint, it is unlikely that some cells were partially sampled and passed the quality threshold (Figure 7.12C). However, the S/N for the yellow cluster is lower than the other clusters, primarily in the region outlined in Figure 7.13, suggesting that the cells in this cluster with high values of Int(734.54)/Int(760.56) may be "nearly-missed" cells rather than a separate subpopulation. The population and subpopulation classifications for the DRG and cerebellum cells were repeated with another pair of animals. This set of data was obtained one month after the first set to ensure that the conclusions were not biased by instrument and animal variations.

Furthermore, since instrument parameters were optimized for each experiment, the difference in the chemical noise profile for each data set may induce non-biological separation among animals. Hence, we did not combine the two data sets in the same analysis. Nevertheless, in the second data set, a total of 324 DRG cells and 1,249 cerebellum cells were subjected to t-SNE analysis and *k*-means clustering. The results are consistent with the data presented above for the first data set, and shown in Figures 7-14-7.16. However, for the cerebellum, the presence of only two cellular subpopulations in the second data set was observed. The values of Int(734.54)/Int(760.56) for the first two populations shown in Figure 7.12 are identical to the two populations shown in Figure 7.16 (i.e., 1.3 and 2.0), suggesting that these two populations are biologically relevant. It would be of great interest to correlate these subpopulations to known cerebellum cellular subtypes through the means of other techniques such as immunohistochemistry.

Conclusions

We describe a high-throughput method for single cell analysis using IL-assisted SIMS with a 20 kV C_{60}^{+} primary ion beam. The IL matrix, composed of 50:50 (v/v) MI-CHCA and TRIP-CHCA, provided uniform coating coverage and robust enhancement of a number of lipid signals that were otherwise undetectable. Cells from three different model systems were studied: the *A. californica* central nervous system, and the rat DRG and cerebellum. For each cell type, characteristic metabolites (e.g., α-tocopherol) and lipids (e.g., PCs) were observed. While PC(34:1) was the most abundant lipid in a majority of the studied DRG cells, there were a few cells with more abundant saturated PC(32:0) than unsaturated PC(34:1). These cells also showed an increase in absolute intensities of lipid signals. Therefore, at least two subpopulations (types) of DRG cells can be classified using the approaches presented here. Analysis of the lipid profiles of cells isolated from the rat cerebellum revealed similar lipid compositions as those found in DRG cells. However, PC(32:0) was more frequently observed as the base lipid peak—a characteristic molecular signature to differentiate these two cell types from each other. Further classification of cerebellar cells based on lipid profiles revealed at least two cellular subpopulations.

Our method provides a unique approach to differentiate cell types and subtypes by utilizing lipid ratios as biomarkers, and is applicable to different classes of neuronal cells. Optically-guided ME-SIMS can profile up to 2,000 cells in one experiment at the rate of ~600 cells/h. These figures of merit are limited by three factors: (a) the size of the sample holder/slide,

(b) the control of the ion beam state and its synchronization to mass spectral acquisition, and (c) the ionization efficiency of the primary ion beam. The maximum number of cells assayed per experiment is proportional to the sample area but is also limited by the degradation of IL matrix inside the vacuum chamber, typically occurring after 3–4 h of analysis. A higher energy ion beam would improve ionization efficiency and decrease the acquisition time per cell. Together with better communication and synchronization between the ion beam, sample stage, and mass analyzer, an acquisition rate as fast as 1 Hz can be achieved. The availability of subcellular resolution with SIMS will aid in the discovery of compartment-specific cellular markers. A combination of SIMS and MALDI MS profiling with other non-MS based techniques, including microscopy and spectroscopy, will allow simultaneous and multidimensional characterization of various analyte classes in the same sample, yielding unique and complementary information for each cell.

Figures and Table

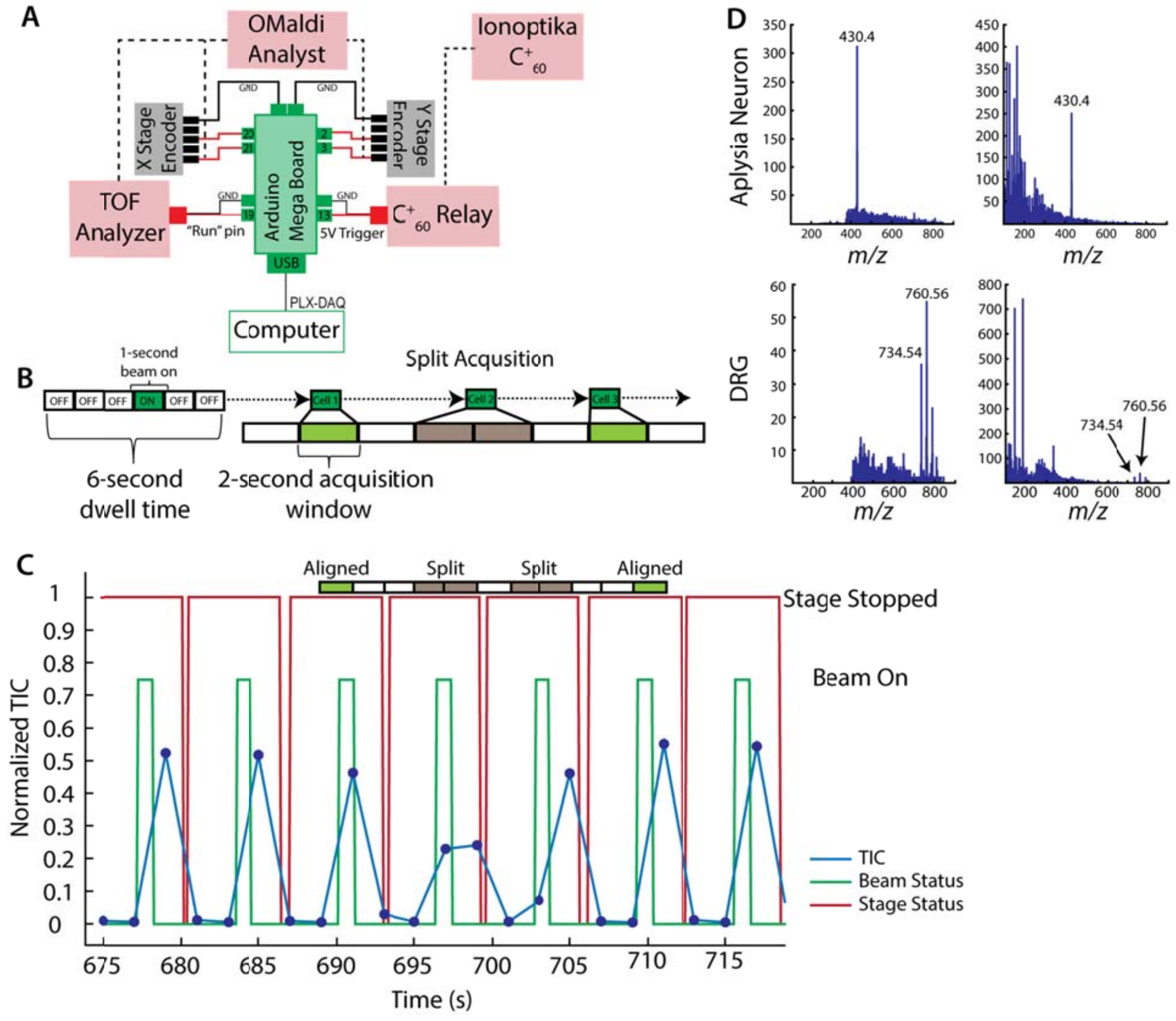


Figure 7.1. (A) Schematic of the microcontroller integration to synchronize motor stage movements, mass spectrometer acquisition, and C_{60}^+ ion beam status. (B) An illustration of a 6-second dwell time and a "split" cell with a 2 s acquisition time. (C) Overlay of total ion current, stage motion, and beam state as a function of time for a representative sample. Instances of aligned and split spectra are highlighted above. (D) Representative examples of "split" cells. In the top two mass spectra, the signals of vitamin E (*m/z* 430.4) are split between the two acquisition windows. In the bottom two mass spectra, the lipid signals are also split.

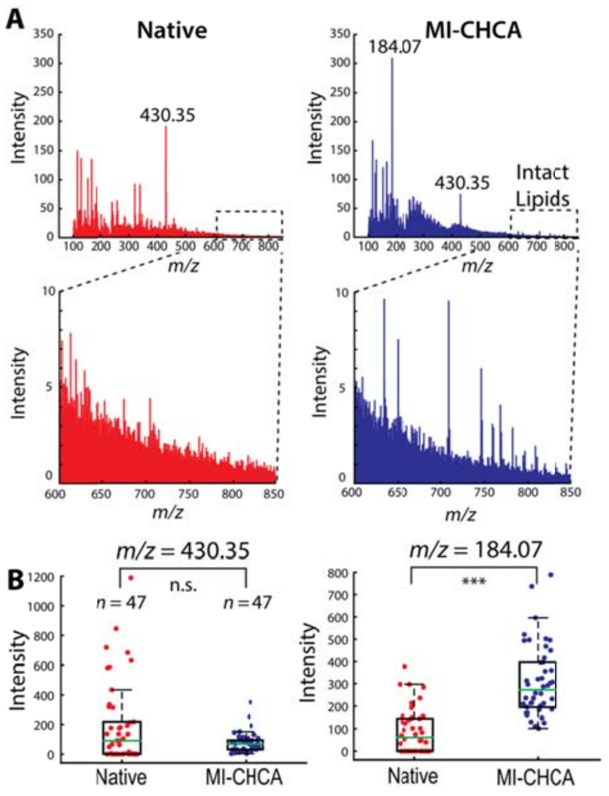


Figure 7.2. Effect of the IL matrix, MI-CHCA, on single cell profiling of *A. californica* pedal ganglion neurons with C_{60}-SIMS. (A) The averaged SIMS mass spectrum acquired from a batch of single *A. californica* pedal ganglion neurons not treated (left, red) and treated (right, blue) with MI-CHCA matrix. (B) Box plots of α-tocopherol (*m/z* 430.35) and phosphocholine head group (*m/z* 184.07) signal intensities detected at two studied conditions. The statistical significance in each comparison was determined by a Wilcoxon rank sum test, *** p <0.0005, n.s. - not significant, n = 47 cells from two animals.

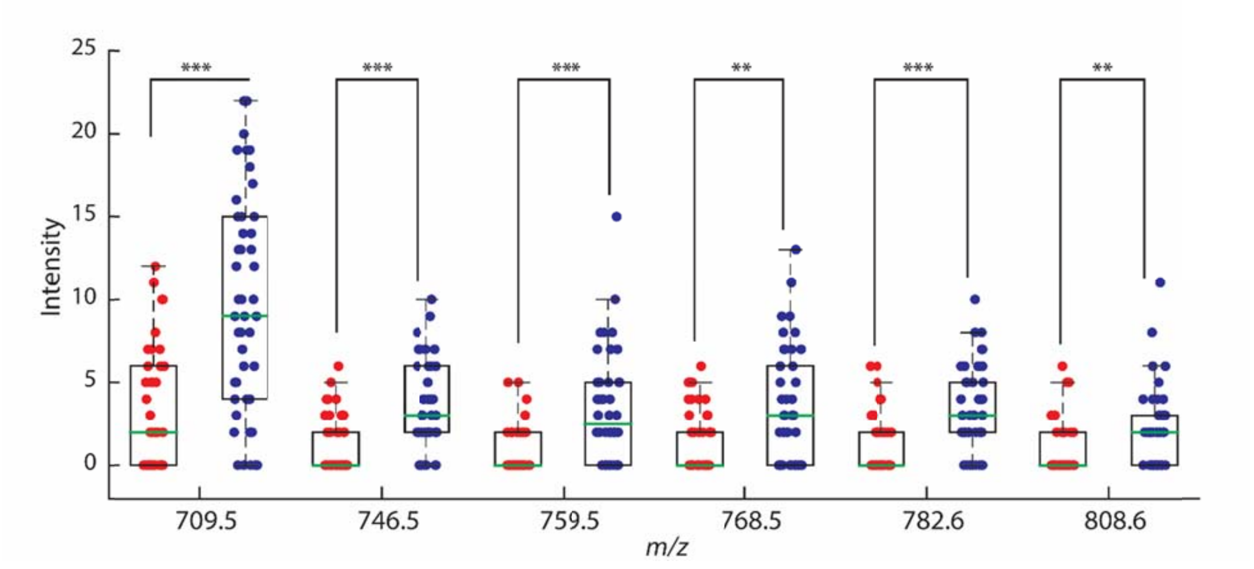


Figure 7.3. Box plot graph summarizing data for no-matrix (red) and matrix-assisted SIMS (blue) for lipid signals identified in a previous SIMS imaging report.[49] The statistical significance in each comparison was determined by a Wilcoxon rank sum test, ** p <0.005, *** p <0.0005, n = 47 cells from two *Aplysia*.

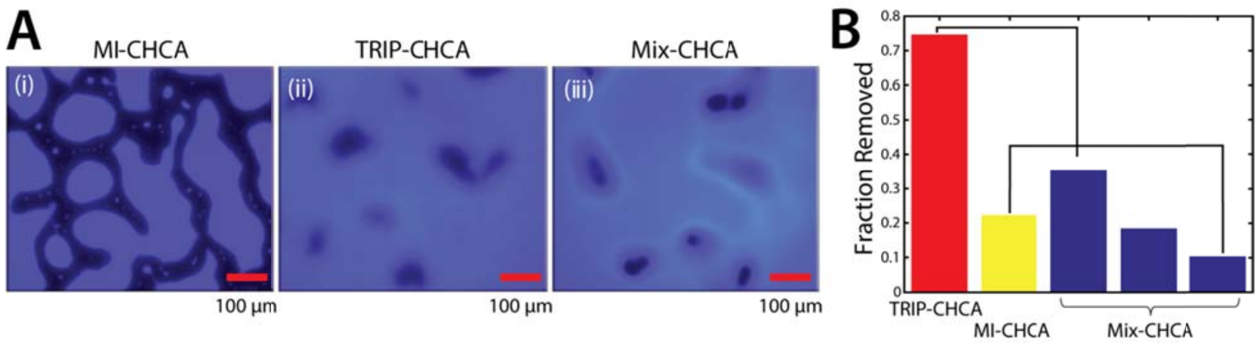


Figure 7.4. (A) Representative fluorescent micrograph of ITO slides coated with (i) MI-CHCA, (ii) TRIP-CHCA, and (iii) Mix-CHCA. (B) The fraction of data acquired from individual DRG cells was removed due to unacceptably low lipid signals. The black lines indicate samples from the same animals.

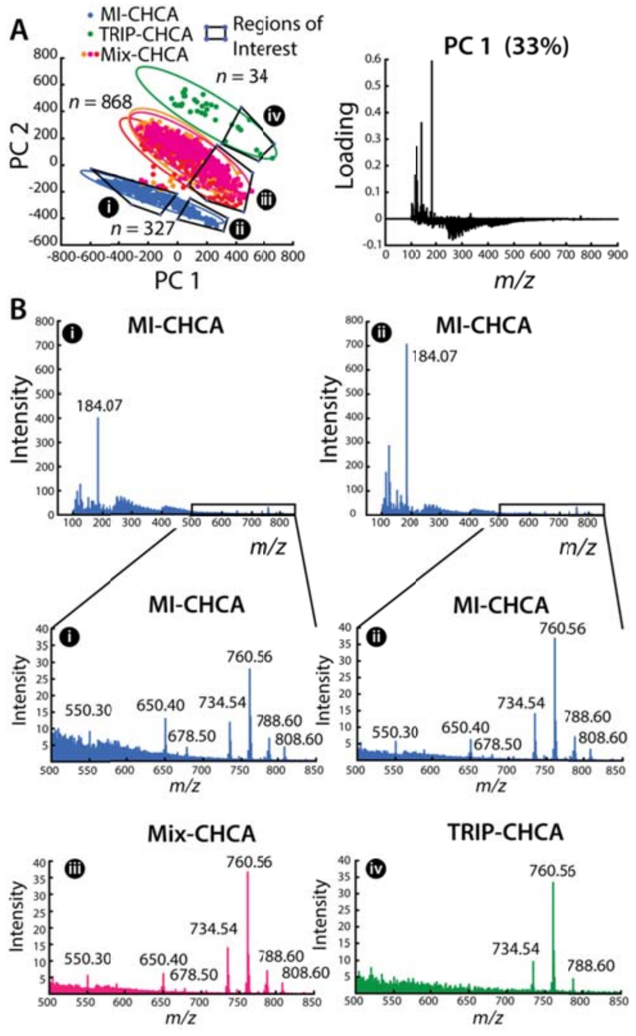


Figure 7.5. Multivariate analysis of the effects of different IL matrixes on ME-SIMS measurements of 1229 individual DRG cells. (A) PCA score and PC1 loading plots showing the separation between the three different IL matrixes. 95% confidence ellipses are shown for each condition with regions of interest outlined and labeled (i-iv). (B) Averaged spectra of DRG in negative (i) and positive (ii) PC1 score regions. The corresponding averaged mass spectra of regions i–iv for *m/z* regions populated by intact lipid signals. The identified lipid peaks are annotated with their *m/z* values.

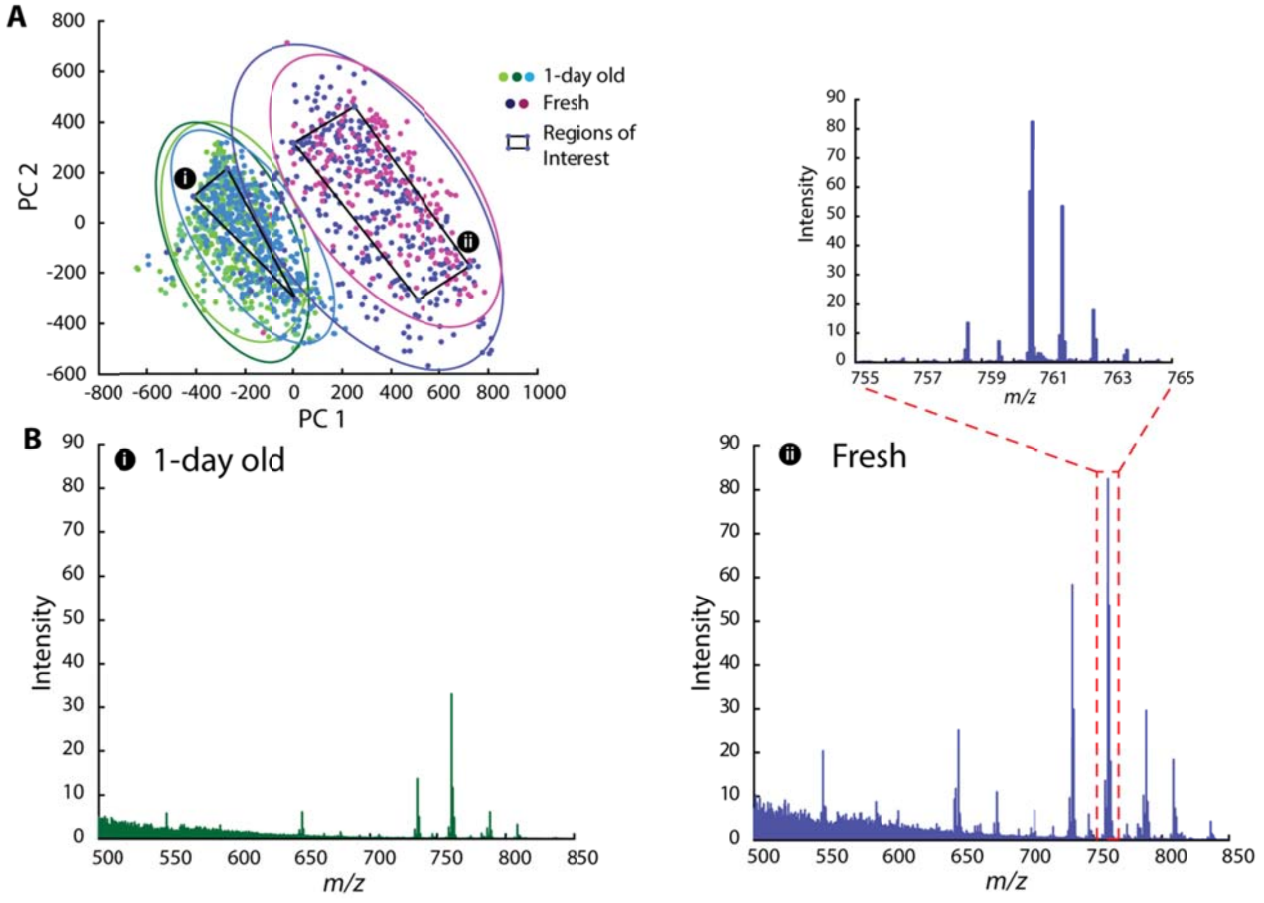


Figure 7.6. (A) PCA score plot of 1-day old and fresh DRG samples. (B) Averaged mass spectra for each sample type, (i) 1-day old samples, (ii) fresh samples.

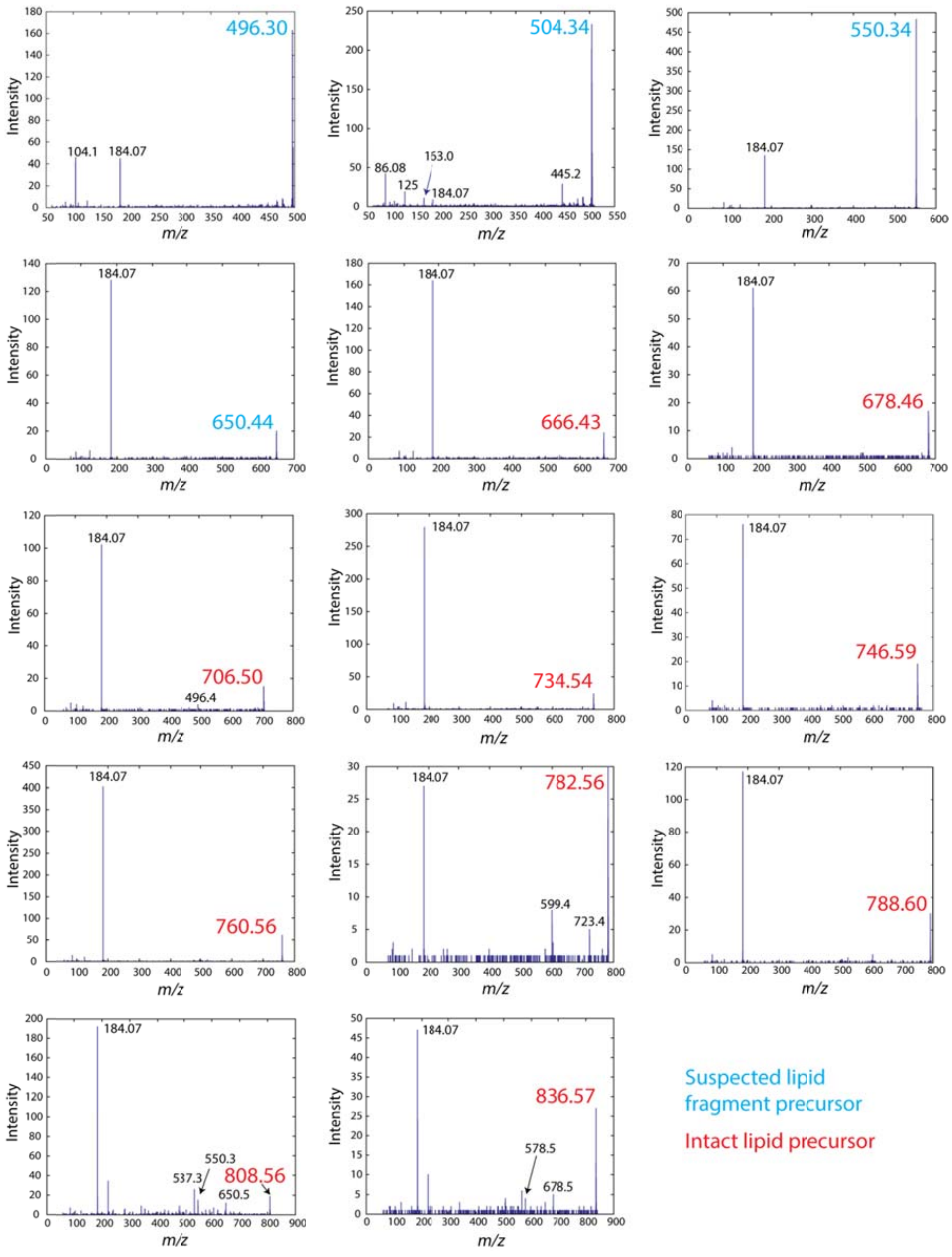


Figure 7.7. Tandem mass spectra of major lipids obtained from SIMS single cell profiling. The tentative identity of each lipid is provided in Table 7.1.

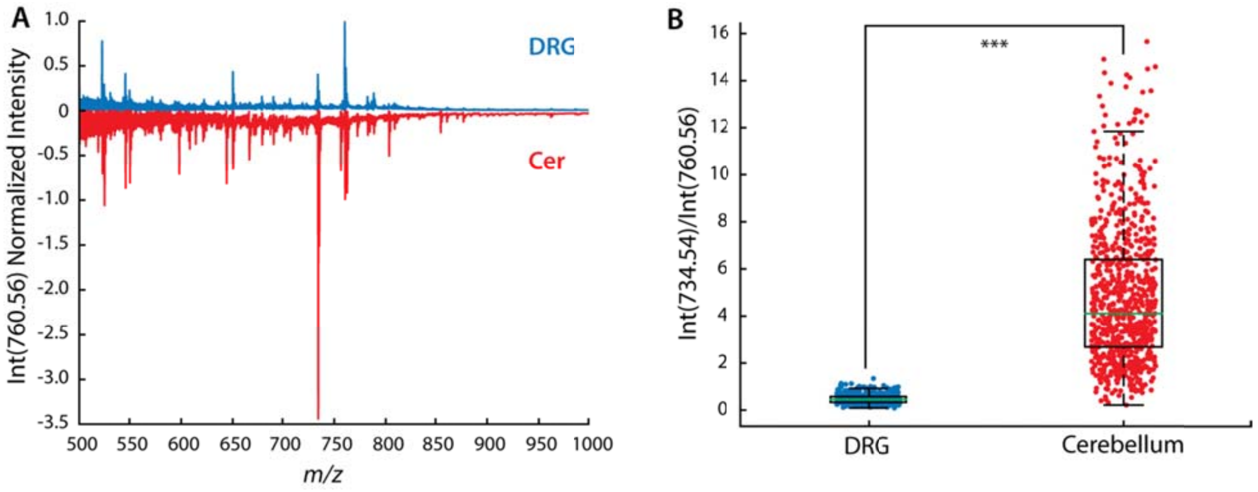


Figure 7.8. (A) Comparison of lipid profiles obtained from MALDI MS of single DRG cells (top) and cerebellum (Cer) cells (bottom) using Mix-CHCA. (B) Quantitative comparison of the ratio of two dominant lipids, PC(32:0) and PC(34:1) at 734.54 and 760.56 *m/z*, respectively, between the two cell types, *** p <0.005 by a Wilcoxon rank sum test.

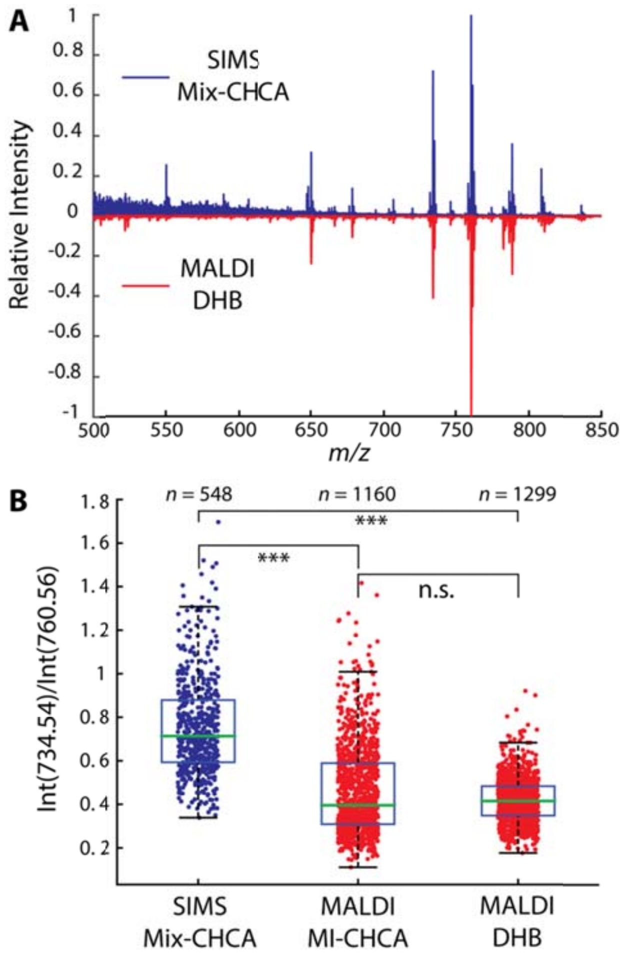


Figure 7.9. Comparison of lipid profiles detected in single DRG cells using ME-SIMS and MALDI MS. (A) The averaged mass spectra acquired from single DRG cells using SIMS (top, blue) and MALDI MS (bottom, red). (B) Quantitative comparison of the ratio of signal intensities of two dominant lipids, PC(32:0) at *m/z* 734.54 and PC(34:1) at *m/z* 760.56, between SIMS and MALDI MS measurements. *** $p < 0.005$, n.s. - not significant as determined by a Wilcoxon rank sum test. The number *n* indicates the total number of DRG cells in each data set.

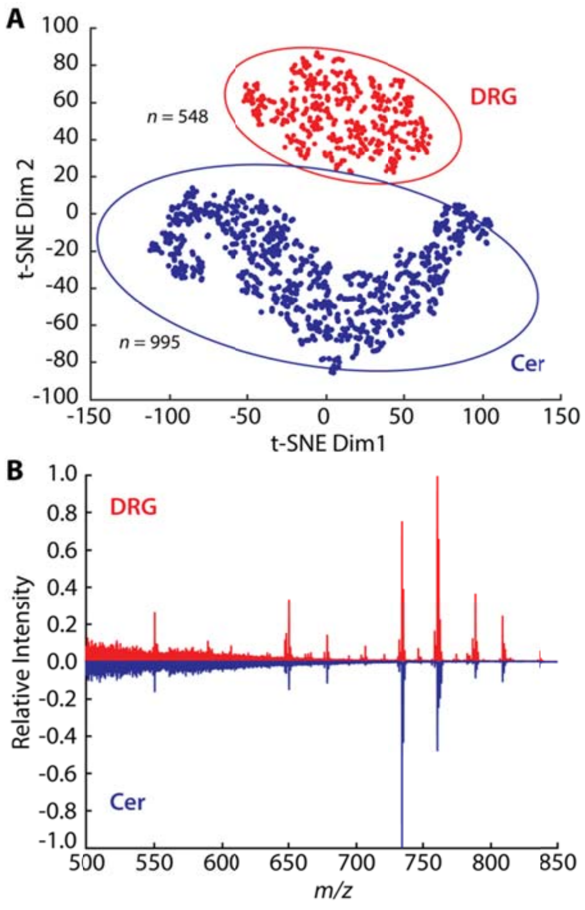


Figure 7.10. Origin-related differentiation of cell types based on single cell SIMS profiling. (A) t-SNE plot of DRG and cerebellar (Cer) cell data sets with 95% confidence ellipses. (B) The averaged mass spectra acquired from single DRG (red) and cerebellar cells (blue).

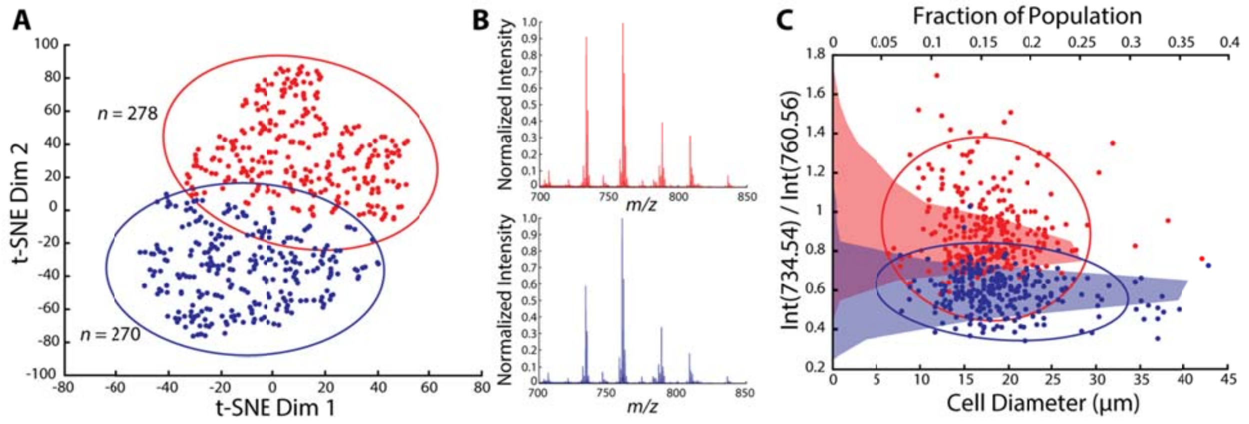


Figure 7.11. Identification of chemically distinct subpopulations of DRG neurons. (A) *k*-means clusters based on t-SNE distribution of data sets acquired from populations of single DRG cells. (B) Representative averaged spectra for each cluster. (C) Plot of ratios of intensities of *m/z* 734.54 and *m/z* 760.56 signals as a function of cell diameter. Cell diameters are determined after glycerol stabilization and drying, and therefore, are smaller than expected *in vivo*. Points on the plot are colored in correspondence with the *k*-means clusters in panel A. The averaged normalized intensities of PC(32:0) signals are 0.91 and 0.59 for cluster 1 (red) and cluster 2 (blue), respectively. A horizontal histogram of the relative population of each cluster based on the relative signal intensity ratio between PC(32:0) and PC(34:1) is shown as an area filled with the corresponding color. The 95% confidence ellipses are shown in scatter plots in A and C.

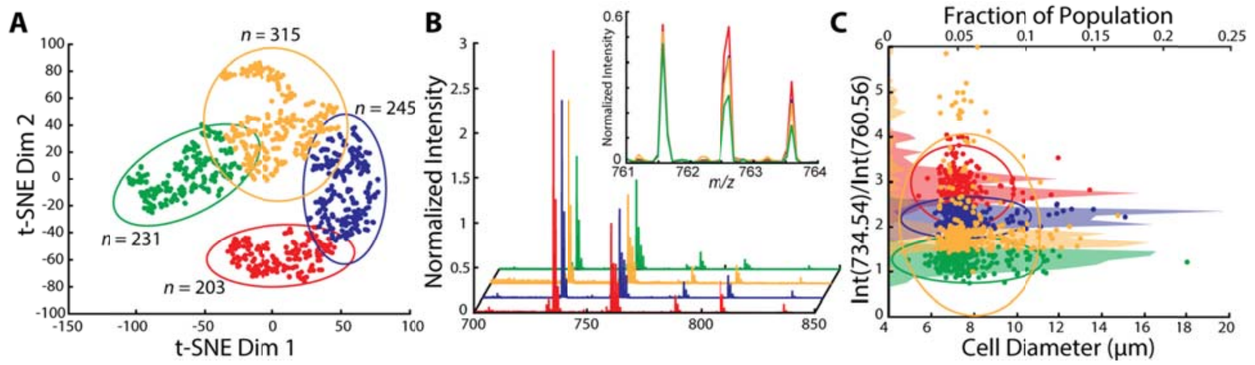
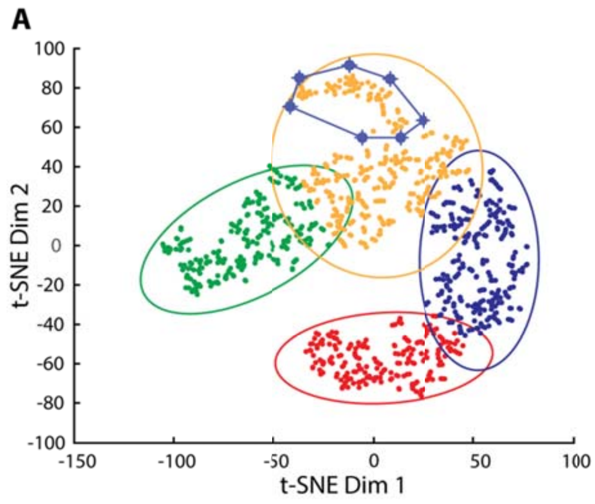


Figure 7.12. Multivariate analysis of data acquired from populations of single cerebellar cells. (A) *k*-means clustering of t-SNE distributions. (B) Representative averaged mass spectra of each cluster. The averaged normalized intensities of PC(32:0) signals are 1.3, 2.0, 2.2 and 2.9 for cluster 1 (green), cluster 2 (yellow), cluster 3 (blue), and cluster 4 (red), respectively. The inset shows the relative intensity of *m/z* 762.56 for each cluster. (C) Plot of ratios of intensities of *m/z* 734.54 and *m/z* 760.56 signals (Y axis) as a function of cell diameter (cell diameters are determined after treatment with glycerol and drying, and therefore, are smaller than expected *in vivo*). A horizontal histogram of the relative population of each cluster based on the relative signal intensity ratio between PC(32:0) and PC(34:1) is shown as an area filled with the corresponding color. The 95% confidence ellipses are shown.



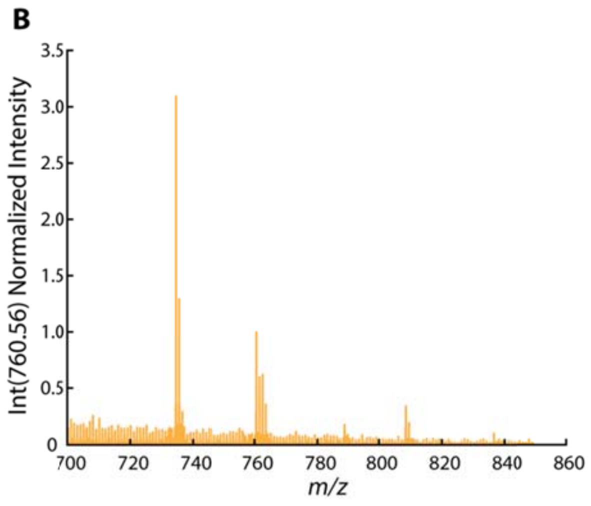


Figure 7.13. (A) Multivariate analysis of single cerebellum cells including *k*-mean clustering reproduced from Figure 7.12. A subsection of the yellow cluster is annotated. (B) The averaged mass spectrum of single cerebellum cells in the subsection highlighted in (A) showing lower S/N.

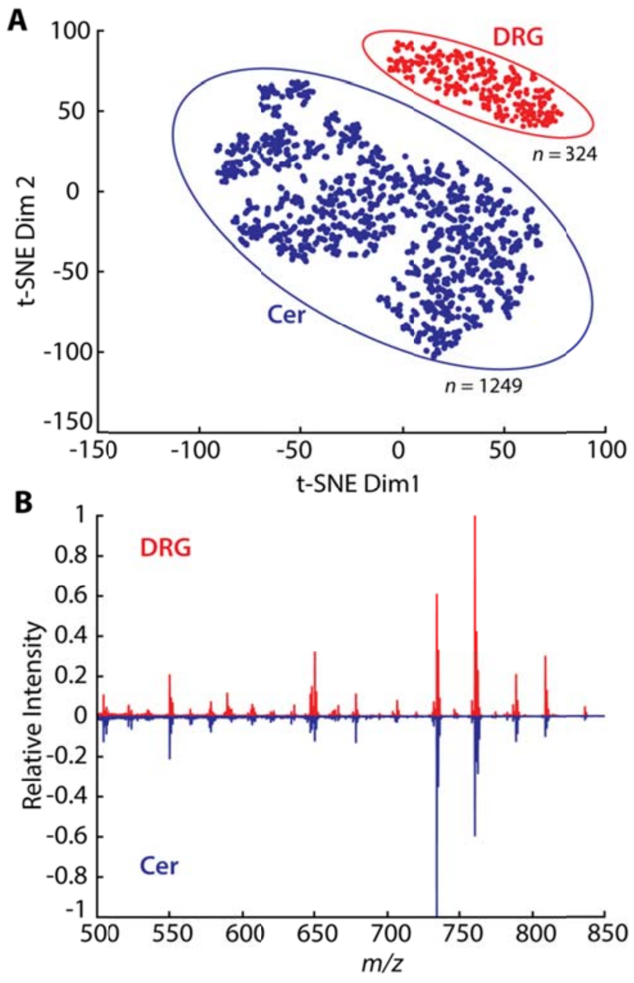


Figure 7.14. Differentiation of two neuronal cell types based on single cell SIMS profiling. (A) t-SNE plot of intact lipids from DRG and cerebellar (Cer) cells with 95% confidence ellipses. (B) The averaged SIMS spectra of single DRG (red) and cerebellar cells (blue).

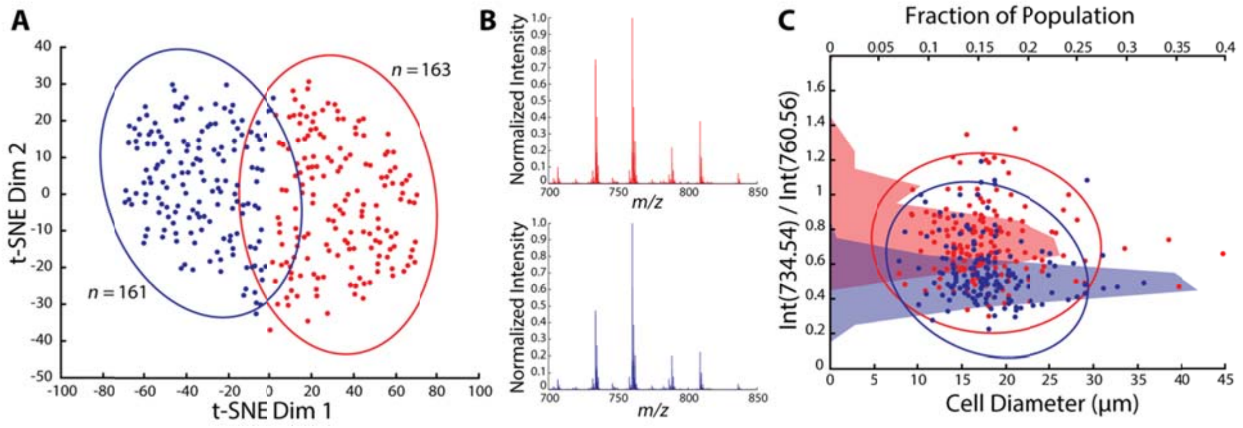


Figure 7.15. Identification of subpopulations within single DRG neurons. (A) *k*-means clusters based on t-SNE distribution of single DRG cells. (B) Averaged spectra of each cluster. The averaged normalized intensities of PC(32:0) are 0.80 and 0.49 cluster 1 (red) and cluster 2 (blue), respectively. (C) Plot of Int(734.56)/Int(760.56) as a function of cell diameter colored based on the *k*-means clusters in panel A. Overlay: a histogram of the relative intensities for each cluster. The 95% confidence ellipses are shown in A and C.

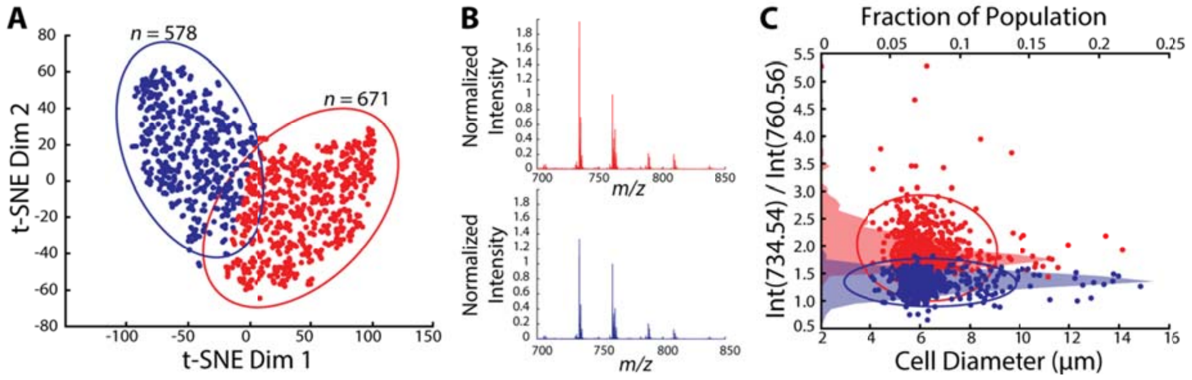


Figure 7.16. Multivariate analysis of single cerebellum cells. (A) *k*-means clustering of *t*-SNE distributions. (B) Averaged spectra of each cluster. The averaged normalized intensities of PC(32:0) are 2.0 and 1.3 for cluster 1 (red) and cluster 2 (blue), respectively. (C) Plot of cell diameter versus Int(734.54)/Int(760.56) showing the relative distribution of cells within each cluster in terms of size. Overlay: histograms of the lipid ratio for each cluster.

Table 7.1. Tentative assignments of lipids observed in SIMS single cell profiling of DRG.

Species	$[M+H]^+$ (Da)	Observed
PC(40:5)	836.54	836.60
PC(38:1)	816.64	816.60
PC(38:5)	808.60	808.60
PC(38:6)	806.56	806.54
PC(36:1)	788.61	788.60
PC(36:2)	786.64	786.60
PC(36:3)	784.58	784.56
PC(36:4)	782.56	782.56
PC(35:1)	774.60	774.58
PC(35:2)	772.58	772.56
PC(34:0)	762.60	762.60
PC(34:1)	760.59	760.56
PC(34:2)	758.56	758.56
PC(33:0)	748.58	748.60
PC(33:1)/PC(O-34:1)	746.56	746.60
PC(32:0)	734.57	734.54
PC(32:1)	732.55	732.50
PC(26:0)/PC(O-28:0)	650.47	650.40
SM(18:0)	731.60	731.60
PC(31:0)	720.55	720.50
PC(30:0)	706.53	706.50
PC(32:0) Frag	478.32	478.30
LysoPC(16:0)/PC(16:0)	496.33	496.40
PC(34:1) Frag/LysoPE(20:3)	504.40	504.30
PC(O-18:1)/PC(18:0)/LysoPC(18:1)	522.34	522.30
PC(O-20:1)/PC(19:1)/	550.35	550.30

References

(1) Navin, N. E. *Genome Research* **2015**, *25*, 1499.

(2) Schwartzman, O.; Tanay, A. *Nature Reviews: Genetics* **2015**, *16*, 716.

(3) Gawad, C.; Koh, W.; Quake, S. R. *Nature Reviews: Genetics* **2016**, *17*, 175.

(4) Rubakhin, S. S.; Romanova, E. V.; Nemes, P.; Sweedler, J. V. *Nature Methods* **2011**, *8*, S20.

(5) Ong, T. H.; Tillmaand, E. G.; Makurath, M.; Rubakhin, S. S.; Sweedler, J. V. *Biochimica et Biophysica Acta (BBA) - Bioenergetics* **2015**, *1854*, 732.

(6) Li, L.; Garden, R. W.; Sweedler, J. V. *Trends in Biotechnology* **2000**, *18*, 151.

(7) Zenobi, R. *Science* **2013**, *342*, 1243259.

(8) Svatos, A. *Analytical Chemistry* **2011**, *83*, 5037.

(9) Gode, D.; Volmer, D. A. *Analyst* **2013**, *138*, 1289.

(10) Kolitz, S. E.; Lauffenburger, D. A. *Biochemistry* **2012**, *51*, 7433.

(11) Patel, A. P.; Tirosh, I.; Trombetta, J. J.; Shalek, A. K.; Gillespie, S. M.; Wakimoto, H.; Cahill, D. P.; Nahed, B. V.; Curry, W. T.; Martuza, R. L.; Louis, D. N.; Rozenblatt-Rosen, O.; Suva, M. L.; Regev, A.; Bernstein, B. E. *Science* **2014**, *344*, 1396.

(12) Heath, J. R.; Ribas, A.; Mischel, P. S. *Nature Reviews: Drug Discovery* **2016**, *15*, 204.

(13) Chen, X.; Love, J. C.; Navin, N. E.; Pachter, L.; Stubbington, M. J. T.; Svensson, V.; Sweedler, J. V.; Teichmann, S. A. *Nature Biotechnology* **2016**, *34*, 1111.

(14) Lanni, E. J.; Rubakhin, S. S.; Sweedler, J. V. *J Proteomics* **2012**, *75*, 5036.

(15) Fujii, T.; Matsuda, S.; Tejedor, M. L.; Esaki, T.; Sakane, I.; Mizuno, H.; Tsuyama, N.; Masujima, T. *Nature Protocols* **2015**, *10*, 1445.

(16) Si, T.; Li, B.; Zhang, K.; Xu, Y.; Zhao, H.; Sweedler, J. V. *Journal of Proteome Research* **2016**, *15*, 1955.

(17) Amantonico, A.; Urban, P. L.; Fagerer, S. R.; Balabin, R. M.; Zenobi, R. *Analytical Chemistry* **2010**, *82*, 7394.

(18) Fernandez, R.; Carriel, V.; Lage, S.; Garate, J.; Diez-Garcia, J.; Ochoa, B.; Castro, B.; Alaminos, M.; Fernandez, J. A. *ACS Chemical Neuroscience* **2016**, *7*, 624.

(19) Rompp, A.; Spengler, B. *Histochemistry and Cell Biology* **2013**, *139*, 759.

(20) Rompp, A.; Guenther, S.; Takats, Z.; Spengler, B. *Analytical and Bioanalytical Chemistry* **2011**, *401*, 65.

(21) Soltwisch, J.; Kettling, H.; Vens-Cappell, S.; Wiegelmann, M.; Muthing, J.; Dreisewerd, K. *Science* **2015**, *348*, 211.

(22) Monroe, E. B.; Jurchen, J. C.; Lee, J.; Rubakhin, S. S.; Sweedler, J. V. *Journal of the American Chemical Society* **2005**, *127*, 12152.

(23) Song, Y.; Nelp, M. T.; Bandarian, V.; Wysocki, V. H. *ACS Cent. Sci.* **2015**, *1*, 477.

(24) Shrestha, B.; Vertes, A. *Analytical Chemistry* **2014**, *86*, 4308.

(25) Hofmann, J.; Hahm, H. S.; Seeberger, P. H.; Pagel, K. *Nature* **2015**, *526*, 241.

(26) Bernstein, S. L.; Dupuis, N. F.; Lazo, N. D.; Wyttenbach, T.; Condron, M. M.; Bitan, G.; Teplow, D. B.; Shea, J. E.; Ruotolo, B. T.; Robinson, C. V.; Bowers, M. T. *Nature Chemistry* **2009**, *1*, 326.

(27) Kopysov, V.; Makarov, A.; Boyarkin, O. V. *Journal of Physical Chemistry Letters* **2016**, *7*, 1067.

(28) Gonzalez Florez, A. I.; Mucha, E.; Ahn, D. S.; Gewinner, S.; Schollkopf, W.; Pagel, K.; von Helden, G. *Angewandte Chemie, International Edition in English* **2016**, *55*, 3295.

(29) Cong, X.; Liu, Y.; Liu, W.; Liang, X.; Russell, D. H.; Laganowsky, A. *Journal of the American Chemical Society* **2016**, *138*, 4346.

(30) Gault, J.; Donlan, J. A.; Liko, I.; Hopper, J. T.; Gupta, K.; Housden, N. G.; Struwe, W. B.; Marty, M. T.; Mize, T.; Bechara, C.; Zhu, Y.; Wu, B.; Kleanthous, C.; Belov, M.; Damoc, E.; Makarov, A.; Robinson, C. V. *Nature Methods* **2016**, *13*, 333.

(31) Boxer, S. G.; Kraft, M. L.; Weber, P. K. *Annu. Rev. Biophys.* **2009**, *38*, 53.

(32) Jansson, E. T.; Comi, T. J.; Rubakhin, S. S.; Sweedler, J. V. *ACS Chemical Biology* **2016**, *11*, 2588.

(33) Ong, T. H.; Kissick, D. J.; Jansson, E. T.; Comi, T. J.; Romanova, E. V.; Rubakhin, S. S.; Sweedler, J. V. *Analytical Chemistry* **2015**, *87*, 7036.

(34) Passarelli, M. K.; Newman, C. F.; Marshall, P. S.; West, A.; Gilmore, I. S.; Bunch, J.; Alexander, M. R.; Dollery, C. T. *Analytical Chemistry* **2015**, *87*, 6696.

(35) Tucker, K. R.; Li, Z.; Rubakhin, S. S.; Sweedler, J. V. *Journal of the American Society for Mass Spectrometry* **2012**, *23*, 1931.

(36) Dunham, S. J.; Comi, T. J.; Ko, K.; Li, B.; Baig, N. F.; Morales-Soto, N.; Shrout, J. D.; Bohn, P. W.; Sweedler, J. V. *Biointerphases* **2016**, *11*, 02A325.

(37) Wehbe, N.; Mouhib, T.; Prabhakaran, A.; Bertrand, P.; Delcorte, A. *Journal of the American Society for Mass Spectrometry* **2009**, *20*, 2294.

(38) Dertinger, J. J.; Walker, A. V. Journal of the American Society for Mass Spectrometry **2013**, 24, 348.

(39) Fitzgerald, J. J.; Kunnath, P.; Walker, A. V. *Analytical Chemistry* **2010**, *82*, 4413.

(40) Bundaleski, N.; Caporali, S.; Chenakin, S. P.; Moutinho, A. M. C.; Teodoro, O. M. N. D.; Tolstogouzov, A. *Int. J. Mass Spectrom.* **2013**, *353*, 19.

(41) Tariq, M.; Freire, M. G.; Saramago, B.; Coutinho, J. A.; Lopes, J. N.; Rebelo, L. P. *Chemical Society Reviews* **2012**, *41*, 829.

(42) Zimmerman, T. A.; Rubakhin, S. S.; Sweedler, J. V. *Journal of the American Society for Mass Spectrometry* **2011**, *22*, 828.

(43) Schober, Y.; Guenther, S.; Spengler, B.; Rompp, A. *Analytical Chemistry* **2012**, *84*, 6293.

(44) Hua, X.; Szymanski, C.; Wang, Z.; Zhou, Y.; Ma, X.; Yu, J.; Evans, J.; Orr, G.; Liu, S.; Zhu, Z.; Yu, X. Y. *Integrative Biology: Quantitative Biosciences from Nano to Macro* **2016**, *8*, 635.

(45) Parry, S.; Winograd, N. *Analytical Chemistry* **2005**, *77*, 7950.

(46) Lanni, E. J.; Dunham, S. J.; Nemes, P.; Rubakhin, S. S.; Sweedler, J. V. *Journal of the American Society for Mass Spectrometry* **2014**, *25*, 1897.

(47) Sjovall, P.; Lausmaa, J.; Johansson, B. *Analytical Chemistry* **2004**, *76*, 4271.

(48) Phan, N. T.; Fletcher, J. S.; Ewing, A. G. *Analytical Chemistry* **2015**, *87*, 4063.

(49) Passarelli, M. K.; Ewing, A. G.; Winograd, N. *Analytical Chemistry* **2013**, *85*, 2231.

(50) Angerer, T. B.; Pour, M. D.; Malmberg, P.; Fletcher, J. S. *Analytical Chemistry* **2015**, *87*, 4305.

(51) Sapunar, D.; Kostic, S.; Banozic, A.; Puljak, L. *Journal of Pain Research* **2012**, *5*, 31.

(52) Kishi, M.; Tanabe, J.; Schmelzer, J. D.; Low, P. A. *Diabetes* **2002**, *51*, 819.

(53) Villiere, V.; McLachlan, E. M. *Journal of Neurophysiology* **1996**, *76*, 1924.

(54) Usoskin, D.; Furlan, A.; Islam, S.; Abdo, H.; Lonnerberg, P.; Lou, D.; Hjerling-Leffler, J.; Haeggstrom, J.; Kharchenko, O.; Kharchenko, P. V.; Linnarsson, S.; Ernfors, P. *Nature Neuroscience* **2015**, *18*, 145.

(55) Werz, M. A.; Macdonald, R. L. *Nature* **1982**, *299*, 730.

(56) Boateng, E. K.; Novejarque, A.; Pheby, T.; Rice, A. S.; Huang, W. *European Journal of Pain (London, England)* **2015**, *19*, 236.

(57) Wang, S. S.; Kloth, A. D.; Badura, A. *Neuron* **2014**, *83*, 518.

(58) Kilkenny, C.; Browne, W. J.; Cuthill, I. C.; Emerson, M.; Altman, D. G. *PLoS Biology* **2010**, *8*, e1000412.

(59) Berman, E. S.; Fortson, S. L.; Checchi, K. D.; Wu, L.; Felton, J. S.; Wu, K. J.; Kulp, K. S. *Journal of the American Society for Mass Spectrometry* **2008**, *19*, 1230.

(60) Chambers, M. C.; Maclean, B.; Burke, R.; Amodei, D.; Ruderman, D. L.; Neumann, S.; Gatto, L.; Fischer, B.; Pratt, B.; Egertson, J.; Hoff, K.; Kessner, D.; Tasman, N.; Shulman, N.; Frewen, B.; Baker, T. A.; Brusniak, M. Y.; Paulse, C.; Creasy, D.; Flashner, L.; Kani, K.; Moulding, C.; Seymour, S. L.; Nuwaysir, L. M.; Lefebvre, B.; Kuhlmann, F.; Roark, J.; Rainer, P.; Detlev, S.; Hemenway, T.; Huhmer, A.; Langridge, J.; Connolly, B.; Chadick, T.; Holly, K.; Eckels, J.; Deutsch, E. W.; Moritz, R. L.; Katz, J. E.; Agus, D. B.; MacCoss, M.; Tabb, D. L.; Mallick, P. *Nature Biotechnology* **2012**, *30*, 918.

(61) Maaten, L. J. P. v. d.; Hinto, G. E. *J. Mach. Learn. Res.* **2008**, *9*, 2579.

(62) Dertinger, J. J.; Walker, A. V. *Journal of the American Society for Mass Spectrometry* **2013**, *24*, 1288.

(63) Calderon, R. O.; Attema, B.; DeVries, G. H. *Journal of Neurochemistry* **1995**, *64*, 424.

(64) Barabas, M. E.; Mattson, E. C.; Aboualizadeh, E.; Hirschmugl, C. J.; Stucky, C. L. *Journal of Biological Chemistry* **2014**, *289*, 34241.

(65) Gnanasekaran, A.; Sundukova, M.; van den Maagdenberg, A. M.; Fabbretti, E.; Nistri, A. *Molecular Pain* **2011**, *7*, 77.

(66) Zhang, Y. H.; Khanna, R.; Nicol, G. D. *Neuroscience* **2013**, *248*, 562.

(67) Patti, G. J.; Yanes, O.; Shriver, L. P.; Courade, J. P.; Tautenhahn, R.; Manchester, M.; Siuzdak, G. *Nature Chemical Biology* **2012**, *8*, 232.

(68) Jackson, S. N.; Wang, H. Y.; Woods, A. S. *Analytical Chemistry* **2005**, *77*, 4523.

CHAPTER 8

MALDI-MS Guided Liquid Microjunction Extraction for CE-MS Analysis of Single Murine Pancreatic Islet Cells

Notes and Acknowledgements

This chapter is adapted from a manuscript in preparation for publication as of March 29, 2017, coauthored by M. A. Makurath, M. C. Philip, S. S. Rubakhin, and J. V. Sweedler. T. J. Comi performed MALDI-MS, data analysis, wrote control software, isolated islets, and assisted with extractions. M. C. Philip performed single cell CE-MS, single cell extractions, and assisted with data analysis. M. A. Makurath designed the initial extraction system and performed radiography experiments. Dr. Ta-Hsuan Ong is acknowledged for his help with initial testing and Dr. Meng Qi for assistance with data analysis. We also thank the UIUC School of Chemical Sciences machine shop for their expertise in designing and machining components for the vacuum chamber and stage. Support from the National Institutes of Health, Award Number P30 DA018310 from the National Institute on Drug Abuse and from the National Institute of Mental Health Award Number 1U01 MH109062 are also acknowledged. T. J. Comi acknowledges support from the National Science Foundation Graduate Research Fellowship Program, the Springborn Fellowship and the Training Program at Chemistry-Interface with Biology (T32 GM070421).

Introduction

Assessing heterogeneity in single cells from biological tissues continues to be a challenging task in the field of physiology.[1-4] Frequently, bulk homogenates mask unique features of individual cells by averaging the population content.[5] While a biological organ or tissue needs many types of distinct cells to function properly, a malfunction can manifest from subpopulations as small as a single cell.[6,7] Furthermore, even cells indistinguishable by histology or fluorescent labeling may possess unique intracellular chemistry.[8-10] Detecting and understanding the heterogeneity between different cells will lead to innovative treatments and biomedical diagnostics.

An important example of single cell heterogeneity exists in pancreatic islets of Langerhans. The cellular composition of islets are traditionally categorized by the main peptide hormones they release into the endocrine duct, typically detected by immunohistochemistry.[11] Recently, further stratification of cell subtypes became possible with the advent of new analytical technologies. Flow cytometry and transcriptomics of insulin-secreting β cells identified up to four different subtypes in humans.[12] Using optically-guided MALDI-MS, differential peptide processing was detected in gamma cells derived from dorsal and ventral regions of the rat pancreas.[13] The development of new methods has revealed the inadequacies of purely histological classifications of islet cells.

Mass spectrometry is among the most popular analytical method for non-targeted single cell analysis.[4] The recent development of single cell MS analysis was made possible due to advances in sensitivity, mass resolution, and sample throughput of mass analyzers. Specifically, the analysis of dispersed cells enables high throughput MS analysis.[14] MALDI-MS is well suited for single cell analysis of a wide range of biological molecules, including for investigating the lipidome and peptidome.[15] The analysis is label-free and consumes a fraction of surface analytes.[16] By locating cells with optical microscopy, the analysis can proceed at acquisition rates of approximately 1 Hz.[17] Most peptides have sufficiently high molecular mass that MALDI matrix interference is minimal, but many metabolites are obscured, limiting investigations of metabolites and peptides from the same cell. Another pertinent method for single cell analysis is capillary electrophoresis (CE)-MS, which can quantitatively identify metabolites at sensitivity relevant to individual cells.[18-20] Sample preparation for CE-MS typically requires the extraction of an entire cell, which has been isolated manually, or through microfluidics.[21] In contrast to MALDI-MS, CE-MS has relatively low throughput, limited to a few cells an hour, which eliminates the possibility of an exhaustive cell-by-cell analysis of even modestly sized populations.

Coupling preliminary classification of cells with MALDI-MS to CE-MS could facilitate targeting of rare and representative cells from a large population. The preselection would allow CE-MS to efficiently analyze large populations by selecting the most informative individuals. Previous attempts to combine MALDI-MS and CE-MS have utilized microfluidic[22,23] or hydrodynamic[24,25] interfaces. Although the same sample was analyzed with both instruments, the methods had relatively low throughput and required excessive manual sample handling. It is critical that the interface method collects small sample volumes with high collection efficiency to reveal chemical heterogeneity from large populations of cells.

Here, we present a semi-automated, microscopy-guided liquid microjunction probe for extraction of single cells targeted by their MALDI-MS profiles. While MALDI-MS is not required for performing liquid extraction, it can complement the microscopy information by providing label-free classification of large populations. The probe has three axes of linear freedom controlled with lab-built graphical user interface to perform microscopy guided cell targeting. The software is an extension of previous microscopy-guided work with MALDI-MS[13] and secondary ion MS.[26] Each extraction takes 1 minute per cell and can locate targets with an accuracy of 42.8 ± 2.3 µm over an area of ~ 12 cm^2 (approximately 2/3 of a standard microscope slide). The probe extracts all visible MALDI matrix in an elliptical area with a major diameter of

422 ± 21 μm and minor diameter of 335 ± 27 μm. Radiography of a standard peptide demonstrates removal of 90.6 ± 0.6% of surface compounds.

The methodology allows selective extraction based on peptide content and enables investigations of specific cell types. Here, small molecules of single cell extracts from individual rat pancreatic α cells were profiled using CE-MS. By interfacing two powerful analytical tools for small-volume samples, the combined data from CE-MS and MALDI-MS successfully classified and analyzed 6 α and 5 β cells. Each cell was identified as a standard histological class by the detection of glucagon and insulin, respectively, by MALDI-MS. Small molecules detected with CE-MS include 18 proteinogenic amino acids as well as dopamine. While the presence of required enzymes for dopamine synthesis has implicated the endogenous presence of dopamine in β cells,[27,28] it has not been directly detected at the single cell level in both α and β cells. Based on the cell-to-cell distance, liquid microjunction extraction accuracy, and removal efficiency, we conclude the extracts are representative of single cell contents and are suitable for CE-MS with a variety of sample systems.

Materials and Methods

Chemicals

All chemicals were purchased from Sigma Aldrich (St. Louis, MO) and used without further purification.

Isolation of Islet of Langerhans and Single Cell Preparation

A four-month old, male Sprague-Dawley outbred rat (*Rattus norvegicus*) was housed on a 12-h light cycle and fed ad libitum. Animal euthanasia was performed in accordance with the appropriate institutional animal care guidelines (the Illinois Institutional Animal Care and Use Committee), and in full compliance with federal guidelines for the humane care and treatment of animals. Islets of Langerhans were manually isolated from digested pancreas as previously reported.[13] Briefly, pancreata are injected through the bile duct with 2 mL of 1.4 mg/mL collagenase P in modified Gey's balanced salt solution (mGBSS) supplemented with 5 mM glucose and 1% (w/v) bovine serum albumin (BSA). mGBSS contains 1.5 mM CaCl2, 4.9 mM KCl, 0.2 mM KH2PO4, 11 mM MgCl2, 0.3 mM MgSO4, 138 mM NaCl, 27.7 mM NaHCO3, 0.8 mM NaH2PO4, and 25 mM HEPES dissolved in Milli-Q water (Millipore, Billerica, MA), with the pH adjusted to 7.2. The pancreas was then surgically dissected and placed into 8 mL of collagenase P solution. Solutions were incubated in a recirculating water bath for 20-30 minutes at 37 °C to dissociate bulk tissue. Excess collagenase P was washed from the resulting tissue with mGBSS containing glucose and BSA, and centrifuged for 3 minutes at 300*g*. The resulting tissue pellet was dispersed into mGBSS and islets were manually isolated with a micropipette. Single islets were incubated in 20 µL of mGBSS with glucose and BSA supplemented further with 0.1 mg/mL Hoechst 33342 and 40% (v/v) glycerol to stain cell nuclei and stabilize their metabolite content.[29] After 30 minutes, single cells were dissociated onto clean ITO-coated glass slides by gentle trituration in the staining solution and allowed to adhere to the slide overnight. Prior to imaging, excess glycerol was aspirated and the surface was rinsed with 150 mM ammonium acetate (pH 10).

Optically-Guided Single Cell Profiling

The first step in the experimental workflow outlined in Scheme 8.1 is to locate cells by optical microscopy. ITO-coated glass slides were prepared for optically-guided single cell profiling by marking the perimeter of dissociated cells with ~20 fiducial marks. Each mark consisted of an etched 'x' which remained visible through MALDI-MS and liquid microjunction extraction. The locations of fiducials and cells were determined by whole-slide brightfield and fluorescence microscopy on an Axio Imager M2 (Carl Zeiss, Jena, Germany). Images were acquired with a 10× objective and tiled to cover the entire region of interest. Florescence imaging of Hoechst 33342 utilized an X-CITE 120 mercury lamp (Lumen Dynamics, Mississauga, Canada) and a 31000v2 DAPI filter set.

Whole slide images were utilized for optically-guided single cell profiling using lab-built software which has been modified to control the liquid microjunction extraction system. The pixel locations of each fiducial are correlated to their physical position in the mass spectrometer. A point-based similarity registration is then utilized to map cell locations on the image to their corresponding physical location.

After optical imaging, samples were coated with MALDI matrix using an artistic airbrush to apply 50 mg/mL 2,5-dihydroxybenzoic acid (DHB) in a 1:1 (v/v) ethanol:water with 0.1% trifluoroacetic acid (TFA) nebulized with 40 psi nitrogen. Coating thickness was assessed optically during matrix application with typical thicknesses of 0.2-0.4 mg/cm^2. Samples were stored at room temperature in a nitrogen dry box until analyzed.

MALDI-MS

Pancreatic cell populations were rapidly profiled with MALDI-MS to stratify the population into traditional histological classes. Specifically, α and β cells were identified based on the detection of glucagon (monoisotopic *m/z* 3481.6) or insulin-1 C peptide (*m/z* 3259.8). To prevent contamination from adjacent cells during follow-up extraction, all cell coordinates were first passed through a 300 µm distance filter. From a single islet dispersed on an ITO-coated glass slide, approximately 200-400 pancreatic cells satisfied the collection criteria.

Mass spectra were acquired on a Bruker ultrafleXtreme MALDI TOF/TOF mass spectrometer with a frequency tripled Nd:YAG solid state laser. Each cell was profiled with

1000 shots using a 1 kHz laser repetition rate with the “Ultra” spot size (~100 μm). The resulting spectra were read into MATLAB 8.6.0 with the readbrukermaldi function (https://github.com/AlexHenderson/readbrukermaldi). Mass windows corresponding to the peptide hormones of interest were extracted and intensities were plotted as shown in Figure 8.1. Cells were classified based on their spectral profiles as α or β using signal intensities at *m/z* 3483.9 and 3259.8 as classifiers. For each mass channel, a threshold value was manually determined to identify cell types with high confidence. Due to stringent filter values, approximately 100 cells were successfully classified by this approach for each islet. Classified cells were then examined to ensure no adjacent cells would contaminate the extraction.

Liquid Microjunction Extraction Probe System

As shown in Figure 8.2, the liquid microjunction extraction system consists of a lab-built, concentric capillary probe coupled to a 3-axis linear actuator positioning system. The single-cell collection setup was designed to transfer cell metabolites from an ITO-coated glass slide into a 0.2 μL microcentrifuge tube. The basic operating principle is similar to a liquid microjunction surface sampling probe except the solution is aspirated by vacuum pressure instead of an electrospray. The diameters of the probe capillaries were selected to be larger than the diameter of individual pancreatic cells to ensure complete extraction, prevent clogging and accommodate the stage accuracy. The sizes of the inner and outer capillaries were 100 μm/170 μm and 250 μm/350 μm (Polymicro Technologies/Molex), and the diameter of pancreatic cells is ~10-15 μm.[30] Sample carryover may result in the detection of unwanted metabolites, therefore, ~5 mm of polyimide coating is thermally removed at the end of each capillary.[31] Following each extraction, the probe was immersed in extraction solution to thoroughly wash out the interior.

The extraction solution consisted of 1:1 methanol:water with 0.5% acetic acid (v/v) which was previously shown to facilitate metabolite extraction and detection with the CE-MS.[32] As shown in Figure 8.3, a small meniscus forms at the probe tip during operation. Eight collections can be performed sequentially without interruption, currently limited by the vacuum chamber capacity. Extraction liquid is delivered at 1.5 μL/min with a PHD 2000 syringe pump (Harvard Apparatus, Holliston, MA) and aspirated with 7-10 mmHg of vacuum, supplied with a diaphragm vacuum/pressure pump (Cole-Parmer, Vernon Hills, IL). The liquid microjunction is positioned with three linear stages (Zaber Technologies, Vancouver, British Colombia) controlled with in-house written software. First, the user moves the *x,y*-translation stage away from the cell deposition and lowers the probe to the surface. The software records the *z*-axis position at the slide surface to enable automatic extraction. The probe position is monitored in real-time with a digital video camera (Sony DFW-X700). Next, coordinates from the whole-slide image and linear actuator positions are correlated with a point-based similarity registration based on more than 18 etched fiducial marks. Once all fiducials have been located on the sample surface, cell positions are loaded into the software. Clicking on a cell position on the image moves the *x,y*-translation stage into position for extraction. The user initiates semi-automatic extractions by moving over target cells and signaling the software with a key press. During extraction, the probe is lowered to the slide for 60s and then retracted. Alternatively, a population of cells may be sequentially extracted and pooled into a single collection vial. Following either collection scheme, the probe is returned to the home position and submerged into a reservoir of extraction solution for 90s to rinse the probe exterior and flush the inner capillary. The cell content at each coordinate travels from the MALDI target, through the inner capillary and into the microcentrifuge tube contained in the vacuum chamber. Inside the vacuum chamber, the microcentrifuge tubes are covered with a thin strip of parafilmM to prevent carryover when moving between collection vials. The inner capillary is retracted from the current collection tube, the tube carousel is indexed to the next position and the inner capillary is placed into the next collection tube without breaking vacuum. Individual samples were dried using Mi-Vac sample concentrator (SP Scientific, Warminster, PA) and stored at -20°C prior to CE-MS analysis.

Characterization of Probe Removal Efficiency

Tritiated (^{3}H) angiotensin was utilized to determine the extraction profile and removal efficiency of the extraction probe. All radioactivity experiments were performed in accordance with Illinois Radiation Protection Act under the University of Illinois at Urbana-Champaign Type A Broad Scope Radioactive Materials License issued by the Illinois Emergency Management Agency (IEMA). To quantify the extraction efficiency, five spots of ~1 μL of 1,000 pCi ^{3}H-angiotensin were deposited onto the surface of an ITO-coated glass slide and allowed to dry for 24 hr at room temperature (~ 22°C). Liquid microjunction extraction of the radioactive material was performed as described above with minor adjustments to minimize the possibility of radioactive contamination of the equipment. A movable stereomicroscope (Leica Wild M3Z) was mounted to monitor the extraction location and the position of the probe in the *z*-direction was adjusted manually. To replicate single cell extraction conditions, each ^{3}H-angiotensin spot was extracted for 60 sec. The pre- and post-extraction radioactivity was monitored with a Storage Phosphor Screen (BAS-IP TR 2025 E Tritium Screen) exposed for 6h. Developing the screen with a phosphorimager (Molecular Dynamics Phosphorimager SI) allowed for relative quantitation of the removal. Image processing was performed with custom MATLAB scripts. The fraction of material removed was determined by the background-corrected, normalized intensity at each pixel before and after extraction. The removal efficiency was estimated by fitting the 2 dimensional distribution to a general Gaussian function, as described in Table 8.1.

Determination of Target Localization Error

To ensure each extraction is from the expected cell, it is imperative to determine the target localization error. Extraction locations were tracked by the removal of MALDI matrix from a MALDI target. Image registration of fiducial markers allowed the correlation of requested target points and realized extraction positions.

A glass slide was etched with 18 fiducial marks for point-based registration, similar to typical cell extractions. An additional six etched marks were placed within these fiducials to assist with image registration as they remain visible after MALDI matrix application and extraction. Eight target locations were manually placed around each of the six, interior etched marks in pairs to assess the effect of repeated registrations. The slide was then coated with DHB and placed into the liquid extraction stage as before. Two users each performed two sets of extractions with twelve targets spread over the six etched marks. This design of experiment allowed evaluation of the user, registration, and location on the target localization error. Each target was extracted for 5 s and the probe was washed for 60 s after each set of twelve extractions.

Following extraction, the sample was imaged again to locate target etched marks and extraction locations. Extraction centers and diameters were manually annotated. A custom MATLAB script was utilized to assess the target localization error of each extraction. Regions surrounding each etched mark were cropped from the whole slide image. Several locations on each mark were utilized to overlay the pre- and post-extraction images. Target locations on the pre-extraction image were then mapped to the post-extraction image with the same coordinate transformation. The pixel distance between the target and actual positions were scaled to microns. A three-way linear ANOVA was utilized to assess the effect of each confounding variable.

CE-MS Analysis

Each cell was resuspended in 1 μL 1% formic acid in water. CE-MS was performed as reported previously using a micrOTOF mass spectrometer (Bruker Daltonics, Billerica, MA).[32] All analyses were performed in positive ion mode using a separation capillary length of 70.7 cm, a separation potential of 17 kV, and a sample and standard injection volume of ~ 15 nL. Extracted ion electropherograms were exported using custom scripts in Bruker DataAnalysis version 4.4. Compounds were identified from the electropherograms by matching the migration order and *m/z* values with standards. In MATLAB, each extracted ion electropherogram was baseline subtracted and smoothed with a 7-point moving average filter. Migration times were aligned to an arbitrary sample (α1) using a linear regression between migration times of a set of amino acids found in each sample (i.e. glycine, alanine, threonine, leucine/isoleucine, histidine, phenylalanine; Figure 8.4). To confirm the presence of dopamine, a standard mix of 10 μM glycine, alanine, threonine, leucine, histidine, and phenylalanine in 1% formic acid in water was analyzed with a 68 cm capillary at 10 kV with and without the addition of 10 μM dopamine. The resulting electropherograms were migration time-corrected and compared to the cell samples.

Results and Discussion

Extraction Accuracy, Area and Efficiency

As described above, MALDI-matrix was utilized as a tracer to monitor the extraction position to assess the target localization error of the liquid microjunction system (Figure 8.5). While the operator and target location did not significantly influence the target localization error ($p = 0.15$ and 0.06 respectively), there was a significant effect on the registration trial ($p = 0.004$). This highlights the importance of accurate determination of fiducial locations as the largest influence on target localization error. The overall target localization error was determined as 42.8 ± 2.3 µm (± SEM, n = 48; range 3.9 to 88.5 µm), which is well within the average extraction radius of 206.3 ± 1.7 µm determined from removal of DHB from the surface. Therefore, it is assumed that each extraction contained the target cell and a cell-to-cell distance filter larger than 250 µm is sufficient to ensure each extraction is free from contamination of neighboring cells.

The same experiment provided a high resolution, qualitative assessment of the extraction area. Figure 8.6 shows a montage of the extracted area. Generally, the footprint was found to be circular, though some irregularities are present, likely due to imperfect construction of the probe or the presence of glass shards from the fiducials. An average radius of 206.3±1.7 µm was found from manual measurement of each extraction spot.

The removal efficiency was investigated quantitatively with ^{3}H-angiotensin spotted onto an ITO-coated glass slide. Fitting the radiographic images to a two-dimensional, general Gaussian function (Figures 8.7 and 8.8) estimates a removal efficiency of 90.6 ± 0.6% (Table 8.1). The extraction footprint was found to be elliptical with major diameter of 422 ± 21 µm and minor diameter of 335 ± 27 µm, in agreement with optical measurements of DHB removal.

Profiles of Small Molecules

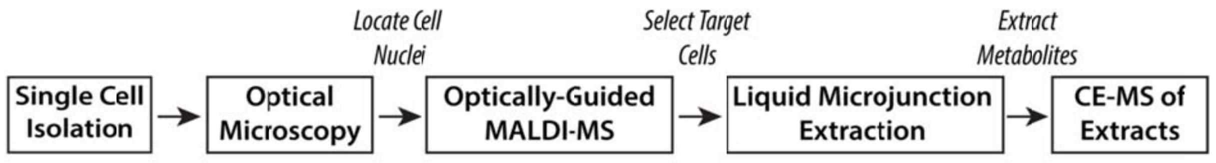
CE-MS complements MALDI-MS analyses by identifying small molecules from a single-cell. We present example extracted ion electropherograms with corresponding MALDI mass spectra in Figure 8.9. The complete collection of electropherograms is in Figure 8.10.

Detected compounds include a majority of the proteinogenic amino acids, precursor molecules, and endocrine signaling molecules. No obvious differences were found between α and β cells, though a more quantitative analysis could identify subtle heterogeneity between each population. An interesting observation was the presence of dopamine in all α and β cells (Figures 8.11 and 8.12). Endogenous dopamine has been detected in single islets via an ELISA assay,[28] but not in single cells. β cells are known to have the required enzymes for synthesis, metabolism, and storage of dopamine, such as tyrosine hydroxylase[33] and vesicular monoamine transporter type 2,[34] thus it is generally accepted that dopamine is produced in β cells.[27] Dopamine within α cells is less studied, and whether dopamine is endogenous to α cells has not yet been investigated. We report the first direct detection of dopamine in single α and β cells, illustrating the unique capabilities of such small-scale analyses.

Conclusions

We present a method to couple high throughput single cell profiling with MALDI-MS with CE-MS metabolomics. The approach leverages the low sample consumption of MALDI-MS to enable follow-up analysis by CE-MS. Combining MALDI-MS and CE-MS resulted in identification of cell types by peptide profile, detection of most amino acids and the signaling molecule dopamine, a difficult task for either technique alone. The approach demonstrates sequential analysis of single cells adhered to a surface. While CE-MS provides insight for small molecules and metabolites, MALDI-MS supplies a label-free classification method at high throughput to highlight individual cells requiring further examination.

Scheme, Figures, and Table



Scheme 8.1. Overview of MALDI-MS guided liquid microjunction extraction approach. Islets of Langerhans are isolated from a rat pancreas and dissociated onto an ITO-coated glass slide. MALDI-MS assays the hormone profile of individual cells from a large population to identify extraction targets. The liquid microjunction probe collects cells from specified locations on the ITO-coated glass slide for follow-up CE-MS analysis.

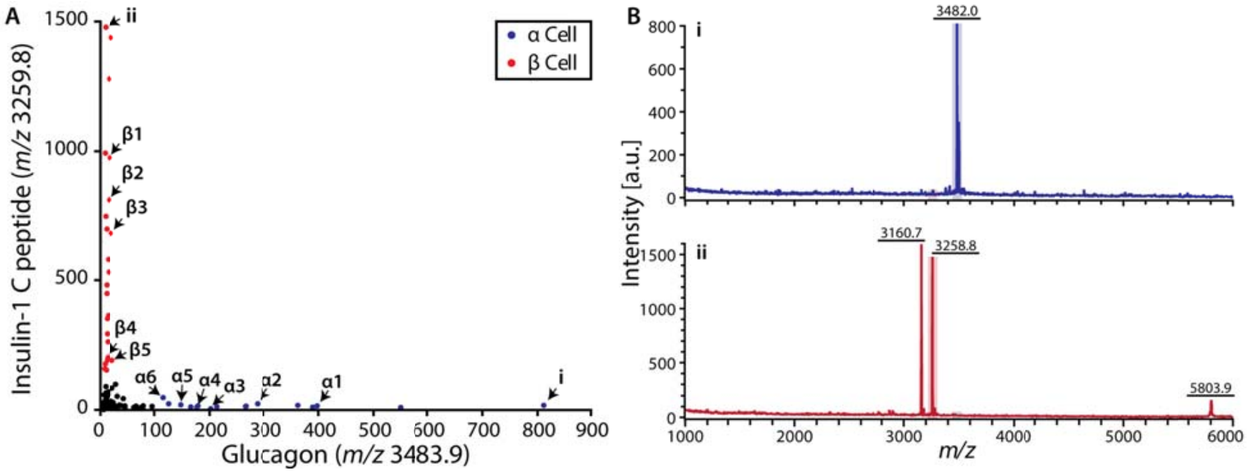


Figure 8.1. MALDI-MS classification of pancreatic cells. (A) A single dorsal islet is composed primarily of α (blue) and β cells (red), containing glucagon and insulin respectively. Classifications are based on a threshold abundance to identify cell types for follow-up CE-MS analysis. Cell identities in remaining figures correspond to labeled α and β cells. (B) Spectra of single pancreatic cells identified in panel A. Mass windows for each peptide are highlighted in the spectra.

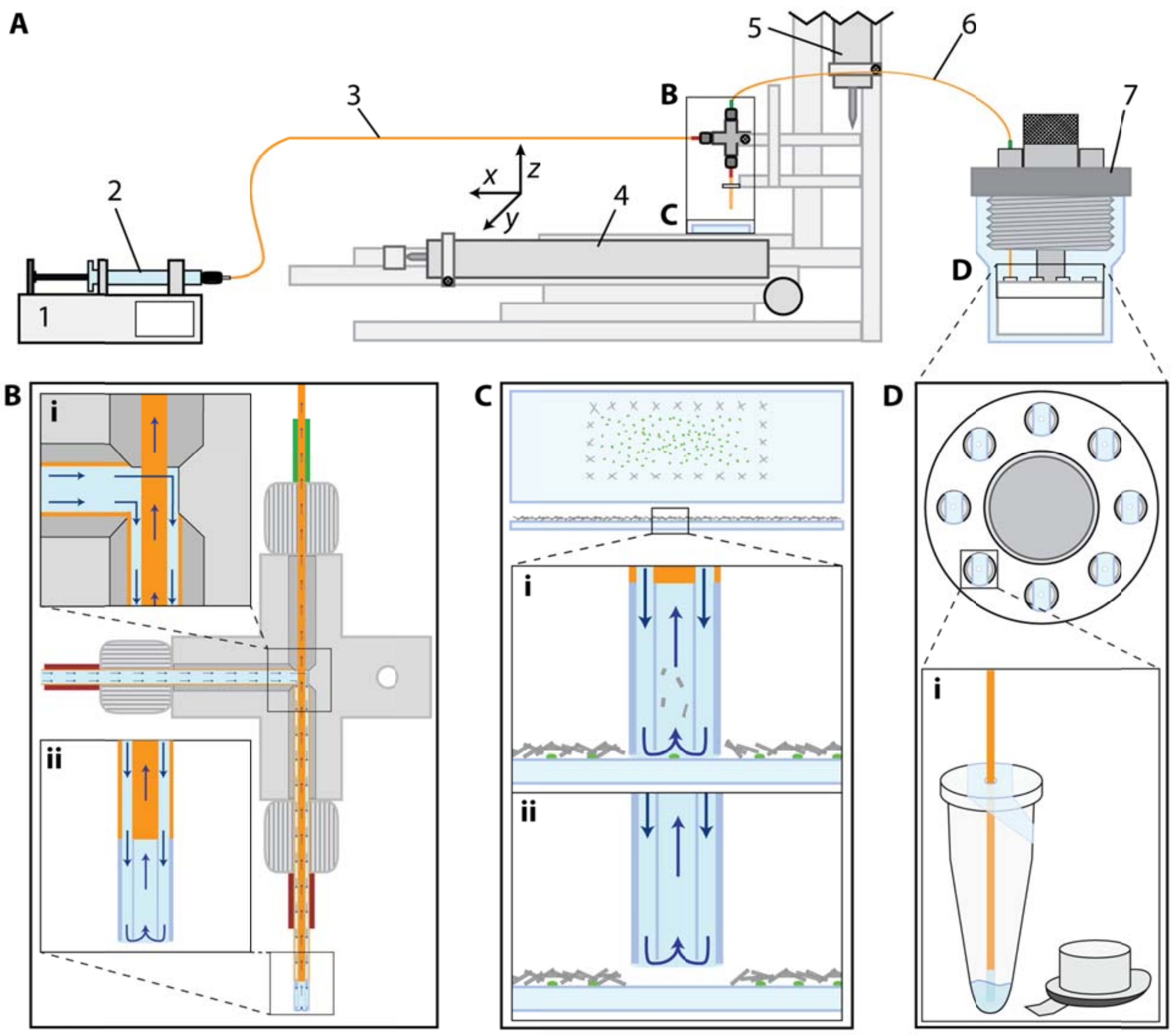


Figure 8.2. Schematic of liquid microjunction extraction instrument. (A) The overall system consists of a syringe pump (1) to deliver extraction solution (2) through the liquid supply side (3) of the liquid microjunction probe (B). The position of the sample and manifold are controlled through an *xyz* system of linear actuators (4,5). Solution is delivered to the surface to extract from target locations before being aspirated through the capillary (6) attached to the collection chamber held at vacuum (7). (B) Expanded view of the liquid microjunction extraction probe. i) The extraction solution is delivered through the size-matched inlet capillary to the outer capillary. Solution aspirated from the surface is drawn through the t-joint to the collection chamber. ii) The inner capillary is slightly withdrawn from the outer capillary. (C) Overview of interaction with target surface. Several x marks are positioned around the perimeter of a field of cells to correlate stage positions with optical images. i) The probe is moved over the top of a cell of interest and lowered to begin extraction. ii) After 60 s, the probe is retracted and solution continues to flush the capillaries and transfer extracts to the collection tube. (D) A custom vacuum chamber aspirates extract solution from the surface into microcentrifuge tubes covered with parafilmM. The collection carousel accommodates eight tubes with indexed positions to simplify alignment. i) ParafilmM prevents droplets hanging on the capillary from transferring between samples, but must be open to provide vacuum.

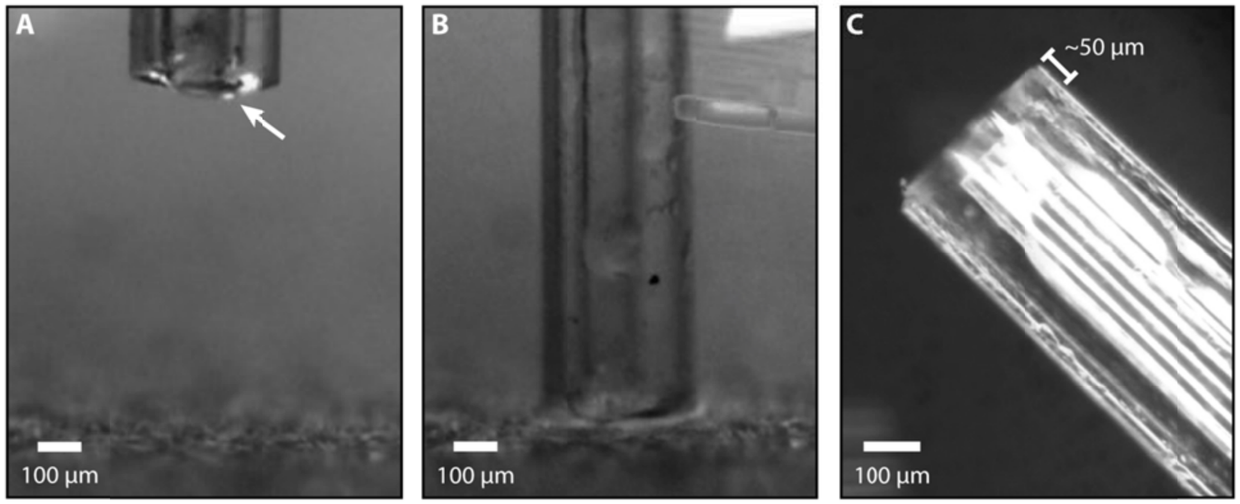


Figure 8.3. Micrographs of the liquid microjunction probe. (A) While operating, the extraction solution forms a meniscus at the probe tip, indicated with a white arrow. (B) When the probe touches the surface, solution flow remains constant, without air entering the capillary. (C) The inner capillary is withdrawn ~50 µm from the outer capillary for stable flow.

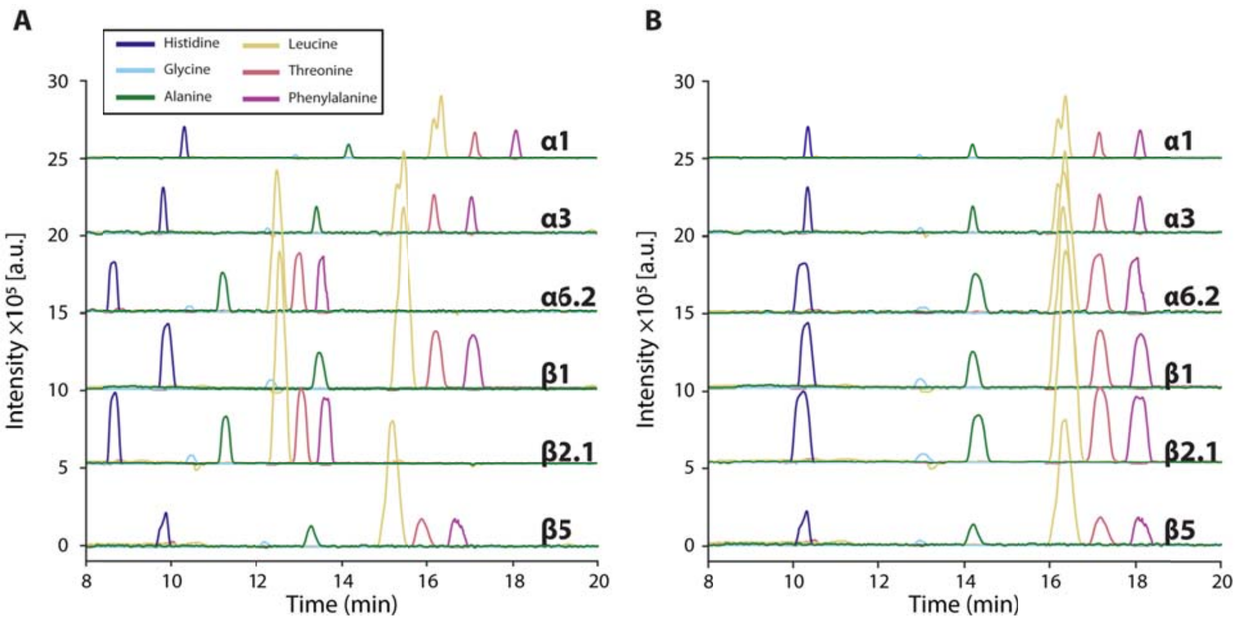


Figure 8.4. Example extracted ion electropherograms of compounds utilized for migration time alignment. The migration time of each *m/z* value is determined from the raw EIEs, shown in (A). The migration times are mapped to one sample (α1) to determine a linear regression between each time scale. The resulting equation is used to calculate a new set of migration times (B).

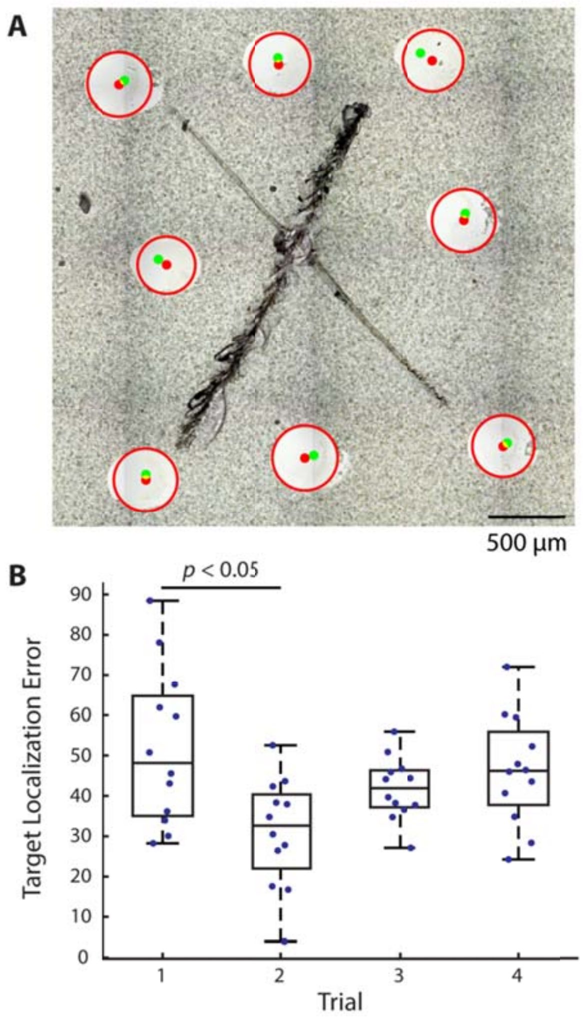


Figure 8.5. Representative extractions for determining target localization error. Target points were positioned around fiducial marks placed in the center of a glass slide. (A) Overlay of microscope image with the position of target locations (green) and extraction areas (red). (B) Box plot of accuracy over four trials of fiducial registration. The registration trial was the only confounding variable found to significantly affect target localization error.

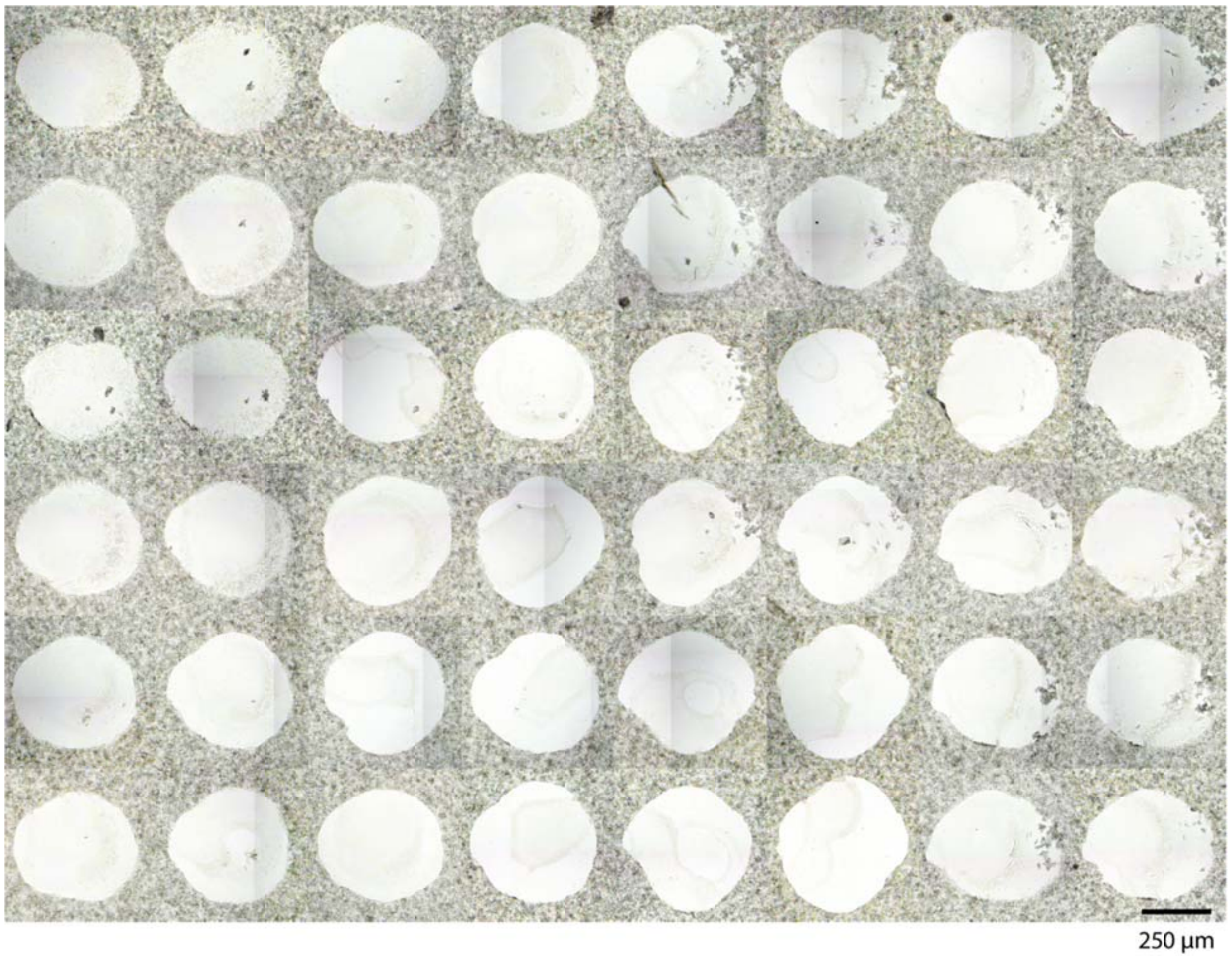


Figure 8.6. Montage of extraction areas. Each spot is the result of 5 seconds of extraction from a DHB-coated microscope slide. The extraction profile is roughly circular with an average radius of 206.3 ± 1.7 µm.

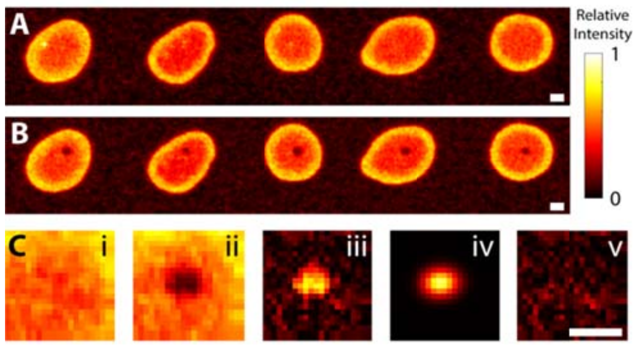


Figure 8.7. Measurement of removal efficiency of ^{3}H-angiotensin. A phosphorimager was utilized to measure angiotensin distributions pre- and post- extraction, shown in (A) and (B) respectively. (C) Sample analysis of the left-most spot. Subregions surrounding each extraction (i and ii) are utilized to determine the distribution of the fraction of radioactivity removed (iii). The distribution is fit to a general two dimensional Gaussian (iv) to determine the fraction removed. Residuals of the fit (v) are non-structured indicating the model is appropriate. All scale bars are 500 μm.

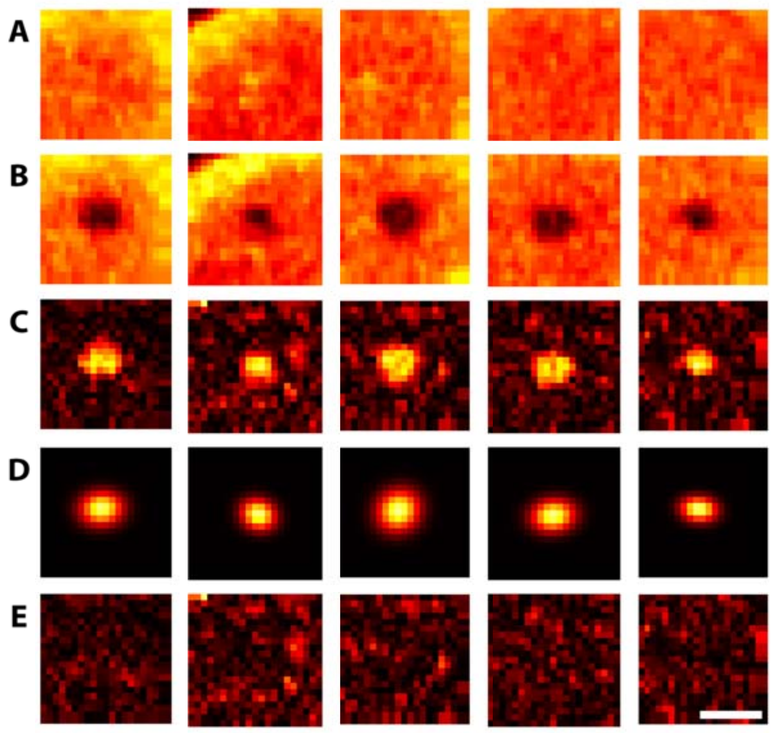


Figure 8.8. Complete set of subregions and Gaussian fits for determination of removal efficiency. Each column corresponds to a single extraction from spot 1 (left) to 5 (right). (A) Normalized intensity distribution prior to extraction. (B) Same region after extraction. (C) Background corrected, fraction removed intensity. (D) Two dimensional Gaussian fit of image in (C). E) Absolute, residual intensity scaled the same as (C). Scale bar is 500 µm.

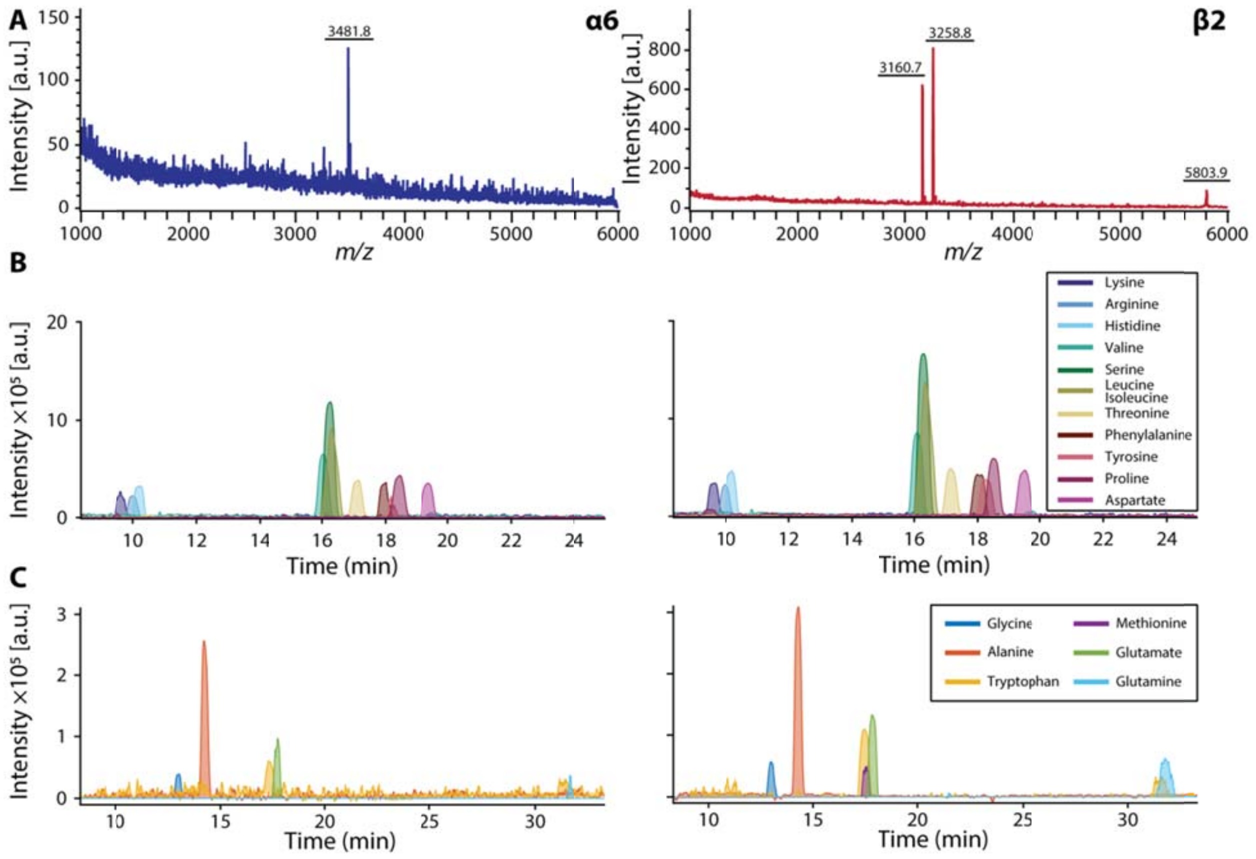


Figure 8.9. (A) Representative single cell MALDI-MS profiles of a single α and β cell. Corresponding CE-MS extracted ion electropherograms of the same cells, showing high intensity amino acids (B) and lower intensity amino acids (C).

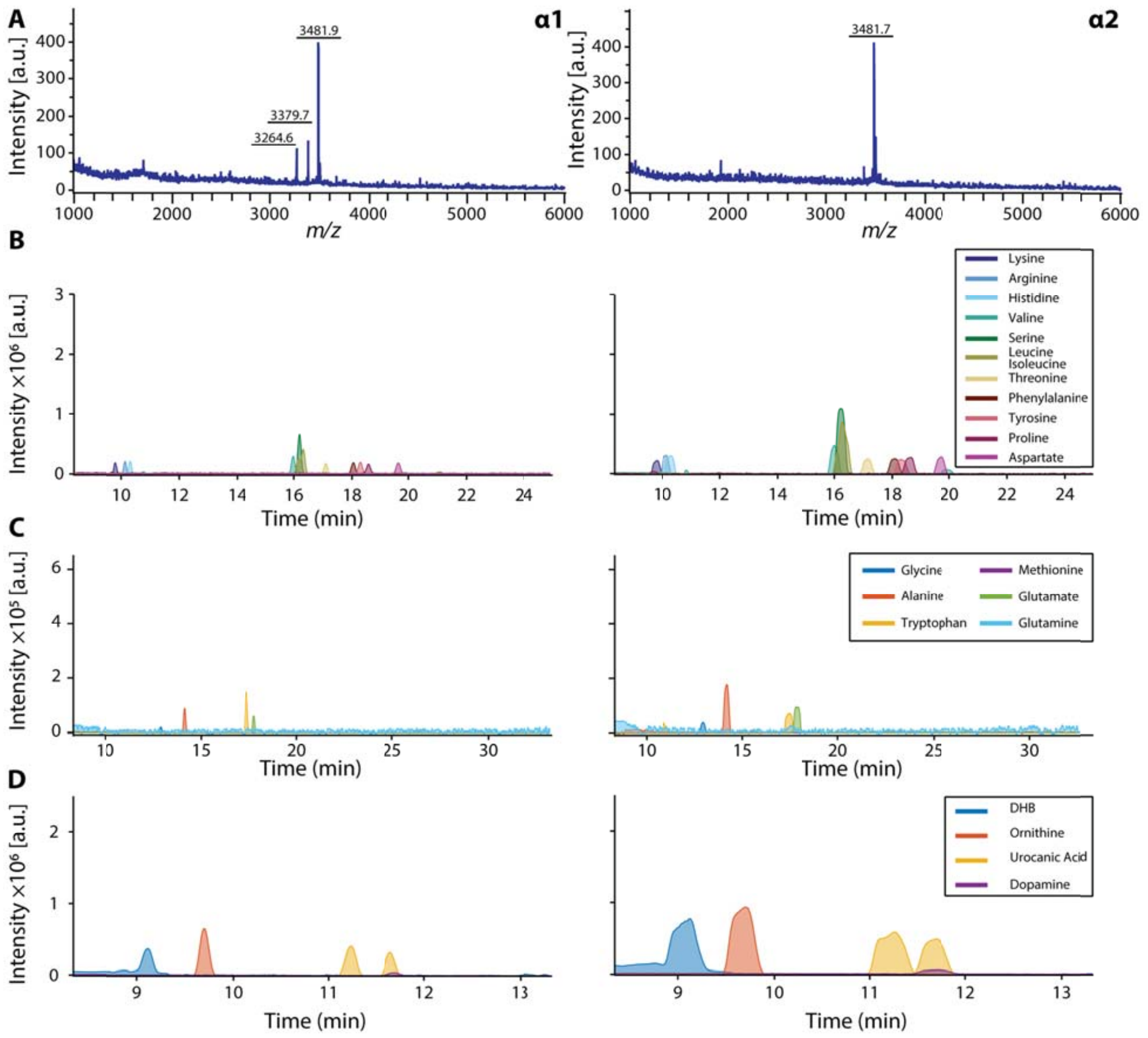


Figure 8.10. Single cell MALDI-MS and extracted ion electropherograms. (A) Single cell MALDI-MS, (B) high intensity amino acids, (C) lower intensity amino acids, and (D) selected metabolites and background signals. The cell identities are displayed in panel (A) and match the annotated points in Figure 8.1. Technical replicates of CE-MS are indicated as decimals after the cell identity, e.g. α6.1 and α6.2.

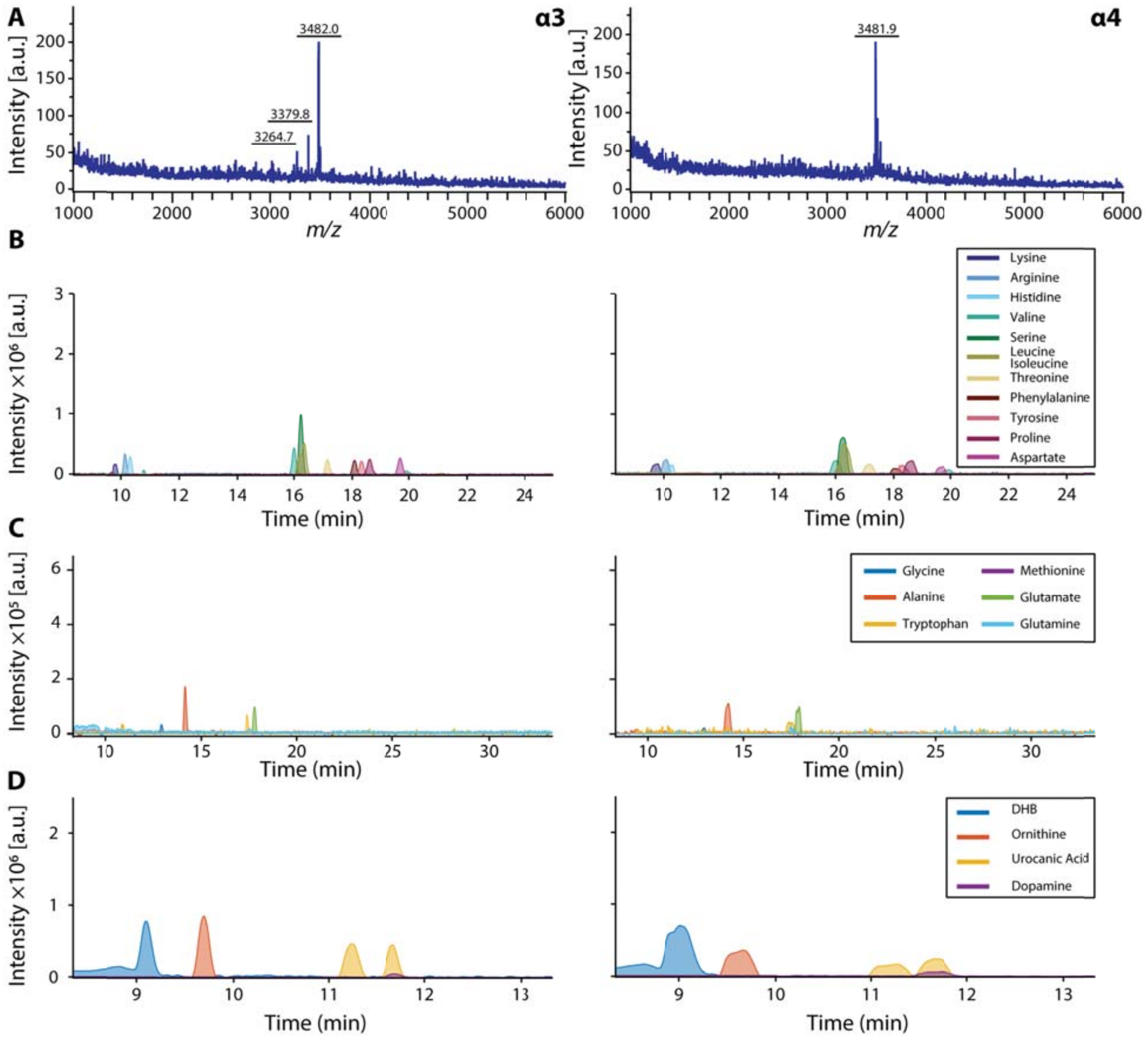


Figure 8.10. (cont.)

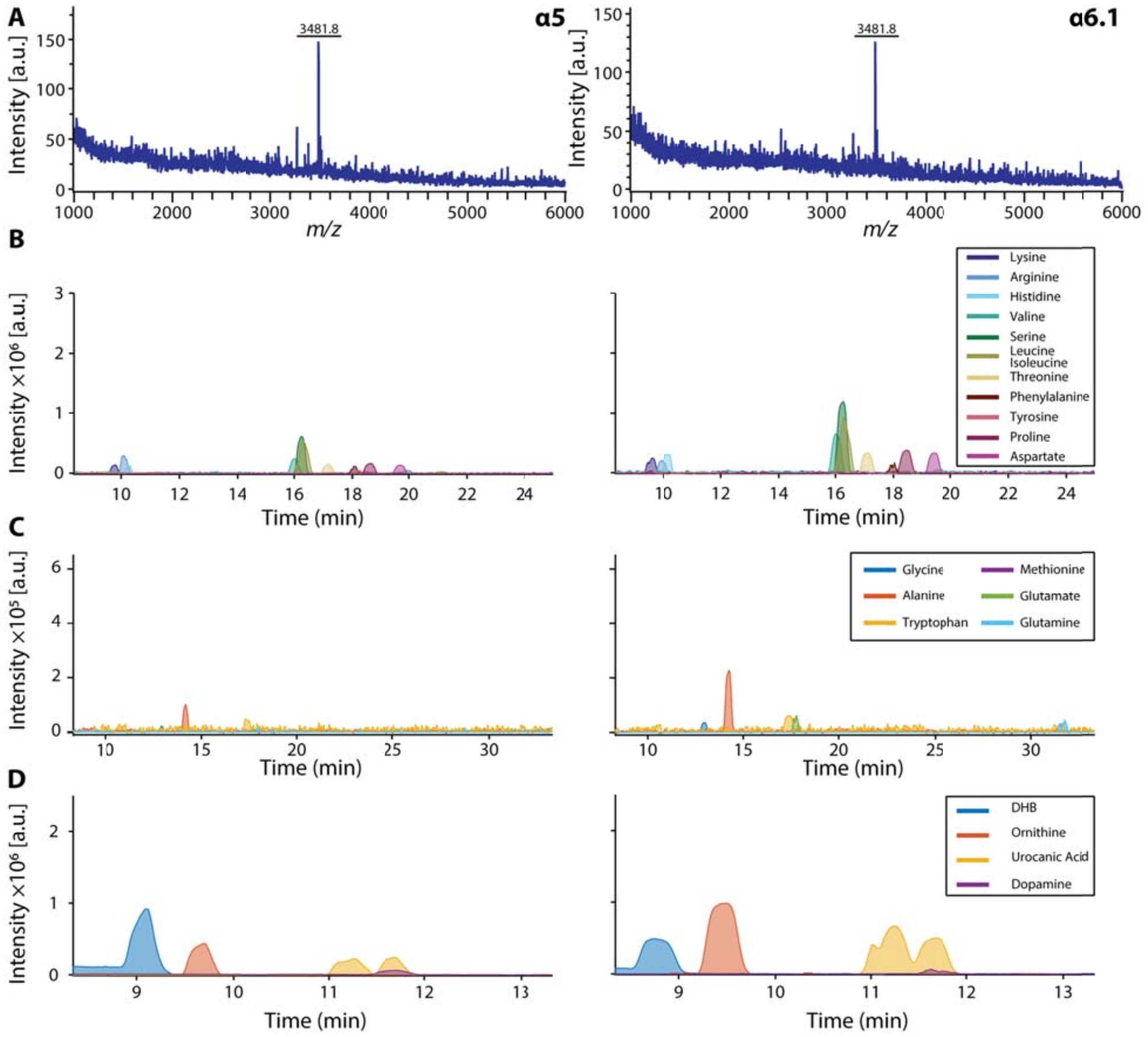


Figure 8.10. (cont.)

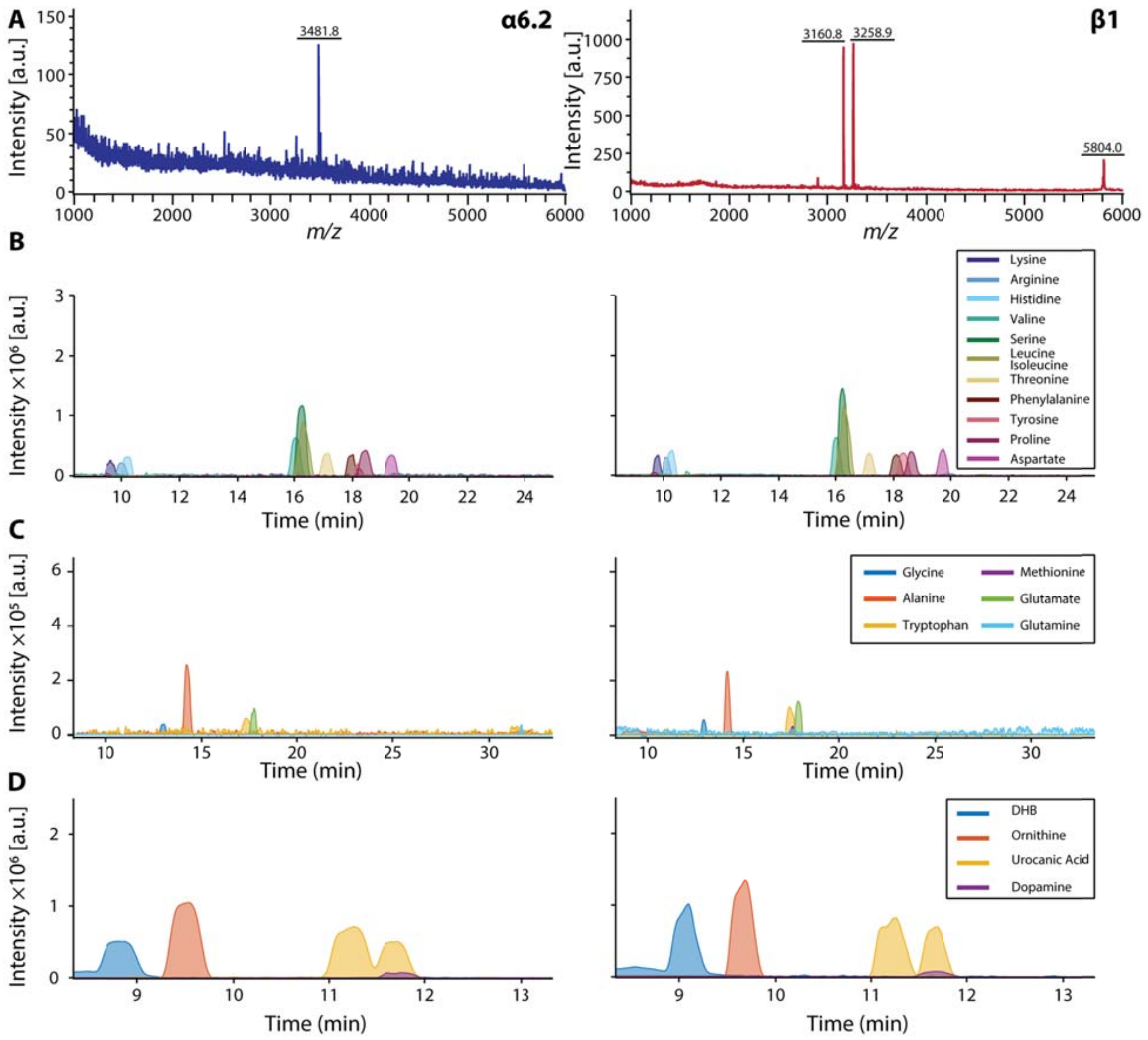


Figure 8.10. (cont.)

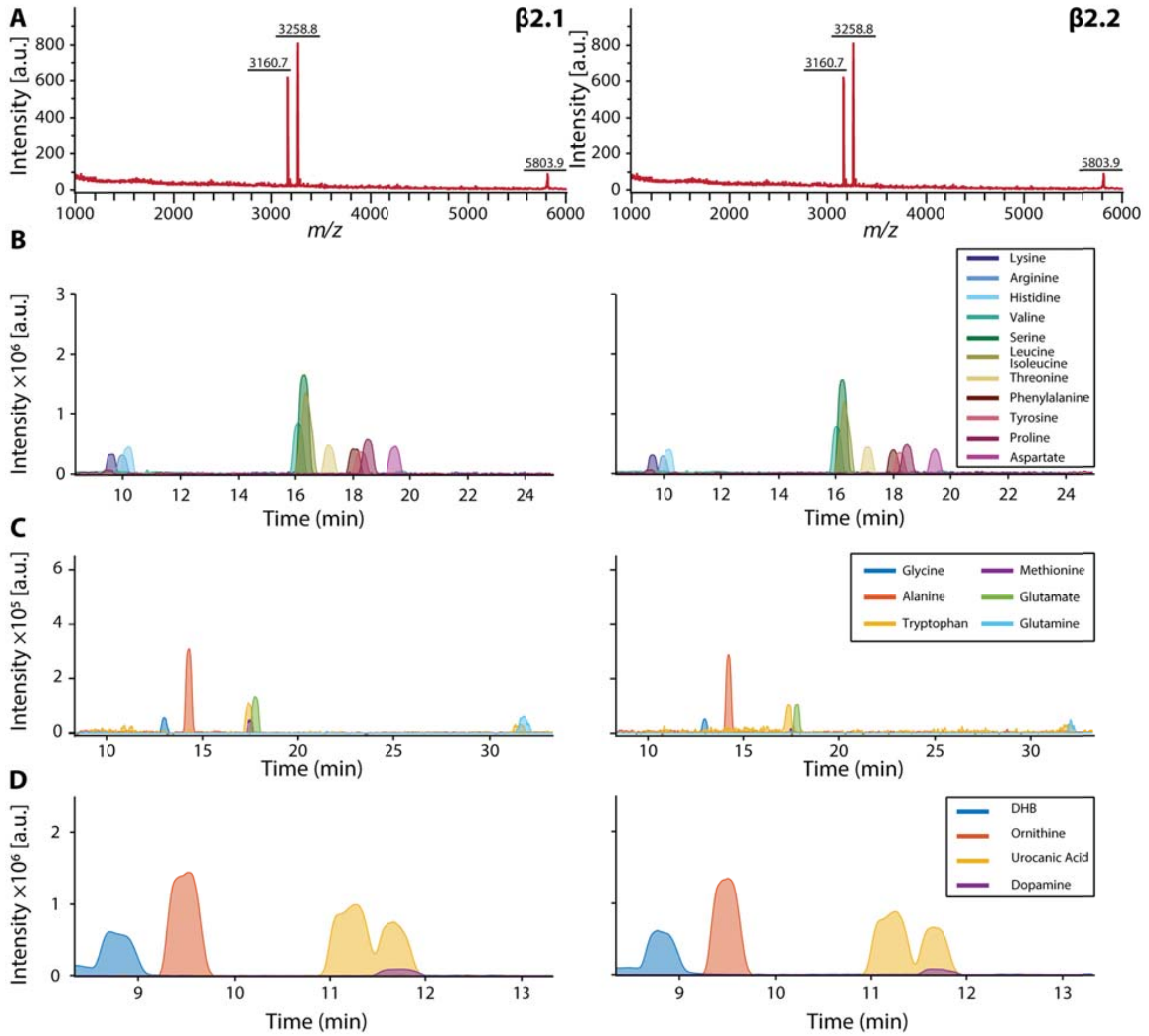


Figure 8.10. (cont.)

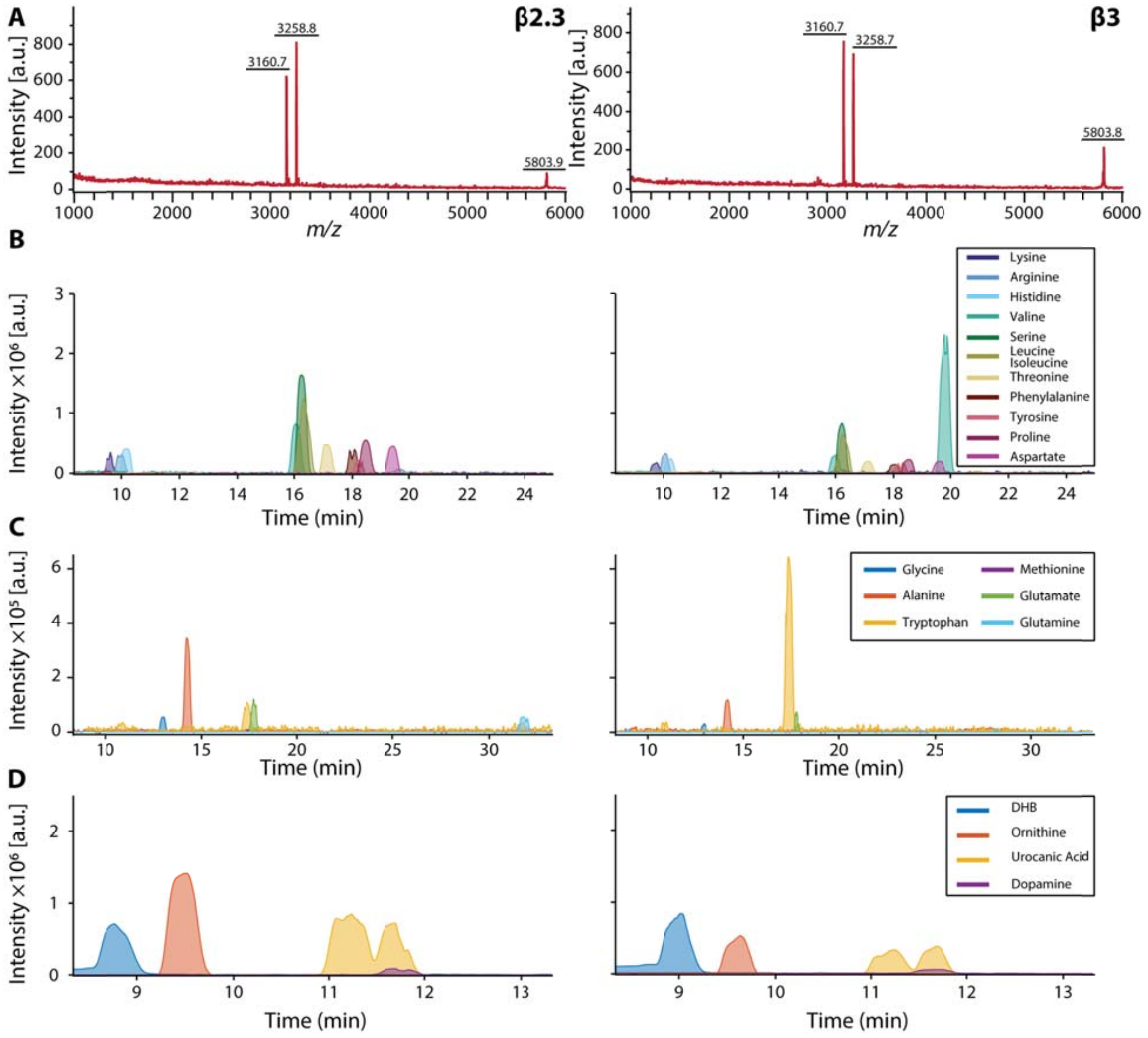


Figure 8.10. (cont.)

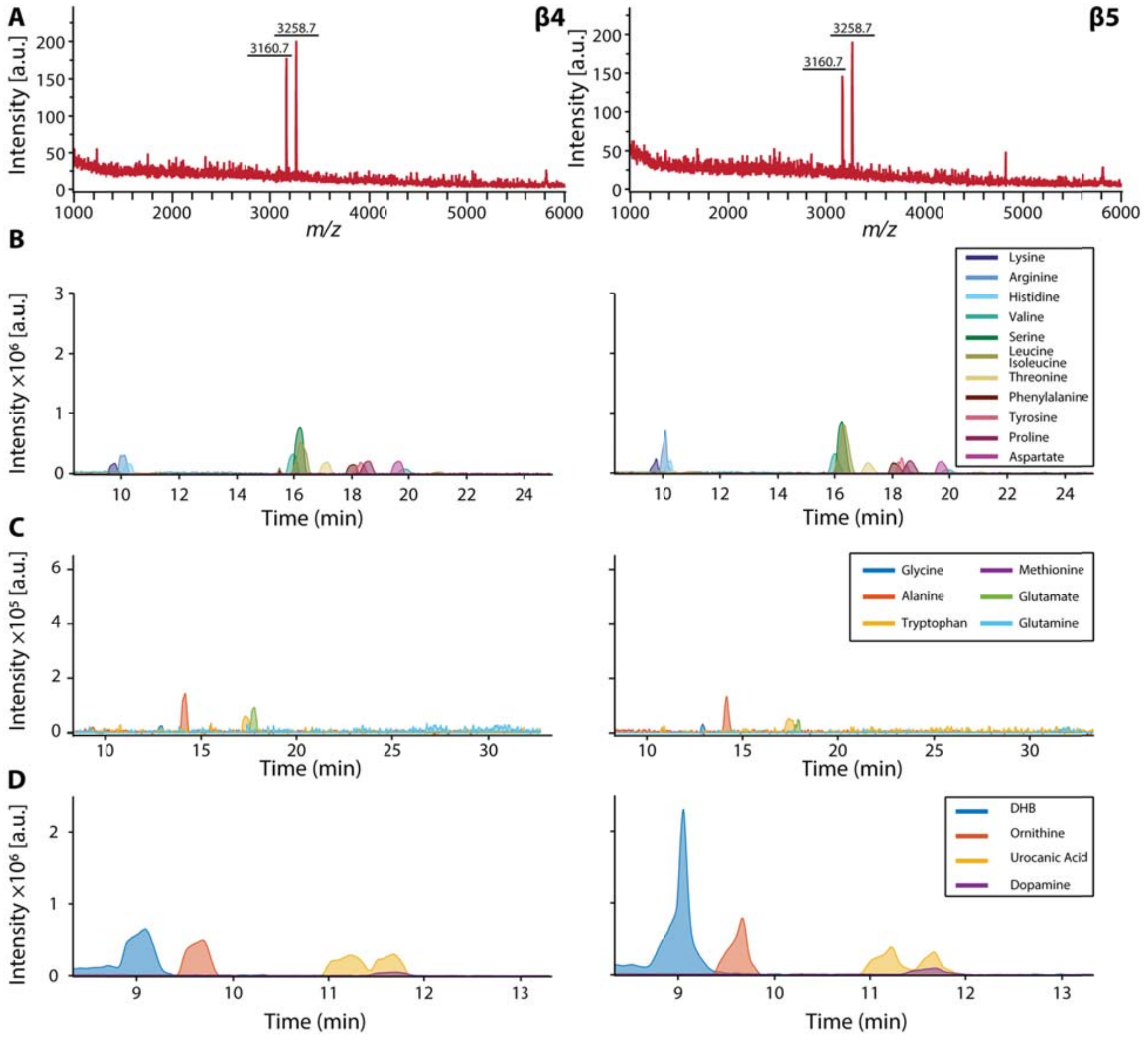


Figure 8.10. (cont.)

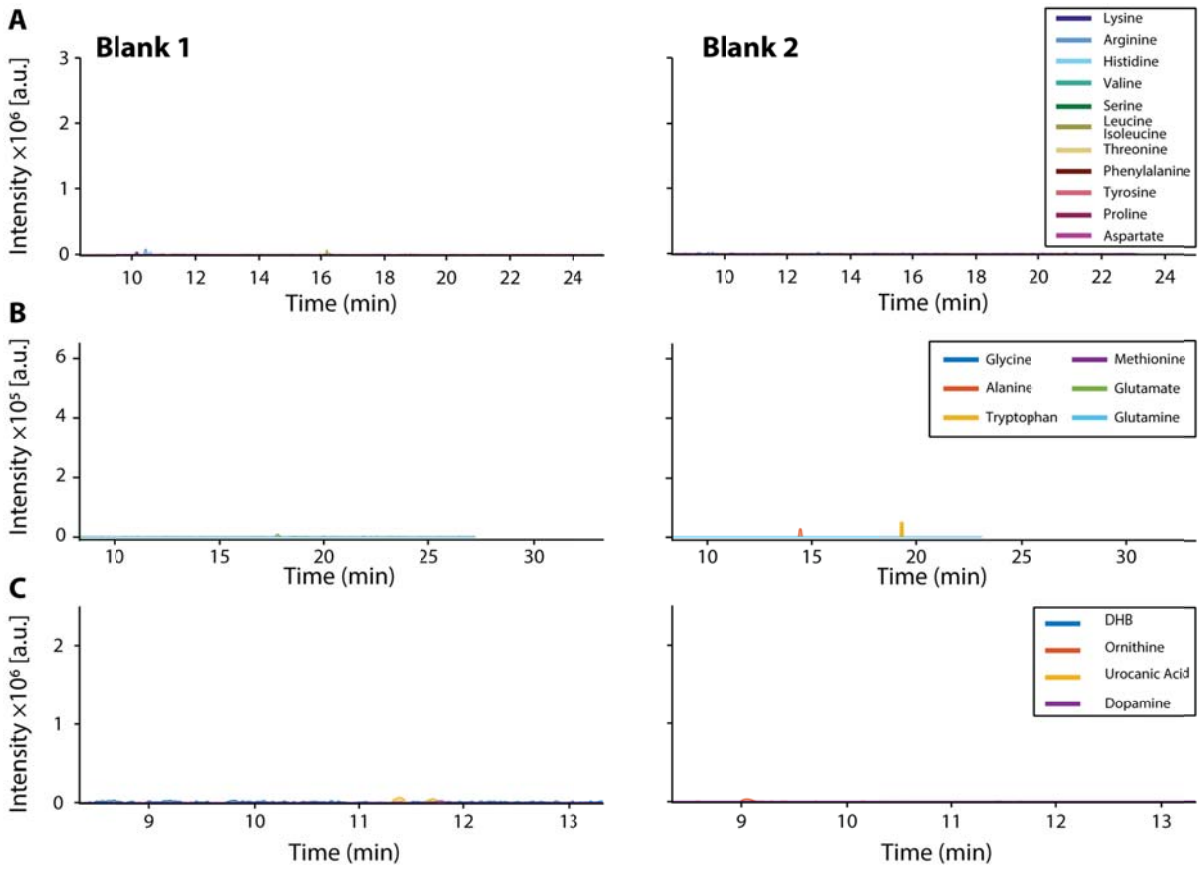
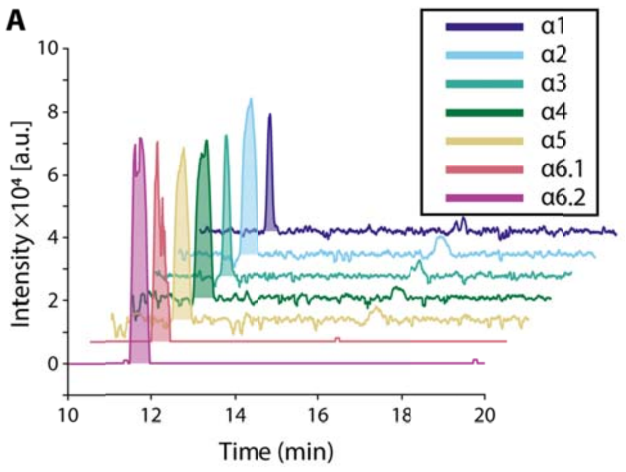


Figure 8.10. (cont.)



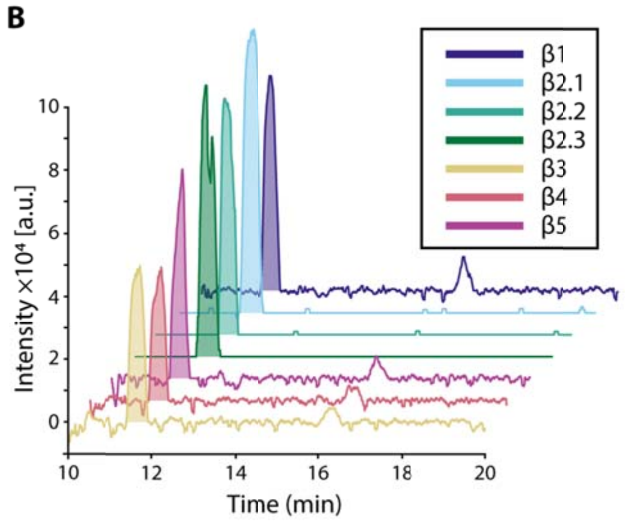


Figure 8.11. Extracted ion electropherograms for *m/z* 154.09 ± 0.01. The peak with migration time matching dopamine standards is filled for each electropherogram. (A) α cells (B) β cells. Dopamine was detectable in every cell analyzed. The peak at ~16 minutes is attributed to sodiated leucine. Cells with technical replicates are annotated with decimals, e.g. α6.1.

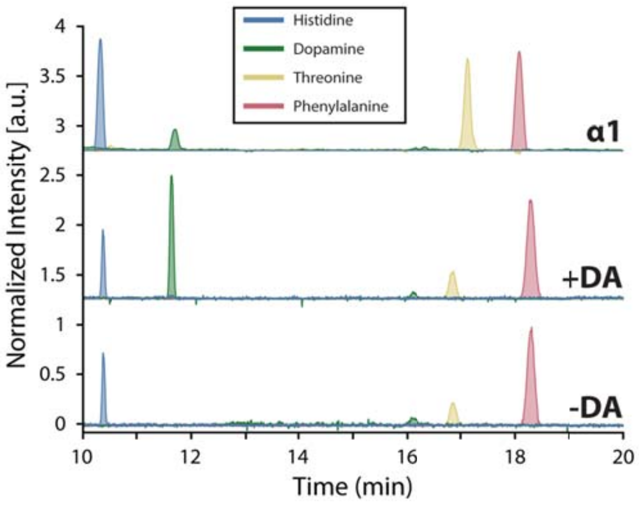


Figure 8.12. Extracted ion electropherograms for amino acid standards compared with α 1. Intensities are normalized to phenylalanine in each sample. A subset of the amino acid standards were utilized for migration time alignment due to low sensitivity in the excluded channels. Dopamine is assigned to the peak migrating at ~12 minutes as highlighted in Figure 6 of the main text.

Table 8.1. Summary of fitting values for determining the fraction of ^{3}H-angiotensin removed. The fraction removed was modeled as a general, two-dimensional Gaussian distribution centered on each sub-image. The model equation is

$$f(x,y) = A\exp\left(-\left(a(x-\mu_x)^2 - 2b(x-\mu_x)(y-\mu_y) + c(y-\mu_y)\right)\right)$$

Where A is the fraction removed at the center, e.g. (x,y) = (μ_x, μ_y) and (μ_x, μ_y) is the center of the distribution. The variables *a,b,c* are further defined as

$$a = \frac{\cos^2\theta}{2\sigma_x^2} + \frac{\sin^2\theta}{2\sigma_y^2}$$

$$b = -\frac{\sin 2\theta}{4\sigma_x^2} + \frac{\sin 2\theta}{4\sigma_y^2}$$

$$c = \frac{\sin^2\theta}{2\sigma_x^2} + \frac{\cos^2\theta}{2\sigma_y^2}$$

Where σ_x, σ_y are the standard deviation of the distribution and θ is the rotation in radians. With the constraints that $A \in [0,2], \mu_x, \mu_y \in [-s,s], \sigma_x, \sigma_y \in [0,s], \theta \in [0,2\pi]$ where s is the subregion size, 500 μm. Reported values represent the 95% confidence intervals for each parameter.

Spot	A	μ_x (μm)	μ_y (μm)	θ (radians)	σ_x (μm)	σ_y (μm)
1	0.889 ± 0.079	15 ± 10	18.9 ± 7.9	1.51 ± 0.25	88.4 ± 7.9	114 ± 10
2	0.91 ± 0.14	24 ± 15	9 ± 13	5.08 ± 0.72	77 ± 13	91 ± 15
3	0.908 ± 0.094	4 ± 12	35 ± 11	5.68 ± 0.90	115 ± 12	106 ± 11
4	0.904 ± 0.099	0 ± 12	16.5 ± 9.0	3.04 ± 0.24	112 ± 12	82.2 ± 9.0
5	0.92 ± 0.13	20 ± 13	24.1 ± 8.9	0.09 ± 0.24	96 ± 13	64.9 ± 8.9

References

(1) Onjiko, R. M.; Moody, S. A.; Nemes, P. *Proceedings of the National Academy of Sciences of the United States of America* **2015**, *112*, 6545.

(2) Armbrecht, L.; Dittrich, P. S. *Analytical Chemistry* **2016**.

(3) Malucelli, E.; Fratini, M.; Notargiacomo, A.; Gianoncelli, A.; Merolle, L.; Sargenti, A.; Cappadone, C.; Farruggia, G.; Lagomarsino, S.; Iotti, S. *Analyst* **2016**.

(4) Zenobi, R. *Science* **2013**, *342*.

(5) Altschuler, S. J.; Wu, L. F. *Cell* **2010**, *141*, 559.

(6) Lawson, D. A.; Bhakta, N. R.; Kessenbrock, K.; Prummel, K. D.; Yu, Y.; Takai, K.; Zhou, A.; Eyob, H.; Balakrishnan, S.; Wang, C.-Y.; Yaswen, P.; Goga, A.; Werb, Z. *Nature* **2015**, *526*, 131.

(7) Powell, A. A.; Talasaz, A. H.; Zhang, H.; Coram, M. A.; Reddy, A.; Deng, G.; Telli, M. L.; Advani, R. H.; Carlson, R. W.; Mollick, J. A.; Sheth, S.; Kurian, A. W.; Ford, J. M.; Stockdale, F. E.; Quake, S. R.; Pease, R. F.; Mindrinos, M. N.; Bhanot, G.; Dairkee, S. H.; Davis, R. W.; Jeffrey, S. S. *PLOS ONE* **2012**, *7*, e33788.

(8) Zeisel, A.; Munoz-Manchado, A. B.; Codeluppi, S.; Lonnerberg, P.; La Manno, G.; Jureus, A.; Marques, S.; Munguba, H.; He, L.; Betsholtz, C.; Rolny, C.; Castelo-Branco, G.; Hjerling-Leffler, J.; Linnarsson, S. *Science* **2015**, *347*, 1138.

(9) Fagerer, S.; Schmid, T.; Ibanez, A.; Pabst, M.; Steinhoff, R.; Jefimovs, K.; Urban, P.; Zenobi, R. *Analyst* **2013**.

(10) Ibáñez, A. J.; Fagerer, S. R.; Schmidt, A. M.; Urban, P. L.; Jefimovs, K.; Geiger, P.; Dechant, R.; Heinemann, M.; Zenobi, R. *Proceedings of the National Academy of Sciences* **2013**, *110*, 8790.

(11) Chiang, M. K.; Melton, D. A. *Developmental Cell* **2003**, *4*, 383.

(12) Dorrell, C.; Schug, J.; Canaday, P. S.; Russ, H. A.; Tarlow, B. D.; Grompe, M. T.; Horton, T.; Hebrok, M.; Streeter, P. R.; Kaestner, K. H.; Grompe, M. *Nature Communications* **2016**, *7*, 11756.

(13) Jansson, E. T.; Comi, T. J.; Rubakhin, S. S.; Sweedler, J. V. *ACS Chemical Biology* **2016**, *11*, 2588.

(14) Comi, T. J.; Do, T. D.; Rubakhin, S. S.; Sweedler, J. V. *Journal of the American Chemical Society* **2017**.

(15) Li, L. J.; Garden, R. W.; Sweedler, J. V. *Trends in Biotechnology* **2000**, *18*, 151.

(16) Page, J. S.; Sweedler, J. V. *Analytical Chemistry* **2002**, *74*, 6200.

(17) Ong, T.-H.; Kissick, D. J.; Jansson, E. T.; Comi, T. J.; Romanova, E. V.; Rubakhin, S. S.; Sweedler, J. V. *Analytical Chemistry* **2015**, *87*, 7036.

(18) Nemes, P.; Knolhoff, A. M.; Rubakhin, S. S.; Sweedler, J. V. *Analytical Chemistry* **2011**, *83*, 6810.

(19) Jankowski, J. A.; Tracht, S.; Sweedler, J. V. *Trac-Trends in Analytical Chemistry* **1995**, *14*, 170.

(20) Lapainis, T.; Rubakhin, S. S.; Sweedler, J. V. *Analytical Chemistry* **2009**, *81*, 5858.

(21) Cecala, C.; Sweedler, J. V. *Analyst* **2012**, *137*, 2922.

(22) Liu, J.; Tseng, K.; Garcia, B.; Lebrilla, C. B.; Mukerjee, E.; Collins, S.; Smith, R. *Analytical Chemistry* **2001**, *73*, 2147.

(23) Rejtar, T.; Hu, P.; Juhasz, P.; Campbell, J. M.; Vestal, M. L.; Preisler, J.; Karger, B. L. *Journal of Proteome Research* **2002**, *1*, 171.

(24) Page, J. S.; Rubakhin, S. S.; Sweedler, J. V. *Analytical Chemistry* **2002**, *74*, 497.

(25) Fan, Y.; Lee, C. Y.; Rubakhin, S. S.; Sweedler, J. V. *Analyst* **2013**, *138*, 6337.

(26) Do, T. D.; Comi, T. J.; Dunham, S. J. B.; Rubakhin, S. S.; Sweedler, J. V. *Analytical Chemistry* **2017**.

(27) Garcia Barrado, M. J.; Iglesias Osma, M. C.; Blanco, E. J.; Carretero Hernández, M.; Sánchez Robledo, V.; Catalano Iniesta, L.; Carrero, S.; Carretero, J. *PLOS ONE* **2015**, *10*, e0123197.

(28) Ustione, A.; Piston, D. W. *Molecular Endocrinology* **2012**, *26*, 1928.

(29) Tucker, K. R.; Li, Z.; Rubakhin, S. S.; Sweedler, J. V. *Journal of The American Society for Mass Spectrometry* **2012**, *23*, 1931.

(30) Giordano, E.; Cirulli, V.; Bosco, D.; Rouiller, D.; Halban, P.; Meda, P. *American Journal of Physiology-Cell Physiology* **1993**, *265*, C358.

(31) Mayer, B. X. *Journal of Chromatography A* **2001**, *907*, 21.

(32) Nemes, P.; Rubakhin, S. S.; Aerts, J. T.; Sweedler, J. V. *Nat. Protocols* **2013**, *8*, 783.

(33) Borelli, M. I.; Rubio, M.; García, M. E.; Flores, L. E.; Gagliardino, J. J. *BMC Endocrine Disorders* **2003**, *3*, 2.

(34) Raffo, A.; Hancock, K.; Polito, T.; Xie, Y.; Andan, G.; Witkowski, P.; Hardy, M.; Barba, P.; Ferrara, C.; Maffei, A.; Freeby, M.; Goland, R.; Leibel, R. L.; Sweet, I. R.; Harris, P. E. *Journal of Endocrinology* **2008**, *198*, 41.

CHAPTER 9

OPTICALLY-GUIDED MALDI-MS PROFILING OF MICROBIAL COLONIES FOR HIGH-THROUGHPUT ENGINEERING OF MULTI-STEP ENZYMATIC REACTIONS

Notes and Acknowledgements

This chapter is adapted from a manuscript in preparation for publication as of March 29, 2017, coauthored by T. Si, B. Li, Y. Ui, P. Hu, Y. Wu, Y. Min, D. A. Mitchell, H. Zhao, and J. V. Sweedler. T. J. Comi developed sampling protocols for MALDI-MS screening and performed data analysis on resulting MALDI-MS datasets. T. Si performed molecular engineering, cloning, mutagenesis, and LC-MS analysis. B. Li performed MALDI-MS and MS/MS experiments, developed the imprinting scheme and analyzed FT-ICR and MS/MS data. E. K. Neumann is acknowledged for her assistance with performing FT-ICR analysis. Support from the National Institutes of Health, Award Number P30 DA018310 from the National Institute on Drug Abuse and from the National Institute of Mental Health Award Number 1U01 MH109062 are also acknowledged. T. J. Comi acknowledges support from the National Science Foundation Graduate Research Fellowship Program, the Springborn Fellowship and the Training Program at Chemistry-Interface with Biology (T32 GM070421).

Introduction

With the current, incomplete understanding of complicated biological systems, it remains indispensable to screen recombinant variant libraries in biological research and engineering.[1-3] Traditional screening methods are either limited to photometrically-active molecules and labeled surrogates, or require chromatographic separation in low throughput[1]. Mass spectrometry (MS) offers label-free analysis of target molecules with high specificity, and matrix-assisted laser desorption/ionization (MALDI)-MS is particularly well suited for rapid inspection of a large number of samples due to its simple sample preparation, high salt tolerance, and wide coverage of diverse biomolecules.[4,5] For rapid profiling of enzymatic reactions, MALDI mass spectrometry imaging (MSI) has been increasingly applied.[6-11] However, MALDI-MSI screening utilizes fixed raster steps for sampling, which requires high-density deposition of reaction components in a regular array to increase throughput[9-11]. Such protocols generate technical challenges in co-localizing multiple enzymes, which may explain the limited reports of MSI-based screening of multi-step biochemical reactions. On the other hand, multi-step biosynthesis is vital for production of many important molecules including fuels, fine chemicals, and pharmaceuticals.[1] In particular, natural products (NPs) synthesized via secondary metabolism often contain complex chemical modifications installed by a number of enzymes. Research on NPs advances fundamental biochemistry and provides a valuable source for medicines.[12,13] NP analogs are widely applied in mechanistic studies focusing on mode-of-action, substrate tolerance, and structure-activity relationships[14]. NP variants are also engineered to develop compounds with improved medicinal properties[14]. To engineer a multi-step reaction such as NP biosynthesis, modified intermediates must be accepted at each step of the catalytic sequence to obtain a final product. Engineering an individual step in isolation ignores possible downstream effects. Therefore, current MSI screening platforms primarily designed for single-step enzymatic reactions may be ill-suited for engineering entire multi-step pathways.

In this work, we sought to apply optically-guided MALDI-MS to engineer multi-step enzymatic reactions via high-throughput, direct profiling of microbial colonies. Using microbial cells as reaction vessels, a set of enzymes can be encoded as a biosynthetic pathway on a DNA vector. Routine molecular biology enables mutagenesis, delivery, and expression of multiple enzymes encapsulated in a single cell. In addition, cell growth and metabolism in colonies facilitates analyte accumulation, which may help to eliminate the current requirement of analyte immobilization/capture for MSI screening. MALDI-MSI has been utilized to study spatial heterogeneity of microbial metabolism in biofilms[15-17] or multi-species co-cultures.[18-20] Such analyses have yet been applied to screen libraries of microbial colonies, which are randomly distributed and widely spaced on agar media when prepared using standard techniques. MSI of such sparse objects is inefficient as most acquisitions occur on the space between colonies due to the fixed raster steps for sampling.[21] Additionally, as beneficial mutations are generally rare, a large number of mutants are often created and screened to isolate desirable variants. Advanced liquid handling systems may be applied to deposit colonies into defined patterns for MSI screening,[9-11] but are costly and time-consuming. Instead, we developed an approach which utilizes microscopy images and simple machine vision to program MALDI-MS acquisition for rapid "colony picking" (Figure 9.1). The approach of optically-guided MS profiling for bacterial colonies is an extension of methodology developed for single-cell MS analysis[22-24].

Engineering multi-step enzymatic pathways may modify the structures and/or quantities of products. Such changes can be reflected in mass spectra, such as mass shifts in non-isomerization reactions, or differences in relative ion intensities of congeners due to altered enzymatic specificities. The mass spectra resulting from microbial screening produce a large, information-rich data set, which requires computational tools to extract, interpret and visualize the most relevant signals to aid mutant recovery. The diverse molecular profiles can be surveyed with targeted, multivariate clustering if the molecular weight information of desired products is available. Alternatively, non-targeted clustering can group colonies exhibiting similar spectra without *a priori* knowledge for discovery efforts. For researchers with limited MS experience, it is highly desirable to visualize screening data in a manner similar to classical, colorimetric assays. Given the wide application of multi-step enzymatic reactions, data analysis pipelines tailored for diverse engineering objectives are needed.

As a proof-of-concept, we applied optically-guided MALDI-MS to study and modify catalytic specificity of multi-step NP biosynthesis. With the designed workflow, we characterized the substrate tolerance of a five-enzyme pathway which synthesizes the antibiotic plantazolicin (**1**) from a precursor peptide. We then applied MALDI-MS screening in directed protein evolution to alter congener compositions of rhamnolipids (RLs) synthesized by a two-enzyme pathway. Custom sampling and analysis algorithms were developed for each system, with the former focusing on structural variations of analogues and the latter on relative abundances of target congeners. We demonstrated successful application of optically guided MALDI-MS profiling in both examples, resulting in the discovery of new compounds and isolation of enzymes with desirable chemical selectivity.

Materials and Methods

Strains, media and cultivation conditions

Zymo 5α Z-competent E. coli (Zymo Research, Irvine, CA) and NEB 10β Electrocompetent *E. coli* (New England Biolabs, Ipswich, MA) were used for general plasmid amplification and library construction, respectively. *E. coli* BL21 (DE3) (Cell Media Facility, UIUC, Urbana) was used as a host for expression of multiple enzymes. For plasmid construction, *E. coli* strains were cultured at 37°C and 250 r.p.m. in Luria broth (LB) liquid media (Fisher Scientific, Pittsburgh, PA), or at 37°C on LB plates solidified with 1.5% (w/v) agar. For plasmid maintenance using antibiotic selection, LB was supplemented with 100 μg mL^{-1} ampicillin and/or 50 μg mL^{-1} kanamycin. For inducible protein expression, BL21 (DE3) cells were cultured at 30°C instead of 37°C, and isopropyl β-D-1-thiogalactopyranoside (IPTG) was supplemented at a final concentration of 1 mM. M9 minimal media supplemented with BME vitamin mix (Sigma-aldrich, cat. #B6891), trace mineral solution (ATCC, cat. #MD-TMS) and 2 g l^{-1} acetate was used for PZN production. All chemicals were purchased through Sigma-Aldrich or Fisher Scientific unless noted otherwise.

DNA and strain construction

The list of primers used can be found in Table 9.1. All enzymes used for recombinant DNA cloning, including Q5 PCR polymerase, restriction digestion enzymes, were from New England Biolabs unless otherwise noted. Plasmid assembly was performed using Gibson Assembly Cloning Kit (New England Biolabs) or T4 ligase following the manufacturer's instructions. QIAprep Spin Plasmid Mini-prep Kits (Qiagen, Valencia, CA) were utilized to isolate plasmid DNA from *E. coli*. PCR, digestion and ligation products were purified by QIAquick PCR Purification and Gel Extraction Kits (Qiagen). Error-prone PCR was performed using GeneMorph II Random Mutagenesis Kits (Agilent Technologies, Santa Clara, CA). The genomic DNA of *Pseudomonas aeruginosa* PAO1c was a kind gift from Prof. Joshua D Shrout at University of Notre Dame.

For PZN production, the partial operon containing the biosynthetic genes essential for PZN production (*ptnC*, *ptnD*, *ptnB*, *ptnE* and *ptnL*) were PCR-amplified in two pieces from a fosmid bearing the complete PZN pathway[25] with primer pairs NP5/NP6 and NP7/NP8. PCR products and the vector pRSFDuet-1 linearized by NdeI and MfeI at the multiple cloning site II (MCSII) were assembled into pRSFDuet-T7-ptnJCDBEL(II), so that expression of the partial operon was under control of a T7 promoter. We have previously discovered N-terminal fusion of the maltose-binding protein (MBP) was necessary for efficient production of the precursor peptide,[25] so pET28a-MBP-bamA was used for precursor expression in trans on a separate plasmid. The kanamycin resistance gene in pRSFDuet-T7-ptnJCDBEL(II) was replaced by the ampicillin resistance gene to allow co-selection with pET28a-MBP-bamA. For site-saturation mutagenesis, the NNK degenerative codons at I34 or I35 positions were introduced in primers NP127 and NP128, respectively. The plasmid libraries assembled using two PCR products amplified from pET28a-MBP-bamA using primer pair #1 (NP119/NP124) and primer pair #2 (NP125/NP127 or NP125/NP128). Gibson assembly products were used to transform NEB 10β cells on agar media to obtain >10^4 independent transformants for each library. Plasmid DNA was isolated and used to transform BL21(DE3) harboring pRSFDuet-T7-pntJCDBEL for strain library creation.

For mono-rhamnolipid production, the *rhlB* gene was PCR-amplified using the primer pair NP21/NP22 from *P. aeruginosa* PAO1c genomic DNA. The PCR product was digested using EcoRI and HindIII, and ligated into MCSI of pRSFDuet-1 treated with the same set of enzymes to create pRSFDuet-T7-rhlB(I). The WT rhlA gene was PCR amplified using the primer pair NP23/NP24 from *P. aeruginosa* PAO1c genomic DNA, digested using MfeI/KpnI, and ligated into MCSII of pRSFDuet-T7-rhlB(I). The resulting plasmid, pRSFDuet-T7-rhlB(I)-T7-rhlA(II), was used to transform BL21 (DE3) to create the 'wild-type' production strain. To introduce random mutagenesis to *rhlA*, error-prone PCR was performed at an average mutation rate of ~1.8 bp kb^{-1} (or 1.6 bp per the 888 bp *rhlA* gene) with the following PCR conditions: 800 ng of pRSFDuet-T7-rhlB(I)-T7-rhlA(II) (WT or mutants) as a template, NP23/NP24 as primers, and 30 PCR cycles. The PCR product was digested using MfeI and KpnI before being ligated into pRSFDuet-T7-rhlB(I) treated with the same set of enzymes. Ligation products were used to transform electrocompetent NEB 10β cells, obtaining >10^5 independent transformants for each strain library on agar media. Plasmid DNA was isolated and transformed into BL21 (DE3) cells for strain library construction.

Microscopy-guided MALDI-MS with microMS

To prepare colonies for MALDI-MS profiling, plasmid DNA libraries were used to transform BL21 (DE3) cells, which were spread on Durapore^{TM} PVDF membrane filters (0.22 µm pore size, 90 mm diameter, EMD Millipore, Kankakee, IL, cat. #GVWP08050) to allow growth using non-inducing LB agar media. After cultivation at 30°C for 16~20 h, the filters were transferred to induction plates containing 1 mM IPTG (M9+acetate media for PZN, and LB media for rhamnolipid) for incubation at 30°C for 24 h. To transfer biomass onto MALDI targets, a colony-bearing filter was placed onto a clean, stainless steel substrate with colonies facing upwards. An indium-tin oxide (ITO)-coated glass slide (Delta Technologies, Loveland, CO) was delicately placed on the filter with ITO coating facing the colonies. Colony patterns were imprinted onto the ITO slide by gently applying ~3.5 N force by hand for 10 s.

High-throughput MALDI-MS screening of *E. coli* colonies was performed using lab-built image analysis software, microMS (available at http://neuroproteomics.scs.illinois.edu/microMS.htm) following previously reported single-cell profiling workflows[22-24] with modifications. Specifically, ITO-coated glass slides were etched with more than 16 fiducials surrounding the imprint region. Auto-fluorescence of *E. coli* colonies in the DAPI channel was used to aid colony finding. Whole-slide bright-field and fluorescence images were acquired on a Ziess Axio Imager M2 (Zeiss, Jena, Germany) using an Ab cam Icc5 camera, a HAL 100 halogen illuminator (Zeiss), and an X-CITE Series 120 Q mercury lamp (Lumen Dynamics, Mississauga, Canada). The 31000v2 DAPI filter set was used for auto-fluorescence excitation. The images were acquired as tiled mosaics using the 10x objective and

10% overlap. Images were processed and exported at 1/8 magnification as tiff files using ZEN software version 2 blue edition (Zeiss). The whole-slide tiff images were loaded into microMS, which performed automatic colony finding, target patterning around each colony, and correlation with the Bruker ultrafleXtreme MALDI-ToF/ToF mass spectrometer (Bruker Daltonics, Billerica, MA). Coordinate registration and correlation is performed by locating the etched fiducials in the mass spectrometer and recording their locations in microMS. The procedure is found to be accurate within ~20 μm when at least 16 fiducials are included in the training set. Target patterning positioned at most 10 targets around each colony, offset from the circumference by 25 pixels (110 μm), with a minimum shot-to-shot spacing of 10 pixels (44 μm). The custom geometry file was then ready to load into the mass spectrometer for automated acquisition.

Before spectra acquisition, the sample slide was coated with MALDI matrix using an artist's airbrush with a 0.2 mm nozzle (Paasche Airbrush Company, Chicago, IL). Several parameters were optimized to ensure even deposition of matrix on the imprinted colonies. The N_2 pressure was set at 40 psi and the imprint glass slides were spray coated at a distance of 30–35 cm with 50 mg mL^{-1} 2,5-dihydroxybenzoic acid (DHB) (Sigma-Aldrich) dissolved in methanol:H_2O (1:1, v/v). After spraying 2 mL of the DHB solution with the airbrush, the sample was dried for 1 min to avoid over-wetting and analyte delocalization. A total of 10 mL of DHB solution was applied in 10 min per target plate.

Measurements were performed using a frequency tripled Nd:YAG solid state laser (λ=355 nm). The laser footprint was set to "Ultra" at a ∼100 μm diameter. Mass spectrometer calibration was performed using Peptide Calibration Standard Kit II (Bruker Daltonics). Data acquisition was run in positive reflection mode with pulsed ion extraction and a mass range of

440-700 Da for the detection of rhamnolipids and 905-2005 Da for PZN variants. Colony imprints were analyzed following the custom geometry file with 500 laser shots fired at 2000 Hz. Resulting spectra were analyzed as detailed below.

Multivariate data analysis

Spectra were read directly into MATLAB 2015b with the readbrukermaldi function (github.com/AlexHenderson/readbrukermaldi) and manually recalibrated with a third order polynomial to correct for mass shifts between analyses. Spectra were resampled with bin widths of 0.025 Da for RL and 0.5 Da for PZN.

For PZN, untargeted t-SNE[26] was performed utilizing each, binned *m/z* value to evaluate population heterogeneity and variance in sample processing. It was determined that mutations resulted in mass shifts from the fully processed form of PZN while other clusters were due to experimental factors, including polymer contamination. Next, targeted t-SNE was performed to locate colonies expressing each point mutation. The maximum intensity of each monoisotopic amino acid substitution was extracted from every spectrum with a tolerance of ±0.25 Da. The reduced dataset was examined with t-SNE to cluster similar spectra. Apparent groups were examined manually to assign spectra to specific substitutions. Spectra without peptide signal were combined into the “N/A” cluster. These correspond to background, imaging artifacts, and spectra acquired on non-circular colonies. Additional filtering of putative colonies at the stage of optical image analysis could reduce the abundance of the N/A cluster. Next, the spectral classifications were mapped onto the optical image, leveraging the pixel positions encoded into the filename of each spectrum. To facilitate mutant recovery, the most common cluster is shown over the optical image, excluding the N/A cluster.

For RL, a targeted analysis was performed to visualize the relative abundance of **5b**. The intensities of protonated, sodiated, and potassiated monoisotopic masses of **5a**, **5b**, **5c**, and **5d** were extracted with a tolerance of ±0.2 Da. Spectra were filtered to remove colonies within 200 μm of each other and total intensity of RL less than 500 (arbitrary counts). The intensities of each RL were summed for spectra surrounding each colony. To visualize the molecular content of mutants on the target, the total RL intensity and relative content of **5b** were mapped onto the optical image of the colonies. The log base 10 of total abundance of RLs determined the size of each data point overlaid on the whole slide image. The relative abundance of **5b** dictated the color of each point. Such a visualization allows rapid assessment of desirable mutants in terms of total expression and relative abundance of **5b**.

In situ high-resolution and tandem MS analysis for PZN analogs

In situ measurement of accurate masses of peptides was conducted on colony imprints using a 7T solariX Fourier transform-ion cyclotron resonance (FT-ICR) mass spectrometer (Bruker Daltonics) equipped with a dual ESI/MALDI source and a Smartbeam II laser. Mass calibrations were performed externally using DHB and Peptide Calibration Standard Kit II (Bruker Daltonics). An *m/z* range of 150-3000 was acquired at 4 Mword. Data was analyzed in Data Analysis version 4.0 software (Bruker Daltonics). *In situ* tandem MS was conducted on the colony imprints using the MALDI ToF/ToF LIFT mode of the mass spectrometer under manual control. Tandem mass spectra were smoothed, baseline-corrected and analyzed in FlexAnalysis 3 (Bruker Daltonics).

Relative quantification of RL congeners using LC-MS/MS with MRM mode

To compare direct MALDI-MS profiling results with LC-MS or MALDI-MS quantification of RL congeners following organic solvent extraction, a filter bearing ~100 colonies with the WT mono-RL pathway subsequent to IPTG induction as described above was extracted using 500 μl of chloroform:ethanol (2:1, v/v) solution. The filter and cell debris were separated from supernatant using centrifugation (5 min, 10,000 r.p.m., 4°C), and the lower organic phase was dried under vacuum in a rotary flash evaporator (MiVac, GeneVac, UK). The samples were reconstituted in 50 μL of acetonitrile:H_2O (1:9, v/v) solution containing 2 mM ammonium acetate. For MALDI-MS analyses, 1 μl of the reconstituted extracts was mixed with 1 μl of DHB solution (50 mg mL^{-1} in acetonitrile:H_2O (1:1, v/v)) and spotted onto a MTP 384 polished steel target (Bruker Daltonics). MALDI spectra were acquired in positive reflection mode as described above. LC-MS quantification was performed as detailed below. The same extraction and quantification procedure was also used to analyze the colonies of mutant strains subsequent to plasmid retransformation.

To characterize the ratios of RL congeners in liquid fermentation products, single colonies were obtained by streaking glycerol frozen stocks of the WT and mutant RL-producing strains on agar plates. Three colonies for each strain were inoculated into 3 mL of LB+Kan media. Following cultivation at 30°C and 250 r.p.m. for 16 h, 60 μl of cell cultures was added to 3 mL of fresh LB media to continue growth until cell densities reached OD_{600}=0.4~0.8. IPTG was added to a final concentration of 1 mM, and induction was performed at 30°C and 250 r.p.m. for 24 h. Culture supernatants were obtained via centrifugation for cell separation (10 min, 4,000 r.p.m., 20°C), and were filtered through 0.22 μm-pore-size cellulose acetate membrane centrifuge tube filters (Sigma) before LC-MS analyses as detailed below.

A multiple reaction monitoring (MRM) assay was performed in negative-ion mode using an ultrahigh performance liquid chromatography–triple quadrupole–electrospray ionization mass spectrometry (UHPLC-QqQ-ESI MS) system (Bruker Daltonics) consisting of an Advance

UHPLC module and an EVOQ Elite triple quadrupole-mass spectrometer. A Kinetex 1.7 μm C18 150 × 2.1 mm internal diameter column (Phenomenex, Torrance, Calif., USA) was used for LC analyte separation. Mobile phase A was H_2O containing of 4 mM ammonium acetate and mobile phase B was acetonitrile. The gradient program was conducted as follows: 0-2 min, 5% B; 2-2.1 min, 5-50% B; 2.1-8 min, 50-90% B; 8-15 min, 90% B; 15-15.1 min, 90-5% B; 15.1-17 min, 5% B. Total run time was 17 min. The injection volume was 2 μL. The EVOQ source parameters were as follows: HESI, spray voltage (-) 4500 V; cone temperature, 250 °C; cone gas flow, 25; heated probe temperature, 450 °C; probe gas flow, 45; nebulizer gas flow, 65; exhaust gas, Off. Monitored MRM transitions for Rha-C8-C10 (and Rha-C10-C8) were 475→305, 475→169; Rha-C10-C10 were 503→333, 503→169; Rha-C10-C12 (and Rha-C12-C10) were 531→333, 531→169 and Rha-C12-12 were 559→395, 559→169. The *m/z* 169 product intensity was utilized as a quantitation transition while the other transitions provided confirmation of RL identities. EVOQ MRM chromatograms were analyzed using Data Review 8.2 (Bruker Daltonics). The peak area of quantitation transition of a specific RL congener was used to calculate its fraction relative to the sum of all RL peak areas.

Results and Discussion

Workflow development

We devised a workflow for high-throughput MALDI-based characterization of bacterial colonies consisting of strain library creation, optically-guided MALDI-MS profiling, and data analysis/visualization. Recombinant variants of a multi-enzymatic pathway are constructed as plasmid DNA libraries, which are used to transform a production host such as *E. coli* (Figure 9.1A). The transformants are plated on a filter membrane,[27] allowing facile manipulation of many colonies simultaneously, such as exchanging culture media or imprinting onto MALDI targets. Microbial cells are initially cultivated on non-inducing agar media to obtain individual colonies and transferred onto induction plates to initiate enzyme expression and target molecule production. Each clonal population contains a single variant of the multi-step pathway. For analysis, colonies are imprinted on conductive, indium tin oxide (ITO)-coated glass slides (Figure 9.1B). The use of transparent MALDI targets allows acquisition of optical images prior to matrix application to determine the relative coordinates of microbial colonies and fiducial markers (Figure 9.1C). MALDI matrix is then applied using an artistic airbrush. Lab-built software, microMS, was developed in Python to generate MALDI laser coordinates for automatic colony profiling (Figure 9.1C). Laser shots are patterned around the peripheries of imprinted colonies for optimal sensitivity, as described below. Resulting mass spectra are processed using multivariant statistical analysis, and high-dimensional data sets are visualized over the optical images to aid mutant recovery (Figure 9.1D).

Substrate Libraries of a Peptidic NP

Ribosomally synthesized and post-translationally modified peptides (RiPPs) form a major class of natural products that are ubiquitous in currently sequenced genomes.[28,29] As the product is synthesized from a ribosomally synthesized peptide, combinatorial variants can be generated by mutagenesis of the precursor gene.[30,31] Plantazolicin (PZN, **1**) is a member of a RiPP subclass termed linear azol(in)e-containing peptides. During biosynthesis of this subclass, a trimeric heterocycle synthetase (BCD) converts select Cys, Ser, and Thr residues in the C-terminal (core) region of the precursor peptide to thiazole, oxazole, and methyloxazol(in)e moieties, respectively. **1** is naturally produced by *Bacillus velezensis* FZB42[32] and exhibits remarkable antibacterial selectivity against *Bacillus anthracis,*[33] the causative agent of anthrax. We previously achieved heterologous production of **1** in *E. coli* using a fosmid bearing the corresponding biosynthetic gene cluster.[25] Analogs of **1** were also created by site-directed mutagenesis of the precursor peptide gene (bamA), followed by a medium-throughput screening involving liquid cultivation and methanol extraction before MS analyses.[25] For successful synthesis of an analog of **1**, a mutant precursor peptide must be accepted as a substrate by multiple steps of the biosynthetic pathways, including cyclodehydration, dehydrogenation, leader peptidolysis, N-terminal demethylation, and export.

To apply optically-guided MALDI-MS screening to *E. coli* colonies producing **1** analogs, we targeted two non-cyclized positions, I34 and I35 (Scheme 9.1, red), where mutations are relatively tolerated by the biosynthesis machinery.[25] Site-saturation mutagenesis was performed using degenerate codon (NNK)-containing primers. Polyclonal plasmid DNA was transformed into competent *E. coli* cells harboring a refactored version of the PZN cluster, where native *Bacillus* promoters were replaced with a strong T7 promoter to enhance production. IPTG was used to induce production of **1** on M9 medium containing acetate as the sole carbon source.

For I34 and I35 libraries, 352 and 393 colonies were screened, respectively, achieving >99.9% probability of full coverage on the NNK libraries.[34] Following the analysis workflow, we first performed unsupervised clustering of the resulting 2389 and 1623 MALDI mass spectra for the I34 and I35 libraries, respectively. We manually examined each spectral class for tentative PZN peaks, and found all the base peaks (Figure 9.2) consistent with single-residual-mutation analogues with 'wild-type-like' modifications: nine azole rings, one azoline ring, leader peptidolysis N-terminal to Arg28, and N-terminal demethylation (Scheme 9.1).

We observed 12 and 9 variant classes of **1**for the I34 (Figure 9.3A) and I35 (Figure 9.2) libraries, respectively, from MALDI-ToF MS data alone. High-resolution MS analysis further revealed both K and Q substitutions at I34, but only Q at I35 (Figure 9.2). Colonies belonging to each class were inoculated in liquid cultures for plasmid isolation and DNA sequencing. Each colony sequenced presented mutations consistent with predicted and observed mass shifts in base peaks assuming full maturation (Figure 9.2). In this study, all previously isolated PZN analogs with single residue mutations at I34 or I35 were detected as well as previously unreported variants (Figure 9.2).[25] Select analogs with sufficient residual analyte were subjected to *in situ* ion identification with tandem MS (Figure 9.4-7). The tandem mass spectra suggested "wild-type-like" modifications (Scheme 9.1) for examined base peaks (Figure 9.4-7). Detection of unreported PZN analogs in this study reflects the improved methodology. First, PZN variant production was increased through pathway refactoring and growth medium optimization, enabling observation of variants that were not detected before due to insufficient amount. Also, enhanced production allowed detection of **1** analogs directly from single colonies, eliminating laborious liquid cultivation and extraction steps that were necessary previously.[25] This improvement was leveraged by optically-guided MALDI-MS to substantially increase analysis throughput, allowing more comprehensive codons (NNK vs NNC) for mutagenesis while retaining high probabilities of full library coverage.

In addition to spectral classes containing base peaks matching predicted *m/z* values of PZN analogs, classes exhibiting low signal to noise or chemical background were also observed (Figure 9.3). This spectral class likely resulted from (1) mutations that are not tolerated by the biosynthetic machinery or lead to analog production below our detection limit, (2) the UAG stop codon contained in the degenerate NNK codon, (3) artifacts during optical image acquisition such as dust, and (4) problems targeting irregularly shaped colony imprints (Figure 9.8, I34M). The first two possibilities were not further studied given the consistency between current and previous results.[25] The latter two can be alleviated through more vigilant colony finding and target patterning. In particular, we found the best sensitivity was obtained when directing MALDI laser to the peripheries of imprinted biomass (Figure 9.8). Direct sampling on the imprinted biomass often yielded mass spectra with low signal to noise (Figure 9.8, I34M and I34R), possibly due to insufficient matrix-to-analyte mixing, poor laser focus, or inefficient ion transfer from the elevated sample heights.

Enzyme libraries in biosynthesis of 4

Next, we sought to engineer enzyme specificity in a two-step biochemical pathway for rhamnolipid (RL) synthesis (Scheme 9.2). Initially discovered from *Pseudomonas aeruginosa*,[35] RLs are a class of biosurfactants extensively studied for potential applications in enhanced oil recovery, biodegradation and bioremediation[36,37]. To form mono-rhamnolipids (mono-RLs, **5**), RhlB (rhamnosyltransferase 1 chain B) catalyzes condensation of **3** and **4**. Different variants of **3** are synthesized by RhlA (rhamnosyltransferase 1 chain A) using **2** of varying chain lengths and degrees of unsaturation,[38] contributing to the structural diversity of RL lipid moieties in nature.[39] The most abundant RL species produced by *P. aeruginosa* and other bacteria consist of β-hydroxydecanoyl-β-hydroxydecanoate (C10-C10) as the fatty acyl moiety, which is attributed to the role of RhlA as a "molecular ruler" with high preference towards β-hydroxydecanoyl-ACP (**2**, n=9) *in vitro.*[38] Different fatty acyl chain lengths affect the physiochemical and biological properties of RLs,[35,40] so it is desirable to produce RLs with custom congener compositions for specific applications. Previous screening assays relied on the link between antimicrobial activity and RL mixture composition and suffered from low chemical specificity.[40] Here we sought to directly measure relative abundances of different RL congeners using MALDI-MS in high throughput.

We first explored the possibility to measure relative abundance of **5** mixtures produced from recombinant *E. coli* colonies using optically-guided MALDI-MS screening. Heterologous production of **5** in *E. coli* was achieved by co-expression of the wild-type rhlA and rhlB genes of *P. aeruginosa* as previously reported (denoted as WT).[41] Following the screening workflow (Figure 9.1), eight peaks in the MALDI mass spectra were tentatively assigned as Na^+ and K^+ adduct ions of **5a-d** based on mass matching (Figure 9.9A) and comparison of tandem MS results with previous reports[15,16] (Figure 9.9B). Compared with LC-MS/MS quantification using multiple reaction monitoring (MRM) after organic solvent extraction, we found that optically-guided MALDI-MS screening provided a good estimate on the fraction of **5b** relative to total mono-RL amount produced from single colonies (Figure 9.9), when ion intensities of **5a-d** peaks in MALDI mass spectra were utilized to calculate relative abundances of RL congeners (see SI for details). For the percentiles of **5a**, **5c** and **5d**, however, significant differences were observed in quantification between MALDI-MS screening of colonies and LC-MS analysis of extracts (Figure 9.10). Such discrepancies may result from two sources—congeners may exhibit different ionization efficiency between MALDI-MS and LC-MS (Figure 9.10) or the relative congener transfer efficiency during imprinting differs from solvent extraction. To confirm phenotypes of mutant strains identified with MALDI-screening, liquid cultivation was performed followed by LC-MS/MS without solvent extraction to quantify RL congener abundance.

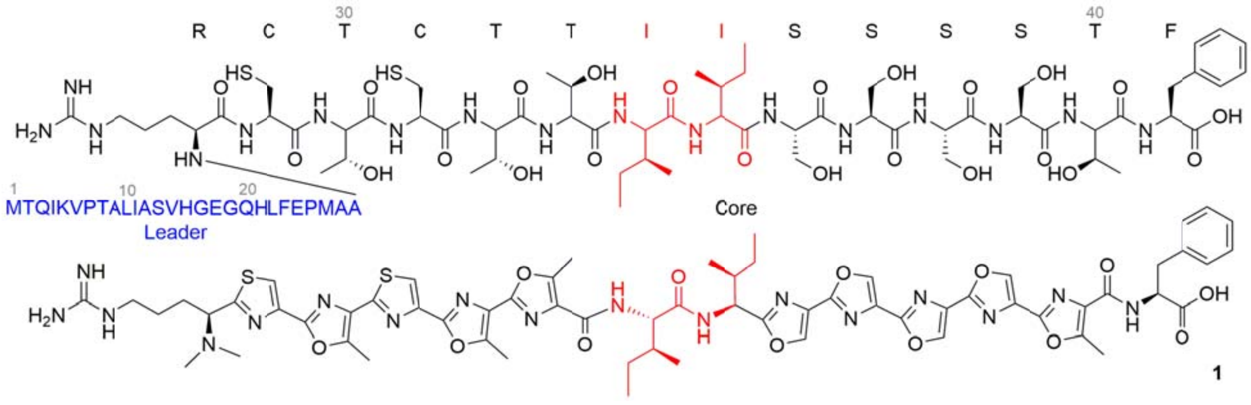
We then applied directed protein evolution to RhlA to engineer relative abundances of **5a-d** in mono-RL production. Random mutations were introduced in the WT *rhlA* gene using error-prone PCR. The PCR product was inserted into a plasmid harboring a WT *rhlB* gene, and the resulting DNA library was used to transform *E. coli* cells. Following the MALDI-MS screening workflow (Figure 9.1), the resulting data sets were visualized by overlaying the optical image with a bubble chart to provide a rapid assessment of the variance in relative abundance and total production of RL molecules (Figure 9.11). Each circle has a radius determined by the log-base 10 intensity of the sum of all RL peaks. The color is determined by the relative abundance of **5b**. Compared with the WT strain (Figure 9.11A), the strain library in the first round of mutagenesis (denoted as R1) exhibited increased diversity in terms of both total intensities of RL ions and relative percentiles of **5b** (Figure 9.11B). These results agree with the description of rhlA as the 'molecular ruler' of RL lipid moiety synthesis.[38] From R1, variant strains producing **5b** at larger fractions relative to WT were recovered with the visual aid of bubble charts (large, red cycles). After plasmid isolation and retransformation into a fresh strain background, two mutant strains (R1#6 and R1#15) were confirmed to produce significantly larger proportions of **5b** than WT in liquid cultures (Figure 9.12). R1#6 and R1#15 each harbors a single amino acid mutation of V10I and A64V, respectively.

The mutated *rhlA* gene from the R1#6 strain was subjected to another round of mutagenesis to further increase relative abundance of **5b**. However, the majority of recovered strains from the second round of screening (R2) were found to contain no additional mutations relative to the parent R1#6. After retransformation, one mutant strain (R2#71) was isolated bearing a single amino acid mutation (L269I) that reduced proportions of **5c** and **5d** in liquid cultures compared with WT, but failed to further enhance the relative abundance of **5b** relative to R1#6 (Figure 9.12). Further investigation is needed to elucidate the mechanisms on why selected mutations confer observed phenotypes.

Conclusions

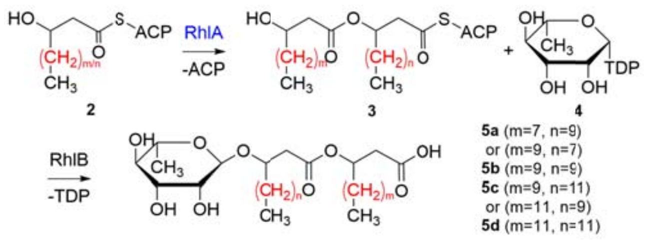
We have developed an integrated workflow for sample preparation, automatic MALDI-MS acquisition, and data processing and visualization for high-throughput screening of multi-step enzymatic reactions in bacterial colonies. MS provides a label-free, highly sensitive platform for monitoring products, reactants, and byproducts with high specificity. Incorporating machine vision and automatic target patterning greatly improves MS acquisition efficiency over traditional MSI assays, especially for randomly distributed colonies. The resulting datasets may be subjected to multivariate clustering or reduced into univariate plots to quickly assess and select mutants with desirable phenotypes. Optically-guided MALDI-MS was successfully applied to screen substrate and enzyme libraries directly from recombinant *E. coli* colonies prepared by standard microbiology methods. The workflow should be applicable to a wide range of multi-step enzyme reactions and facilitate high-throughput screening in microbial systems. Currently, it takes ~1 h to acquire a whole-slide microscopy image of a 25 mm × 75 mm ITO-coated glass target, on which ~1000 colonies can be screened with a MS sampling rate of 1~2.5 s (2,000~10,000 MALDI laser shots) per colony. The upper limit of colony numbers per slide is due to the manual step for mutant recovery, as it becomes more challenging for a human researcher to locate a specific strain with higher colony density. Robotic recovery of mutant colonies may help to overcome this limitation. Further improvement may also be achieved through faster acquisition of optical images, automatic imprinting/matrix coating to enhance sample uniformity, and derivatization of analytes with poor native MALDI-MS sensitivity.

Schemes, Figures, and Table



1

Scheme 9.1. Precursor and structure of **1**.



Scheme 9.2. Biosynthesis of **4**.

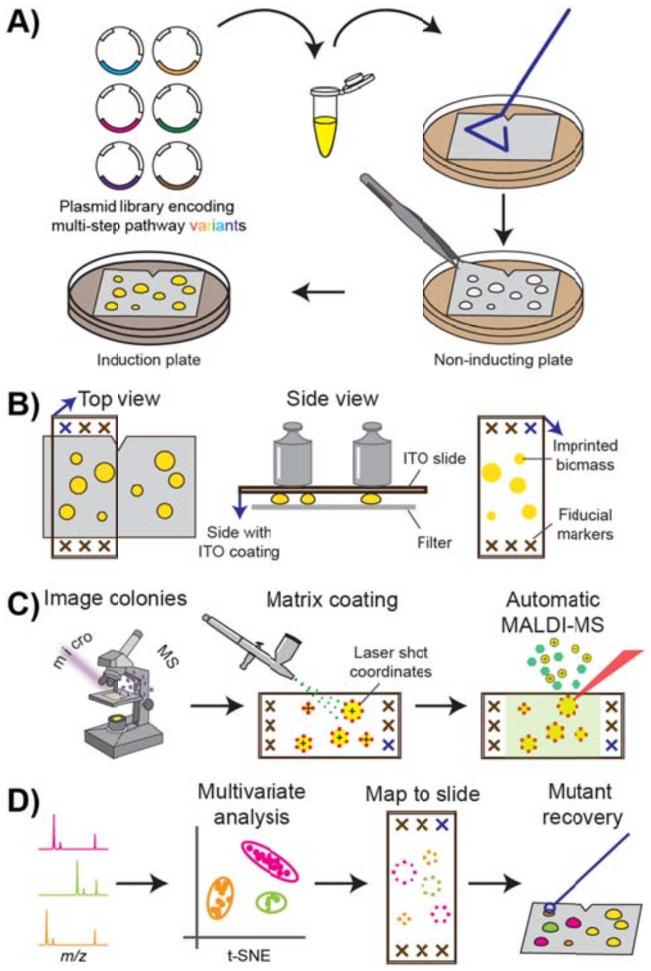
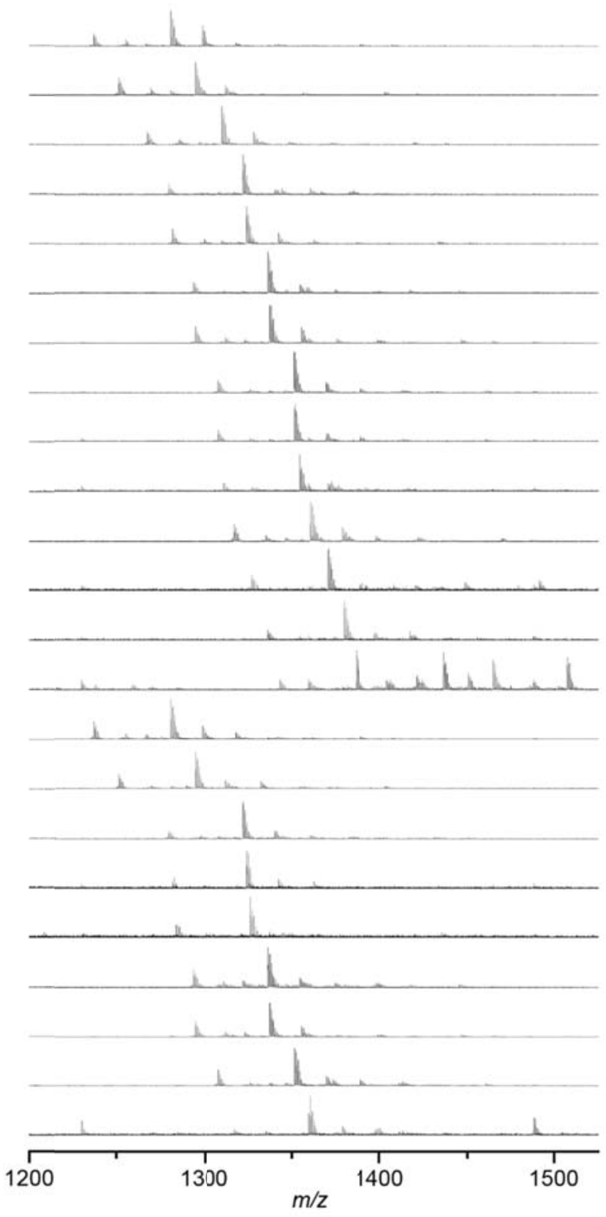


Figure 9.1. Optically-guided MALDI-MS screening. Blue arrows indicate the ITO-coated side of the glass slide.



AA mutaton by MS	Theoretical monoisotopic *m/z* [M+H]⁺	TOF *m/z* [M+H]⁺	FT-ICR *m/z* [M+H]⁺	FT-ICR mass error (ppm)	DNA sequencing	
					Codon	AA
I34G	1280.41490	1280.452			GGT	I34G
I34A	1294.43055	1294.469			GCT	I34A
I34S	1310.42546	1310.541	1310.42424	-0.93	AGT	I34S
I34V	1322.46185	1322.528	1322.46206	0.16	GTT	I34V
I34T	1324.44111	1324.544	1324.44063	-0.37	ACG	I34T
I34L	1336.47750	1336.590	1336.47823	0.55	CTG	I34L
I34N	1337.43636	1337.512			AAT	I34N
I34Q	1351.45201	1351.389	1351.45219	0.13	CAG	I34Q
I34K	1351.48840	1351.571	1351.48848	0.06	AAG	I34K
I34M	1354.43392	1354.522	1354.43408	0.12	ATG	I34M
I34H	1360.45235	1360.441			CAT	I34H
I34F	1370.46185	1370.587			TTT	I34F
I34R	1379.49455	1379.576	1379.49482	0.20	AGG	I34R
I34Y	1386.45676	1386.520	1386.45743	0.48	TAT	I34Y
I35G	1280.41490	1280.521	1280.41506	0.13	GGG	I35G
I35A	1294.43055	1294.533			GCT	I35A
I35V	1322.46185	1322.497			GTT	I35V
I35T	1324.44111	1324.544	1324.44065	-0.35	ACT	I35T
I35C	1326.40262	1326.520			TGT	I35C
I35L	1336.47750	1336.536			CTT	I35L
I35N	1337.43636	1337.538			AAT	I35N
I35Q	1351.45201	1351.510	1351.45201	0.81	CAG	I35Q
I35H	1360.45235	1360.596			CAT	I35H

Figure 9.2. Summary of PZN analogs observed in this study. For each **1** analog, a typical MALDI-ToF mass spectrum of the *E. coli* colony is included on the left. The table on the right summarizes the information of **1** analogs including AA mutation assignment based on mass spectra, theoretical and measured monoisotopic *m/z* value of the $[M+H]^+$ ion (by MALDI-ToF or MALDI-FT-ICR), and DNA sequencing results. High-resolution FT-ICR was performed on select colonies from the same sample target subsequent to MALDI-MS screening, focusing on tentative Q and K mutants, as well as select **1** analogs (shown in red) not reported in a previous study[25].

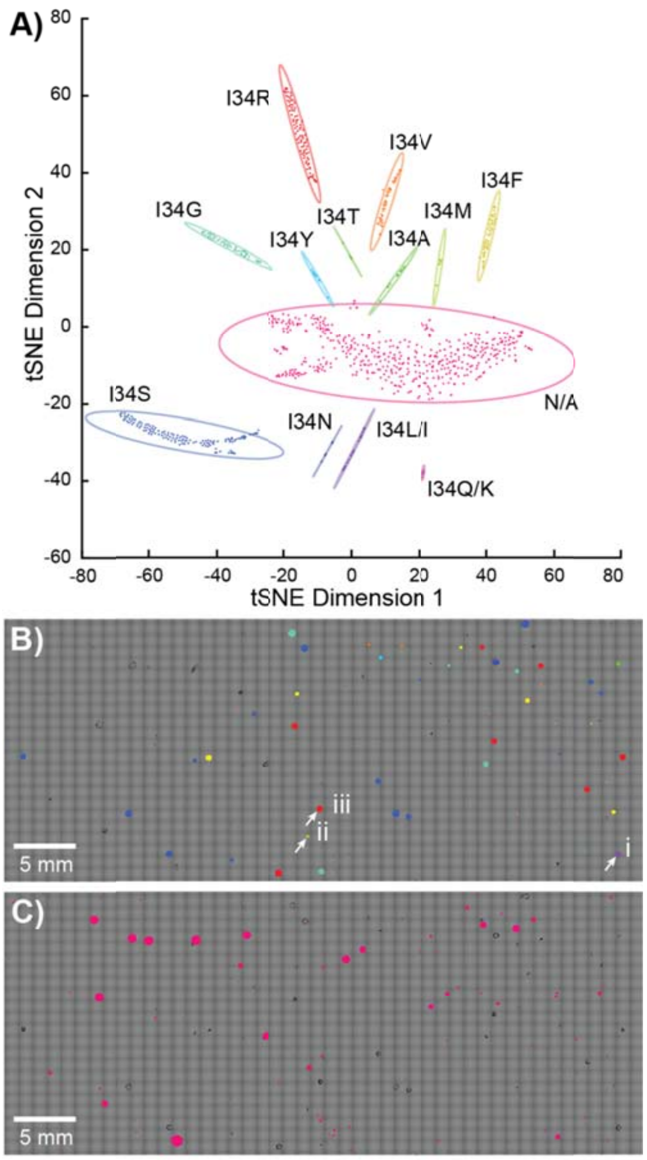


Figure 9.3. Multivariate analysis of PZN analogs. (A) Visualization with targeted t-SNE clustering of the I34 library from a single experiment. Each point corresponds to a single mass spectrum with each cluster surrounded by a 95% confidence ellipsoid. The N/A cluster contains spectra without observable peptide signals. The position of each mutant (B) or N/A colony (C) is mapped onto the optical image to aid mutant recovery. Three colonies are highlighted in panel B which are displayed in more detail in Figure 9.8.

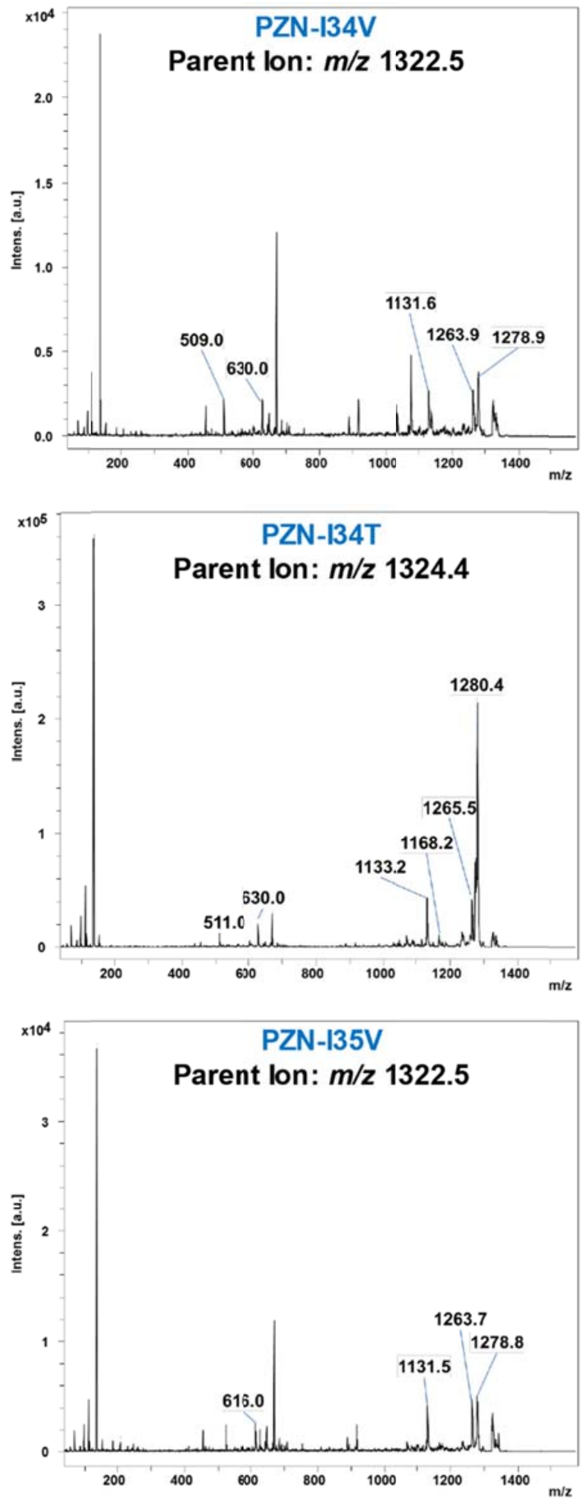
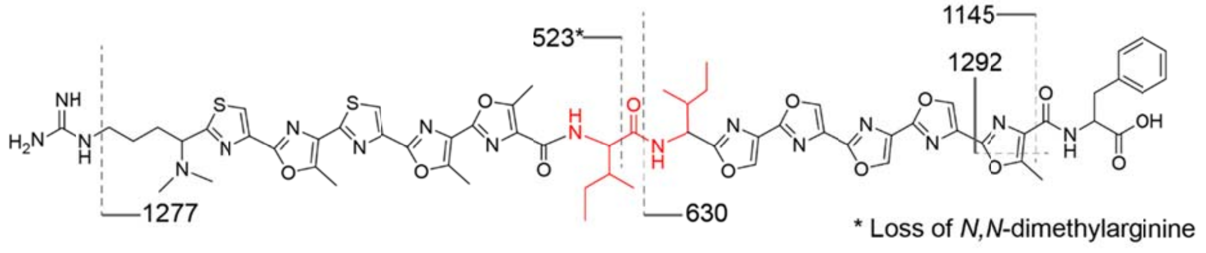
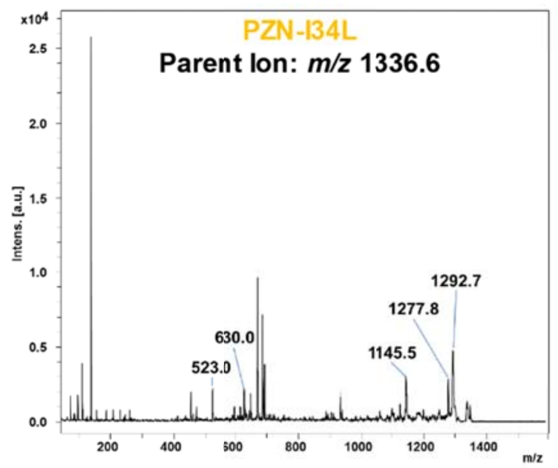


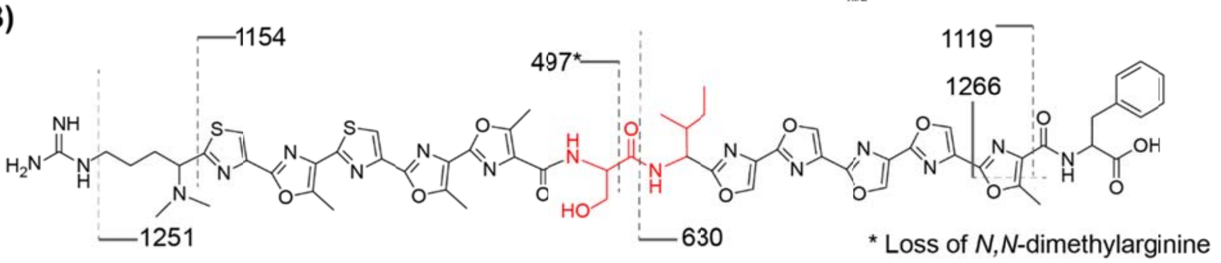
Figure. 9.4. In situ MALDI-ToF/ToF tandem mass spectra of PZN-I34V, PZN-I34T and PZN-I35V. These analogs were reported previously[25] with tandem mass spectra (denoted as blue). Peaks with *m/z* values consistent with previous results were labeled.

A)





B)



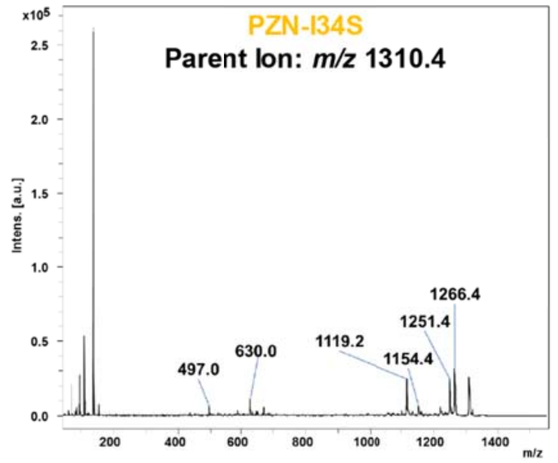
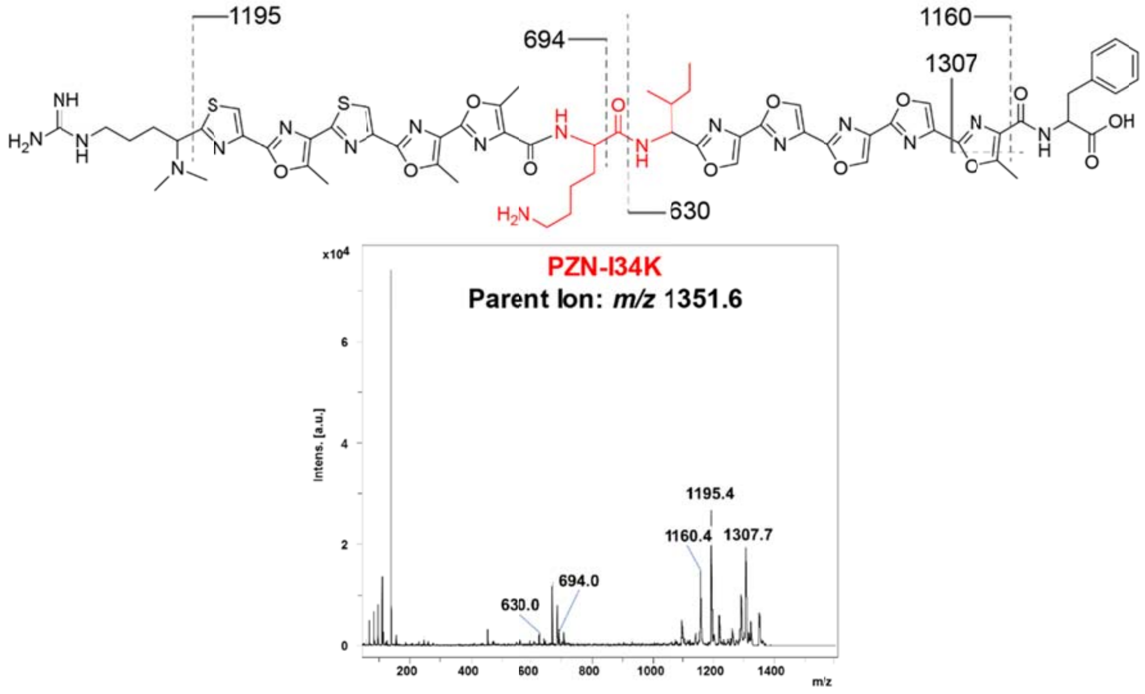


Figure 9.5. *In situ* MALDI-ToF/ToF tandem mass spectra of (A) PZN-I34L and (B) PZN-I34S. These analogs were reported previously[25] without tandem mass spectra (denoted as orange). Fragments derived from multiple bond cleavages are denoted by asterisks.

A)



B)

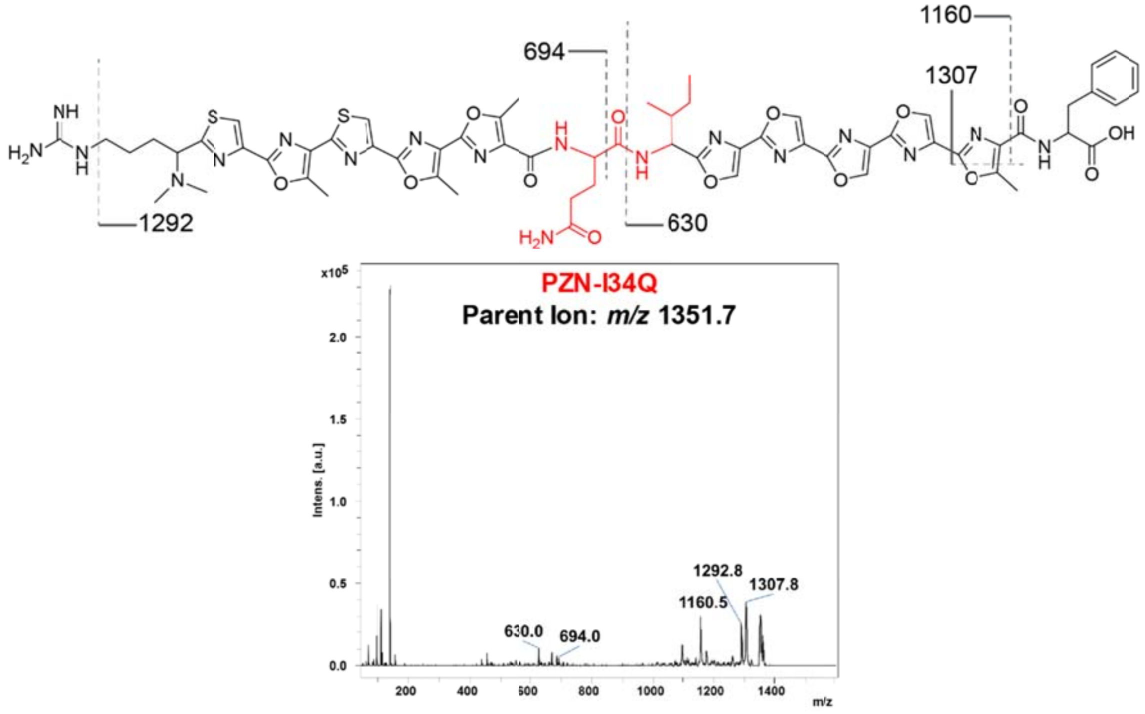
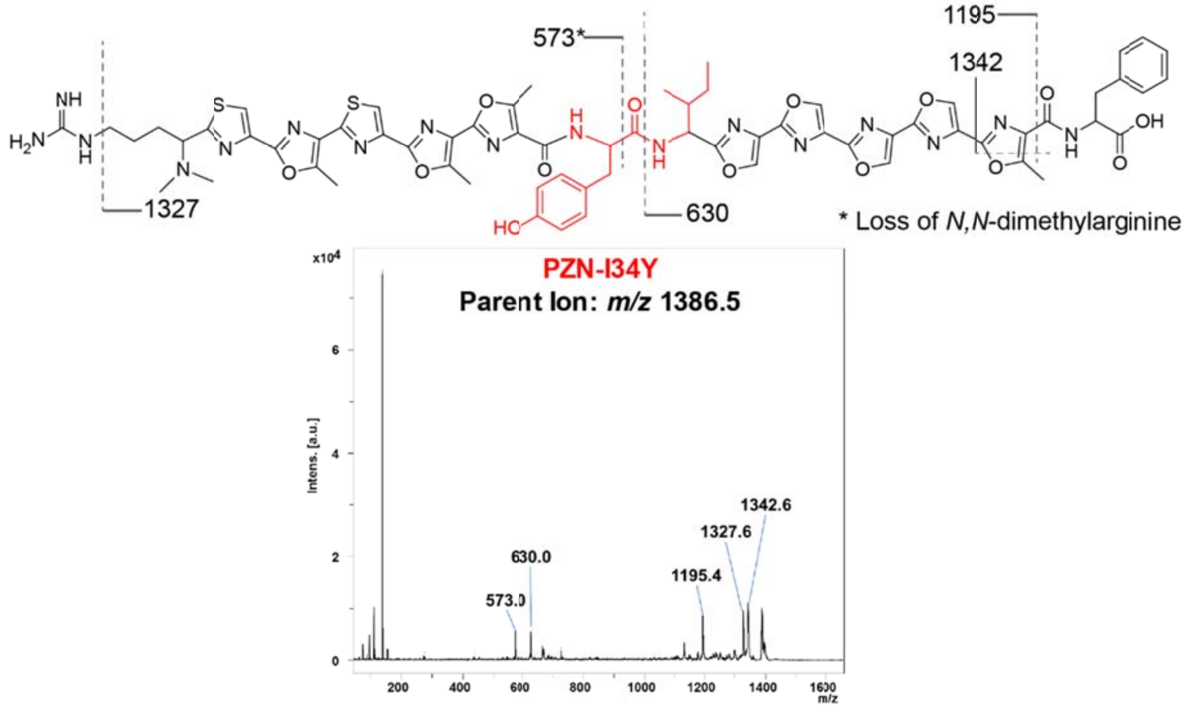


Figure 9.6. *In situ* MALDI-ToF/ToF tandem mass spectra of (A) PZN-I34K and (B) PZN-I34Q. These analogs were not reported previously[25] (denoted as red).

A)



B)

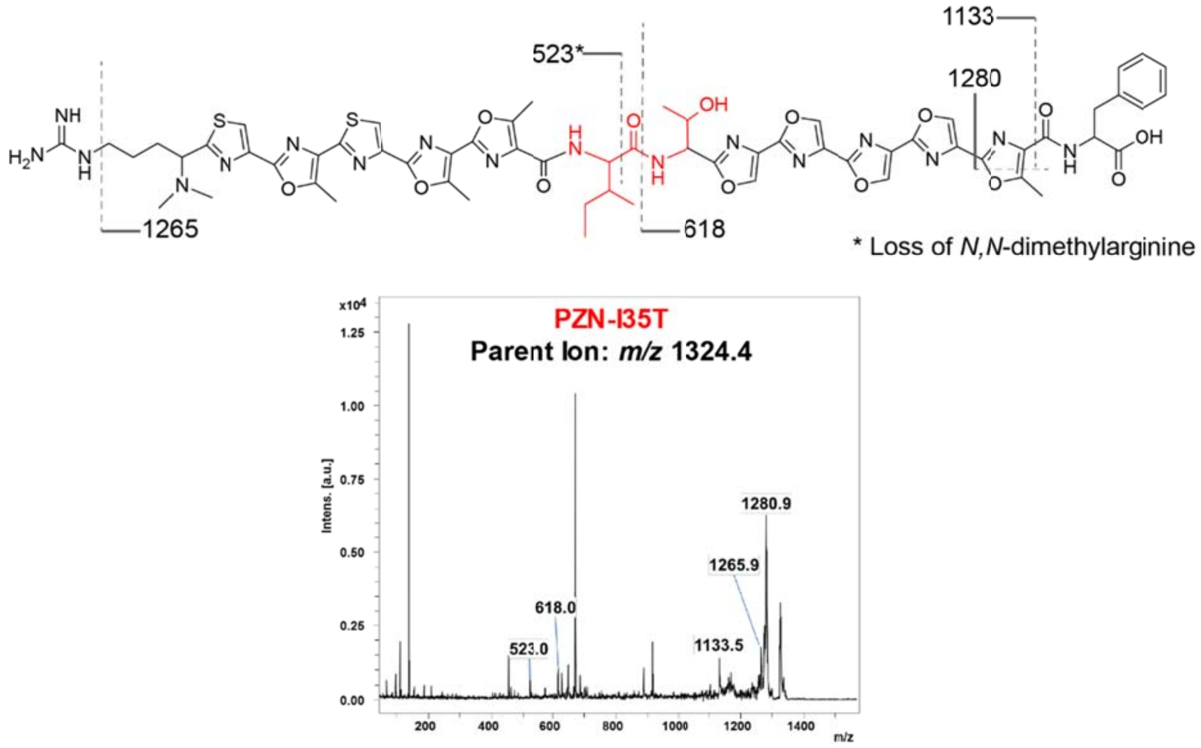


Figure 9.7. *In situ* MALDI-ToF/ToF tandem mass spectra of (A) PZN-I34Y and (B) PZN-I35T. These analogs were not reported previously[25] (denoted as red). Fragments derived from multiple bond cleavages are denoted by asterisks.

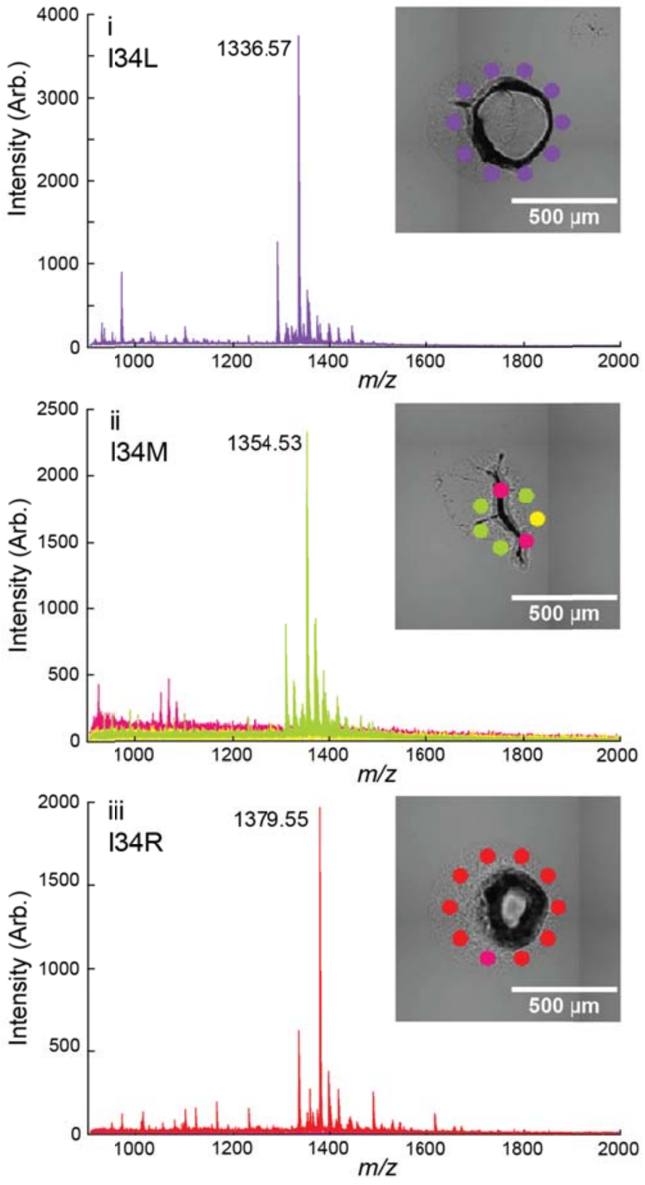
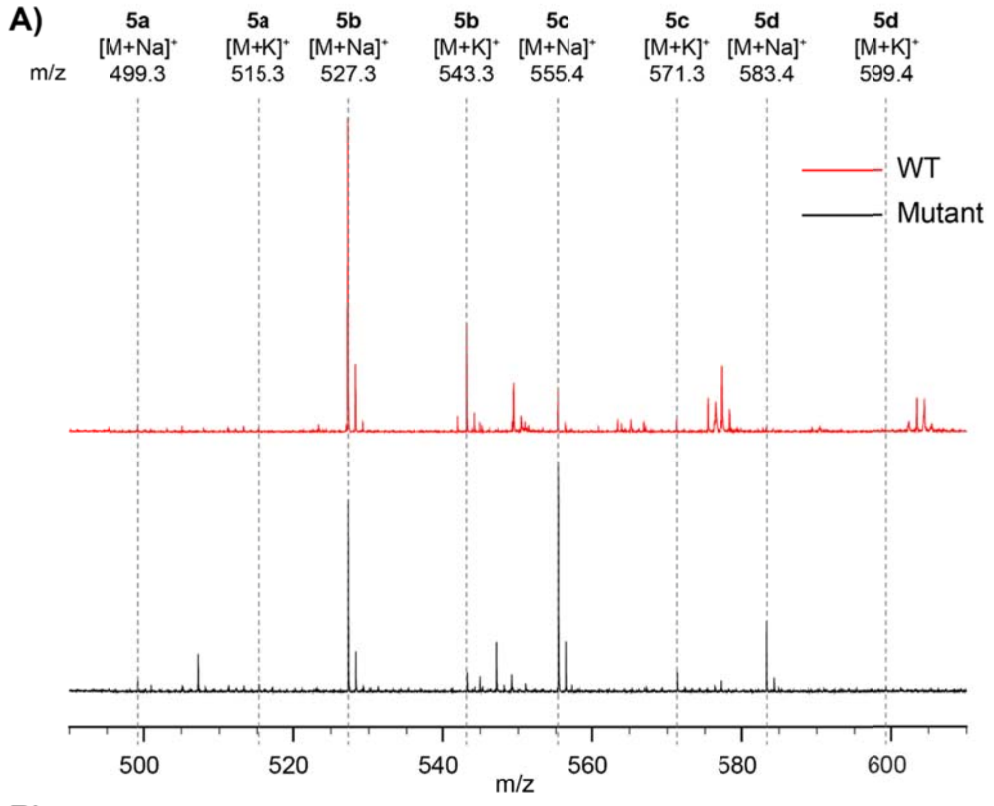


Figure 9.8. Detailed view of annotated colonies from Figure 9.3. The color of each cluster corresponds to the t-SNE plot in Figure 9.3. Average spectra of each cluster for the colony are displayed with the base peak labeled. Mutations were confirmed by DNA sequencing.



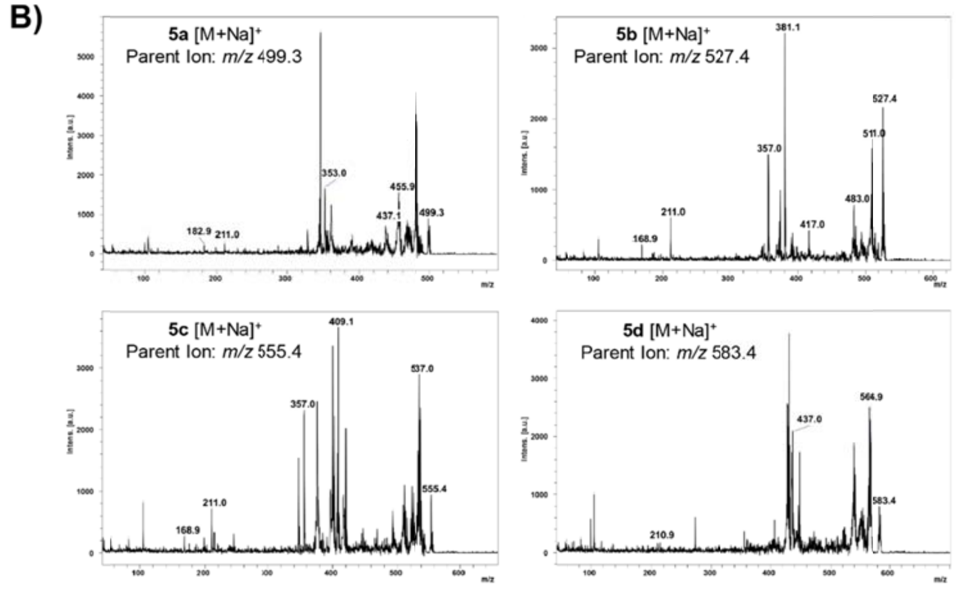


Figure 9.9. MALDI mass spectra of mono-RLs produced from *E. coli* colonies. (A) Typical colony-MALDI-ToF mass spectra from WT and a mutant strain. The mutant produced greater portions of **5c** and **5d** relative to WT. Tentative peak assignments of **5a-d** are labeled with dashed lines. (B) *In situ* LIFT ToF/ToF analysis on tentative sodiated ions of **5a-d**. Fragment ions with *m/z* values consistent with previous reports[15,16] were labeled.

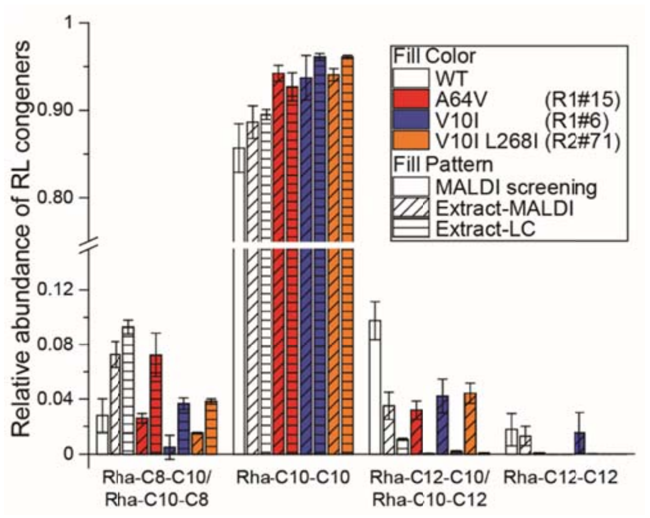


Figure 9.10. Relative abundance of RL congeners produced from colonies following IPTG induction. Optically-guided MALDI-MS was performed on WT colonies. For WT and each mutant strain, ~100 colonies on a filter after IPTG induction were subjected to organic solvent extraction, and the extract was analyzed using either MALDI-MS or LC-MS/MS. The fraction of a RL species relative to total RL amount was calculated as described in Methods. Error bars indicate standard deviations of biological replicates (n=81) for MALDI-MS screening, or technical triplicates (n=3) for MALDI-MS or LC-MS/MS measurement of the same extract.

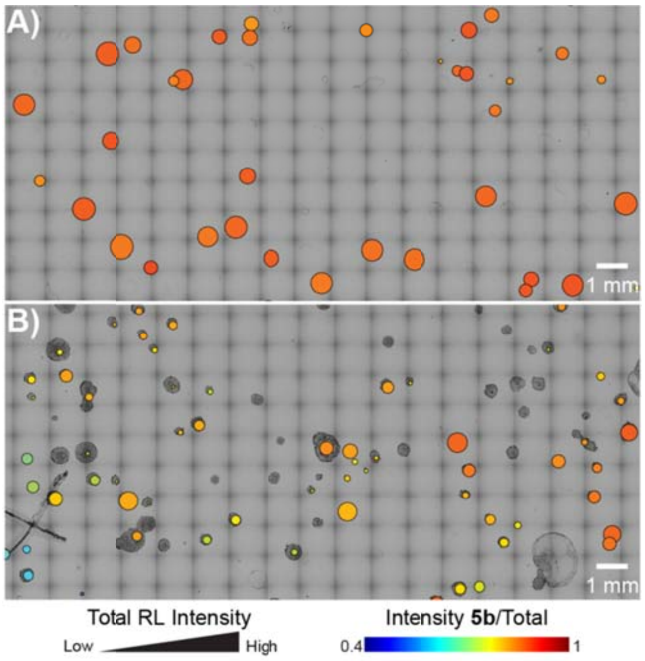


Figure 9.11. Visualization of MALDI-MS screening results of mono-RL-producing *E. coli* colonies. (A) WT. (B) Strain library in the first round of mutagenesis (R1).

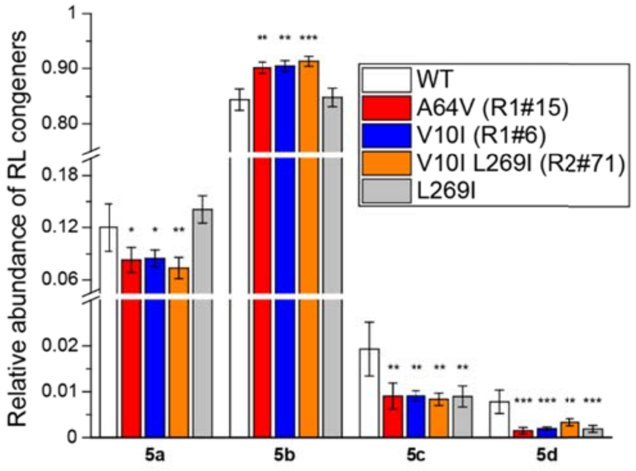


Figure 9.12. Comparison of RL production in liquid cultures between WT and isolated mutant strains quantified using LC-MS/MS in MRM mode. Error bars indicate the standard deviations of biological triplicates. Significant differences were determined between WT and mutants using an independent two-tailed, two-sample *t*-test for equal sample sizes and equal variance. Significance levels: * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$.

Table 9.1. DNA sequences in this study.

Primers		
NP5	T7-PZN-2 For	tataagaaggagatatacatgtgaaaattcactacatgggag
NP6	T7-PZN-2 Rev	tcgcgtggccggccgatatctcacgtatacctttttgtttttttataatcc
NP7	PZN-2-mid For	gatgtgaattcttctccgag
NP8	PZN-2-mid Rev	ctcggagaagaattcacatc
NP119	ptnA-I34 Rev	tgtggtacaggtacagcgtg
NP124	pET28-Mid For2	catcctgcgatgcagatccggaacataatggtg
NP125	pET28-Mid Rev2	ctgcatcgcaggatgctgc
NP127	ptnA-I34-NNK For	cacgctgtacctgtaccacannkatctctagttcatctacgttttaagcgg
NP128	ptnA-I35-NNK For	cacgctgtacctgtaccacaatcnnktctagttcatctacgttttaagcgg
RL021	pRSF-MCSI-RhlB For	atcaccacagccaggatccgaattcgatgcacgccatcctcatc
RL022	MCSI-RhlB Rev	ttaagcattatgcggccgcaagctttcaggacgcagccttcag
RL023	MCSII-RhlA For	atatacatatggcagatctcaattggatgcggcgcgaaagtctg
RL024	MCSII-RhlA Rev	tttaccagactcgagggtacctcaggcgtagccgatggc

References

(1) Dietrich, J. A.; McKee, A. E.; Keasling, J. D. *Annu Rev Biochem* **2010**, *79*, 563.

(2) Romero, P. A.; Arnold, F. H. *Nat Rev Mol Cell Biol* **2009**, *10*, 866.

(3) Macarron, R.; Banks, M. N.; Bojanic, D.; Burns, D. J.; Cirovic, D. A.; Garyantes, T.; Green, D. V.; Hertzberg, R. P.; Janzen, W. P.; Paslay, J. W.; Schopfer, U.; Sittampalam, G. S. *Nat Rev Drug Discov* **2011**, *10*, 188.

(4) Bothner, B.; Chavez, R.; Wei, J.; Strupp, C.; Phung, Q.; Schneemann, A.; Siuzdak, G. *J Biol Chem* **2000**, *275*, 13455.

(5) Greis, K. D. *Mass Spectrom Rev* **2007**, *26*, 324.

(6) Northen, T. R.; Lee, J.-C.; Hoang, L.; Raymond, J.; Hwang, D.-R.; Yannone, S. M.; Wong, C.-H.; Siuzdak, G. *Proceedings of the National Academy of Sciences* **2008**, *105*, 3678.

(7) Ban, L.; Pettit, N.; Li, L.; Stuparu, A. D.; Cai, L.; Chen, W.; Guan, W.; Han, W.; Wang, P. G.; Mrksich, M. *Nat Chem Biol* **2012**, *8*, 769.

(8) Heins, R. A.; Cheng, X.; Nath, S.; Deng, K.; Bowen, B. P.; Chivian, D. C.; Datta, S.; Friedland, G. D.; D'Haeseleer, P.; Wu, D.; Tran-Gyamfi, M.; Scullin, C. S.; Singh, S.; Shi, W.; Hamilton, M. G.; Bendall, M. L.; Sczyrba, A.; Thompson, J.; Feldman, T.; Guenther, J. M.; Gladden, J. M.; Cheng, J. F.; Adams, P. D.; Rubin, E. M.; Simmons, B. A.; Sale, K. L.; Northen, T. R.; Deutsch, S. *ACS Chem Biol* **2014**, *9*, 2082.

(9) Gurard-Levin, Z. A.; Scholle, M. D.; Eisenberg, A. H.; Mrksich, M. *ACS Comb Sci* **2011**, *13*, 347.

(10) Greving, M.; Cheng, X.; Reindl, W.; Bowen, B.; Deng, K.; Louie, K.; Nyman, M.; Cohen, J.; Singh, A.; Simmons, B.; Adams, P.; Siuzdak, G.; Northen, T. *Analytical and Bioanalytical Chemistry* **2012**, *403*, 707.

11) de Rond, T.; Danielewicz, M.; Northen, T. *Current Opinion in Biotechnology* **2015**, *31*, 1.

(12) Rodrigues, T.; Reker, D.; Schneider, P.; Schneider, G. *Nat Chem* **2016**, *8*, 531.

(13) Newman, D. J.; Cragg, G. M. *J Nat Prod* **2012**, *75*, 311.

(14) Kim, E.; Moore, B. S.; Yoon, Y. J. *Nat Chem Biol* **2015**, *11*, 649.

(15) Masyuko, R. N.; Lanni, E. J.; Driscoll, C. M.; Shrout, J. D.; Sweedler, J. V.; Bohn, P. W. *Analyst* **2014**, *139*, 5700.

(16) Lanni, E. J.; Masyuko, R. N.; Driscoll, C. M.; Aerts, J. T.; Shrout, J. D.; Bohn, P. W.; Sweedler, J. V. *Anal Chem* **2014**, *86*, 9139.

(17) Si, T.; Li, B.; Zhang, K.; Xu, Y.; Zhao, H.; Sweedler, J. V. *J Proteome Res* **2016**, *15*, 1955.

(18) Yang, Y. L.; Xu, Y.; Straight, P.; Dorrestein, P. C. *Nat Chem Biol* **2009**, *5*, 885.

(19) Dunham, S. J.; Ellis, J. F.; Li, B.; Sweedler, J. V. *Acc Chem Res* **2017**, *50*, 96.

(20) Watrous, J. D.; Dorrestein, P. C. *Nat Rev Microbiol* **2011**, *9*, 683.

(21) Yan, C.; Parmeggiani, F.; Jones, E. A.; Claude, E.; Hussain, S. A.; Turner, N. J.; Flitsch, S. L.; Barran, P. E. *J Am Chem Soc* **2017**, *139*, 1408.

(22) Ong, T. H.; Kissick, D. J.; Jansson, E. T.; Comi, T. J.; Romanova, E. V.; Rubakhin, S. S.; Sweedler, J. V. *Anal Chem* **2015**, *87*, 7036.

(23) Jansson, E. T.; Comi, T. J.; Rubakhin, S. S.; Sweedler, J. V. *ACS Chem Biol* **2016**, *11*, 2588.

(24) Do, T. D.; Comi, T. J.; Dunham, S. J.; Rubakhin, S. S.; Sweedler, J. V. *Anal Chem* **2017**, *89*, 3078.

(25) Deane, C. D.; Melby, J. O.; Molohon, K. J.; Susarrey, A. R.; Mitchell, D. A. *ACS Chem Biol* **2013**, *8*, 1998.

(26) Maaten, L. v. d.; Hinton, G. *Journal of Machine Learning Research* **2008**, 2579.

(27) Cornvik, T.; Dahlroth, S. L.; Magnusdottir, A.; Herman, M. D.; Knaust, R.; Ekberg, M.; Nordlund, P. *Nat Methods* **2005**, *2*, 507.

(28) Skinnider, M. A.; Johnston, C. W.; Edgar, R. E.; Dejong, C. A.; Merwin, N. J.; Rees, P. N.; Magarvey, N. A. *Proc Natl Acad Sci U S A* **2016**, *113*, E6343.

(29) Arnison, P. G.; Bibb, M. J.; Bierbaum, G.; Bowers, A. A.; Bugni, T. S.; Bulaj, G.; Camarero, J. A.; Campopiano, D. J.; Challis, G. L.; Clardy, J.; Cotter, P. D.; Craik, D. J.; Dawson, M.; Dittmann, E.; Donadio, S.; Dorrestein, P. C.; Entian, K. D.; Fischbach, M. A.; Garavelli, J. S.; Goransson, U.; Gruber, C. W.; Haft, D. H.; Hemscheidt, T. K.; Hertweck, C.; Hill, C.; Horswill, A. R.; Jaspars, M.; Kelly, W. L.; Klinman, J. P.; Kuipers, O. P.; Link, A. J.; Liu, W.; Marahiel, M. A.; Mitchell, D. A.; Moll, G. N.; Moore, B. S.; Muller, R.; Nair, S. K.; Nes, I. F.; Norris, G. E.; Olivera, B. M.; Onaka, H.; Patchett, M. L.; Piel, J.; Reaney, M. J.; Rebuffat, S.; Ross, R. P.; Sahl, H. G.; Schmidt, E. W.; Selsted, M. E.; Severinov, K.; Shen, B.; Sivonen, K.; Smith, L.; Stein, T.; Sussmuth, R. D.; Tagg, J. R.; Tang, G. L.; Truman, A. W.; Vederas, J. C.; Walsh, C. T.; Walton, J. D.; Wenzel, S. C.; Willey, J. M.; van der Donk, W. A. *Nat Prod Rep* **2013**, *30*, 108.

(30) Young, T. S.; Dorrestein, P. C.; Walsh, C. T. *Chem Biol* **2012**, *19*, 1600.

(31) Ruffner, D. E.; Schmidt, E. W.; Heemstra, J. R. *ACS Synth Biol* **2015**, *4*, 482.

(32) Scholz, R.; Molohon, K. J.; Nachtigall, J.; Vater, J.; Markley, A. L.; Sussmuth, R. D.; Mitchell, D. A.; Borriss, R. *J Bacteriol* **2011**, *193*, 215.

(33) Molohon, K. J.; Blair, P. M.; Park, S.; Doroghazi, J. R.; Maxson, T.; Hershfield, J. R.; Flatt, K. M.; Schroeder, N. E.; Ha, T.; Mitchell, D. A. *ACS Infect Dis* **2016**, *2*, 207.

(34) Nov, Y. *Appl Environ Microbiol* **2012**, *78*, 258.

(35) Howe, J.; Bauer, J.; Andra, J.; Schromm, A. B.; Ernst, M.; Rossle, M.; Zahringer, U.; Rademann, J.; Brandenburg, K. *FEBS J* **2006**, *273*, 5101.

(36) Dobler, L.; Vilela, L. F.; Almeida, R. V.; Neves, B. C. *New Biotechnol* **2016**, *33*, 123.

(37) Muller, M. M.; Kugler, J. H.; Henkel, M.; Gerlitzki, M.; Hormann, B.; Pohnlein, M.; Syldatk, C.; Hausmann, R. *J Biotechnol* **2012**, *162*, 366.

(38) Zhu, K.; Rock, C. O. *J Bacteriol* **2008**, *190*, 3147.

(39) Abdel-Mawgoud, A. M.; Lepine, F.; Deziel, E. *Appl Microbiol Biotechnol* **2010**, *86*, 1323.

(40) Han, L.; Liu, P.; Peng, Y.; Lin, J.; Wang, Q.; Ma, Y. *J Appl Microbiol* **2014**, *117*, 139.

(41) Cabrera-Valladares, N.; Richardson, A. P.; Olvera, C.; Trevino, L. G.; Deziel, E.; Lepine, F.; Soberon-Chavez, G. *Appl Microbiol Biotechnol* **2006**, *73*, 187.

APPENDIX A

SELECTED SOURCE CODE

Notes and Acknowledgements

The following are a collection of source code used throughout the preceding chapters and for other projects which did not appear in this dissertation but could be useful in the future. The organization follows the presentation through the dissertation where possible. Code is heavily commented but additional notes will appear when necessary along with a brief introduction on the motivation. Source code has been formatted to ease reading and may not run by simply copying and pasting from the document due to line numbers and indenting.

Simulation of Analyte Movement During DESI-MSI

Motivation, Overview and Extensions

During the development of synchronized DESI MSI, a question that was commonly posed is what effect the synchronization had on spatial resolution of the resulting image. A complication to the question is that the effect of fundamental properties of an analyte system on imaging resolution had not been developed in a systematic way. While fluid simulations of droplets impacting a thin layer were performed and extrapolated for the desorption process of DESI, simulations of analyte movement on a surface had not been performed. As an initial attempt to understand the DESI imaging process, we developed a simplified model considering only desorption and washing (redistribution) under the movement of a circular DESI spray. The details of the model may be found in Chapter 4, here is the complete source code for the simulation.

While image convolution to assess aperture effects is well developed for optical systems, DESI is fundamentally different due to the spray interactions with the analyte of interest. In addition to acquiring an average intensity from the spray plume, analytes must dissolve into the thin liquid film which develops during the DESI process. Within the film, compounds of interest are removed by desorption (some amount forms gas phase ions which are ultimately detected) while those remaining on the surface may redistribute. The influence of the nebulizing gas accelerates this process, causing rivulets of solvent and delocalization, especially when probing smooth surfaces. Due to the resemblance of cleaning a surface with a pressure washer, this phenomenon has been named the “washing effect”.

As it pertains to MSI, the washing effect can cause smearing to the leading edge of the plume or redistribution to the next row. The choice of a sampling period (pixel size) must be considered against the longer period of time for both washing and desorption of material. In contrast to MALDI MSI, oversampling with DESI is frequently unsuccessful as analytes may be removed from the outer ridges of the desorption area before they enter the area with high ionization efficiency. As such, there appears to be a minimum pixel size for a given sample and set of operating parameters, below which no image is produced. The minimum size is larger than would be expected based on sensitivity and pixel size, and appears to be governed largely by the washing effect. Synchronizing desorption with ion injection appeared to improve the situation by preventing analyte delocalization when the mass analyzer was not actively collecting ions. To better understand the mechanisms behind apparent improvements in resolution, the washing and desorbing efficiencies were adjusted with both continuous and synchronized DESI sources.

The simulation code is written in MATLAB. The main method is batchRun, which performs several simulations with altered parameters in each set. Following the completion of each MSI simulation, the image distributions and a video of the simulation are saved. At the start of batchRun, Initialize is run to set all imaging parameters and read in the input image. The actual simulation is performed by MSIsimulation, which steps through an entire MSI run, iteratively updating the analyte distribution and recording a simulated output. MSIsimulation utilizes the helper functions cosDistribution and UpdateIntensities. cosDistribution generates a cosine distributed probability function which forms the washing efficiency distribution. UpdateIntensities performs a single time step of analyte desorption and migration on the input image.

The simulation was successful in determining how washing and desorption affected output image intensity and as a phenomenological model, captured many experimental results. Improvements could be made on the formulation of the model, to define unitless parameters as physical, measureable properties of the analytes and surface. The spray profile could also be refined into an ellipse, which would more closely mimic actual DESI sprays. The washing produces an artifact which moves analyte to the corners more than should be expected, causing slight deviations from circular symmetry. Implementing a similar model in a hexagonal basis set would eliminate some of the issues with rectangular pixels.

batchRun.m

```
%refresh workspace
close all
clear

%the base filename for all output
start = 'pulImg100_';

%load image and simulation parameters
Initialize;

%experiment 1, pulsed with specified efficiencies
continuous = false;
spotEff = .005;
ionEff = .01;
washingEff = 1;

%perform simulation to populate msi, alter inputImg, and figure movie (F)
MSIsimulation;

%save output images
save([start '005_01_1_p.mat'], 'msi');
save([start '005_01_1_pPost.mat'], 'inputImg');

%save figure movie
wo = VideoWriter([start '005_01_1_p.avi']);
wo.FrameRate = 10;
open(wo);
writeVideo(wo,F);
close(wo)

%experiment 2, continuous with efficiencies from above
continuous = true;

%perform simulation to populate msi, alter inputImg, and figure movie (F)
MSIsimulation;

%save output images
save([start '005_01_1_c.mat'], 'msi');
save([start '005_01_1_cPost.mat'], 'inputImg');

%save figure movie
wo = VideoWriter([start '005_01_1_c.avi']);
wo.FrameRate = 10;
open(wo);
writeVideo(wo,F);
close(wo)

%etc...
```

Initialize.m

```
%all units in µm, s, and fractional

%sizes
spotD = 250; %spot diameter
spotSTD = 125; %spot stdev/dropoff
ionD = 25; %ionization diameter
ionSTD = 25; %ionization dropoff
T = 20; %sampling period/pixel size

%depletion efficiencies
%.1 .5 good for slick
%.01 .05 for tissue
%high wash 5.625 low 0.01
spotEff = .01;%removal from spot
ionEff = .05;%removal from ionization area, not accounting spot
washingEff = .01;%washing effect movement

%times
deltaT = 0.1; %simulation time step
MST = 2.2; %scan length (excluding ion injection)
IT = 0.1; %ion injection time

%read image, convert to grayscale
% test = 256-transpose(double(rgb2gray(imread('swirl.jpg', 'JPEG'))));
% imgDPI = 9600;
% test = 256-transpose(double(imread('iso-rag.jpg', 'JPEG')));
% imgDPI = 478;
% test = 256-transpose(double(rgb2gray(imread('AF-TT-4x6.jpg', 'JPEG'))));
% imgDPI = 1200;
test = double((imread('pulsedInput.png')));

%find some derived values
% pixelSize = 25400/imgDPI;
pixelSize = 1000/460;

%how quickly to move the plume (µm/s) for the given pixel size and scan
time
scanRate = T/(MST+IT);
%direction of neighboring pixels
directions = [ 1 0; 1 1; 0 1; -1 0; -1 -1; 0 -1; 1 -1; -1 1];
%unit vectors of each direction.
dirUnit = [ 1 0; 1/sqrt(2) 1/sqrt(2); 0 1; -1 0; -1/sqrt(2)
   -1/sqrt(2); 0 -1; 1/sqrt(2) -1/sqrt(2); -1/sqrt(2) 1/sqrt(2)];

continuous = true;%if spray is continuous or not

%how to distribute analytes in the center most pixel
centerUnit = [1 1 1 1 1 1 1 1];
```

MSIsimulation.m

```
%generate masks of intensities, 50% larger than spotD for washing

%spot is the combined effect of desorption and ionization
%Gaussian of specified size
spot = fspecial('gaussian', round(spotD/pixelSize*2),
  round(spotSTD/pixelSize));
%normalize and scale by efficiency
spot = spot/max(spot(:))*spotEff;

%ionization leads to signal output along with desorption
ion = fspecial('gaussian', round(spotD/pixelSize*2),
  round(ionSTD/pixelSize));
ion = ion / max(ion(:))*ionEff;
%include ionization with desorption (spot)
spot=spot+ion;

%washing mask with cosine distribution
wash = cosDistribution(round(spotD/pixelSize*2),
              round(4*ionSTD/pixelSize))*washingEff;

%dot products of each neighbor for washing
dots = zeros(size(spot,1),size(spot,2),8);

%washing shouldn't change through iterations, just the position which is
%taken care of in update

%middle position
middle = ceil(size(spot)/2);
%make radial vectors for each spot in mask
%shift row and column numbers to place middle at 0
rr = (1:size(spot,1))-middle(1);
rc = (1:size(spot,2))-middle(2);

%populate dots with the dot product of radial vector with unit
for ii = 1:8
    [x,y] = meshgrid(rc*dirUnit(ii,2),rr*dirUnit(ii,1));
    dots(:,:,ii) = x+y;
end

%set middle of mask to the centerUnit
dots(middle(1),middle(2),:) = centerUnit;

%normalize dot products, temp is the sum
tempDots = zeros(size(dots));
%remove negative values (would indicate movement into pixel)
dots(dots<0)= 0;
%should be a repmat here
tempDots(:,:,1) = sum(dots,3);
for ii = 2 :8
    tempDots(:,:,ii) = tempDots(:,:,ii-1);
end
dots = dots./tempDots;

current = [1 1]; %current/start position in top left
currentT = 0; %current time
```

```
%output intensity image
msi = zeros(round(size(inputImg)*pixelSize/T));

%add border to input image to facilitate masking edges
inputImg = zeros(size(test) + ceil(size(spot)/2));
%copy over test to prevent repeated reading
inputImg(1:size(test,1),1:size(test,2)) = test;

%clear F movie from previous runs
clear F;
%index counter
ii =1;
%handle to output image
h = figure('units','normalized','outerposition',[0 0 1 1]);
%start timer to track elapsed time
tic
%while the current probe position is within the vertical bounds of the
  image
while current(2) < size(inputImg,2)*pixelSize
  %while probe in horizontal bounds
    while current(1) < size(inputImg,1)*pixelSize
    %  convert current (μm) to pixel position
       pix = ceil(current/pixelSize);
    %  convert current to pixel of output image
       outPix = ceil(current/T);
    %  check if in bounds
       if(pix(1)+size(ion,1)-1 < size(inputImg,1) &&...
              pix(2)+size(ion,2)-1 < size(inputImg,2))
           %  if in IT, add to output pixel
           if(mod(currentT,(MST+IT)) < IT)
                %add intensity to output
              msi(outPix(1),outPix(2)) =  msi(outPix(1),outPix(2)) + ...
                                                %previous intens
                        %analyte intens, scaled by ionization
                  sum(sum(inputImg(pix(1):pix(1)+size(ion,1)-1,
                                pix(2):pix(2)+size(ion,2)-1) .* ion))...
                  %times deltaT
                        *deltaT;
                %update intensities on input
              UpdateIntensities;
           %For non-continuous DESI, this is skipped
           elseif(continuous)
              UpdateIntensities;
           end
       end

     %move probe position in x direction
       current(1) = current(1) + scanRate*deltaT;
     %update time
       currentT = currentT + deltaT;
    end
  %probe has run over the x bounds
  %reset x to start
    current(1) = 1;
  %step y by T
    current(2) = current(2) + T;
  %display percent completed after each row
```

```
    current(2)/(size(inputImg,2)*pixelSize)*100
  %make currentT the next scan of the instrument to ensure first column
  is the start of a scan
    currentT = ceil(currentT/2.3)*2.3;
  %update figure every 10th iteration
    if mod(ii,10) == 0
     subplot(1,2,1);
     %show input
     imshow(transpose(imresize(inputImg,size(msi)*2)), [0 256]);
     subplot(1,2,2);
     %show output
     imshow(transpose(imresize(mat2gray(msi),size(msi)*2,'bicubic')));
     %record frame
     F(ii/10) = getframe(h);
    end
  %update iteration counter
    ii = ii+1;
  %report elapsed time
  toc
end

```

cosDistribution.m

```
function [ dist ] = cosDistribution( SIZE, radius)
%Generates a cosine distribution mask of given SIZE with
%radius = the zero crossing of cosine
   %initialize mask
    dist = zeros(SIZE);

   %calculate radius from center
    r = (1:SIZE)-SIZE/2;
    [x,y] = meshgrid(r,r);

   %fill distribution with cosine of radius
    dist = cos(sqrt(x.^2+y.^2)/(radius*2)*pi);
   %remove values outside of radius
    dist(sqrt(x.^2+y.^2) > radius) = 0;
end

```

UpdateIntensities.m

```matlab
%%washing
%sub image of size of mask
Img = inputImg(pix(1):pix(1)+size(spot,1)-1,pix(2):pix(2)+size(spot,2)-1);
%washing effect (image scaled by washing distribution)
%this is the total amount of analyte REMOVED from a given pixel
temp = wash.*Img;
%elements which will lose more than the initial value are set to initial
   value
temp(temp > Img/deltaT) = Img(temp>Img/deltaT)/deltaT;
%change in intensity by change in time (d in / d t). negate as intensity
   is leaving
dindt = -temp;

%copy temp to match dots size
temps = zeros(size(spot,1),size(spot,2),8);
%should be a repmat
temps(:,:,1) = temp;
for j = 2:8
   temps(:,:,j) = temps(:,:,j-1);
end

%account for direction of washing
temps = temps.*dots;

%distribute temps to each neighbor as an offset of dindt
% 1 0
dindt(2:end,:) = dindt(2:end,:) + temps(1:end-1,:,1);
% 1 1
dindt(2:end,2:end) = dindt(2:end,2:end) + temps(1:end-1,1:end-1,2);
% 0 1
dindt(:,2:end) = dindt(:,2:end) + temps(:,1:end-1,3);
% -1 0
dindt(1:end-1,:) = dindt(1:end-1,:) + temps(2:end,:,4);
% -1 -1
dindt(1:end-1,1:end-1) = dindt(1:end-1,1:end-1) + temps(2:end,2:end,5);
% 0 -1
dindt(:,1:end-1) = dindt(:,1:end-1) + temps(:,2:end,6);
% 1 -1
dindt(2:end,1:end-1) = dindt(2:end,1:end-1) + temps(1:end-1,2:end,7);
% -1 1
dindt(1:end-1,2:end) = dindt(1:end-1,2:end) + temps(2:end,1:end-1,8);

%depletion
% exponential decay based on spot
dindt =dindt -spot.*inputImg(pix(1):pix(1)+size(spot,1)-1,...
                                       pix(2):pix(2)+size(spot,2)-1);

%update input image by dindt * dt
tempInput = ...
   inputImg(pix(1):pix(1)+size(spot,1)-1,pix(2):pix(2)+size(spot,2)-1)...
   + dindt *deltaT;

%set negative values to 0
tempInput(tempInput<0) = 0;

```

```
%copy back into input image
inputImg(pix(1):pix(1)+size(spot,1)-1,pix(2):pix(2)+size(spot,2)-1) =
tempInput;
```

Monitoring DESI-MSI During Acquisition

Motivation, Overview and Extensions

DESI MSI was performed on a Thermo LTQ Orbitrap XL which produced proprietary .raw files. In most offline analysis, these were converted to mzXML files for subsequent analysis. The file structure consisted of a series of files, one per row of MS image, containing a chromatogram with uniform sampling. During acquisition, the sample stage would scan at a constant rate to produce a given pixel width. After each row, the file would be saved and the next row started with a new file.

Particularly for high resolution images, acquisition can take several hours and common output consisted of just monitoring the current spectrum or loading a previous row and attempting to glean the image quality from that. What was missing was a lightweight program capable of monitoring image quality during acquisition so that changes or loss of intensity could be addressed immediately. This led to the development of ImageMonitor, which was a small GUI executable which ran during MSI acquisition, shown in Figure A.1. The target *m/z* value and data directory were provided along with optional scaling factors for minimum and maximum intensities. When the update checkbox was enabled, the software would be triggered upon the generation of a new file in the target directory. This caused the raw file to be directly read and the displayed image updated. Of note, all previous rows stayed in memory as simply the intensity of the requested *m/z* value so the executable did not consume much memory. Since only one file was read upon update, disk usage was also minimized. Changing the *m/z* value would force the entire data set to be read again from disk. As the underlying code to generate the image was from the same library as the offline analysis code, the output displayed by the monitor would be similar to the expected output. While this particular implementation is not generally applicable to different instrument systems, for MSI instruments which produce images a row at a time, a similar algorithm could be useful in monitoring image acquisition.

ImageMonitor is written in C#. The main method in Program.cs simply creates a new form and runs it. The From1 is the main GUI window, designed in Visual Studio. The automatically generated From1.Designer.cs is included for completeness. The main logic is contained in Form1.cs. The only dependency of Form1.cs is DoubleImage, which was part of an MSI library. DoubleImage models an intensity matrix with double precision.

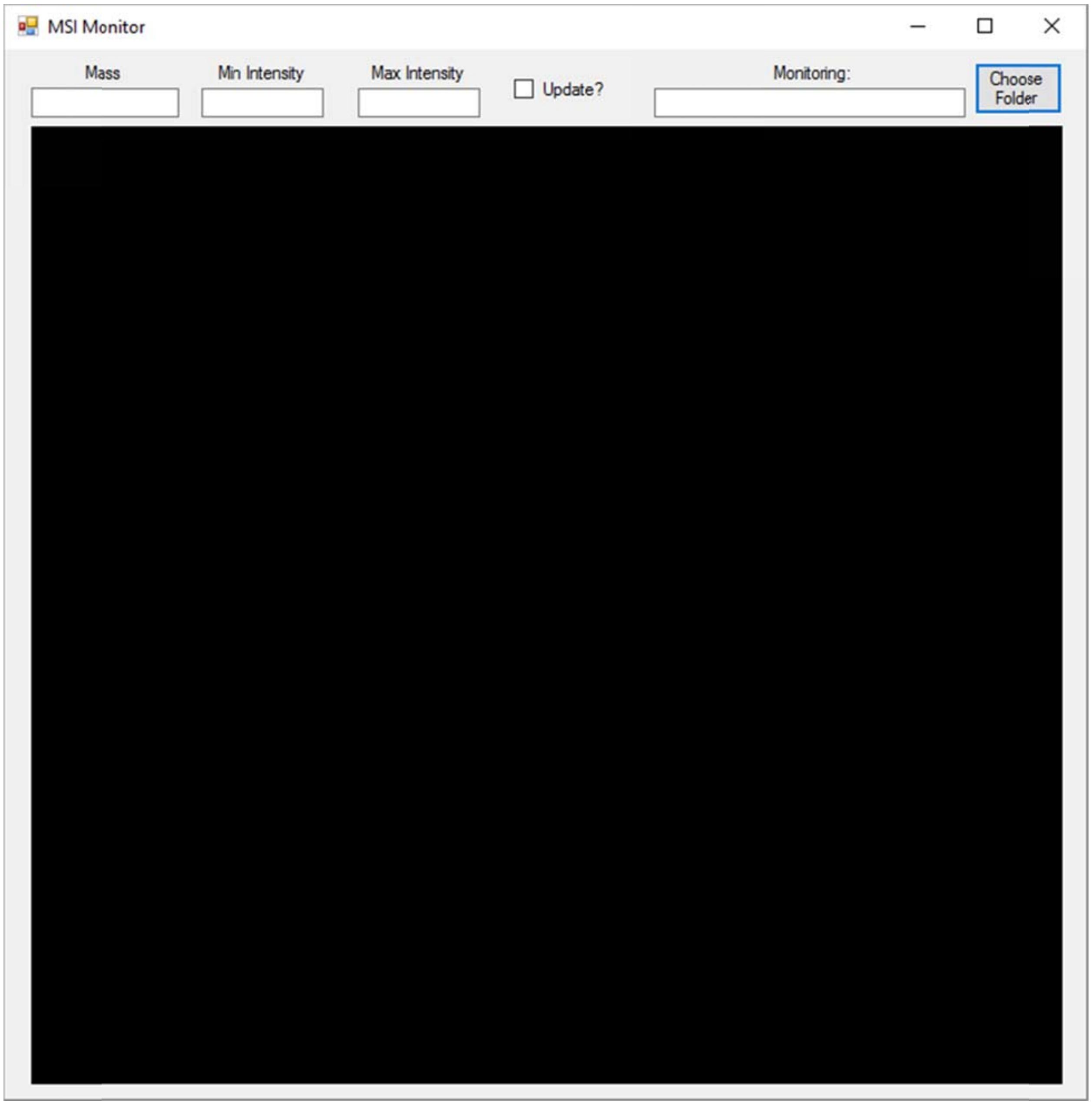


Figure A.1. ImageMonitor GUI. The textboxes across the top are filled in with the desired values. Once all values are set, clicking update causes the GUI to update when a new file is produced in the target directory. This causes the monitor to read just the most recent file and update the image with another row.

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ImageMonitor
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Form1.Designer.cs

```csharp
using System.Windows.Forms;
namespace ImageMonitor
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.btDirectory = new System.Windows.Forms.Button();
            this.lblMonitoring = new System.Windows.Forms.Label();
            this.tbMass = new System.Windows.Forms.TextBox();
            this.pbImage = new System.Windows.Forms.PictureBox();
            this.lblMass = new System.Windows.Forms.Label();
            this.tbDirectory = new System.Windows.Forms.TextBox();
            this.lblMin = new System.Windows.Forms.Label();
            this.tbMin = new System.Windows.Forms.TextBox();
            this.lblMax = new System.Windows.Forms.Label();
            this.tbMax = new System.Windows.Forms.TextBox();
            this.cbUpdate = new System.Windows.Forms.CheckBox();
((System.ComponentModel.ISupportInitialize)(this.pbImage)).BeginInit();
            this.SuspendLayout();
            //
            // btDirectory
            //
            this.btDirectory.Location = new System.Drawing.Point(658, 9);
            this.btDirectory.Name = "btDirectory";
            this.btDirectory.Size = new System.Drawing.Size(60, 35);
            this.btDirectory.TabIndex = 0;
            this.btDirectory.Text = "Choose Folder";
            this.btDirectory.UseVisualStyleBackColor = true;
            this.btDirectory.Click += new
              System.EventHandler(this.btDirectory_Click);
            //
            // lblMonitoring
            //
            this.lblMonitoring.AutoSize = true;
            this.lblMonitoring.Location = new System.Drawing.Point(519,
                                                                     9);
            this.lblMonitoring.Name = "lblMonitoring";
            this.lblMonitoring.Size = new System.Drawing.Size(62, 13);
            this.lblMonitoring.TabIndex = 1;
            this.lblMonitoring.Text = "Monitoring: ";
            //
            // tbMass
            //
            this.tbMass.Location = new System.Drawing.Point(18, 26);
            this.tbMass.Name = "tbMass";
            this.tbMass.Size = new System.Drawing.Size(100, 20);
            this.tbMass.TabIndex = 2;
            this.tbMass.KeyUp += new
        System.Windows.Forms.KeyEventHandler(this.tbMass_TextChanged);
            //
            // pbImage
            //
            this.pbImage.BackColor = System.Drawing.Color.Black;
            this.pbImage.Location = new System.Drawing.Point(18, 52);
            this.pbImage.Name = "pbImage";
            this.pbImage.Size = new System.Drawing.Size(700, 650);
            this.pbImage.TabIndex = 3;
            this.pbImage.TabStop = false;
            //
            // lblMass
            //
            this.lblMass.AutoSize = true;
            this.lblMass.Location = new System.Drawing.Point(51, 9);
            this.lblMass.Name = "lblMass";
            this.lblMass.Size = new System.Drawing.Size(32, 13);
            this.lblMass.TabIndex = 1;
            this.lblMass.Text = "Mass";
            //
            // tbDirectory
            //
            this.tbDirectory.Location = new System.Drawing.Point(441,26);
            this.tbDirectory.Name = "tbDirectory";
            this.tbDirectory.Size = new System.Drawing.Size(211, 20);
            this.tbDirectory.TabIndex = 2;
            this.tbDirectory.TextChanged += new
              System.EventHandler(this.tbDirectory_TextChanged);
            this.tbDirectory.KeyUp += new
        System.Windows.Forms.KeyEventHandler(this.tbMass_TextChanged);
            //
            // lblMin
            //
            this.lblMin.AutoSize = true;
            this.lblMin.Location = new System.Drawing.Point(141, 9);
            this.lblMin.Name = "lblMin";
            this.lblMin.Size = new System.Drawing.Size(66, 13);
            this.lblMin.TabIndex = 1;
            this.lblMin.Text = "Min Intensity";
            //
            // tbMin
            //
            this.tbMin.Location = new System.Drawing.Point(133, 26);
            this.tbMin.Name = "tbMin";
            this.tbMin.Size = new System.Drawing.Size(83, 20);
            this.tbMin.TabIndex = 2;
            this.tbMin.TextChanged += new
                    System.EventHandler(this.tbMin_TextChanged);
            this.tbMin.KeyUp += new
        System.Windows.Forms.KeyEventHandler(this.tbMass_TextChanged);
            //
            // lblMax
            //
            this.lblMax.AutoSize = true;
            this.lblMax.Location = new System.Drawing.Point(246, 9);
            this.lblMax.Name = "lblMax";
            this.lblMax.Size = new System.Drawing.Size(69, 13);
            this.lblMax.TabIndex = 1;
            this.lblMax.Text = "Max Intensity";
            //
            // tbMax
            //
            this.tbMax.Location = new System.Drawing.Point(239, 26);
            this.tbMax.Name = "tbMax";
            this.tbMax.Size = new System.Drawing.Size(83, 20);
            this.tbMax.TabIndex = 2;
            this.tbMax.TextChanged += new
                    System.EventHandler(this.tbMax_TextChanged);
            this.tbMax.KeyUp += new
        System.Windows.Forms.KeyEventHandler(this.tbMass_TextChanged);
            //
            // cbUpdate
            //
            this.cbUpdate.AutoSize = true;
            this.cbUpdate.Location = new System.Drawing.Point(346, 19);
            this.cbUpdate.Name = "cbUpdate";
            this.cbUpdate.Size = new System.Drawing.Size(67, 17);
            this.cbUpdate.TabIndex = 4;
            this.cbUpdate.Text = "Update?";
            this.cbUpdate.UseVisualStyleBackColor = true;
            this.cbUpdate.CheckedChanged += new
                    System.EventHandler(this.cbUpdate_CheckedChanged);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(734, 712);
            this.Controls.Add(this.cbUpdate);
            this.Controls.Add(this.pbImage);
            this.Controls.Add(this.tbDirectory);
            this.Controls.Add(this.tbMax);
            this.Controls.Add(this.lblMax);
            this.Controls.Add(this.tbMin);
```

```
                this.Controls.Add(this.lblMin);
                this.Controls.Add(this.tbMass);
                this.Controls.Add(this.lblMass);
                this.Controls.Add(this.lblMonitoring);
                this.Controls.Add(this.btDirectory);
                this.Name = "Form1";
                this.Text = "MSI Monitor";
        ((System.ComponentModel.ISupportInitialize)(this.pbImage)).EndInit();
                this.ResumeLayout(false);
                this.PerformLayout();

            }

            #endregion

            private System.Windows.Forms.Button btDirectory;
            private System.Windows.Forms.Label lblMonitoring;
            private System.Windows.Forms.TextBox tbMass;
            private System.Windows.Forms.PictureBox pbImage;
            private Label lblMass;
            private TextBox tbDirectory;
            private Label lblMin;
            private TextBox tbMin;
            private Label lblMax;
            private TextBox tbMax;
            private CheckBox cbUpdate;
        }
    }
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MassSpecLibrary;
using ThermoRawFileReaderDLL.FinniganFileIO;
using System.Runtime.CompilerServices;

namespace ImageMonitor
{
    public partial class Form1 : Form
    {
        //trigger to detect file creation
        private FileSystemWatcher watcher = new FileSystemWatcher();
        //image data
        private List<List<double>> image;
        //target m/z value
        private double? mzValue;
        //path to monitor
        private String path;
        //list of files already analyzed
        private List<String> processed = new List<String>();
        //out variable for file information
        private FinniganFileReaderBaseClass.udtScanHeaderInfoType header;
        //raw file with data
        private XRawFileIO rawFile;
        //m/z tolerance
        private double tolerance = 0.05;
        //min and max scaling
        private double? min;
        private double? max;
        //lock for multithreading
        private object thisLock = new object();

        //constructor
        public Form1()
        {
            InitializeComponent();
              //add a function for when the file system changes
            watcher.Created += watcher_Changed;
              //only consider raw files
            watcher.Filter = "*.raw";
        }

        private void watcher_Changed(object sender,FileSystemEventArgs e)
        {
              //reset the image when a new file is created
            RefreshImage(reset: false);
        }

        //handle changes in mass values
        private void tbMass_TextChanged(object sender, KeyEventArgs e)
        {
            double temp;

              //parse value
            if (Double.TryParse(tbMass.Text, out temp) == true)
                mzValue = temp;
            else
                mzValue = null;
              //refresh image
            RefreshImage(reset: true);
        }

        //handle clicking on the browse directory button
        private void btDirectory_Click(object sender, EventArgs e)
        {
              //popup a new folder browser
            FolderBrowserDialog diag = new FolderBrowserDialog();
            if (diag.ShowDialog() == DialogResult.OK)
            {
                    //get directory, set watcher path and text box
                tbDirectory.Text = watcher.Path = path =
                                      diag.SelectedPath;
                    //enable watcher to respond
                watcher.EnableRaisingEvents = true;
                    //refresh image
                RefreshImage(reset: true);
            }
        }

        [MethodImpl(MethodImplOptions.Synchronized)]
        private void RefreshImage(bool reset)
        {
              //set image to blank if no m/z value is specified, the
              directory doesn't exist, or the update box is unchecked
            if (mzValue.HasValue == false || !Directory.Exists(path) ||
                    cbUpdate.Checked == false)
            {
                pbImage.Image = null;
                return;
            }

              //if reset is passed in (when the entire image needs to be
                    reset)
            if (reset == true)
            {
                    //initialize new image
                image = new List<List<double>>();
                    //initialize new list of processed raw files (none
                          are processed)
                processed = new List<String>();
            }

              //get target m/z value
            double mz = mzValue.Value;
```

```

              //create list of all files
            var files = Directory.EnumerateFiles(path, "*.raw")
                    //not in processed list
                .Except(processed)
                    //ordered by creation time
                .OrderBy(f => File.GetCreationTime(f))
                .ToList();

              //if fewer than 2 files, keep blank image
            if (files.Count < 2)
            {
                pbImage.Image = null;
                return;
            }

            //don't include last file, it will usually be actively
              written
            files.RemoveAt(files.Count - 1);

              //initialize variables which will be passed by ref or out
              in COM methods
            rawFile = new XRawFileIO();
            double[] mzs, intens;
            int scans, numPeaks;
              //flag to check if m/z is in range
            bool checkmz = false;

              //for each file
            foreach (String file in files)
            {
              //try to open
                if (rawFile.OpenRawFile(file) == true)
                {
                    //get number of scans
                    scans = rawFile.GetNumScans();
                    //initialize new row in image
                    image.Add(new List<double>(scans));

                    //for each scan (pixel) in row (note 1 based
                          indexing)
                    for (int i = 1; i <= scans; i++)
                    {
                          //read in header
                        rawFile.GetScanInfo(i, out header);
                          //get number of peaks
                        numPeaks = header.NumPeaks;
                          //initialize m/z and intensity arrays
                        mzs = new double[numPeaks];
                        intens = new double[numPeaks];

                          //read in scan data to arrays
                        rawFile.GetScanData(i, ref mzs, ref intens, ref
                                                             header);

                          //have not previously checked if m/z in range
                        if (checkmz == false)
```

```
        {
         //check value
          checkmz = true;
          if( mzs[0] > mz || mzs[mzs.Length - 1] < mz)
          {
            //make image blank if outside
                        of range
            pbImage.Image = null;
            return;
          }
        }

         //iterate through each m/z
        double maxInt = 0;
        for (int j = 0; j < mzs.Length; j++)
         //record max intensity for m/zs in range
          if (mzs[j] >= mz - tolerance &&
            mzs[j] <= mz + tolerance)
            maxInt = intens[j] > maxInt ? intens[j] :
                           maxInt;

         //add in pixel intensity
        image.Last().Add(maxInt);
      }
      //add to the processed list
      processed.Add(file);
    }
   //close file at end
    rawFile.CloseRawFile();
  }

   //create new double image of current image data
  DoubleImage dblImg = new DoubleImage(image);

   //set min and max values
  if (min.HasValue == true)
    dblImg.SetMin(min.Value);

  if (max.HasValue == true)
    dblImg.SetMax(max.Value);

   //draw image
  pbImage.Image = dblImg
    .DrawBitmap(pbImage.Width, pbImage.Height,
    DoubleImage.Coloring.Rainbow,
 System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic);
}

//handle changes in directory text box
private void tbDirectory_TextChanged(object sender, EventArgs e)
{
  if(Directory.Exists(tbDirectory.Text) == true)
  {
      //set new watcher path and trigger new events
    watcher.Path = path = tbDirectory.Text;
    watcher.EnableRaisingEvents = true;
      //refresh image
```

```
                RefreshImage(reset: true);
            }
        }

        //handle min text box changes
        private void tbMin_TextChanged(object sender, EventArgs e)
        {
              //try to parse value
            double temp;
            if (Double.TryParse(tbMin.Text, out temp) == true)
            {
              //refresh image if changed
                min = temp;
                RefreshImage(reset: true);
            }
            else
                min = null;
        }

        //handle max text box changes
        private void tbMax_TextChanged(object sender, EventArgs e)
        {
              //try to parse value
            double temp;
            if (Double.TryParse(tbMax.Text, out temp) == true)
            {
                    //refresh image if needed
                max = temp;
                RefreshImage(reset: true);
            }
            else
                max = null;
        }

        //handle interactions with check box
        private void cbUpdate_CheckedChanged(object sender, EventArgs e)
        {
              //force reset (will be blank if box is unchecked)
            RefreshImage(reset: true);
        }
    }
}
```

DoubleImage.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.IO;


namespace MassSpecLibrary
{
    /// <summary>
    /// A 2D array representing an image, but has double precision.
    /// </summary>
    public class DoubleImage
    {
        /// <summary>
        /// Gets the image values.
        /// </summary>
        /// <value>
        /// The image array with double values.
        /// </value>
        public double[,] Image {get; private set;}

        public double min = Double.MaxValue, max = Double.MinValue;
        private int intensMax = 255, intensMin = 0, width, height;

        /// <summary>
        /// An enum of the possible color values.
        /// </summary>
        public enum Coloring {
            /// <summary>
            /// The rainbow color map
            /// </summary>
            Rainbow,
            /// <summary>
            /// The green color map
            /// </summary>
            Green,
            /// <summary>
            /// The greyscale color map
            /// </summary>
            Greyscale ,

            WhiteAndBlack
        }

        public DoubleImage(List<List<double>> image)
        {
            width = image.Min(row => row.Count);
            height = image.Count;

            max = image.AsParallel().Max(row => row.Max());
            min = image.AsParallel().Min(row => row.Min());

            Image = new double[height, width];

            for (int i = 0; i < height; i++)
                for (int j = 0; j < width; j++)
                    Image[i, j] = image[i][j];
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="DoubleImage"/> class.  Simply allocates the array space of specified size.
        /// </summary>
        /// <param name="rows">The number of rows to allocate.</param>
        /// <param name="cols">The number of columns to allocate.</param>
        public DoubleImage(int rows, int cols)
        {
            Image = new double[rows, cols];
            width = cols;
            height = rows;
        }

        /// <summary>
        /// Adds the specified value to [row,col]. Has minimal error checking and updates the value of min and max.
        /// </summary>
        /// <param name="row">The row to add to.</param>
        /// <param name="col">The column to add to.</param>
        /// <param name="val">The value to add to [row,col].</param>
        public void Add(int row, int col, double val)
        {
            if (row >= 0 && row < Rows() && col >= 0 && col < Cols())
            {
                Image[row, col] = val;
                max = max < val ? val : max;
                min = min > val ? val : min;
            }
        }

        /// <summary>
        /// Subtracts the values of one double image from the instance, ie this - other.  Returns if dimensions do not match.
        /// </summary>
        /// <param name="other">The other double image.</param>
        public void Subtract(DoubleImage other){
            if(other.Rows() != Rows() || other.Cols() != Cols())
                return;
            for (int i = 0; i < Rows(); i++)
                for (int j = 0; j < Cols(); j++)
                    Image[i, j] -= other.Image[i, j];
        }

        /// <summary>
        /// Draws the bitmap of the internal data in black and white with no scaling.
        /// </summary>
        /// <returns>A bitmap of the data in black and white with no scaling.</returns>
        public Bitmap DrawBitmap()
        {
            int rows = Image.GetLength(0), cols = Image.GetLength(1);
            Bitmap result = new Bitmap(cols, rows); // this is transposed for the bitmap

            if (max == min)
                return result; //this would cause an error, just return a blank image

            //populate bitmap image scaled by min and max

            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    //scale intensities linearly from min to max, if min == max intens is 0
                    //int intens = max == min ? 0 : (int)((intensMax – intensMin) / (max - min) * (image[i, j])) + intensMin;
                    int intens = max == min ? 0 : (int)((255) / (max – min) * (Image[i, j]));
                    //rescale between min and max intensity, < min is 0 > max = 255
                    intens = intensMax == intensMin ? 0 : (int)((double)255 / (intensMax-intensMin) * (intens -intensMin)) ;
                    intens = intens < 0 ? 0 : (intens > 255 ? 255 : intens);
                    result.SetPixel(j, i, Color.FromArgb(intens, intens, intens));

                }
            }
            return result;
        }

        /// <summary>
        /// Draws the unscaled bitmap with specified color.
        /// </summary>
        /// <param name="color">The colormap to use for drawing.</param>
        /// <returns>A colored bitmap with no resizing</returns>
        public Bitmap DrawBitmap(Coloring color)
        {
            Bitmap result = DrawBitmap();

            Graphics g = Graphics.FromImage(result);

            ImageAttributes attr = new ImageAttributes();
            if(color == Coloring.Rainbow)
                attr.SetRemapTable(RecolorPallette.RainbowMap);
            if (color == Coloring.Green)
                attr.SetRemapTable(RecolorPallette.GreenMap);
            if (color == Coloring.WhiteAndBlack)
                attr.SetRemapTable(RecolorPallette.WhiteAndBlackMap);
            g.DrawImage(result, new Rectangle(0, 0, result.Width,
              result.Height), 0, 0, result.Width, result.Height,
              GraphicsUnit.Pixel, attr);

            return result;
        }

        /// <summary>
        /// Draws the scaled bitmap with specified interpolation and
              coloring.  Width and height are the allowable maximums, the
              resulting bitmap will fit in this area maintaining the
              aspect ratio.
        /// </summary>
        /// <param name="width">The desired maximum width.</param>
        /// <param name="height">The desired maximum height.</param>
        /// <param name="color">The colormap to use.</param>
        /// <param name="interp">The interpolation mode.</param>
        /// <returns>Scaled, colored, interpolated bitmap.</returns>
        public Bitmap DrawBitmap(int width, int height, Coloring color,
              InterpolationMode interp)
        {
            int mag = Magnification(width, height);
            Bitmap temp = DrawBitmap();
            Bitmap result = new Bitmap(this.width * mag, this.height *
                                                         mag);
            //resize

            Graphics g = Graphics.FromImage(result);
            g.InterpolationMode = interp;

            g.DrawImage(temp, 0, 0, result.Width, result.Height);

            ImageAttributes attr = new ImageAttributes();
            if (color == Coloring.Rainbow)
                attr.SetRemapTable(RecolorPallette.RainbowMap);
            if (color == Coloring.Green)
                attr.SetRemapTable(RecolorPallette.GreenMap);
            if (color == Coloring.WhiteAndBlack)
                attr.SetRemapTable(RecolorPallette.WhiteAndBlackMap);
            g.DrawImage(result, new Rectangle(0, 0, result.Width,
                          result.Height), 0, 0, result.Width,
                          result.Height, GraphicsUnit.Pixel, attr);

            return result;
        }

        public Bitmap DrawBitmap(int magnification, Coloring color,
                                  InterpolationMode interp)
        {
            Bitmap temp = DrawBitmap();
            Bitmap result = new Bitmap(this.width * magnification,
                                       this.height * magnification);
            //resize

            Graphics g = Graphics.FromImage(result);
            g.InterpolationMode = interp;

            g.DrawImage(temp, 0, 0, result.Width, result.Height);

            ImageAttributes attr = new ImageAttributes();
            if (color == Coloring.Rainbow)
                attr.SetRemapTable(RecolorPallette.RainbowMap);
            if (color == Coloring.Green)
                attr.SetRemapTable(RecolorPallette.GreenMap);
            if (color == Coloring.WhiteAndBlack)
                attr.SetRemapTable(RecolorPallette.WhiteAndBlackMap);
            g.DrawImage(result, new Rectangle(0, 0, result.Width,
                                result.Height), 0, 0, result.Width,
                                result.Height, GraphicsUnit.Pixel, attr);

            return result;
        }

        /// <summary>
        /// Draws the bitmap with a colored ROI.
        /// </summary>
        /// <param name="coloring">The coloring of the image to
              use.</param>
        /// <param name="inROI">The locations that are in the ROI.  True
              implies the pixel is in the ROI.</param>
        /// <param name="ROIColor">Color of the ROI.</param>
        /// <returns>A bitmap with the ROI colored in</returns>
        public Bitmap DrawBitmap(Coloring coloring, bool[,] inROI, Color
                                ROIColor)
        {
            //start with normal bitmap
            Bitmap result = DrawBitmap(coloring);
            //recolor points in list
            for (int i = 0; i < inROI.GetLength(0); i++)
                for (int j = 0; j < inROI.GetLength(1); j++)
                    if (inROI[i, j] == true)
                        result.SetPixel(j, i, ROIColor); //transpose for
                                                              image!
            return result;
        }

        /// <summary>
        /// Draws the bitmap at the desired scale, specified color, with
              ROI colored in.
        /// </summary>
        /// <param name="width">The desired maximum width.</param>
        /// <param name="height">The desired maximum height.</param>
        /// <param name="color">The colormap to use.</param>
        /// <param name="interp">The interpolation mode.</param>
        /// <param name="inROI">The locations that are in the ROI.  True
              implies the pixel is in the ROI.</param>
        /// <param name="ROIColor">Color of the ROI.</param>
        /// <returns>Bitmap that is rescaled, colored and has ROI
              colored.</returns>
        public Bitmap DrawBitmap(int width, int height, Coloring color,
                                InterpolationMode interp, bool[,] inROI,
                                Color ROIColor)
        {
            //start with normal bitmap, resized as needed
            Bitmap result = DrawBitmap(width, height, color, interp);
            int mag = Magnification(width, height);
            Graphics g = Graphics.FromImage(result);
            Brush b = new SolidBrush(ROIColor);

            //draw in rectangular pixels
            for (int i = 0; i < inROI.GetLength(0); i++)
                for (int j = 0; j < inROI.GetLength(1); j++)
                    if (inROI[i, j] == true)
                        g.FillRectangle(b, j * mag -mag/2, i * mag -
                          mag/2,mag, mag); //transpose for image!


            //return
            return result;
        }

        public Bitmap DrawBitmap(int width, int height, Coloring color,
                                 InterpolationMode interp,
                                 List<List<Point>> ROI, Color ROIColor)
        {
            //start with normal bitmap, resized as needed
            Bitmap result = DrawBitmap(width, height, color, interp);
            int mag = Magnification(width, height);
            Graphics g = Graphics.FromImage(result);
            Brush b = new SolidBrush(ROIColor);
            SolidBrush sb = new SolidBrush(Color.White);

            for (int i = 0; i < ROI.Count; i++)
            {
                float? x = null, y = null; //hold the maxes
                foreach (Point p in ROI[i])
                {
                    x = x > p.X ? x : p.X;
                    y = y > p.Y ? y : p.Y;
                    g.FillRectangle(b, p.Y * mag - mag / 2, p.X * mag -
                                          mag / 2, mag, mag);
                }
                g.DrawString(i.ToString(), new Font("Arial", 6), sb, new
                                      PointF(y.Value * mag - mag / 2,
                                      x.Value * mag - mag / 2));
            }



            //return
            return result;
        }



        /// <summary>
        /// Determines the magnification to the target width and height
             that maintains the aspect ratio and integer pixel values
        /// </summary>
        /// <param name="targetWidth">Target width.</param>
        /// <param name="targetHeight">Target height.</param>
        /// <returns>The scaling magnification as an integer.</returns>
        public int Magnification(int targetWidth, int targetHeight)
        {
            return (targetWidth / width < targetHeight / height ?
                    targetWidth / width : targetHeight / height);
        }

        /// <summary>
        /// Get the number of rows of the image.
        /// </summary>
        /// <returns>The number of rows.</returns>
        public int Rows()
        {
            return Image.GetLength(0);
        }

        /// <summary>
        /// Get the number of columns of the image.
        /// </summary>
        /// <returns>The number of columns of the image.</returns>
        public int Cols()
        {
            return Image.GetLength(1);
        }

        /// <summary>
        /// Sets the intensity maximum of the resulting bitmap.  Checks
                    the supplied value is greater than the minimum and
                    less than 255.
        /// </summary>
        /// <param name="val">The new intensity maximum.</param>
        public void SetIntMax(int val)
        {
            //max should be between min and 255
            intensMax = val < intensMin ? intensMin : (val > 255 ? 255 :
                                                       val);
        }

        /// <summary>
        /// Sets the intensity minimum of the resulting bitmap. Checks
              the supplied value is less than the maximum and greater
              than 0.
        /// </summary>
        /// <param name="val">The new intensity minimum.</param>
        public void SetIntMin(int val)
        {
            //min should be between 0 and max
            intensMin = val < 0 ? 0 : (val > intensMax ? intensMax :
                                                             val);
        }

        public void SetMin(double val)
        {
            min = val;
        }

        public void SetMax(double val)
        {
            max = val;
        }

        /// <summary>
        /// Determines the difference between two double images.
              Intensities are scaled between 0 and 1, and the square
              difference is calculated.  Min and max are set to 0 and 1.
        /// </summary>
        /// <param name="other">The other DoubleImage to subtract with.
              Assumes the instances are the same size.</param>
        /// <returns>A new double image with values equal to the square
              difference between pixels in the input images.</returns>
        public DoubleImage ScaledSquareDiff(DoubleImage other)
        {
            DoubleImage result = new DoubleImage(Rows(), Cols());
            for (int i = 0; i < Rows(); i++)
            {
                for (int j = 0; j < Cols(); j++)
                {
                    result.Image[i,j] = Math.Pow((Image[i, j] - min) /
                          (max - min) - (other.Image[i, j] - other.min) /
                          (other.max - other.min), 2);
                }
            }
            result.min = 0;
            result.max = 1;
            return result;
        }

        /// <summary>
        /// Saves the double image as an ascii tab delineated file.
        /// </summary>
        /// <param name="filename">The filename to save to.</param>
        public void Save(String filename)
        {
            StreamWriter writer = new StreamWriter(filename);
            for (int i = 0; i < Rows(); i++)
            {
                for (int j = 0; j < Cols(); j++)
                {
                    writer.Write(Image[i, j] + "\t");
                }
                writer.WriteLine();
            }
            writer.Close();
        }

        public static void SaveScale(string file, int pixelheight,
                                         Coloring color)
        {
            DoubleImage di = new DoubleImage(pixelheight, 256);
            for (int i = 0; i < 256; i++)
                for (int j = 0; j < pixelheight; j++)
                    di.Image[j,i] = i;
            di.SetMin(0);
            di.SetMax(255);
            di.DrawBitmap(color).Save(file);
            }
        }
}
```

microMS

Motivation, Overview and Extensions

After the initial demonstration of single cell profiling with MALDI MS using fluorescence imaging to locate cells, it was clear that a more user friendly option was necessary to promote widespread adoption. The first reported workflow required significant manual effort on the user and could result in inaccurate target localization due to handling of tiled images, a coordinate registration which was vulnerable to small errors, and utilizing fixed fiducial marks generated prior to microscopy. While these issues were addressed by microMS, a continuing goal was to make optically-guided single cell profiling approachable to novice users on a variety of instruments. The result is a feature rich GUI which displays each step of the process and provides ample feedback to ensure quality data results. On the back end, the implementation of coordinate mappers as an abstract base class greatly simplifies the addition of new instruments; only a handful of functions need implementation before the image analysis features are available to new systems. Decoupling target positions from instrument positions also has the advantage of simplifying repeated analysis of the same targets on different instruments.

The complete source code, written in Python, is presented here, though updates may be found at http://neuroproteomics.scs.illinois.edu/microMS.htm. Also note the full user manual is found in Appendix B with examples of usage and implementing new coordinate mappers. A further discussion of microMS can be found in Chapter 5, which also shows the project organization in Figure 5.1. Briefly, the main method in microMS.py creates a new microMSQTwindow object and starts the main thread. microMSQTwindow contains the widgets slideCanvas and histCanvas, which allow interactions with a microscope image and population level statistics of found blobs. microMSQTwindow also has code for the menu bar, handles interactions between widgets, and spawns popup option windows. Each GUI component interacts with the controller, microMSModel which models a single imaging experiment, complete with an image, collection of targets and a coordinate mapper. As such, it contains instance variables of many classes in ImageUtilities (handling image interactions and blob methods) and CoordinateMappers (which model a physical instrument). The source code is presented as the main method, followed by the packages GUICanvases, ImageUtilities, and CoordinateMappers in that order. Each source file is heavily documented including descriptions of each class.

microMS.py

```
#! /usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
import ctypes
from PyQt5 import QtGui, QtCore, QtWidgets

from GUICanvases.microMSQTWindow import MicroMSQTWindow

def main():
    '''
    main method that begins execution of the QApplication
    '''
    qApp = QtWidgets.QApplication(sys.argv)

    #set up icon
    if sys.platform == 'win32':
        myappid = 'uiuc.sweedlerlab.microms.v1'
        ctypes.windll.shell32
              .SetCurrentProcessExplicitAppUserModelID(myappid)
    qApp.setWindowIcon(QtGui.QIcon(r'GUICanvases/Icon/icon_sm.png'))

    #start application
    aw = MicroMSQTWindow()
    aw.setWindowTitle("MicroMS")
    aw.show()
    sys.exit(qApp.exec_())

if __name__ == '__main__':
    main()
```

GUICanvases/__init__.py

```
'''
The GUICanvases package contains classes for display and user interaction
GUIConstants.py:     a collection of constant variables for display
histCanvas.py:       a widget for displaying and interacting with
                     population level metrics
microMSModel.py:     controller class for a single experiment
microMSQTWindow.py:  main GUI window
mplCanvas.py:        an abstract class extending figure canvas for
                     displaying matplotlib figures
popup.py:            a collection of small, custom windows for user IO to
                     set parameters with blob finding, histogram display,
                     and intermediate maps
slideCanvas.py       a widget to display and interact with a slideWrapper
                     object.
                     Contains most of the programming logic
'''
```

GUICanvases/GUIConstants.py

```

###colors and sizes of GUI components in slideCanvas
#default and boarder color of the slide image
IMAGE_BACKGROUND        =   'black'
#color of temporary or test blob find
TEMP_BLOB_FIND          =   'turquoise'
#predicted locations from the current mapper
PREDICTED_POINTS        =   'yellow'
#color of circle and text of highest FLE
FIDUCIAL_WORST          =   'red'
#color of the rest of the fiducial circles and text
FIDUCIAL                =   'blue'
#background label of fiducial marks
FIDUCIAL_LABEL_BKGRD    =   'white'
#ROI boundary
ROI                     =   'yellow'
#ROI minimum distance
ROI_DIST                =   10 #pixels
#colors of blob list
MULTI_BLOB              =   ['lime', 'salmon', 'skyBlue', 'orangeRed',
                             'plum', 'hotPink', 'aqua', 'yellow',
                             'olive', 'green']
#text display of grouped targets from expanding blobs
EXPANDED_TEXT          =   'purple'
#default blob radius in pixels
DEFAULT_RADIUS          =   8
#default fiducial radius in pixels
FIDUCIAL_RADIUS         =   100
#maximum number of blobs to draw when limit is selected
DRAW_LIMIT              =   150
#maximum number of blobs to check prior to deselecting TSP optimization
TSP_LIMIT               =   1000


###colors of GUI components in histCanvas
#colors of  bars in histogram for red, green, blue, size, circularity,
                                                         and distance
BAR_COLORS              =   ['red', 'green', 'blue', 'gray', 'gray',
                                                         'gray']
#color of bars and blobs with values less than the cutoff
LOW_BAR                 =   'cyan'
#color of bars and blobs with values greater than the cutoff
HIGH_BAR                =   'hotpink'
#color of bars and blobs with values in a single bar
SINGLE_BAR              =   'darkorange'
#color of line to indicate a single blob position
SINGLE_BLOB             =   'red'

###constants for blob shapes
DEFAULT_BLOB_RADIUS     =   DEFAULT_RADIUS
DEFAULT_PATTERN_RADIUS  =   DEFAULT_RADIUS

###standard test files for the debug load
#directory to check for prior to trying to load
DEBUG_DIR               =   r'T:\Cerebellum One Left Stitched _'
```

```
#image file
DEBUG_IMG_FILE          =   r'T:\Cerebellum One Left Stitched
                        _\Cerebellum One Left Stitched __c1.tif'
#blob find file
DEBUG_BLOB_FIND         =   r'T:\Cerebellum One Left Stitched
                        _\sol_find.txt'
#registration file
DEBUG_REG_FILE          =   r'T:\Cerebellum One Left Stitched
                        _\sol.msreg'

###help message text
IMAGE_HOTKEYS           =   ("w,s,a,d\t\tMove\n"
        "W,S,A,D\tMove Farther\n"
        "q,e\t\tZoom out/in\n"
        "r\t\tReset view\n"
        "t\t\tSwitch views\n"
        "b\t\tTest blob find\n"
        "B\t\tSwitch to threshold view\n"
        "m\t\tMirror x axis\n"
        "p\t\tToggle predicted location\n"
        "o\t\tToggle drawn shapes\n"
        "O\t\tToggle drawing all blob lists\n"
        "Ctrl + C\tClear all found blobs\n"
        "C\t\tClear current blob list\n"
        "c\t\tClear ROI\n\n"
        "#\t\tToggle channel\n"
        "Ctrl+#\t\tSet channel\n"
        "Alt+#\t\tSet manual blob list\n\n"
        "LMB\t\tMove to center\n"
        "LMB+Shift\tAdd/remove points\n"
        "LMB+Ctrl\tDraw ROI\n"
        "MMB\t\tGet pixel values\n"
        "RMB\t\tAdd slide coordinate\n"
        "RMB+Shift\tRemove slide coordinate\n"
        "Scroll\t\tZoom in/out"
        )

INSTRUMENT_HOTKEYS      =   ("i,k,j,l\t\tMove\n"
        "Ctrl + I,K,J,L\tMove Far\n"
        "I,K,J,L\t\tMove Farther\n"
        "+,-\t\tMove probe up/down\n"
        "V\t\tSet probe position\n"
        "v\t\tToggle probe position\n"
        "h\t\tHome stage\n"
        "H\t\tFinal position\n"
        "x\t\tSingle analysis\n\n"
        "LMB+Alt\tMove to spot\n"
        "RMB\t\tAdd coordinate\n"
        "RMB+Shift\tRemove coordinate\n"
        )

HISTOGRAM_HOTKEYS       =   ("LMB\t\tSet lower threshold\n"
        "LMB+Shift\tSet lower cutoff\n"
        "MMB\t\tSet single bar\n"
        "RMB\t\tSet upper threshold\n"
        "RMB+Shift\tSet upper cutoff\n"
        "Scroll\t\tZoom in/out")
```

GUICanvases/histCanvas.py

```python
from __future__ import unicode_literals

import numpy as np
from PyQt5 import QtGui, QtCore, QtWidgets
from copy import copy

from GUICanvases.mplCanvas import MplCanvas
from GUICanvases import GUIConstants

from ImageUtilities.blobFinder import blobFinder
from ImageUtilities.blobList import blobList

class HistCanvas(MplCanvas):
    '''
    HistCanvas is an implementation of MplCanvas that interacts with a slideCanvas
    to display population level information on a collection of blob objects.
    Most of the control logic is contained here as the view is simply a bar chart
    '''
    def __init__(self, master, model, *args, **kwargs):
        '''
        Initialize and connect listeners
        master: the master widget, a microMSQT
        slideCanvas: the connected slideCanvas to interact with
        '''
        MplCanvas.__init__(self, *args, **kwargs)
        self.draw()
        #start by showing the blob areas
        self.populationMetric = 3
        self.populationValues = None
        self.blobSet = None
        #the image of the collection from slideWrapper to analyze
        self.imgInd = 1
        #toggle to move the slide position to the first single blob
        self.moveSlide = False

        #listeners for mouse interaction
        self.mpl_connect('button_release_event', self.mouseUp)
        self.mpl_connect('scroll_event', self.mouseZoom)

        self.master = master
        self.model = model

        #offset from blob radius to consider when extracting fluorescence
        self.offset = 0

        #x axis limits for zooming
        self.xlo = None
        self.xhi = None

        #toggle to indicate if the maximum or average intensity should be displayed
        self.reduceMax = False

        #a list of the currently available metrics
        self.metrics = ['Red', 'Green', 'Blue', 'Size', 'Circularity', 'Distance']

        #initialize display variables
        self.resetVariables()

    def resetVariables(self, resetZoom = True, resetBlobs = False):
        '''
        reset variables related to splitting the population and display
        resetZoom: reset the zoom on the x axis
        resetBlobs: reset the list of blobs currently investigated
        '''
        #lowIntens and lowLimit hold thresholds for low values of the population
        #low blobs have I such that lowLimit < I < lowIntens
        self.lowIntens = None
        self.lowLimit = None
        #high blobs have I such that highIntens < I < highLimit
        self.highIntens = None
        self.highLimit = None
        #single bar is a value bin in the histogram
        self.singleBar = None
        #single blob contains the index of a single blob to show the position of in the histogram
        self.singleBlob = None

        if resetZoom:
            #zoom level on the x axis
            self.zoomLvl = 0
            #center of the x axis
            self.xcent = None

        if resetBlobs:
            #the color channel or morphology
            self.populationMetric = 3
            #set of population values
            self.populationValues = None
            #the actual blob list
            self.blobSet = None

    def removeBlob(self, index):
        #return immediately if globalBlbs is not set
        if self.populationValues is None or self.populationValues.size < index:
            return

        self.populationValues = np.delete(self.populationValues, index)
        self._calculateHist(resetVars = False)

    def _calculateHist(self, resetVars = True):
        #return immediately if globalBlbs is not set
        if self.populationValues is None:
            self.update_figure()
            return

        #metric >= 3 -> look at morphology
        if self.populationMetric >= 3:
            self.counts, self.bins, patches =
              self.axes.hist(self.populationValues, bins = 100)

        #metric == [0, 1, 2] -> look at intensities of [r, g, b] channel
        of image at imgInd
        else:
            self.counts, self.bins, patches =
                    self.axes.hist(self.populationValues, bins=100,
                    range=(0,255))

        self.bins = self.bins[1:]

        #reset limits and redraw
        if resetVars == True:
            self.resetVariables()
        self.update_figure()



    def calculateHist(self):
        '''
        calculate the population values with either the current set of
        blobs from the model
        this can require some calculation time to complete due to
        repeated disk reads on the image
        '''
        #set a new set of global blbs
        self.blobSet = self.model.blobCollection[self.model.currentBlobs]

        #return immediately if globalBlbs is not set
        if self.blobSet is None or len(self.blobSet.blobs) == 0:
            self.populationValues = None
            self._calculateHist()
            return

        #metric == 3 -> look at the area (= pi * r^2)
        if self.populationMetric == 3:
            self.populationValues = np.array([x.radius*x.radius*3.14 for
                         x in self.blobSet.blobs])

        #metric == 4 -> look at circularity
        elif self.populationMetric == 4:
            self.populationValues = np.array([x.circularity for x in
                                        self.blobSet.blobs])

        #metric == 5 -> look at minimum distance between samples
        elif self.populationMetric == 5:
            self.populationValues =
                    np.array(self.blobSet.minimumDistances())

        #metric == [0, 1, 2] -> look at intensities of [r, g, b] channel
              of image at imgInd
        else:
            self.populationValues =
                          np.array(self.model.slide.getFluorInt(self.blob
                          Set.blobs, self.populationMetric, self.imgInd,
                          self.offset, self.reduceMax))

        self._calculateHist()

    def mouseUp(self,event):
        '''
        handles click events by updating the high and low limits
        event: mpl mouse click event
        '''
        #click out of bounds
        if event.xdata is None or event.ydata is None or
                          self.populationValues is None:
            return

        #LMB to set low values
        if event.button == 1:
            #shift LMB to set the lower limit
            if QtWidgets.QApplication.keyboardModifiers() ==
                                    QtCore.Qt.ShiftModifier:
                self.lowLimit = event.xdata
            #LMB to set a lower threshold
            else:
                self.lowIntens = event.xdata
            #display a message if a lower threshold is set
            if self.lowIntens is not None:
                if self.lowLimit is not None:
                    self.master.statusBar().showMessage(
                          str(sum((self.populationValues <
                          self.lowIntens) & (self.populationValues >
                          self.lowLimit)))
                          + ' below {:.1f} and above
                          {:.1f}'.format(self.lowIntens, self.lowLimit))
                else:
                    self.master.reportFromModel(
                          str(sum(self.populationValues <
                          self.lowIntens))
                          + ' below {:.1f}'.format(self.lowIntens))

        #MMB to select a single bin
        if event.button == 2:
            self.singleBar = event.xdata
            self.master.reportFromModel('Clicked on {:.1f}'
                                    .format(event.xdata))
            self.moveSlide = True

        #RMB to set high values
        if event.button == 3:
            #shift RMB to set the higher limit
            if QtWidgets.QApplication.keyboardModifiers() ==
                    QtCore.Qt.ShiftModifier:
                self.highLimit = event.xdata

            #RMB to set the higher threshold
            else:
                self.highIntens = event.xdata

            #display message if a higher threshold is set
            if self.highIntens is not None:
                if self.highLimit is not None:
                    self.master.statusBar().showMessage(
                            str(sum((self.populationValues >
                            self.highIntens) & (self.populationValues <
                            self.highLimit)))
                                        + ' above {:.1f} and below {:.1f}'
                                              .format(self.highIntens,
                                              self.highLimit))
                else:
                    self.master.statusBar().showMessage(
                            str(sum(self.populationValues >
                            self.highIntens))
                          + ' above {:.1f}'.format(self.highIntens))

        #redraw figure
        self.update_figure()

    def mouseZoom(self,event):
        '''
        handle mouse scrolling by zooming in and out
        event: mpl mouse scroll event
        '''
        if self.populationValues is not None and event.xdata is not None:

            if event.button == 'up':
                self.zoomLvl += 1#zoom in
            else:
                self.zoomLvl -= 1#zoom out

            self.zoomLvl = 0 if self.zoomLvl < 0 else
                          (10 if self.zoomLvl > 10 else self.zoomLvl)
            self.xcent = int(event.xdata)

            #redraw the zoom lvl
            self.redraw_zoom()

    def setBlobNum(self, target):
        '''
        automatically sets a high and low threshold to select
        approximately the same number of blobs in each condition
        target: the target number of blobs to find
        '''
        #return when values not set
        if self.populationValues is None:
            return
        #subdivide the population by a larger factor
        counts, bins= np.histogram(self.populationValues, bins = 2560)

        #find lower cutoff, binary search
        left = 0
        right = len(counts)
        c = 0
        lowlimit = 0 if self.lowLimit is None else
```

```
              np.argmin(np.abs(bins - self.lowLimit))
        while left < right and c < 20:
            ind = (left + right) // 2
            tempCount = sum(counts[lowlimit:ind])
            if tempCount < target:
                left = ind +1
            elif tempCount > target:
                right = ind -1
            else:
                break
            c += 1
        self.lowIntens = bins[ind+1]
        tclow =   sum(counts[lowlimit:ind])

        #find upper cutoff, binary search
        left = 0
        right = len(counts)
        c = 0
        highlimit = len(counts)-1 if self.highLimit is None
                    else np.argmin(np.abs(bins - self.highLimit))
        while left < right and c < 20:
            ind = (left + right) // 2
            tempCount = sum(counts[ind:highlimit])
            if tempCount < target:
                right = ind -1
            elif tempCount > target:
                left = ind +1
            else:
                break
            c += 1
        self.highIntens = bins[ind-1]
        tchigh = sum(counts[ind:highlimit])
        self.update_figure()

        self.master.reportFromModel('Found ' + str(tclow) + ' below and '
                                      + str(tchigh) + ' above')

    def clearFilt(self):
        '''
        Clear the current set of filter parameters and redraw figure
        '''
        self.resetVariables(False)
        self.update_figure();

    def savePopulationValues(self, filename):
        '''
        saves the population values of the currently displayed histogram
        filename: text file to save
        '''
        if self.populationValues is None or len(self.populationValues) ==
0:
            return 'Nothing to save'
        output = open(filename, 'w')
        output.write('Blob\t{}\n'
                    .format(self.metrics[self.populationMetric]))
        for i,b in enumerate(self.model.blobCollection[
                                self.model.currentBlobs].blobs):
```

```
            output.write('0_x_{0:.0f}y_{1:.0f}\t{2}\n'.format(b.X, b.Y,
                         self.populationValues[i]))
        return 'Saved histogram values'

    def saveHistImage(self, filename):
        '''
        Saves the current histogram image
        filename: image file to write
        '''
        self.fig.savefig(filename)

    def getFilteredBlobs(self):
        '''
        Get the set of blobs which pass the current filters
        returns list of blobLists with the filters already set
        '''
        result = []
        #low intensity
        if self.lowIntens is not None:
            if self.lowLimit is not None:
                tempbool = (self.populationValues < self.lowIntens) &
                                      (self.populationValues >
                                            self.lowLimit)
            else:
                tempbool = self.populationValues < self.lowIntens

            lowblbs = [self.blobSet.blobs[i] for i in
                           np.where(tempbool)[0]]
            result.append(self.blobSet.partialDeepCopy(lowblbs))
            result[-1].filters
                    .append(self._getFilterDescription(self.lowLimit,
                                                        self.lowIntens))

        #high intensity
        if self.highIntens is not None:
            if self.highLimit is not None:
                tempbool = (self.populationValues >  self.highIntens) &
                                 (self.populationValues < self.highLimit)
            else:
                tempbool = self.populationValues >  self.highIntens

            highblbs = [self.blobSet.blobs[i] for i in
                                 np.where(tempbool)[0]]
            result.append(self.blobSet.partialDeepCopy(highblbs))
            result[-1].filters
                    .append(self._getFilterDescription(self.highIntens,
                                                        self.highLimit))

        return result

    def _getFilterDescription(self, lowVal, highVal):
        '''
        returns a succinct string description of the current filter set
        lowVal: low value, part of # < channel < #
        highVal: high value, other part of # < channel < #
        '''
        result = ''
        if lowVal is None and highVal is None:
            return None

        channel = 'c{}[{}]'.format(self.imgInd,
              self.metrics[self.populationMetric])

        if lowVal is not None:
            result += "{:.1f}<".format(lowVal)
        result += channel
        if highVal is not None:
            result += "<{:.1f}".format(highVal)
        result += ';'

        result += 'max' if self.reduceMax else 'mean'
        result += ';offset={}'.format(self.offset)

        return result

    def redraw_zoom(self):
        '''
        redraw the widget after a zoom change,
        does not update the underlying graph
        '''
        #find range of the x axis scaled by zoom
        rng = (self.xhi - self.xlo) / 2**(self.zoomLvl+1)
        #determine center position
        if self.xcent is None:
            self.xcent = (self.xhi - self.xlo) / 2
        #find high and low positions, center +/- range
        (low, high) = (self.xcent - rng, self.xcent + rng)
        #keep low and high bounded by the min and max
        (low, high) = (self.xlo if low < self.xlo else low,
                                self.xhi if high > self.xhi else high)
        #if no zoom, autoscale
        if self.zoomLvl == 0:
            self.axes.autoscale(True, 'both')
        else:#autoscale y but use high and low for x
            self.axes.set_xlim([low,high])
            self.axes.autoscale(True, 'y')

        self.draw()

    def update_figure(self):
        '''
        redraw the figure by recalculating the graph and recoloring
        The blob subsets are passed back to the model
        '''
        if self.populationValues is None:
            self.axes.cla()
        else:
            #draw bar chart of entire population
            self.axes.bar(self.bins, self.counts,
                          width = self.bins[0] - self.bins[1],
                          color = GUIConstants
                                .BAR_COLORS[self.populationMetric])
            self.axes.hold(True)
            blbSubset = []
            blbColors = []
            #handle low intens
            if self.lowIntens is not None:
                if self.lowLimit is not None:
                    #tempbool is the bins that pass the filter
                    tempbool = (self.bins < self.lowIntens) &
                                (self.bins > self.lowLimit)
                    #tempbool2 is the blobs that pass the filter
                    tempbool2= (self.populationValues < self.lowIntens) &
                                (self.populationValues > self.lowLimit)
                else:
                    tempbool = self.bins < self.lowIntens
                    tempbool2 = self.populationValues < self.lowIntens
                #draw the low threshold bars
                self.axes.bar(self.bins[tempbool], self.counts[tempbool],
                              width = self.bins[0]-self.bins[1],
                              color = GUIConstants.LOW_BAR)
                #add the low threshold blobs to the blob subset
                          to pass to slideCanvas
                if np.any(tempbool2):
                    blbSubset.append(copy(self.blobSet))
                    blbSubset[-1].blobs = [self.blobSet.blobs[i] for i in
                                                  np.where(tempbool2)[0]]
                    blbSubset[-1].description = 'low'
                    blbSubset[-1].threshCutoff = int(self.lowIntens)
                    blbColors.append(GUIConstants.LOW_BAR)

            #handle high intens
            if self.highIntens is not None:
                if self.highLimit is not None:
                    tempbool = (self.bins >  self.highIntens) &
                                (self.bins < self.highLimit)
                    tempbool2=(self.populationValues > self.highIntens) &
                                (self.populationValues < self.highLimit)
                else:
                    tempbool = self.bins >  self.highIntens
                    tempbool2 = self.populationValues >  self.highIntens
                #draw the high threshold bars
                self.axes.bar(self.bins[tempbool], self.counts[tempbool],
                              width = self.bins[0]-self.bins[1],
                              color = GUIConstants.HIGH_BAR)
                #add the high threshold blobs to the blob subset
                    to pass to slideCanvas
                if np.any(tempbool2):
                    blbSubset.append(copy(self.blobSet))
                    blbSubset[-1].blobs = [self.blobSet.blobs[i] for i in
                                                  np.where(tempbool2)[0]]
                    blbSubset[-1].color = GUIConstants.HIGH_BAR
                    blbSubset[-1].description = 'high'
                    blbSubset[-1].threshCutoff = int(self.highIntens)
                    blbColors.append(GUIConstants.HIGH_BAR)

            #handle single bar selected
            if self.singleBar is not None:
                temp = self.bins - self.singleBar
                ind = int(np.sum(temp < 0))
                ind = 0 if ind < 0 else len(self.bins)-1 if
```

```
                ind >= len(self.bins) else ind
            #draw the single bar
            self.axes.bar(self.bins[ind], self.counts[ind],
                          width = self.bins[0]-self.bins[1],
                          color = GUIConstants.SINGLE_BAR)
            #add the single bar blobs to the subset for slideCanvas
            tempbool = (self.populationValues < self.bins[ind])
            if ind == len(self.bins) -1:
                tempbool = self.populationValues >= self.bins[ind-1]
            elif ind != 0:
                tempbool = tempbool & (self.populationValues
                                        >= self.bins[ind-1])
            if np.any(tempbool):
                blbSubset.append(copy(self.blobSet))
                blbSubset[-1].blobs = [self.blobSet.blobs[i] for i in
                                              np.where(tempbool)[0]]
                blbSubset[-1].description = 'single'
                blbSubset[-1].threshCutoff = int(self.bins[ind])
                if self.moveSlide == True:
                    firstBlob = blbSubset[-1].blobs[0]
                    self.model.slide.pos = [firstBlob.X, firstBlob.Y]
                    self.moveSlide = False
                blbColors.append(GUIConstants.SINGLE_BAR)


        #draw lines displaying the values used for filtering
        #a single blob to highlight
        if self.singleBlob is not None:
            if self.singleBlob >= 0 and
                      self.singleBlob < len(self.populationValues):
                self.axes.vlines(
                      self.populationValues[self.singleBlob], 0,
                      self.axes.get_ylim()[1],
                      colors = GUIConstants.SINGLE_BLOB)
        #draw limits
        if self.lowLimit is not None:
            self.axes.vlines(self.lowLimit, 0,
                            self.axes.get_ylim()[1],
                            colors = GUIConstants.LOW_BAR,
                            linestyles='dashed')
        if self.highLimit is not None:
            self.axes.vlines(self.highLimit, 0,
                            self.axes.get_ylim()[1],
                            colors = GUIConstants.HIGH_BAR,
                            linestyles='dashed')

        #draw thresholds
        if self.lowIntens is not None:
            self.axes.vlines(self.lowIntens, 0,
                            self.axes.get_ylim()[1],
                            colors = GUIConstants.LOW_BAR,
                            linestyles='dashdot')
        if self.highIntens is not None:
            self.axes.vlines(self.highIntens, 0,
                self.axes.get_ylim()[1],
                colors = GUIConstants.HIGH_BAR,
                linestyles='dashdot')

        #tell slide canvas about the new subset
        self.master.report_blbsubset((blbSubset, blbColors))

    self.axes.hold(False)
    #update the axes labels and x axis limits
    self.axes.set_ylabel('Count')
    if self.populationMetric == 3:
        self.axes.set_xlabel('Size')
        self.xlo, self.xhi = self.axes.get_xlim()
    elif self.populationMetric == 4:
        self.axes.set_xlabel('Circularity')
        self.xlo, self.xhi = self.axes.get_xlim()
    elif self.populationMetric == 5:
        self.axes.set_xlabel('Distance')
        self.xlo, self.xhi = self.axes.get_xlim()

    #colors are labeled as intensity and limited to 0,255
    else:
        self.xlo, self.xhi = 0,255
        self.axes.set_xlabel('Intensity')

    self.redraw_zoom()

```

GUICanvases/microMSModel.py

```python
from PIL import ImageDraw, ImageFont
import matplotlib as mpl
from matplotlib.path import Path
from matplotlib.collections import PatchCollection
import matplotlib.pyplot as plt
import os
import random
from scipy.spatial.distance import pdist
import numpy as np
from copy import deepcopy, copy

from GUICanvases import GUIConstants

from ImageUtilities import slideWrapper
from ImageUtilities import blobFinder
from ImageUtilities import blob
from ImageUtilities import TSPutil
from ImageUtilities.enumModule import Direction, StepSize
from ImageUtilities import blobList

from CoordinateMappers import supportedCoordSystems

class MicroMSModel(object):
    '''
    The model of a microMS experiment consisting of
        a slide, blob finder, and blobs
    Performs several vital functions for interacting
        with each object and maintains a list of blobs
    '''
    def __init__(self, GUI):
        '''
        Initialize a new model setup.  Slide starts as None.
        The coordinateMapper is set as the first mapper
              of the supported mappers.
        Also calls self.resetVariables to clear other instance variables.
        GUI: the supporting GUI
        '''
        self.slide = None
        self.coordinateMapper = supportedCoordSystems.supportedMappers[0]
        self.GUI = GUI
        self.resetVariables()

    def setupMicroMS(self, filename):
        '''
        Loads an image and sets up a new session
        filename: the image to load
        '''
        self.slide = slideWrapper.SlideWrapper(filename)
        self.resetVariables()

    def resetVariables(self):
        '''
        Clears and initializes all instance variables
        '''
        self.blobCollection = [blobList.blobList(self.slide) for i in
                  range(10)]

        self.setCurrentBlobs(0)
        self.tempBlobs = None
        self.histogramBlobs = None
        self.histColors = None
        self.coordinateMapper.clearPoints()
        self.mirrorImage = False
        self.showPatches = True
        self.drawAllBlobs = False
        self.showPrediction = False
        self.showThreshold = False

    def setCoordinateMapper(self, newMapper):
        '''
        Sets a new coordinate mapper and clears its points
        newMapper: the new instance of coordinateMapper to use
        '''
        self.coordinateMapper = newMapper
        self.coordinateMapper.clearPoints()

    def saveEntirePlot(self, fileName):
        '''
        saves the entire slide image at the current zoom level
        fileName: the file to write to
        *NOTE: this can take a while to run and generate
              large files at max zoom
        '''
        #save the current size and position
        size, pos = self.slide.size, self.slide.pos
        #match size to whole slide, position at center
        self.slide.size, self.slide.pos = \
            (self.slide.dimensions[0]//2**self.slide.lvl,
             self.slide.dimensions[1]//2**self.slide.lvl), \
            (self.slide.dimensions[0]//2, self.slide.dimensions[1]//2)

        #get whole image
        wholeImg = self.slide.getImg()
        draw = ImageDraw.Draw(wholeImg)

        #markup image
        linWid = 1 if 6-self.slide.lvl < 1 else 6-self.slide.lvl
        tfont = ImageFont.truetype("arial.ttf",linWid+6)
        #for each blob list
        for ii in range(len(self.blobCollection)):
            if self.blobCollection[ii].length() > 0:
                    drawnlbls = set()
                    drawlbl = self.blobCollection[ii].blobs[0].group
                          is not None
                    #for each blob
                    for i,gb in enumerate(self.blobCollection[ii].blobs):
                        p = self.slide.getLocalPoint((gb.X,gb.Y))
                        rad = gb.radius/2**self.slide.lvl
                        #draw blob outline
                        draw.ellipse((p[0]-rad, p[1]-rad,
                                p[0]+rad, p[1]+rad),
                                outline=GUIConstants.MULTI_BLOB[ii])
                        #draw label if group exists
                        if drawlbl and gb.group not in drawnlbls:
                            draw.text((p[0]+10/2**self.slide.lvl,
                                      p[1]-10/2**self.slide.lvl)
                                      str(gb.group),
                                            font=tfont,
                                      fill=GUIConstants.EXPANDED_TEXT)
                            drawnlbls.add(gb.group)

        #save image
        wholeImg.save(fileName)

        #restore size and position
        self.slide.size, self.slide.pos = size, pos

    def saveCurrentBlobFinding(self, filename):
        '''
        Save the current blob finder and currently selected blob list
        filename: file to save to
        '''
        #slide not set up
        if self.slide is None:
            return "No slide loaded"
        #current list is empty
        if self.blobCollection[self.currentBlobs].length() == 0:
            return "List {} contains no blobs!"
                    .format(self.currentBlobs +1)
                          #plus one for GUI display
        #save blobs
        self.blobCollection[self.currentBlobs].saveBlobs(filename)
        return "Saved blob information of list {}"
              .format(self.currentBlobs+1)

    def saveHistogramBlobs(self, filename):
        '''
        Save up to 3 files for different histogram filters
        filename: the filename to save
        '''
        #slide not set up
        if self.slide is None:
            return "No slide loaded"
        #no histogram blobs to save
        if self.histogramBlobs is None or len(self.histogramBlobs) == 0:
            return "No histogram divisions provided"
        #save different divisions
        f, ex = os.path.splitext(filename)
        for blbs in self.histogramBlobs:
            if blbs.length() > 0:
                blbs.saveBlobs('{}_{}_{}{}'
                          .format(f, blbs.description,
                          blbs.threshCutoff, ex))
        return "Saved histogram divisions with base name {}"
              .format(os.path.split(f)[1])

    def saveAllBlobs(self, filename):
        '''
        Save each list of blobs in its own list
        filename: a full filename with extension.
```

```
            The list number will be added as such:
            dir/test.txt -> dir/test_1.txt
        '''
        #slide not set up
        if self.slide is None:
            return "No slide loaded"
        f, ex = os.path.splitext(filename)
        #save each blob list
        for i, blbs in enumerate(self.blobCollection):
            if blbs.length() > 0:
                blbs.saveBlobs('{}_{}{}'.format(f, i, ex))

        return "Saved blobs with base name '{}'"
              .format(os.path.split(f)[1])

    def saveCoordinateMapper(self, filename):
        '''
        Save the current coordinate mapper
        filename: file to save to
        '''
        #no fiducials trained
        if len(self.coordinateMapper.pixelPoints) < 1:
            return "No coordinates to save"

        self.coordinateMapper.saveRegistration(filename)
        return "Saved coordinate mapper"

    def saveInstrumentPositions(self,filename, tspOpt, maxPoints = None):
        '''
        save positions of blobs in instrument coordinate system
        fileName: file to save to
        tspOpt: bool indicating whether or not
              to perform traveling salesman optimization
        maxPoints: maximum number of blobs to save.
              Default (None) saves all
        '''
        #check if the file can be saved
        if len(self.coordinateMapper.physPoints) < 2:
            return "Not enough training points to save instrument file"

        if self.blobCollection[self.currentBlobs].length() == 0:
            return "No blobs to save"

        #get current blob list
        blobs = self.blobCollection[self.currentBlobs].blobs
        #if maxPoints is valid
        if maxPoints is not None and maxPoints > 0 and
                            maxPoints < self.currentBlobLength():
            #obtain a random sample of blobs
            blobs = random.sample(blobs,maxPoints)

        #if tspOpt is requested
        if tspOpt == True:
            #reorder visit order
            soln = TSPutil.TSPRoute(blob.blob.getXYList(blobs))
            blobs = [blobs[i] for i in soln]

        #save list of blobs
        self.coordinateMapper.saveInstrumentFile(filename,
                                                  blobs)
        return "Saved instrument file of list {}"
                    .format(self.currentBlobs +1 )

    def saveInstrumentRegistrationPositions(self, filename):
        '''
        Save fiducial locations in the instrument coordinate system
        '''
        if len(self.coordinateMapper.physPoints) < 2:
            return "Not enough training points to save fiducial
                    locations"
        self.coordinateMapper.saveInstrumentRegFile(filename)
        return "Saved instrument registration positions"

    def loadCoordinateMapper(self,filename):
        '''
        load a prior registration file
        changes the current mapper to the one specified in the file
        filename: file to load
        returns a status display string, and the index of the new mapper
        '''
        #get old index
        old = supportedCoordSystems.supportedMappers
                          .index(self.coordinateMapper)
        #get first line in file
        reader = open(filename,'r')
        line = reader.readline().strip()
        reader.close()
        #see if that is a name of a coordinatemapper
        try:
            i = supportedCoordSystems.supportedNames.index(line)
        except:
            return 'Unsupported instrument: {}'.format(line), old

        #See if mapper has changed to warn the user
        result = 'Loaded {} registration'.format(line)
        if i != old:
            result = 'Warning, changing instrument to {}'.format(line)
            self.coordinateMapper =
                    supportedCoordSystems.supportedMappers[i]
        self.coordinateMapper.loadRegistration(filename)
        return result, i

    def loadBlobFinding(self, filename):
        '''
        Loads the blobs in the provided filename to
              the current list of blobs and sets
              the blobfinder to the previous values
        filename: file to load
        '''
        self.blobCollection[self.currentBlobs].loadBlobs(filename)
        return "Finished loading blob positions into list {}"
                          .format(self.currentBlobs+1)

    def loadInstrumentPositions(self, filename):
        '''
        Load an instrument position file to the current blob list.
        Will not have proper radius, but should retain the groups.
        filename: file to load
        '''
        self.blobCollection[self.currentBlobs].blobs = \
            self.coordinateMapper.loadInstrumentFile(filename)
        self.blobCollection[self.currentBlobs].generateGroupLabels()
        return "Finished loading instrument file into list {}"
                                  .format(self.currentBlobs+1)

    def currentBlobLength(self):
        '''
        Gets the length of the current blob list
        '''
        return self.blobCollection[self.currentBlobs].length()

    def currentInstrumentExtension(self):
        '''
        Gets the instrument extension of the current coordinate mapper
        '''
        return self.coordinateMapper.instrumentExtension

    def runGlobalBlobFind(self):
        '''
        Performs global blob finding on the current slide
              and sets to current blob list
        '''
        if self.slide is None:
            return "No slide was open"
        return self.blobCollection[self.currentBlobs].blobSlide() +
                    " in list {}".format(self.currentBlobs+1)

    def updateCurrentBlobs(self, newBlobs):
        if not isinstance(newBlobs, blobList.blobList):
            raise ValueError('New blobs must be a blobList')

        #find first unused blob index
        for i in range(len(self.blobCollection)):
            if self.blobCollection[i].length() == 0:
                #add new blobs
                self.blobCollection[i] = newBlobs
                self.setCurrentBlobs(i)
                return

    def distanceFilter(self, distance):
        '''
        filters the global blob list to remove blobs
              which are closer than 'distance' pixels
        the prior list is stored in previous current index
        distance: distance threshold
        '''
        if self.currentBlobLength() == 0:
            return "No blobs to filter"

        self.updateCurrentBlobs(
              self.blobCollection[self.currentBlobs]
```

```
                        .distanceFilter(distance, verbose = True))

        return "Finished distance filter in list {}"
                        .format(self.currentBlobs+1)

    def roiFilter(self):
        '''
        filters the blob list to remove blobs outside the current ROI
        '''
        if self.currentBlobLength() == 0:
            return "No blobs to filter"
        if len(self.blobCollection[self.currentBlobs].ROI) < 3:
            return "No ROI selected"
        startLen = self.currentBlobLength()
        self.updateCurrentBlobs(
              self.blobCollection[self.currentBlobs]
                    .roiFilter())
        endLen = self.currentBlobLength()
        return "{} blobs removed, {} remain in list {}"
              .format(startLen - endLen, endLen, self.currentBlobs+1)

    def roiFilterInverse(self):
        '''
        filters the blob list to remove blobs inside the current ROI
        '''
        if self.currentBlobLength() == 0:
            return "No blobs to filter"
        if len(self.blobCollection[self.currentBlobs].ROI) < 3:
            return "No ROI selected"
        startLen = self.currentBlobLength()
        self.updateCurrentBlobs(
              self.blobCollection[self.currentBlobs]
                    .roiFilterInverse())
        endLen = self.currentBlobLength()
        return "{} blobs removed, {} remain in list {}"
              .format(startLen - endLen, endLen, self.currentBlobs+1)


    def hexPackBlobs(self, separation, layers, dynamicLayering = False):
        '''
        expands each blob into hexagonally closest packed positions
        sep: minimum separation between points
        layers: number of layers to generate
        dynamicLayering: adjust the number of layers with the blob radius
        '''
        self.updateCurrentBlobs(self.blobCollection[self.currentBlobs]\
            .hexagonallyClosePackPoints(separation,
                                        layers,
                                        dynamicLayering = dynamicLayering))


    def rectPackBlobs(self, separation, layers, dynamicLayering = False):
        '''
        expands each blob into rectangularly packed positions
        sep: minimum separation between points
        layers: number of layers to generate
        dynamicLayering: adjust the number of layers with the blob radius
        '''
        self.updateCurrentBlobs(self.blobCollection[self.currentBlobs]\
            .rectangularlyPackPoints(separation,
                                    layers,
                                    dynamicLayering = dynamicLayering))

    def circularPackBlobs(self, separation, maxShots, offset):
        '''
        expands each blob into circularly packed
        positions around the blob
        sep: minimum separation between spots
        shots: maximum number of spots to place around each blob
        offset: offset from the current circumference,
        offset > 0 places spots outside the current blob
        '''
        self.updateCurrentBlobs(self.blobCollection[self.currentBlobs]\
            .circularPackPoints(separation, maxShots,
                                offset, minSpots = 4))

    def analyzeAll(self):
        '''
        if the current mapper is connected to an instrument,
        triggers analysis of all blobs currently found
        '''
        #get all pixel points and translate to motor coords
        if self.currentBlobLength() == 0:
           return "No targets currently selected"
        if len(self.coordinateMapper.physPoints) <= 2:
           return "Not enough training points"
        if self.coordinateMapper.connectedInstrument is None or \
            self.coordinateMapper.connectedInstrument.connected == False:
           return "No connected instrument"

        targets = list(map(lambda b:
                          self.coordinateMapper.translate((b.X, b.Y)),
                          self.blobCollection[self.currentBlobs].blobs))

        #send to connected instrument
        self.coordinateMapper.connectedInstrument.collectAll(targets)

        return "Finished collection"

    def setBlobSubset(self, blobSubset):
        '''
        Sets the histogram blobs supplied by a histcanvas
        blobSubset: an odd object, tuple of lists,
              first is a blobList, second a list of colors
        '''
        self.histogramBlobs = blobSubset[0]
        self.histColors = blobSubset[1]

    def reportSlideStep(self, direction, stepSize):
        '''
        Moves the slide in the specified direction,
              taking into account mirroring
        direction: a slideWrapper.direction in the observed direction
        stepSize: enum dictating if the step size
        '''
        if self.slide is not None:
            if self.mirrorImage:
                if direction == Direction.left:
                    self.slide.step(Direction.right, stepSize)
                elif direction == Direction.right:
                    self.slide.step(Direction.left, stepSize)
                else:
                    self.slide.step(direction, stepSize)

            self.slide.step(direction, stepSize)

    def testBlobFind(self):
        '''
        Performs a test blob find on the current position
        Sets the zoom level to the maximum
              value to match test blob finding
        '''
        self.slide.lvl = 0
        if self.slide is not None:
            self.tempBlobs = self.blobCollection[self.currentBlobs]
                                                 .blobFinder.blobImg()

    def setCurrentBlobs(self, ind):
        '''
        Sets the current blob index to the specified value
        ind: integer value of list to show
        '''
        self.currentBlobs = ind
        if self.GUI is not None:
            self.GUI.setTitle(self.currentBlobs)

    def reportSize(self, newSize):
        '''
        Sets the size of the slidewrapper to the specified value.
        Sets the max number of pixels to 600 but keeps the aspect ratio
        newSize = (width, height)
        '''
        w,h = newSize
        factor = 600/max(w,h)
        w, h = int(w*factor), int(h*factor)
        self.slide.size = [w, h]

    def getCurrentImage(self):
        '''
        gets the image to display, accounting for showing thresholds
        '''
        #show the threshold image produced by blobfinder helper method
        if self.showThreshold:
            im, num = blobFinder
                    .blobFinder._blbThresh(self.slide.getImg(),
          self.blobCollection[self.currentBlobs].blobFinder.colorChannel,
          self.blobCollection[self.currentBlobs].blobFinder.threshold)
            return im
        #else, use current image view
        else:
            return self.slide.getImg()

    def getPatches(self, limitDraw):
        '''
        Gets the patches of all blobs, registration
        marks and predicted points.
        limitDraw: boolean toggle to limit the number of blobs to draw
        '''
        ptches = []
        #nothing requested or nothing to show
        if self.showPatches == False or self.slide is None:
            return PatchCollection(ptches)

        #temp blobs from blob finding test.  Only drawn once
        if self.tempBlobs is not None:
            ptches = [plt.Circle((blb.X, blb.Y),
                                  blb.radius,
                                  color = GUIConstants.TEMP_BLOB_FIND,
                                  linewidth = 1,
                                  fill = False)
                       for blb in self.tempBlobs]
            #reset temp blobs
            self.tempBlobs = None
            #return patches, if none to show match_original
                    needs to be false
            return PatchCollection(ptches,
                          match_original=(len(ptches) != 0))

        #draw predicted points from coordinate mapper
        lineWid = 1 if 6-self.slide.lvl < 1 else 6-self.slide.lvl
        if self.showPrediction and
                    len(self.coordinateMapper.physPoints) >= 2:
            points, inds = self.slide
                          .getPointsInBounds(
                          self.coordinateMapper.predictedPoints())
            ptches.extend(
                    [plt.Circle(p,
                          GUIConstants.FIDUCIAL_RADIUS/2**self.slide.lvl,
                                color = GUIConstants.PREDICTED_POINTS,
                                linewidth = lineWid,
                                fill = False)
                     for p in points]
                )

        #draw fiducial locations, showing the worst
              FLE in a different color
        worstI = -1
        if len(self.coordinateMapper.physPoints) > 2:
            worstI = self.coordinateMapper.highestDeviation()
        points, inds = self.slide
                    .getPointsInBounds(self.coordinateMapper.pixelPoints)
        for i,p in enumerate(points):
            if inds[i] == worstI:
                ptches.append(
                    plt.Circle(p,
                          GUIConstants.FIDUCIAL_RADIUS/2**self.slide.lvl,
                          color = GUIConstants.FIDUCIAL_WORST,
                          linewidth = lineWid,
```

```python
                               fill=False)
                    )
            else:
                ptches.append(
                    plt.Circle(p,
                          GUIConstants.FIDUCIAL_RADIUS/2**self.slide.lvl,
                          color = GUIConstants.FIDUCIAL,
                          linewidth = lineWid,
                          fill=False)
                    )

        #draw region of interest
        ptches.extend(self.getROIPatches())

        #draw histogram blobs
        if self.histogramBlobs is not None and
                    len(self.histogramBlobs) != 0:
            for i, blbs in enumerate(self.histogramBlobs):
                ptches.extend(blbs.getPatches(limitDraw,
                                            self.slide,
                                            self.histColors[i]))

        #draw blobs
        else:
            #draw all blob lists with their own color
            if self.drawAllBlobs == True:
                for j, blobs in enumerate(self.blobCollection):
                    ptches.extend(blobs.getPatches(limitDraw,
                                            self.slide,
                                            GUIConstants.MULTI_BLOB[j]))

            #show only the current blob list
            else:
                ptches.extend(
                          self.blobCollection[self.currentBlobs]
                                .getPatches(limitDraw, self.slide,
                            GUIConstants.MULTI_BLOB[self.currentBlobs]))

        #return list of patches as a patch collection,
              if none match_original must be false
        return PatchCollection(ptches, match_original=(len(ptches) != 0))

    def getROIPatches(self, newPoint = None, append = False):
        ptches = []
        tROI = self.blobCollection[self.currentBlobs]
                    .getROI(newPoint,
                    GUIConstants.ROI_DIST *2**self.slide.lvl,
                    append)

        if len(tROI) > 1:
            verts = []
            for roi in tROI:
                verts.append(self.slide.getLocalPoint(roi))
            verts.append(self.slide.getLocalPoint(tROI[0]))
            ptches.append(
                    mpl.patches.PathPatch(Path(verts, None),
                                      color = GUIConstants.ROI,
                                        fill = False))

        return ptches

    def reportROI(self, point, append = False):
        '''
        Handles ROI additions and removals based on position
        point: the point in global coordinates
        '''
        self.blobCollection[self.currentBlobs].ROI = \
            self.blobCollection[self.currentBlobs]
                    .getROI(point,
                                GUIConstants.ROI_DIST *2**self.slide.lvl,
                                append)

    def drawLabels(self, axes):
        '''
        draw text labels on the supplied axis.
        Assume the axis is displaying the
        slide image and blobs of the current state of everything.
        '''
        if self.slide is None or self.showPatches == False:
            return

        #fiducial labels
        lineWid = 1 if 6-self.slide.lvl < 1 else 6-self.slide.lvl
        #draw fiducial labels, showing the worst FLE in a different color
        worstI = -1
        if len(self.coordinateMapper.physPoints) > 2:
            worstI = self.coordinateMapper.highestDeviation()
        points, inds = self.slide.getPointsInBounds(
                                self.coordinateMapper.pixelPoints)
        for i,p in enumerate(points):
            if inds[i] == worstI:
                col = GUIConstants.FIDUCIAL_WORST
            else:
                col = GUIConstants.FIDUCIAL
            axes.text(p[0] +
                    GUIConstants.FIDUCIAL_RADIUS/2**self.slide.lvl,
                      p[1] -
                    GUIConstants.FIDUCIAL_RADIUS/2**self.slide.lvl,
                    self.coordinateMapper.predictLabel(
                        self.coordinateMapper.physPoints[inds[i]]),
                        fontsize = lineWid + 6,
                        fontweight='bold',
                        color = col,
                        bbox=dict(
                          facecolor=GUIConstants.FIDUCIAL_LABEL_BKGRD))
        #show group labels
        #hist blobs have no text
        if self.histogramBlobs is not None and
                    len(self.histogramBlobs) != 0:
            pass
        #normal blobs can have group labels
        else:
            pass
            #show group names of all lists
            if self.drawAllBlobs == True:
                for blobs in self.blobCollection:
                    self._drawBlobLabels(axes, blobs, lineWid)

            #show only the current list
            else:
                self._drawBlobLabels(axes,
                                self.blobCollection[self.currentBlobs],
                                lineWid)

    def _drawBlobLabels(self, axes, blobs, lineWid):
        '''
        Helper method to draw blob labels onto the provided axis
        axes: matplotlib axes to draw text to
        blobs: blobList with labels to draw
        lineWid: the linewidth to use for drawing
        '''
        #get grouplabels from blobs
        labels = list(blobs.groupLabels.keys())
        pos = list(blobs.groupLabels.values())
        if len(labels) != 0:
            points, inds = self.slide.getPointsInBounds(pos)
            for i,p in enumerate(points):
                #add offset from normal position
              axes.text(p[0]+
                    GUIConstants.DEFAULT_RADIUS/2**self.slide.lvl,
                        p[1]-
                    GUIConstants.DEFAULT_RADIUS/2**self.slide.lvl,
                    labels[inds[i]],
                    fontsize=lineWid+6,
                    color=GUIConstants.EXPANDED_TEXT)

    def reportInfoRequest(self, localPoint):
        '''
        Handles a request for image/blob
              information at the supplied local point
        localPoint: (x,y) tuple of the query point
            in the local coordinate space
            of the slide image
        returns a string description of the point
        '''
        #nothing to query against
        if self.slide is None:
            return "No slide loaded"

        point = self.slide.getGlobalPoint(localPoint)
        #if the histogram canvas is shown, highlight that blob's location
        if self.GUI is not None and self.GUI.showHist:
            #find blob if user clicked in bounds
            if self.blobCollection[self.currentBlobs] is not None and \
                self.blobCollection[self.currentBlobs].length() > 0:

                points, inds = self.slide
                          .getPointsInBounds(
                          blob.blob.getXYList(
                          self.blobCollection[self.currentBlobs].blobs))
                found = False
                for i,p in enumerate(points):
                    #see if click point is within radius
                    if (localPoint[0]-p[0])**2 +
                        (localPoint[1] - p[1])**2 <= \
                        (self.blobCollection[self.currentBlobs]
                          .blobs[inds[i]].radius/2**self.slide.lvl)**2:
                            self.GUI.histCanvas.singleBlob = inds[i]
                            found = True
                            break
                #if not found, set to None
                if not found:
                    self.GUI.histCanvas.singleBlob = None

        #get pixel color and alpha (discarded)
        try:
            r,g,b,a = self.slide.getImg().getpixel(localPoint)
        except IndexError:
            r,g,b = 0,0,0

        #get the size and circ of an area > thresh if on blb view
        if self.showThreshold:
            area,circ = self.blobCollection[self.currentBlobs]
                          .blobFinder.getBlobCharacteristics(localPoint)
            return "x = %d, y = %d r,g,b = %d,
                    %d,%d\tArea = %d\tCirc = %.2f"
                    %(point[0], point[1], r, g, b, area, circ)
        #show rgb and x,y location
        else:
            return "x = %d, y = %d r,g,b = %d,%d,%d"
                                %(point[0], point[1], r, g, b)

    def reportFiducialRequest(self, localPoint,
                                removePoint, extras = None):
        '''
        handles a fiducial request.
        localpoint: (x,y) tuple in the image coordinate system
        removePoint: boolean toggle.  If true,
               the closest fiducial is removed
        extras: a debugging object to bypass GUI display.
        Must define text and ok
        '''
        #no slide to register against
        if self.slide is None:
            return "No slide loaded"

        globalPos = self.slide.getGlobalPoint(localPoint)

        #shift RMB to remove closest fiducial
        if removePoint:
            if len(self.coordinateMapper.physPoints) == 0:
                return "No points to remove"
            self.coordinateMapper.removeClosest(globalPos)
            return "Removed fiducial"

        #get physical location from user
        else:
            #mapper returns predicted location
            predicted = self.coordinateMapper.predictName(globalPos)

            #prompt user
            if self.GUI is None and extras is None:
                return "No input provided"
            if extras is not None:#make this check first for debugging
                text = extras.text
                ok = extras.ok
            elif self.GUI is not None:
                text, ok = self.GUI.requestFiducialInput(predicted)

            if ok:
                #validate entry
                if self.coordinateMapper.isValidEntry(text):
                    #add position to mapper
                    self.coordinateMapper
                          .addPoints(globalPos,
                                self.coordinateMapper.extractPoint(text))
                    return "%s added at %d,%d"
                          % (text, globalPos[0], globalPos[1])
                else:
                    return "Invalid entry: {}".format(text)

    def reportBlobRequest(self, localPoint, radius):
        '''
        Tries to add the blob to the current blob list.
        If overlap with current blob, remove that point
        localPoint: (x,y) tuple in the image coordinate space
        radius: the radius of the new blob to be added
        '''
        #no slide to add blobs onto
        if self.slide is None:
            return "No slide loaded"

        globalPnt = self.slide.getGlobalPoint(localPoint)
        added, removeInd = self.blobCollection[self.currentBlobs]
                                .blobRequest(globalPnt, radius)
        if added == True:
            if self.GUI is not None and self.GUI.showHist:
                self.GUI.toggleHistWindow()
            return "Adding blob at {}, {}"
                    .format(globalPnt[0], globalPnt[1])
        else:
            if self.GUI is not None and self.GUI.showHist:
                self.GUI.histCanvas.removeBlob(removeInd)
            return "Removed blob at {}, {}"
                    .format(globalPnt[0], globalPnt[1])

    def requestInstrumentMove(self, localPoint):
        '''
        Handles requests for moving the connected instrument
        localPoint: (x,y) tuple in the current image coordinate system
        returns a string summarizing the effect of the action
        '''
        #no slide is set up
        if self.slide is None:
            return "No slide loaded"

        #the connected instrument isn't initialized or present
        if self.coordinateMapper.connectedInstrument is None or \
            not self.coordinateMapper.connectedInstrument.connected:
            return "Instrument not connected"

        #perform actual movement
        pixelPnt = self.slide.getGlobalPoint(localPoint)
        if len(self.coordinateMapper.physPoints) >= 2:
            motorPnt = self.coordinateMapper.translate(pixelPnt)
            self.coordinateMapper
                    .connectedInstrument
                    .moveToPositionXY(motorPnt)
            return "Moving to {:.0f}, {:.0f}"
                    .format(motorPnt[0], motorPnt[1])
        #not enough registration points
        else:
            return "Not enough training points"
```

GUICanvases/microMSWindow.py

```
from __future__ import unicode_literals
import os
from PyQt5 import QtGui, QtCore, QtWidgets

from CoordinateMappers import supportedCoordSystems
from CoordinateMappers import connectedInstrument

from ImageUtilities.slideWrapper import SlideWrapper
from ImageUtilities.enumModule import Direction, StepSize

from GUICanvases.histCanvas import HistCanvas
from GUICanvases.slideCanvas import SlideCanvas
from GUICanvases.popup import blbPopupWindow,
    gridPopupWindow, histPopupWindow
from GUICanvases.microMSModel import MicroMSModel
from GUICanvases import GUIConstants

class MicroMSQTWindow(QtWidgets.QMainWindow):
    '''
    A QT implementation of the MicroMS window.
    Interacts with the MicroMSModel, a SlideCanvas and a HistCanvas
    Mainly handles the menu, key presses, and coordinating canvases
    '''
    def __init__(self):
        '''
        initialize a new microMSQT window,
             setting up the layout and some instance variables
        '''
        QtWidgets.QMainWindow.__init__(self)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.main_widget = QtWidgets.QWidget(self)
        self.fileName = None

        #model with slide and blob data
        self.model = MicroMSModel(self)

        self.layout = QtWidgets.QHBoxLayout(self.main_widget)

        #new slide canvas for displaying the image
        #and handling mouse interactions.
        self.slideCanvas = SlideCanvas(self,
                                       self.model,
                                       self.main_widget,
                                       width=6, height=6, dpi=100)
        self.layout.addWidget(self.slideCanvas, stretch = 1)

        #histogram canvas for showing and interacting
        #with population level measurements
        self.histCanvas = HistCanvas(master=self,
                                 model=self.model,
                                 width=6, height=6, dpi=100)
        self.layout.addWidget(self.histCanvas, stretch = 1)
        self.histCanvas.hide()

        self.showHist = False


        self.main_widget.setFocus()
        self.setCentralWidget(self.main_widget)

        #dictionary of popup windows to keep handle of each
        self.popups = {
            'imageHelp'     :   self.createMessageBox(
                              GUIConstants.IMAGE_HOTKEYS,
                                          'Image Help'),
            'instHelp'      :   self.createMessageBox(
                              GUIConstants.INSTRUMENT_HOTKEYS,
                                          'Instrument Help'),
            'histHelp'      :   self.createMessageBox(
                              GUIConstants.HISTOGRAM_HOTKEYS,
                                          'Histogram Help'),
            'blobFind'      :   blbPopupWindow(self),
            'grid'          :   gridPopupWindow(self),
            'histOpts'      :   histPopupWindow(self.histCanvas, self)
            }

        self.setupMenu()

    def setupMenu(self):
        '''
        setup the menubar and connect instance functions
        '''
        #file menu
        self.file_menu = QtWidgets.QMenu('&File', self)

        #open button
        openFile = QtWidgets.QAction(QtGui.QIcon('open.png'),
                                                'Open', self)
        openFile.setShortcut('Ctrl+O')
        openFile.setStatusTip('Open new File')
        openFile.triggered.connect(self.fileOpen)
        self.file_menu.addAction(openFile)

        #decimation submenu
        decSub = QtWidgets.QMenu('Decimate...',self)
        self.file_menu.addMenu(decSub)
        decSub.addAction('Single Image', self.decimateImageSingle)
        decSub.addAction('Image Group', self.decimateImageGroup)
        decSub.addAction('Directory', self.decimateDirectory)

        #instrument selection
        instSub = QtWidgets.QMenu('&Instrument...', self)
        self.file_menu.addMenu(instSub)
        self.instruments = QtWidgets.QActionGroup(self, exclusive=True)
        self.instruments.triggered.connect(self.mapperChanged)
        #populate with all instruments currently supported
        for s in supportedCoordSystems.supportedNames:
            a = self.instruments.addAction(QtWidgets.QAction(s,
                                          instSub, checkable=True))
            instSub.addAction(a)
        self.instruments.actions()[0].setChecked(True)

        #save submenu
        saveSub = QtWidgets.QMenu('&Save...',self)
        self.file_menu.addMenu(saveSub)


        saveSub.addAction('&Instrument Positions',
                                     self.saveInstrumentPositions)
        saveSub.addAction('&Fiducial Positions',
                                     self.saveFiducialPositions)

        saveSub.addSeparator()

        saveSub.addAction('&Registration', self.saveReg)
        saveSub.addAction('&Current Blobs', self.saveCurrentFind)
        saveSub.addAction('&Histogram Divisions',
                               self.saveHistogramBlobs)
        saveSub.addAction('All Lists of Blobs', self.saveAllBlobs)

        saveSub.addSeparator()

        saveSub.addAction('&Image', self.saveImg,
                          QtCore.Qt.CTRL + QtCore.Qt.Key_S)
        saveSub.addAction('&Whole Image', self.saveWholeImg)

        saveSub.addSeparator()
        saveSub.addAction('Histogram Image',self.histSaveImage)
        saveSub.addAction('Histogram Values',self.histSaveValues)

        #load submenu
        loadSub = QtWidgets.QMenu('&Load...',self)
        self.file_menu.addMenu(loadSub)

        loadSub.addAction('&Registration', self.loadReg)
        loadSub.addAction('&Found Blobs', self.loadBlobFind)
        loadSub.addAction('&Instrument Positions',
                                     self.loadInstrumentPositions)

        #quit button
        self.file_menu.addAction('&Quit', self.fileQuit,
                                 QtCore.Qt.CTRL + QtCore.Qt.Key_Q)
        self.menuBar().addMenu(self.file_menu)

        #tools menu
        self.tools_menu = QtWidgets.QMenu('&Tools',self)

        #blob find
        self.tools_menu.addAction('&Blob Find', self.globalBlob)
        #blob options
        self.tools_menu.addAction('&Blob Options',self.blbPopup,
                                  QtCore.Qt.CTRL + QtCore.Qt.Key_B)
        #limit drawn blobs toggle
        self.limitDraw = QtWidgets.QAction('Limit Drawn Blobs',
                                     self.tools_menu, checkable=True)
        self.limitDraw.setChecked(True)
        self.tools_menu.addAction(self.limitDraw)
        self.tools_menu.addSeparator()
        #Histogram options
        self.tools_menu.addAction(
                         'Histogram Window',self.toggleHistWindow,
                          QtCore.Qt.CTRL + QtCore.Qt.Key_H)
        self.tools_menu.addAction('Histogram Options',self.histOptions)
        self.tools_menu.addAction('Pick Extremes',self.histSelect)
        self.tools_menu.addAction('Apply Filter',self.histFilter,
                                  QtCore.Qt.CTRL + QtCore.Qt.Key_A)
        #blob position options
        self.tools_menu.addSeparator()
        self.tools_menu.addAction('Distance Filter',self.distanceFilter)
        self.tools_menu.addAction('ROI Filter Retain',self.roiFilter)
        self.tools_menu.addAction('ROI Filter Remove',
                                              self.roiFilterInverse)
        self.tools_menu.addSeparator()
        self.tools_menu.addAction('Rectangular Pack', self.rectPack)
        self.tools_menu.addAction('Hexagonal Pack', self.hexPack)
        self.tools_menu.addAction('Circular Pack', self.circPack)

        #instrument settings
        self.tools_menu.addSeparator()
        self.tools_menu.addAction('Instrument Setting',self.gridPopup,
                                  QtCore.Qt.CTRL + QtCore.Qt.Key_G)
        self.menuBar().addSeparator()
        self.menuBar().addMenu(self.tools_menu)

        #device submenu
        self.inst_menu = QtWidgets.QMenu('Device', self)
        self.inst_menu.addAction('Establish Connection',
                               self.initializeInstrument)
        self.inst_menu.addAction('Set Dwell Time', self.setDwell)
        self.inst_menu.addAction('Set Wash Time', self.setWash)
        self.inst_menu.addAction('Analyze All', self.analyzeAll)

        self.menuBar().addSeparator()
        self.menuBar().addMenu(self.inst_menu)
        self.inst_menu.setEnabled(self.model.coordinateMapper
                                     .isConnectedToInstrument)

        #help menu
        self.help_menu = QtWidgets.QMenu('&Help', self)
        self.menuBar().addSeparator()
        self.menuBar().addMenu(self.help_menu)
        self.help_menu.addAction('&Image Hotkeys', self.imgHotkeyMsg)
        self.help_menu.addAction('&Instrument Hotkeys',
                                     self.instHotkeyMsg)
        self.help_menu.addAction('&Histogram Hotkeys',
                                     self.histHotkeyMsg)

    ###most of the following functions are simple
    #popups to parse input and pass to canvases
    def fileOpen(self, extras=None):
        '''
        open and setup a slide.  only ndpi and tif are supported
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getOpenFileName(
                self, 'Open File',
                filter='Slide Scans (*.ndpi *.tif)')[0]

        else:
            fileName = extras.fileName

        if fileName:
            self.setupCanvas(fileName)

    def decimateImageGroup(self, extras = None):
        '''
        decimate a tif file (to speed up zooming out) and open the file
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getOpenFileName(
                self, 'Open File to Decimate',
                filter='Slide Scans (*.tif)') [0]

        else:
            fileName = extras.fileName

        if fileName:
            SlideWrapper.generateDecimatedImgs(fileName)
            #open file once done
            self.setupCanvas(fileName)
            self.raise_()
            self.activateWindow()

    def decimateImageSingle(self, extras = None):
        '''
        decimate a single file and open the image group
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getOpenFileName(
                self, 'Open File to Decimate',
                filter='Slide Scans (*.tif)')[0]

        else:
            fileName = extras.fileName

        if fileName:
            (path, file) = os.path.split(fileName)
            SlideWrapper.generateDecimatedImage(path, file)
            #open file once done
            self.setupCanvas(fileName)
            self.raise_()
            self.activateWindow()

    def decimateDirectory(self, extras = None):
        '''
        decimate a tif file (to speed up zooming out) and open the file
        '''
        if extras is None or not hasattr(extras, 'directory'):
            directory = QtWidgets.QFileDialog
                                         .getExistingDirectory(self,
                                               'Open Directory to Decimate')

        else:
            directory = extras.directory

        if directory:
            SlideWrapper.decimateDirectory(directory)
            self.raise_()
            self.activateWindow()

    def setupCanvas(self, fileName):
        '''
        opens the file specified by filename and
             sets up some instance variables
        '''
        self.model.setupMicroMS(fileName)
        self.statusBar().showMessage("Opened {}".format(fileName))
        self.directory = os.path.dirname(fileName)
        self.fileName = os.path.splitext(os.path.basename(fileName))[0]
        self.setTitle(self.model.currentBlobs)
        self.showHist = False
        self.histCanvas.resetVariables(True, True)
        self.histCanvas.hide()
        self.model.reportSize((float(self.slideCanvas.size().width()),
                               float(self.slideCanvas.size().height())))
        self.model.slide.resetView()
        self.slideCanvas.draw()

    def setTitle(self, blobList):
        if self.fileName is not None:
            self.setWindowTitle('MicroMS: {}    (List #{})'
                                .format(self.fileName, blobList+1))
        else:
            self.setWindowTitle('MicroMS')


    def mapperChanged(self, action):
        '''
        action triggered by the mapper
             changing in the instrument submenu
        Changes the mapper of imagecanvas to the selected one
             and updates the device menu and canvas
        '''
        i = supportedCoordSystems.supportedNames.index(action.text())
        self.model.setCoordinateMapper(supportedCoordSystems
                                      .supportedMappers[i])
        self.inst_menu.setEnabled(self.model.coordinateMapper
                                .isConnectedToInstrument)
        self.slideCanvas.draw()

    def saveImg(self, extras = None):
        '''
        save the image of the image canvas to the selected location
        '''
        if self.model.slide is None:
            self.statusBar().showMessage("No image to save")
            return
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog
                                      .getSaveFileName(self,
                                               "Select save file",
                                               self.directory,
                                               filter='*.png')
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex

        else:
            fileName = extras.fileName

        if fileName:
            self.slideCanvas.savePlt(fileName)

    def saveWholeImg(self, extras = None):
        '''
        saves the entire image at the selected zoom
             to the selected location
        Can produce large images!!
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getSaveFileName(self,
                                                         "Select save file",
                                                         self.directory,
                                                         filter='*.png')
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex

        else:
            fileName = extras.fileName

        if fileName:
            self.model.saveEntirePlot(fileName)
            if extras is None or not hasattr(extras, 'fileName'):
                msg = QtWidgets.QMessageBox(self)
                msg.setText("Finished saving")
                msg.setWindowTitle("")
                msg.exec_()

    def saveAll(self, extras = None):
        '''
        saves files necessary for replicating the blob finding:
        -Blob finding file with pixel locations of
             spots and find parameters
        -Registration file with pixel to physical locations of fiducials
        '''
        if extras is None or not hasattr(extras, 'text'):
            text, ok = QtWidgets.QInputDialog
                                       .getText(self,'Save All',
                                               'Enter base filename:')

        else:
            text = extras.text
            ok = extras.ok

        if ok:
            self.statusBar().showMessage(
                self.model.saveCurrentBlobFinding(
                        os.path.join(self.directory, text+".txt"))
            )
            self.statusBar().showMessage(
                self.model.saveCoordinateMapper(
                        os.path.join(self.directory, text+".msreg"))
            )

    def saveReg(self, extras = None):
        '''
        save just the msreg file
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getSaveFileName(self,
                                                             "Select save file",
                                                             self.directory,
                                                             filter='*.msreg')
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex

        else:
            fileName = extras.fileName

        if fileName:
            self.statusBar().showMessage(
                self.model.saveCoordinateMapper(fileName)
            )

    def saveCurrentFind(self, extras = None):
        '''
        save blob finding of the current blob list
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getSaveFileName(self,
                                                             "Select save file",
                                                             self.directory,
                                                             filter='*.txt')
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex
        else:
            fileName = extras.fileName

        if fileName:
            self.statusBar().showMessage(
                self.model.saveCurrentBlobFinding(fileName)
            )

    def saveHistogramBlobs(self, extras = None):
        '''
        save blob finding of all histogram filters
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getSaveFileName(self,
                                                             "Select save file",
                                                             self.directory,
                                                        filter='*.txt')
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex
        else:
            fileName = extras.fileName

        if fileName:
            self.statusBar().showMessage(
                self.model.saveHistogramBlobs(fileName)
            )

    def saveAllBlobs(self, extras = None):
        '''
        save blob finding of all blob lists
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getSaveFileName(self,
                                                        "Select save file",
                                                        self.directory,
                                                        filter='*.txt')
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex
        else:
            fileName = extras.fileName

        if fileName:
            self.statusBar().showMessage(
                self.model.saveAllBlobs(fileName)
            )

    def saveInstrumentPositions(self, extras = None):
        '''
        save instrument-specific file for sample positions
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets
                          .QFileDialog.getSaveFileName(self,
                                 "Select save file",
                                 self.directory,
                                 filter='*' +
                                      self.model
                                      .currentInstrumentExtension())
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex
        else:
            fileName = extras.fileName

        if fileName:
            if extras is None or not hasattr(extras, 'fileName'):
                text,ok = QtWidgets
                          .QInputDialog.getText(self, "Input Required",
                          "Input max number of spots  or OK for all " +
                                str(self.model.currentBlobLength()) )

                if ok and not text == '':
                    maxnum = int(float(text))
                elif ok:
                    maxnum = self.model.currentBlobLength()
                else:
                    return

                maxnum = min(self.model.currentBlobLength(), maxnum)

                reply = QtWidgets
                         .QMessageBox
                         .question(self, 'Run optimization?',
                            '''Perform TSP optimization?
                               Not recommended for over {} targets
                               Currently have {}'''.format(
                            GUIConstants.TSP_LIMIT, maxnum),
                            buttons = QtWidgets.QMessageBox.No |
                                     QtWidgets.QMessageBox.Yes,
                            defaultButton = QtWidgets.QMessageBox.Yes
                                     if maxnum < GUIConstants.TSP_LIMIT
                                     else QtWidgets.QMessageBox.No)
                tsp = reply == QtWidgets.QMessageBox.Yes
            else:
                maxnum = extras.maxnum
                tsp = extras.tsp
            self.statusBar().showMessage(
                self.model.saveInstrumentPositions(
                    fileName,
                    tsp,
                    maxnum)
            )

            self.raise_()
            self.activateWindow()

    def saveFiducialPositions(self, extras = None):
        '''
        save instrument specific file
             for fiducial locations to check registration
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getSaveFileName(self,
                            "Select save file",
                            self.directory,
                            filter='*' +
                            self.model.currentInstrumentExtension())
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex
        else:
            fileName = extras.fileName

        if fileName:
            self.statusBar().showMessage(
                self.model.saveInstrumentRegistrationPositions(fileName)
            )

    def loadReg(self, extras = None):
        '''
        load a registration file
        sets the instrument and loads pixel and
             physical positions of fiducials
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getOpenFileName(
                self, 'Open File',
                self.directory,
                filter='*.msreg')[0]
        else:
            fileName = extras.fileName

        if fileName:
            message, index = self.model.loadCoordinateMapper(fileName)
            self.statusBar().showMessage(
                message
            )

            self.inst_menu.setEnabled(
                    self.model.coordinateMapper.isConnectedToInstrument)
            self.instruments.actions()[index].setChecked(True)
            self.slideCanvas.draw()

    def loadBlobFind(self, extras = None):
        '''
        load sample positions and blob finding parameters
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getOpenFileName(
                self, 'Open File',
                self.directory,
                filter='*.txt')[0]
        else:
            fileName = extras.fileName

        if fileName:
            self.statusBar().showMessage(
                self.model.loadBlobFinding(fileName)
            )
            self.slideCanvas.draw()

            if self.showHist == True:
                self.toggleHistWindow()

    def loadInstrumentPositions(self, extras = None):
        '''
        loads samples from an instrument file to display pixel positions
        '''
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getOpenFileName(
                self, 'Open File',
                self.directory,
                filter='*' + self.model.currentInstrumentExtension())[0]
        else:
            fileName = extras.fileName

        if fileName:
            self.statusBar().showMessage(
                self.model.loadInstrumentPositions(fileName)
                )
            self.slideCanvas.draw()

            if self.showHist == True:
                self.toggleHistWindow()

    def fileQuit(self):
        '''
        quit through the file -> quit button
        '''
        self.close()

    def closeEvent(self, ce):
        '''
        print the filename of the image that
was displayed prior to closing
        '''
        if self.model.coordinateMapper
                 .isConnectedToInstrument == True  and\
            self.model.coordinateMapper
                 .connectedInstrument.connected == True:
            self.model.coordinateMapper.connectedInstrument.homeAll()
        if self.fileName is not None:
            print("Exiting from file {}".format(self.fileName))

    def globalBlob(self, extras = None):
        '''
        blob find over the entire slide area or ROI
        '''
        self.statusBar().showMessage('Starting blob finding')
        self.statusBar().showMessage(
            self.model.runGlobalBlobFind()
        )
        self.saveAll(extras)
        self.slideCanvas.draw()

        if self.showHist == True:
            self.toggleHistWindow()

    def blbPopup(self):
        '''
        popup the blob finding parameters
        '''
        blbFind = self.model
                 .blobCollection[self.model.currentBlobs].blobFinder
        if blbFind is not None:
            self.popups['blobFind'].loadParams(blbFind)
            self.popups['blobFind'].show()
            self.popups['blobFind'].activateWindow()

    def toggleHistWindow(self):
        '''
        toggles the display of the histogram canvas
```

```
            and initializes the instance
        '''
        self.showHist = not self.showHist
        if self.showHist:
            #reset histogram to default values
            self.histCanvas.resetVariables(resetBlobs=True)
            self.histCanvas.show()
            self.histCanvas.calculateHist()
        else:
            self.histCanvas.hide()
            self.histCanvas.clearFilt()
            if self.popups['histOpts'].isVisible():
                self.popups['histOpts'].hide()

    def histOptions(self):
        '''
        pops up a window to adjust histogram canvas display
        also resets the sample positions (globalBlbs)
        '''
        if self.showHist:
            self.popups['histOpts'].loadParams(self.histCanvas)
            self.popups['histOpts'].show()
            self.popups['histOpts'].activateWindow()

    def histSelect(self, extras =  None):
        '''
        select the top and bottom X blobs from the histogram
        '''
        if extras is None or not hasattr(extras, 'text'):
            text,ok = QtWidgets
                          .QInputDialog.getText(self,
                                "Input Required",
                                "Input number of highest and \
                                lowest value blobs to find")
        else:
            text = extras.text
            ok = extras.ok
        if ok and not text == '':
            self.histCanvas.setBlobNum(int(text))

    def histSaveImage(self, extras = None):
        '''
        Saves the current figure image as a png
        extras: optional extra parameters to bypass GUI input
        '''
        if self.showHist == False:
            return

        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getSaveFileName(self,
                                "Select image file to save",
                                self.directory,
                                filter='*.png')
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex
        else:
            fileName = extras.fileName

        if fileName:
            self.statusBar().showMessage(
                self.histCanvas.saveHistImage(fileName)
            )

    def histSaveValues(self, extras = None):
        '''
        Saves all the blob locations and values of
             the current histogram metric
        extras: optional data to bypass GUI display
        '''
        if self.showHist == False:
            return
        if extras is None or not hasattr(extras, 'fileName'):
            fileName = QtWidgets.QFileDialog.getSaveFileName(self,
                                "Select save file",
                                self.directory,
                                filter='*.txt')
            f = os.path.splitext(fileName[0])[0]
            ex = os.path.splitext(fileName[1])[1]
            fileName = f+ex
        else:
            fileName = extras.fileName

        if fileName:
            self.statusBar().showMessage(
                self.histCanvas.savePopulationValues(fileName)
            )

    def histFilter(self):
        '''
        applies the filter to the histogram,
             updating the blob find positions to
             those matching the filter
        the filter is also recorded for writing the blob find file
        '''
        filt = self.histCanvas.getFilteredBlobs()
        if len(filt) == 0:
            self.statusBar().showMessage('Invalid histogram filter')
        else:
            for blbs in filt:
                self.model.updateCurrentBlobs(blbs)
            self.statusBar().showMessage('Applied {} filter'
                                        .format(len(filt)))
            self.histCanvas.calculateHist()
            self.slideCanvas.draw()

    def distanceFilter(self, extras = None):
        '''
        Performs distance filter of the
        sample positions and updates histogram display
        '''
        if extras is None or not hasattr(extras, 'text'):
            text,ok = QtWidgets.QInputDialog
                              .getText(self, "Input Required",
                                      "Input distance filter in pixels")
        else:
            text = extras.text
            ok = extras.ok

        if ok and not text == '':
            self.statusBar().showMessage('Starting distance filter')
            self.statusBar().showMessage(
                self.model.distanceFilter(int(text))
            )
            if self.showHist:
                self.histCanvas.calculateHist()
            self.slideCanvas.draw()
            self.raise_()
            self.activateWindow()

    def roiFilter(self):
        '''
        Performs filtering of blobs falling within the ROI
        '''
        self.statusBar().showMessage(
            self.model.roiFilter()
            )
        if self.showHist:
            self.histCanvas.calculateHist()
        self.slideCanvas.draw()

    def roiFilterInverse(self):
        '''
        Performs filtering of blobs falling within the ROI
        '''
        self.statusBar().showMessage(
            self.model.roiFilterInverse()
            )
        if self.showHist:
            self.histCanvas.calculateHist()
        self.slideCanvas.draw()

    def rectPack(self, extras = None):
        '''
        expand each spot into a rectangularly packed grid
        Get the separation and number of layers from the user
        '''
        if self.model.currentBlobLength() > 0:

            if extras is None or not hasattr(extras, 'sep'):
                text,ok = QtWidgets.QInputDialog
                                .getText(self,
                                      "Input Required",
                                      "Input separation in pixels" )
                if ok:
                    sep = int(text)
                else:
                    sep = 50

                text,ok = QtWidgets.QInputDialog
                                      .getText(self,
                                        "Input Required",
                                        "Input number of layers" )
                if ok:
                    layers = int(text)
                else:
                    layers = 1

                dynamicLayering = QtWidgets.QMessageBox
                                .question(self, 'Input Required',
                                       'Adjust layering to blob size?',
                                       QtWidgets.QMessageBox.Yes,
                                       QtWidgets.QMessageBox.No)

                if dynamicLayering == QtWidgets.QMessageBox.Yes:
                    dynamicLayering = True
                else:
                    dynamicLayering = False

            else:
                sep = extras.sep
                layers = extras.layers
                dynamicLayering = extras.dynamicLayering

            self.model.rectPackBlobs(sep, layers, dynamicLayering)
            self.slideCanvas.draw()
            if self.showHist:
                self.toggleHistWindow()

    def hexPack(self, extras = None):
        '''
        expand each spot into a hexagonally closed packed grid
        Get the separation and number of layers from the user
        '''
        if self.model.currentBlobLength() > 0:
            if extras is None or not hasattr(extras, 'sep'):
                text,ok = QtWidgets.QInputDialog
                                      .getText(self, "Input Required",
                                      "Input separation in pixels" )
                if ok:
                    sep = int(text)
                else:
                    sep = 50

                text,ok = QtWidgets.QInputDialog
                                      .getText(self, "Input Required",
                                      "Input number of layers" )
                if ok:
                    layers = int(text)
                else:
                    layers = 1

                dynamicLayering = QtWidgets.QMessageBox
                                      .question(self, 'Input Required',
                                       'Adjust layering to blob size?',
                                             QtWidgets.QMessageBox.Yes,
                                             QtWidgets.QMessageBox.No)

                if dynamicLayering == QtWidgets.QMessageBox.Yes:
                    dynamicLayering = True
                else:
                    dynamicLayering = False

            else:
                sep = extras.sep
                layers = extras.layers
                dynamicLayering = extras.dynamicLayering

            self.model.hexPackBlobs(sep, layers, dynamicLayering)
            self.slideCanvas.draw()
            if self.showHist:
                self.toggleHistWindow()

    def circPack(self, extras = None):
        '''
        expand each spot into circularly spaced positions
             around the spot circumference
        get separation, max number of spots and offset from user
        '''
        if self.model.currentBlobLength() > 0:
            if extras is None or not hasattr(extras, 'sep'):
                text,ok = QtWidgets.QInputDialog
                               .getText(self, "Input Required",
                               "Input minimum separation in pixels" )
                if ok:
                    sep = int(text)
                else:
                    sep = 50

                text,ok = QtWidgets.QInputDialog
                                     .getText(self, "Input Required",
                                           "Input max number of spots" )
                if ok:
                    shots = int(text)
                else:
                    shots = 10

                text,ok = QtWidgets.QInputDialog
                                     .getText(self, "Input Required",
                                           "Input offset in pixels" )
                if ok:
                    offset = int(text)
                else:
                    offset = 10

            else:
                sep = extras.sep
                shots = extras.shots
                offset = extras.offset

            self.model.circularPackBlobs(sep, shots, offset)
            self.slideCanvas.draw()
            if self.showHist:
                self.toggleHistWindow()

    def gridPopup(self):
        '''
        popup a window to edit the
        intermediate map of the mapper instance
        '''
        self.popups['grid'].loadParams(self.model)
        self.popups['grid'].show()
        self.popups['grid'].activateWindow()

    def initializeInstrument(self, extras = None):
        '''
        Initialize instrument on the user specified COM port
        '''
        if extras is None or not hasattr(extras, 'text'):
            text,ok = QtWidgets.QInputDialog
                            .getText(self, "Enter COM Port",
                                   "Connections at {}"
                                   .format(
                                        self.model.coordinateMapper
                                        .connectedInstrument.findPorts())
                                         )
        else:
            text = extras.text
            ok = extras.ok

        if ok:
            try:
                self.model.coordinateMapper
                         .connectedInstrument.initialize(text)
                self.statusBar().showMessage(
                              'Connected to {}'.format(text))
            except:
                self.statusBar().showMessage(
                              'Error connecting to {}'.format(text))

    def setDwell(self, extras = None):
        '''
        Set the dwell time for analysis with a connected instrument
        '''
        if extras is None or not hasattr(extras, 'text'):
            text,ok = QtWidgets.QInputDialog
                                        .getText(self, "Input Required",
                                                  "Set dwell time (s)"
                                                  )
        else:
            text = extras.text
            ok = extras.ok

        if ok:
            try:
                self.model.coordinateMapper
                         .connectedInstrument.dwellTime = float(text)
            except:
                self.statusBar().showMessage('Input error')

    def setWash(self, extras = None):
        '''
        Set the dwell time for analysis with a connected instrument
        '''
        if extras is None or not hasattr(extras, 'text'):
            text,ok = QtWidgets.QInputDialog
                          .getText(self, "Input Required",
                                 "Set wash time (s), -1 for continuous"
                                             )
        else:
            text = extras.text
            ok = extras.ok

        if ok:
            try:
                self.model.coordinateMapper
                              .connectedInstrument
                              .postAcqusitionWait = float(text)
            except:
                self.statusBar().showMessage('Input error')

    def analyzeAll(self):
        '''
        analyze all positions of the specified samples,
             acquire for time specified by dwell time
        '''
        self.statusBar().showMessage(
            self.model.analyzeAll()
            )

    def report_blbsubset(self, blbSubset):
        self.model.setBlobSubset(blbSubset)
        self.slideCanvas.draw()

    def createMessageBox(self, message, title):
        msg = QtWidgets.QMessageBox(self)
        msg.setWindowIcon(self.windowIcon())
        msg.setText(message)
        msg.setWindowTitle(title)
        msg.setStandardButtons(QtWidgets.QMessageBox.Ok)
        msg.setModal(False)
        return msg

    '''
    These are popup messages with the
       hotkeys defined in the included canvases
    '''
    def imgHotkeyMsg(self):
        self.popups['imageHelp'].show()
        self.popups['imageHelp'].activateWindow()

    def instHotkeyMsg(self):
        self.popups['instHelp'].show()
        self.popups['instHelp'].activateWindow()

    def histHotkeyMsg(self):
        self.popups['histHelp'].show()
        self.popups['histHelp'].activateWindow()

    def keyPressEvent(self, event):
        '''
        key press event handler
        '''
        if self.model.slide is not None:
            shift = event.modifiers() & QtCore.Qt.ShiftModifier
            #move with wsad
            if shift and event.modifiers() &
             QtCore.Qt.ControlModifier and\
                event.modifiers() & QtCore.Qt.AltModifier:
                stepSize = StepSize.giant
            elif shift and event.modifiers() &
              QtCore.Qt.ControlModifier:
                stepSize = StepSize.medium
            elif shift:
                stepSize = StepSize.large
            else:
                stepSize = StepSize.small
            if event.key() == QtCore.Qt.Key_A:
                self.model.reportSlideStep(Direction.left, stepSize)
            elif event.key() == QtCore.Qt.Key_D:
                self.model.reportSlideStep(Direction.right, stepSize)
            elif event.key() == QtCore.Qt.Key_W:
                self.model.reportSlideStep(Direction.up, stepSize)
            elif event.key() == QtCore.Qt.Key_S:
                self.model.reportSlideStep(Direction.down, stepSize)

            #zoom in and out
            elif event.key() == QtCore.Qt.Key_Q:
                self.model.slide.zoomOut()
            elif event.key() == QtCore.Qt.Key_E:
                self.model.slide.zoomIn()

            #reset view to top left corner
            elif event.key() == QtCore.Qt.Key_R:
                self.model.slide.resetView()

            #toggle display of target blob locations
            elif event.key() == QtCore.Qt.Key_O:
                if shift:
                    self.model.drawAllBlobs =
                         not self.model.drawAllBlobs
                else:
                    self.model.showPatches= not self.model.showPatches

            #cycle between image channels with t or z
            elif event.key() == QtCore.Qt.Key_T or
                         event.key() == QtCore.Qt.Key_Z:
                self.model.slide.switchType()

            #toggle display of predicted locations from mapper
            elif event.key() == QtCore.Qt.Key_P:
                self.model.showPrediction =
                         not self.model.showPrediction

            #toggle left/right mirror
            elif event.key() == QtCore.Qt.Key_M:
```

```
            self.model.mirrorImage = not self.model.mirrorImage

        elif event.key() == QtCore.Qt.Key_B:
            #toggle threshold view
            if shift:
                self.model.showThreshold =
                     not self.model.showThreshold
            #perform blob finding on max zoom image
            else:
                self.model.testBlobFind()

        elif event.key() == QtCore.Qt.Key_C:
            #clears all target positions
            if event.modifiers() & QtCore.Qt.ShiftModifier and \
                event.modifiers() & QtCore.Qt.ControlModifier:
                self.model.resetVariables()
                self.histCanvas.resetVariables(True, True)
                if self.showHist == True:
                    self.histCanvas.calculateHist()
            #clears current target positions
            elif event.modifiers() & QtCore.Qt.ShiftModifier:
                self.model
                           .blobCollection[self.model.currentBlobs]
                           .blobs = []
                self.histCanvas.resetVariables(True, True)
                if self.showHist == True:
                    self.histCanvas.calculateHist()
            #clears filters and ROI positions
            else:
                self.model
                     .blobCollection[self.model.currentBlobs]
                     .ROI = []
                self.histCanvas.clearFilt()

        keys = [QtCore.Qt.Key_1, QtCore.Qt.Key_2, QtCore.Qt.Key_3,
                QtCore.Qt.Key_4, QtCore.Qt.Key_5, QtCore.Qt.Key_6,
                QtCore.Qt.Key_7, QtCore.Qt.Key_8, QtCore.Qt.Key_9,
                QtCore.Qt.Key_0]

        #for each numeric key
        for i,k in enumerate(keys):
            if event.key() == k:
                #set global blobs to the multiblob specified
                if event.modifiers() & QtCore.Qt.AltModifier:
                    self.model.setCurrentBlobs(i)
                    self.statusBar()
                     .showMessage(
                           'Picking blobs into list #{},
                           contains {} blobs'
                           .format(i+1,
                                 self.model.currentBlobLength()))
                    if self.showHist:
                        self.histCanvas.calculateHist()
                #switch to image channel i
                elif event.modifiers() & QtCore.Qt.ControlModifier:
                    self.model.slide.switchToChannel(i)

                #toggle image channel on and off
                else:
                    self.model.slide.toggleChannel(i)
                break
        mapper = self.model.coordinateMapper
        if mapper.isConnectedToInstrument == True and \
            mapper.connectedInstrument.connected == True:

            #move instrument position with ikjl
            if event.key() == QtCore.Qt.Key_I:
                mapper.connectedInstrument.move(
                    Direction.up,
                    stepSize)
            elif event.key() == QtCore.Qt.Key_K:
                mapper.connectedInstrument.move(
                    Direction.down,
                    stepSize)
            elif event.key() == QtCore.Qt.Key_J:
                mapper.connectedInstrument.move(
                    Direction.left,
                    stepSize)
            elif event.key() == QtCore.Qt.Key_L:
                mapper.connectedInstrument.move(
                    Direction.right,
                    stepSize)

            elif event.key() == QtCore.Qt.Key_V:
                #set probe position
                if shift:
                    mapper.connectedInstrument.setProbePosition()

                #toggle probe position
                else:
                    mapper.connectedInstrument.toggleProbe()

            #perform single collection
            elif event.key() == QtCore.Qt.Key_X:
                mapper.connectedInstrument.collect()

            #move probe up and down
            elif event.key() == QtCore.Qt.Key_Equal:
                mapper.connectedInstrument.moveProbe(
                    Direction.up,
                    stepSize)
            elif event.key() == QtCore.Qt.Key_Minus:
                mapper.connectedInstrument.moveProbe(
                    Direction.down,
                    stepSize)
            elif event.key() == QtCore.Qt.Key_Plus:
                mapper.connectedInstrument.moveProbe(
                    Direction.up,
                    stepSize)
            elif event.key() == QtCore.Qt.Key_Underscore:
                mapper.connectedInstrument.moveProbe(
                    Direction.down,
                    stepSize)

            #home all positions
            elif event.key() == QtCore.Qt.Key_H:
                if event.modifiers() & QtCore.Qt.ShiftModifier:
                    mapper.connectedInstrument.finalPosition()
                else:
                    mapper.connectedInstrument.homeAll()

            elif event.key() == QtCore.Qt.Key_F and \
                event.modifiers() & QtCore.Qt.ControlModifier:
                x,y = mapper.connectedInstrument.getPositionXY()
                z = mapper.connectedInstrument.getProbePosition()
                self.statusBar()
                           .showMessage(
                                 'Stage at ({}, {}); probe at {}'
                                       .format(x,y,z))

        self.slideCanvas.draw()
    else:
        #debug autoload
        if event.key() == QtCore.Qt.Key_D and
               event.modifiers() & QtCore.Qt.ControlModifier:
            self.debugLoad()

def debugLoad(self):
    '''
    a debugging function that automatically
         sets up an example image and data set
    '''
    #check if debug data exists
    if os.path.isdir(GUIConstants.DEBUG_DIR):
        #image filename
        fileName = GUIConstants.DEBUG_IMG_FILE
        self.setupCanvas(fileName)
        #preset position and zoom level
        self.model.slide.pos = [30500, 30000]
        self.model.slide.lvl = 0

        #the blob finding file
        self.model.loadBlobFinding(GUIConstants.DEBUG_BLOB_FIND)
        #the registration file
        self.model.loadCoordinateMapper(GUIConstants.DEBUG_REG_FILE)
        self.slideCanvas.draw()

def reportFromModel(self, message = "",
                                 redrawSlide = False,
                                 redrawHist = False):
    '''
    Method for the model to interact with the GUI and windows.
    Displays the supplied message and redraws selected canvases
    message: String message to display
    redrawSlide: boolean to dictate if slideCanavas should redraw
    redrawHist: boolean to dictate if histCanavas should be redrawn
    '''
    self.statusBar().showMessage(message)
    if redrawSlide:
        self.slideCanvas.draw()
    if redrawHist and self.showHist:
        self.histCanvas.update_figure()

def requestFiducialInput(self, defaultStr):
    '''
    Method for microMSModel to receive input from the user
    defaultStr: the string to initially display to the user
    '''
    return QtWidgets.QInputDialog
                     .getText(self,'Coordinate Dialog',
                                       'Enter plate coordinate:',
                                       text=defaultStr)
```

GUICanvases/mplCanvas.py

```
from __future__ import unicode_literals

from PyQt5 import QtGui, QtWidgets
from matplotlib.figure import Figure
from matplotlib.backends.backend_qt5agg import
               FigureCanvasQTAgg as FigureCanvas

class MplCanvas(FigureCanvas):
    """Ultimately, this is a QWidget
         (as well as a FigureCanvasAgg, etc.)."""
    def __init__(self, parent=None, width=5, height=4, dpi=100):
        self.fig = Figure(figsize=(width, height),
                                 dpi=dpi,tight_layout=True)

        self.axes = self.fig.add_subplot(111)
        # We want the axes cleared every time plot() is called
        self.axes.hold(False)

        self.compute_initial_figure()

        FigureCanvas.__init__(self, self.fig)
        self.setParent(parent)

        FigureCanvas.setSizePolicy(self,
                                   QtWidgets.QSizePolicy.Expanding,
                                   QtWidgets.QSizePolicy.Expanding)
        FigureCanvas.updateGeometry(self)

    def compute_initial_figure(self):
        pass
```

GUICanvases/popup.py

```

"""
a collection of small, custom popup windows used by microMSQT
"""

from PyQt5 import QtWidgets

class blbPopupWindow(QtWidgets.QDialog):
    '''
    Window for setting blob finding parameters
    '''
    def __init__(self, parent=None):
        '''
        setup GUI and populate with current values
        blobFinder: the blob finding object
        parent: the parent, calling widget, a MicroMSQTWindow
        '''
        super(blbPopupWindow,self).__init__(parent)

        self.master = parent

        self.setWindowTitle("Blob Find Entry")

        #user input widgets
        self.minText = QtWidgets.QLineEdit(self)
        self.maxText = QtWidgets.QLineEdit(self)
        self.minCirText = QtWidgets.QLineEdit(self)
        self.maxCirText = QtWidgets.QLineEdit(self)
        self.intens = QtWidgets.QLineEdit(self)
        self.imgInd = QtWidgets.QLineEdit(self)
        self.channel = QtWidgets.QComboBox(self)
        self.channel.addItem("Red")
        self.channel.addItem("Green")
        self.channel.addItem("Blue")

        #add to vbox layout with labels
        vbox = QtWidgets.QVBoxLayout()
        vbox.addWidget(QtWidgets.QLabel("Minimum Size",self))
        vbox.addWidget(self.minText)
        vbox.addWidget(QtWidgets.QLabel("Maximum Size",self))
        vbox.addWidget(self.maxText)
        vbox.addWidget(QtWidgets.QLabel("Minimum Circularity",self))
        vbox.addWidget(self.minCirText)
        vbox.addWidget(QtWidgets.QLabel("Maximum Circularity",self))
        vbox.addWidget(self.maxCirText)
        vbox.addWidget(QtWidgets.QLabel("Threshold",self))
        vbox.addWidget(self.intens)
        vbox.addWidget(QtWidgets.QLabel("Image Channel",self))
        vbox.addWidget(self.imgInd)
        vbox.addWidget(QtWidgets.QLabel("Color",self))
        vbox.addWidget(self.channel)
        self.setButton = QtWidgets.QPushButton("Set Parameters",self)
        self.setButton.clicked.connect(self.setParams)
        vbox.addWidget(self.setButton)

        self.setLayout(vbox)

    def loadParams(self, blbFinder):
        self.blobFinder = blbFinder
        self.minText.setText( str(blbFinder.minSize))
        self.maxText.setText('' if blbFinder.maxSize is None
                                                    else str(blbFinder.maxSize))
        self.minCirText.setText( str(blbFinder.minCircularity))
        self.maxCirText.setText('' if blbFinder.maxCircularity is None
                                        else str(blbFinder.maxCircularity))
        self.intens.setText(str(blbFinder.threshold))
        self.imgInd.setText(str(blbFinder.imageIndex+1))
        self.channel.setCurrentIndex(blbFinder.colorChannel)


    def setParams(self):
        '''
        sets the parameters for blob finding
              based on the current GUI values
        Calls on the slideCanvas to perform
              blob finding on the current image
        '''
        try:
            self.blobFinder.minSize = int(self.minText.text())
        except:
            self.minText.setText(str(self.blobFinder.minSize))

        try:
            self.blobFinder.maxSize = None if self.maxText.text() is ''
                                                    else int(self.maxText.text())
        except:
            self.maxText.setText('' if self.blobFinder.maxSize is None
                                              else str(self.blobFinder.maxSize))

        try:
            self.blobFinder.minCircularity =
                    float(self.minCirText.text())
        except:
            self.minCirText.setText( str(self.blobFinder.minCircularity))

        try:
            self.blobFinder.maxCircularity = None
                                              if self.maxCirText.text() is ''
                                              else float(self.maxCirText.text())
        except:
            self.maxCirText.setText('' if self.blobFinder.maxCircularity
                                                    is None
                                        else str(self.blobFinder.maxCircularity))

        try:
            self.blobFinder.threshold = int(self.intens.text())
        except:
            self.intens.setText(str(self.blobFinder.threshold))

        try:
            self.blobFinder.imageIndex = int(self.imgInd.text())-1
        except:
            self.imgInd.setText(str(self.blobFinder.imageIndex+1))

        self.blobFinder.colorChannel = int(self.channel.currentIndex())
        #blob find
        if self.master is not None:
            self.master.model.testBlobFind()
            self.master.slideCanvas.draw()

class gridPopupWindow(QtWidgets.QDialog):
    '''
    displays a table with the current intermediate
        map of the mapper for the user to edit
    '''
    def __init__(self, parent = None):
        '''
        populate the GUI with previous points
        previousPoints: list of triples of the set
              coordinate and its x and y physical position
        parent: the microMSQT window calling the popup
        '''
        super(gridPopupWindow,self).__init__(parent)

        self.setWindowTitle("Stage Locations")
        vbox = QtWidgets.QVBoxLayout()
        self.table = QtWidgets.QTableWidget(self)
        vbox.addWidget(self.table)
        self.setLayout(vbox)

    def loadParams(self, model):

        self.model = model
        previousPoints = model.coordinateMapper.getIntermediateMap()

        self.table.setRowCount(len(previousPoints))
        self.table.setColumnCount(3)
        self.table.setHorizontalHeaderLabels(["Coord","X","Y"])
        self.table.update()
        for i,m in enumerate(previousPoints):
            for j,el in enumerate(m):
                self.table.setItem(i,j,
                          QtWidgets.QTableWidgetItem(str(el)))

    def closeEvent(self,evnt):
        '''
        parse the information in the table and return
              it to the current mapper
        '''
        result = []
        for i in range(self.table.rowCount()):
            coord = self.table.item(i,0).text()
            x = self.table.item(i,1).text()
            y = self.table.item(i,2).text()
            result.append((coord, x, y))

        self.model.coordinateMapper.setIntermediateMap(result)
        #close
        self.hide()
```

```python

class histPopupWindow(QtWidgets.QDialog):
    '''
    a popup window to adjust histogram options
        such as display image and metric
    '''
    def __init__(self, histCanvas, parent=None):
        '''
        setup GUI and initialize it with the current settings
        histCanvas: a histCanvas widget contained within parent
        parent: a microMSQT window
        '''
        super(histPopupWindow,self).__init__(parent)

        self.hist = histCanvas
        self.master = parent

        self.setWindowTitle("Histogram Options")

        #generate user io widgets
        self.imgInd = QtWidgets.QLineEdit(self)
        self.channel = QtWidgets.QComboBox(self)
        for m in self.hist.metrics:
            self.channel.addItem(m)

        self.offset = QtWidgets.QLineEdit(self)
        self.max = QtWidgets.QRadioButton(self)
        self.max.setText('Max Intensity')
        self.mean = QtWidgets.QRadioButton(self)
        self.mean.setText('Average Intensity')

        #add to vbox layout with labels
        vbox = QtWidgets.QVBoxLayout()
        vbox.addWidget(QtWidgets.QLabel("Image Channel",self))
        vbox.addWidget(self.imgInd)
        vbox.addWidget(QtWidgets.QLabel("Color or Morphology",self))
        vbox.addWidget(self.channel)
        vbox.addWidget(QtWidgets.QLabel("Offset (pixels)",self))
        vbox.addWidget(self.offset)
        vbox.addWidget(self.max)
        vbox.addWidget(self.mean)

        btn = QtWidgets.QPushButton("Set Parameters",self)
        btn.clicked.connect(self.setParams)
        vbox.addWidget(btn)

        self.setLayout(vbox)


    def loadParams(self, histCanvas):

        self.hist = histCanvas

        #generate user io widgets
        self.imgInd.setText(str(self.hist.imgInd+1))

        self.channel.setCurrentIndex(self.hist.populationMetric)
        self.offset.setText(str(self.hist.offset))
        self.mean.setChecked(not self.hist.reduceMax)
        self.max.setChecked(self.hist.reduceMax)

    def setParams(self):
        '''
        trigger to set the new histogram
              parameters and redraw the histogram
        '''
        try:
            self.hist.imgInd = int(self.imgInd.text())-1
        except:
            self.imgInd.setText(str(self.hist.imgInd+1))

        try:
            self.hist.offset = int(self.offset.text())
        except:
            self.offset.setText(str(self.hist.offset))

        self.hist.populationMetric = int(self.channel.currentIndex())

        if self.master is not None and
          self.master.model.slide is not None:
            self.master.model.slide.switchToChannel(self.hist.imgInd)

        self.hist.reduceMax = self.max.isChecked()
        self.hist.calculateHist()
```

GUICanvases/slideCanvas.py

```python
from __future__ import unicode_literals

from PyQt5 import QtCore, QtWidgets, QtGui
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QCursor

import numpy as np
import random
import os

import matplotlib.pyplot as plt
from matplotlib.collections import PatchCollection
from PIL import ImageDraw, ImageFont, Image

from GUICanvases.mplCanvas import MplCanvas
from GUICanvases import GUIConstants
from ImageUtilities import blobFinder
from ImageUtilities import TSPutil
from ImageUtilities import blob

from CoordinateMappers import supportedCoordSystems
from CoordinateMappers import connectedInstrument

class SlideCanvas(MplCanvas):
    '''
    A QWidget for displaying and interfacing slide images
    This also has quite a bit of control code
    '''
    def __init__(self, master, model, *args, **kwargs):
        '''
        initialize a new instance of a slide canvas
        sets up several instance variables and default display settings
        model: the microMSModel shared with the window GUI
        '''
        MplCanvas.__init__(self, *args, **kwargs)

        #modify display defaults
        self.axes.xaxis.set_visible(False)
        self.axes.yaxis.set_visible(False)
        self.axes.set_axis_bgcolor(GUIConstants.IMAGE_BACKGROUND)
        self.setCursor(QCursor(Qt.CrossCursor))

        #temporary image for drawing rectangles, circles, etc quickly
        self.tempIm = None

        #variables related to mouse actions
        self.mDown = False          #mouse pressed for drawing a ROI
        self.startP = None          #starting position of a mouse drag
        self.endP = None            #end position of a mouse drag
        self.mDownCirc = False      #mouse down for drawing a global blob
        self.mMoveCirc = False      #mouse moved drawing a global blob
        self.mMoveROI = False       #mouse moved ROI with control alt

        self.model = model
        self.master = master

        #connect mouse events
        self.mpl_connect('button_release_event', self.mouseUp)
        self.mpl_connect('button_press_event', self.mouseDown)
        self.mpl_connect('motion_notify_event', self.mouseMove)
        self.mpl_connect('scroll_event', self.mouseZoom)

    def compute_initial_figure(self):
        '''
        Draw the initial image shown before anything is loaded.
        Shows a high res version of the icon image
        '''
        tdir,f = os.path.split(__file__)
        icon = Image.open(os.path.join(tdir, 'Icon', 'icon.png'))
        self.axes.imshow(icon)

    def draw(self):
        '''
        redraw canvas with markups using current settings
        '''
        if self.mMoveROI == True:
            return#redrawROI handles redraws here
        if self.model.slide is not None:
            #reset size as needed
            self.model.reportSize((float(self.size().width()),
                                   float(self.size().height())))

            #get base image from slideWrapper and show
            self.tempIm = self.model.getCurrentImage()
            self.axes.imshow(self.tempIm)

            #add on the blobs, predicted coordinates, and fiducial set
            self.axes.add_collection(self.model
                                      .getPatches(
                                      self.master.limitDraw.isChecked()))
            #the text labels can't be patches,
              #have to pass in the axes object to draw
            self.model.drawLabels(self.axes)

        #mirror left/right as needed
        if self.model.mirrorImage:
            self.axes.invert_xaxis()
        super().draw()

    def mouseUp(self,event, extras = None):
        '''
        handles mouse events when the user releases
        event: an mpl mouse event
        '''
        if event.xdata is None or event.ydata is None:
            return

        if self.model.slide is None:
            return

        if extras is not None and hasattr(extras, 'modifiers'):
            modifiers = extras.modifiers
        else:
            modifiers = QtWidgets.QApplication.keyboardModifiers()

        #left mouse button click without dragging
        #generally handles image movement
        #and interaction with target locations
        if(event.button == 1 and not self.mDown):
            #remove or add global blob with shift click
            if modifiers == QtCore.Qt.ShiftModifier:

                #check if global blbs exists,
                    #if any points are within click
                globalPnt = self.model.slide
                          .getGlobalPoint((event.xdata, event.ydata))

                #if shift click and drag, add blob with specified radius
                if self.mDownCirc and self.mMoveCirc:
                    rad = np.sqrt((globalPnt[0]-self.startPC[0])**2
                                + (globalPnt[1]-self.startPC[1])**2)
                    #minimum size of default radius pixels
                    rad = GUIConstants.DEFAULT_RADIUS if \
                        rad < GUIConstants.DEFAULT_RADIUS else rad

                #just a shift click
                else:
                    rad = GUIConstants.DEFAULT_RADIUS

                #reset manual drawing flags
                self.mDownCirc = False
                self.mMoveCirc = False

                self.master.reportFromModel(
                    self.model.reportBlobRequest(
                                  (event.xdata, event.ydata),
                                  radius = rad)
                   )

            #control + alt + LMB to add ROI point
            elif modifiers & QtCore.Qt.AltModifier and \
                modifiers & QtCore.Qt.ControlModifier:
                self.model.reportROI(self.model.slide.getGlobalPoint(
                                   (event.xdata, event.ydata)))

            #control + shift + LMB to append ROI point
            elif modifiers & QtCore.Qt.ShiftModifier and \
                modifiers & QtCore.Qt.ControlModifier:
                self.model.reportROI(self.model.slide.getGlobalPoint(
                                   (event.xdata, event.ydata)),
                                    append = True)

            #alt + LMB to move connected instrument to specified position
            elif modifiers == QtCore.Qt.AltModifier:
                self.master.reportFromModel(
                    self.model.requestInstrumentMove((event.xdata,
                                                      event.ydata))
                    )

            #plain LMB moves the image center to the mouse position
            else:
                self.model.slide.moveCenter((event.xdata, event.ydata))

        #right button to interact with fiducial registration
        elif(event.button == 3):
            self.master.reportFromModel(
                self.model.reportFiducialRequest(
                    (event.xdata, event.ydata),
                     removePoint = modifiers == QtCore.Qt.ShiftModifier,
                        extras = extras)
                )

        #middle mouse button to get information on mouse position
        elif(event.button == 2):
            self.master.reportFromModel(
                self.model.reportInfoRequest((event.xdata, event.ydata)),
                redrawHist = self.master.showHist
                )

        #mouse was dragged to draw an ROI
        if(self.mDown):
            #convert two point to a 4point rectangle
            p1 = self.model.slide.getGlobalPoint((event.xdata,
                                                   event.ydata))
            p2 = self.model.slide.getGlobalPoint(self.ROI)
            xlow, ylow = min(p1[0], p2[0]), min(p1[1], p2[1])
            xhigh, yhigh = max(p1[0], p2[0]), max(p1[1], p2[1])
            self.model.blobCollection[self.model.currentBlobs]
                              .ROI = [  (xlow, ylow),
                                            (xlow, yhigh),
                                            (xhigh, yhigh),
                                            (xhigh, ylow)]

            self.mDown = False
        self.draw()

    def mouseDown(self, event, extras = None):
        '''
        mouseDown sets variables for drawing ROIs
              or target positions with variable radii
        event: an mpl mouse down event
        '''
        if event.xdata is None or event.ydata is None:
            return

        if self.model.slide is None:
            return

        if extras is not None and hasattr(extras, 'modifiers'):
            modifiers = extras.modifiers
        else:
            modifiers = QtWidgets.QApplication.keyboardModifiers()

        #ROI drawing
        if event.button == 1 and \
            modifiers == QtCore.Qt.ControlModifier:
```

```python
                self.mDown = True
                self.ROI = (event.xdata, event.ydata)

            #target drawing
            elif event.button == 1 and \
                modifiers == QtCore.Qt.ShiftModifier:
                self.mDownCirc = True
                self.startPC = self.model.slide
                              .getGlobalPoint((event.xdata, event.ydata))

        def mouseMove(self,event, extras = None):
            '''
            mouse moves redraw ROI or blob positions as appropriate
            event: an mpl mouse move event
            '''
            if event.xdata is None or event.ydata is None:
                return

            if self.model.slide is None:
                return

            #ROI movement

            if extras is not None and hasattr(extras, 'modifiers'):
                modifiers = extras.modifiers
            else:
                modifiers = QtWidgets.QApplication.keyboardModifiers()

            if self.mDown == True:
                self.redrawRect((event.xdata, event.ydata))

            elif modifiers & QtCore.Qt.AltModifier and \
                    modifiers & QtCore.Qt.ControlModifier:
                self.redrawROI((event.xdata, event.ydata))
                self.mMoveROI = True

            elif modifiers & QtCore.Qt.ShiftModifier and \
                    modifiers & QtCore.Qt.ControlModifier:
                self.redrawROI((event.xdata, event.ydata), append = True)
                self.mMoveROI = True

            elif self.mMoveROI == True:
                self.mMoveROI = False
                self.draw()


            #target drawing
            elif self.mDownCirc == True:
                self.redrawCirc((event.xdata, event.ydata))
                self.mMoveCirc = True

        def mouseZoom(self,event):
            '''
            handle scroll wheel movement, which zooms the slide in and out
            event: an mpl mouse wheel event
            '''
            if event.xdata is None or event.ydata is None:
                    return

            if self.model.slide is None:
                return

            #zoom in or out
            if event.button == 'up':
                self.model.slide.zoomIn()
                self.model.slide.moveCenter((event.xdata, event.ydata))
            else:
                self.model.slide.zoomOut()

            #reset temporary blobs and update
            self.draw()

        def redrawROI(self, pnt, append = False):
            '''
            helper method to draw ROI polygon during mouse movement
            pnt: the current point in local (image) coordinates
            '''
            if self.tempIm is not None:
                self.axes.imshow(self.tempIm)
                roi = self.model.getROIPatches(
                        self.model.slide.getGlobalPoint(pnt),
                                                append)
                self.axes.add_collection(PatchCollection(roi,
                                    match_original=(len(roi) != 0)))
                if self.model.mirrorImage:
                    self.axes.invert_xaxis()
                super().draw()

        def redrawRect(self, pnt):
            '''
            helper method to draw the yellow ROI
                  rectangle during mouse movement
            pnt: the current point in local (image) coordinates
            '''
            if self.tempIm is not None:
                tempStartP = self.ROI
                self.axes.imshow(self.tempIm)
                lowerL = ((min(tempStartP[0], pnt[0]),
                                  min(tempStartP[1], pnt[1])))
                x = abs(tempStartP[0]- pnt[0])
                y = abs(tempStartP[1]- pnt[1])
                r = plt.Rectangle(lowerL, x, y,
                                                color=GUIConstants.ROI,
                                                fill=False)
                self.axes.add_patch(r)
                if self.model.mirrorImage:
                    self.axes.invert_xaxis()
                super().draw()

        def redrawCirc(self, pnt):
            '''
            helper method to draw the green circle for manually added blobs
            pnt: the current mouse position in local (image) coordinates
            '''
            if self.tempIm is not None:
                tempStartP = self.model.slide.getLocalPoint(self.startPC)
                self.axes.imshow(self.tempIm)
                rad = np.sqrt((tempStartP[0]-pnt[0])**2 +
                                          (tempStartP[1]-pnt[1])**2)

                c = plt.Circle(pnt, rad,
                     color=GUIConstants.MULTI_BLOB[self.model.currentBlobs],
                           linewidth=1,
                           fill=False)
                self.axes.add_patch(c)
                if self.model.mirrorImage:
                    self.axes.invert_xaxis()
                super().draw()

        def savePlt(self, fileName):
            '''
            saves the current figure
            fileName: the file to write to
            '''
            self.fig.savefig(fileName)
```

ImageUtilities/__init__.py

```
'''
The ImageUtilities package contains the classes required
   to display and analyze microscope images
blob.py:            object model of the blob objects found with
                           blobFinder and some helpful methods
blobList.py:        a collection of blobs
blobFinder.py:      performs blob finding with a simple
                           threshold and group algorithm
enumModule.py:       a collection of enums for movement
slideWrapper.py:    wraps and extends the openslide functions
                           to handle ndpi and tif images
TSPutil.py:         implements traveling salesperson optimization
                           of a collection of tuples
'''
```

ImageUtilities/blob.py

```
from GUICanvases import GUIConstants
import matplotlib as mpl

class blob(object):
    """
    Representation of a target point
    """
    def __init__(self, x = float(0), y = float(0),
                 radius = float(GUIConstants.DEFAULT_BLOB_RADIUS),
                 circularity = float(1), group = None):
        '''
        Initialize a new blob with the specified position,
                     shape and group
        x: x coordinate, default 0.0
        y: y coordinate, default 0.0
        radius: effective radius of the blob,
                     default to value specified in GUIConstants
        circularity: 0 < circ < 1, default value is 1 (perfect circle)
        '''
        self.X = x
        self.Y = y
        self.radius = float(radius)
        #keep circularity in bounds
        self.circularity = 1 if circularity > 1 else \
            (0 if circularity < 0 else circularity)
        self.group = group

    @staticmethod
    def getXYList(blobs):
        '''
        Method to convert a list of blobs to their x,y coordinates
        blobs: list of blobs
        returns a list of (x,y) tuples of each blob in order
        '''
        if blobs is None:
            return None
        return list(map(lambda b: (b.X, b.Y), blobs))

    @staticmethod
    def blobFromSplitString(instrings):
        '''
        Tries to parse all information from
               a split string to make a new blob
        instrings: list of strings, produced from
               splitting a blob.toString()
        returns a new blob with the indicated x,y,r and circularity
        '''
        result = blob()

        if instrings is None:
            return result

        if (len(instrings) == 3 or len(instrings) == 4):
            result.X = float(instrings[0])
            result.Y = float(instrings[1])
            result.radius = float(instrings[2])
        if len(instrings) == 4:
            result.circularity = float(instrings[3])

        return result

    def toString(self):
        '''
        Generates a tab delimited string with the x, y,
               radius and circularity of the blob
        '''
        return "{0:.3f}\t{1:.3f}\t{2:.3f}\t{3:.3f}"
                     .format(self.X, self.Y,
                                  self.radius, self.circularity)
```

ImageUtilities/blobFinder.py

```python
import skimage
from skimage import measure
import numpy as np
from itertools import product
import time
import scipy

from ImageUtilities.blob import blob

import matplotlib
import matplotlib.pyplot as plt

class blobFinder(object):
    '''
    performs blob finding on a slidewrapper object
    '''
    def __init__(self, slide, minSize = 50, maxSize = None,
                 minCircularity = 0.6, maxCircularity = None,
                 colorChannel = 2, threshold = 75, imageIndex = 1):
        '''
        set up the slidewrapper
        slide: slidewrapper to interact with
        minSize: minimum blob size in pixels
        maxSize: maximum blob size in pixels, None for no maximum
        minCircularity: minimum blob circularity
        maxCircularity: maximum blob circularity, None for no max
        colorChannel: [0, 1, 2] -> [R, G, B] channel to select
        threshold: maximum pixel intensity to consider a blob
        imageIndex: index of multi-slide object to consider
        '''
        self.slide = slide
        self.minSize = minSize
        self.maxSize = maxCircularity
        self.minCircularity = minCircularity
        self.maxCircularity = maxCircularity
        self.colorChannel = colorChannel
        self.threshold = threshold
        self.imageIndex = imageIndex

    def copyParameters(self, other):
        '''
        Copies blob finding parameters from another blobFinder instance
        other: blobFinder object to copy parameters from
        '''
        self.minSize = other.minSize
        self.maxSize = other.maxSize
        self.minCircularity = other.minCircularity
        self.maxCircularity = other.maxCircularity
        self.colorChannel = other.colorChannel
        self.threshold = other.threshold
        self.imageIndex = other.imageIndex


    def getParameters(self):
        '''
        get the set of parameters as a dictionary
        returns a dictionary of string -> value
              pairs of all parameters for blob finding
        '''
        return {
                'minSize' : self.minSize,
                'maxSize' : self.maxSize,
                'minCir' : self.minCircularity,
                'maxCir' : self.maxCircularity,
                'channel' : self.colorChannel,
                'thresh' : self.threshold,
                'ImageInd' : self.imageIndex}

    def setParameterFromSplitString(self, toks):
        '''
        Sets the parameters dictated in the toks list.
              String must match from getParameters
        toks: list of strings generated from string.split
        '''
        if toks is None or len(toks) < 2:
            return

        if toks[0] == 'minSize':
            if toks[1] == 'None\n':
                raise(ValueError('None type not acceptable for minSize'))
            self.minSize = int(toks[1])

        elif toks[0] == 'maxSize':
            if toks[1] == 'None\n':
                self.maxSize = None
            else:
                self.maxSize = int(toks[1])

        elif toks[0] == 'minCir':
            if toks[1] == 'None\n':
                raise(ValueError('None type not acceptable for minCirc'))
            self.minCircularity = float(toks[1])

        elif toks[0] == 'maxCir':
            if toks[1] == 'None\n':
                self.maxCircularity = None
            else:
                self.maxCircularity = float(toks[1])

        elif toks[0] == 'channel':
            if toks[1] == 'None\n':
                raise(ValueError('None type not acceptable for channel'))
            self.colorChannel = int(toks[1])

        elif toks[0] == 'thresh':
            if toks[1] == 'None\n':
                raise(ValueError('None type not acceptable for thresh'))
            self.threshold = int(toks[1])

        elif toks[0] == 'ImageInd':
            if toks[1] == 'None\n':
                raise(ValueError('None type not accepted for ImageInd'))
                self.imageIndex = int(toks[1])


    def getBlobCharacteristics(self, pnt):
        '''
        Gets the area and circularity for a
              blob containing the supplied point
        Returns 0,0 if no blob containing point
        pnt: (x,y) of the requested point
        '''
        #get current image
        img = self.slide.getImg()
        #threshold image
        lbl, num = blobFinder._blbThresh(img,
                                  self.colorChannel, self.threshold)
        slices = scipy.ndimage.find_objects(lbl)
        area, circ = 0,0
        #for each blob in region
        for i in range(num):
            s = slices[i]
            dx, dy = s[:2]
            #check if point is within the bounds of the blob
            if dx.start < pnt[1] and dx.stop > pnt[1] and \
                dy.start < pnt[0] and dy.stop > pnt[0]:
                #convert blob to boolean image
                region = lbl[dx.start-1:dx.stop+1, dy.start-1:dy.stop+1]
                region = region == i+1
                #get area and circularity
                area = np.sum(region)
                perim = skimage.measure.perimeter(region)
                if perim == 0:
                    circ = 1
                else:
                    circ = min(4*np.pi * area / perim**2, 1)
                #scale area by zoom level
                #these get  less accurate with higher zoom level
                area = area * 2**(2*self.slide.lvl)
                break
        return area, circ

    @staticmethod
    def _blbHelp(img, sizes, channel = 2, threshold = 200,
                 circs = (0.7,None), xShift=0, yShift = 0):
        '''
        helper function to perform blob finding on the image
        returns a list of blobs
        img: the image to blob find
        sizes: (min, max) size to consider max == None means no max size
        channel: r,g,b channel to threshold
        threshold: minimum pixel intensity to count as blob
        circs: (min, max) circularity to consider
                    max == None means no max
        xShift: amount to add to x coordinate
              to shift into global coordinate
        yShift: amount to add to y coordinate
              to shift into global coordinate
        '''
        #blob find
        lbl, num = blobFinder._blbThresh(img, channel, threshold)
        slices = scipy.ndimage.find_objects(lbl)
        result = []
        #for each blob
        for i in range(num):
            #convert to boolean image
            s = slices[i]
            dx, dy = s[:2]
            region = lbl[dx.start-1:dx.stop+1, dy.start-1:dy.stop+1]
            region = region == i+1
            #area is total number of true pixels
            area = np.sum(region)
            #if passes size threshold
            if area > sizes[0] and (sizes[1] is None or area < sizes[1]):
                #calculate circularity = 4 pi area / perimeter^2
                perim = skimage.measure.perimeter(region)
                circ = 4*np.pi * area / perim**2
                #if passes circularity threshold
                if circ > circs[0] and
                      (circs[1] is None or circ < circs[1]):
                    #determine center of mass, ignoring intensity
                    (x,y) = scipy.ndimage
                                .measurements.center_of_mass(region)
                    #calculate radius assuming circle
                    r = np.sqrt(area/np.pi)
                    #add to result, note x,y transpose!
                    result.append(blob(y=x+dx.start-1+yShift,
                                       x = y+dy.start-1+xShift,
                                       radius = r,
                                       circularity = circ))
        return result

    @staticmethod
    def _blbThresh(img, channel = 2, threshold = 200):
        '''
        helper function to threshold and group image
        returns the label and total number of objects from ndimage.label
        img: image to consider
        channel: r,g,b channel to threshold
        threshold: min intensity cutoff
        '''
        img = np.array(img.split()[channel])
        thresh = img > threshold
        return scipy.ndimage.label(thresh)

    def blobImg(self):
        '''
        perform blob finding on the current
              position of slideWrapper at max zoom
        returns a list of blobs in image
        '''
        inputImg = self.slide.getMaxZoomImage(imgInd = self.imageIndex)
        return blobFinder._blbHelp(inputImg,
                                (self.minSize, self.maxSize),
                                      self.colorChannel,
                                      self.threshold,
                                    (self.minCircularity,
                                     self.maxCircularity))

    def blobSlide(self, subSize = 8192, ROI = None):
        '''
        perform blob finding on the entire image bounded by ROI
        only reads a subregion of the image at once,
              which causes an initial grouping of blobs
        returns a list of blobs in image
        subSize: size in pixels of one side of the
              subregion to iterate over
            larger values may use up lots of RAM
        ROI: a list of points for ROI polygon.
              Only used to determine bounding box.
        '''
        #the amount of overlap between regions,
        #would matter with larger objects but is currently ignored
        overlap = 0

        #if ROI is none, get max size and (0,0)
        if ROI is None or len(ROI) < 2:
            botR = self.slide.getSize()
            topL = (0,0)
            ROI = [topL, botR]
        else:
            topL = (min(map(lambda x: x[0], ROI)),
                    min(map(lambda x: x[1], ROI)))
            botR = (max(map(lambda x: x[0], ROI)),
                    max(map(lambda x: x[1], ROI)))

        #set of x and y values of the center of each sub image
        xs = np.arange(topL[0] + subSize//2,
                           botR[0]+subSize//2, subSize-overlap)
        ys = np.arange(topL[1] + subSize//2,
                           botR[1]+subSize//2, subSize-overlap)

        #Cartesian product of xs and ys
        centers = product(xs,ys)

        #initialize time, blob list, and iterator count
        start = time.time()
        total = len(xs) * len(ys)
        print("starting %d images" % total)
        blbs = []
        i = 1

        #for each subregion
        for cent in centers:
            #get max zoom image
            inputImg = self.slide
                           .getMaxZoomImage((int(cent[0]),int(cent[1])),
                                       (subSize,subSize),
                                       imgInd = self.imageIndex)
            #blob find
            blb = blobFinder._blbHelp(inputImg,
                                    (self.minSize, self.maxSize),
                                    self.colorChannel, self.threshold,
                                    (self.minCircularity,
                                      self.maxCircularity),
                                    cent[0]-subSize/2, cent[1]-subSize/2)
            blbs.extend(blb)
            #print out expected time remaining, not super accurate
            if i % 10 == 0 or i == 1:
                print("finished %d of %d subareas, %d seconds left"
                    % (i, total, (time.time()-start)/ i * (total-i)))
            i = i+1

        print("took {:.3f} minutes".format((time.time() - start)/60))

        return blbs
```

ImageUtilities/blobList.py

```
import numpy as np
import scipy
from scipy.spatial.distance import pdist
import matplotlib.pyplot as plt
from matplotlib.path import Path
from copy import deepcopy
import ast

from GUICanvases import GUIConstants

from ImageUtilities import blob
from ImageUtilities import blobFinder

class blobList(object):
    """
    A collection of blob objects.
    Underlying data is the list self.blobs and supplies several
    utilities for filtering, drawing and expanding blobs.
    Each bloblist also contains its own blobfinder and filters
    """

    def __init__(self, slide = None):
        self.blobs = []
        self.blobFinder = blobFinder.blobFinder(slide)
        self.filters = []
        self.description = None
        self.threshCutoff = None
        self.ROI = []
        self.groupLabels = dict()
        ##Add any new instance vars to deepcopy!

    def append(self, blb):
        if isinstance(blb, blob.blob):
            self.blobs.append(blb)

    def length(self):
        return len(self.blobs)

    def __copy__(self):
        cls = self.__class__
        result = cls.__new__(cls)
        result.__dict__.update(self.__dict__)
        return result

    def __deepcopy__(self, memo):
        cls = self.__class__
        result = cls.__new__(cls)
        memo[id(self)] = result

        result.blobs = deepcopy(self.blobs)
        result.filters = deepcopy(self.filters)
        result.description = deepcopy(self.description)
        result.ROI = deepcopy(self.ROI)
        result.threshCutoff = self.threshCutoff
        result.groupLabels = deepcopy(self.groupLabels)
        result.blobFinder = blobFinder.blobFinder(self.blobFinder.slide)
        result.blobFinder.copyParameters(self.blobFinder)

        return result

    def partialDeepCopy(self, newBlobs):
        cls = self.__class__
        result = cls.__new__(cls)

        result.blobs = newBlobs
        result.generateGroupLabels()
        result.filters = deepcopy(self.filters)
        result.description = deepcopy(self.description)
        result.ROI = deepcopy(self.ROI)
        result.threshCutoff = self.threshCutoff

        result.blobFinder = blobFinder.blobFinder(self.blobFinder.slide)
        result.blobFinder.copyParameters(self.blobFinder)

        return result


    def saveBlobs(self, filename):
        '''
        save the current blob coordinates in pixels
              and the set of blob find parameters
        and histogram filters applied to generate the set
        fileName: file to save to
        '''
        if len(self.blobs) == 0:
            return
        output = open(filename,'w')
        #save blob finding parameters
        for key, val in self.blobFinder.getParameters().items():
            output.write("{}\t{}\n".format(key,val))
        #save ROI
        output.write('ROI: {}\n'.format(self.ROI))
        #save histogram filters
        if len(self.filters) != 0:
            output.write("->{}->\n".format('->'.join(self.filters)))
        else:
            output.write("->\n")
        #blb parameter header
        output.write("x\ty\tr\tc\n")
        #save blobs
        for b in self.blobs:
            output.write("{}\n".format(b.toString()))

        output.close()

    def loadBlobs(self, filename):
        '''
        Loads the blobs and sets the blob finding
              parameters from a filename
        filename: the txt file to read in.  Formatted from saveBlobs
        '''
        reader = open(filename,'r')
        lines = reader.readlines()
        self.blobs = []
        for l in lines:
            toks = l.split('\t')
            if len(toks) == 2:
                #set blob finder parameters
                self.blobFinder.setParameterFromSplitString(toks)
            elif toks[0] != 'x' and len(toks) > 2:
                #add new blob
                self.blobs.append(blob.blob.blobFromSplitString(toks))
            else:
                #get filters
                toks = l.split('->')
                if len(toks) > 1:
                    self.filters = toks[1:-1]
                elif l[0:3] == 'ROI':
                    self.ROI = ast.literal_eval(l[5:])

        self.generateGroupLabels()

    def blobRequest(self, globalPoint, radius):
        '''
        Tries to add the blob to the current blob list.
        If overlap with current blob, remove that point
        globalPoint: (x,y) tuple in the image coordinate space
        radius: the radius of the new blob to be added
        returns true if a blob was added, false if one was removed
        '''
        for i,b in enumerate(self.blobs):
            if (globalPoint[0]-b.X)**2 + (globalPoint[1]-b.Y)**2 <= \
                b.radius**2:
                self.blobs.pop(i)
                return False, i

        self.blobs.append(blob.blob(globalPoint[0],
                                    globalPoint[1], radius))
        return True, -1


    def blobSlide(self):
        if len(self.ROI) < 3:
            self.blobs = self.blobFinder.blobSlide()
            return "Finished blob finding on whole slide, found {} blobs"
                        .format(len(self.blobs))
        else:
            self.blobs = self.blobFinder.blobSlide(ROI = self.ROI)
            if len(self.blobs) != 0:
                roi = Path(self.ROI)
                points = np.array([ (b.X,b.Y) for b in self.blobs])
                self.blobs = [self.blobs[i]
                                for i in np.where(
                                    roi.contains_points(points))[0]]
            return "Finished blob finding in ROI, found {} blobs"
                                .format(len(self.blobs))

    def getROI(self, point, distCutoff, append = False):
        '''
        Performs checks and additions to interacting with an ROI.
              Does not alter ROI
        point: global point to check
        returns a new list of tuples of the ROI
        '''
        result = self.ROI.copy()
        if point is not None and len(self.ROI) > 2 and append == False:
            #find distances between point and ROI
            dists = pdist([point] + result)[:len(result)]
            #remove first point with dist <= ROI_DIST
            for i,d in enumerate(dists):
                if d < distCutoff:
                    result.pop(i)
                    return result

            #add between the two closest dists
            dists = np.append(dists, dists[0])
            dist2 = []
            for i in range(len(dists) -1):
                dist2.append(dists[i] + dists[i+1])
            #quick, no check for intersection
            #result.insert(np.argmin(dist2)+1, point)

            #slower, checks for overlapping,
              #returns the shortest distance without overlap
            pos = np.argsort(dist2)
            for p in pos:
                #check first leg of path
                segment = Path([result[p], point])
                testSeg = Path(result[p+1:] + result[:p],
                          [Path.MOVETO] + [Path.LINETO]*(len(result)-2))
                if testSeg.intersects_path(segment):
                    continue

                #check second leg of path
                if p+1 == len(result):
                    segment = Path([point, result[0]])
                    testSeg = Path(result[1:],
                          [Path.MOVETO] + [Path.LINETO]*(len(result)-2))
                    if testSeg.intersects_path(segment):
                        continue
                else:
                    segment = Path([point, result[p+1]])
                    testSeg = Path(result[p+2:] + result[:p+1],
                          [Path.MOVETO] + [Path.LINETO]*(len(result)-2))
                    if testSeg.intersects_path(segment):
                        continue

                #passed, return:
                result.insert(p+1, point)
                return result

        elif point is not None:
            result.append(point)

        return result
```

```

    def roiFilter(self):
        if len(self.ROI) < 3:
            return deepcopy(self)
        roi = Path(self.ROI)
        points = np.array([ (b.X,b.Y) for b in self.blobs])
        if points.size == 0:
            return self.partialDeepCopy([])
        result = self.partialDeepCopy([self.blobs[i]
                                      for i in np.where(
                                       roi.contains_points(points))[0]])
        return result

    def roiFilterInverse(self):
        if len(self.ROI) < 3:
            return deepcopy(self)
        roi = Path(self.ROI)
        points = np.array([ (b.X,b.Y) for b in self.blobs])
        if points.size == 0:
            return self.partialDeepCopy([])
        result = self.partialDeepCopy(
              [self.blobs[i]
                    for i in np.where(
                     np.logical_not(roi.contains_points(points)))[0]])
        return result

    def distanceFilter(self, dist, subblocks = None, verbose = False):
        '''
        Filter blob positions based on a set separation distance.
        Implemented by dividing the area into different subregions.
        Blobs are binned into at least one region,
              then all pairwise distances
        are compared to the distance cutoff.
        Returns list of bool with result[i] == true
              if i has a neighbor too close (< dist away)

        blobs: list of blobs
        dist: the distance cutoff
        subblocks: specify the number of sublocks to divide the area.
            Divides the x and y into subblocks sections
            = None allows the function to dynamically
                    determine number of subblocks
        verbose: set if output message is printed to console
        '''
        if self.blobs is None or len(self.blobs) == 0:
            return
        #initialize result and determine subblocks
        result = [False] * len(self.blobs)
        if subblocks is None:
            subblocks = int(np.ceil(np.sqrt(len(self.blobs)/100)))
            subblocks = min(subblocks, 5)

        subLocs = self._groupBlobs(dist, subblocks)

        #perform distance filtering on each sub block
        for i in range(subblocks+1):
            for j in range(subblocks+1):
                #number of points in sub region
                n = len(subLocs[i][j])
                #np array of points
                locs = np.zeros((n,2))
                #populate locs
                for ii,v in enumerate(subLocs[i][j]):
                    locs[ii,0] = self.blobs[v].X
                    locs[ii,1] = self.blobs[v].Y
                #distance filter the sub region list
                #tooClose[i] == true if too close to a neighbor
                tooClose = blobList._distFilter(locs, dist)
                #set result, index is the subLocs[x][y]
                    #and the kth point in that subregion list
                for k in np.where(tooClose)[0]:
                    result[subLocs[i][j][k]] = True

        #determine number of blobs passing filter
        count = np.sum(result)
        #report to console
        if verbose: print("Done! {} blobs within {} pixels, {} remaining"
                          .format(count, dist, len(result) - count))

        newList = self.partialDeepCopy([self.blobs[i]
                                for i in np.where(~np.array(result))[0]])
        newList.filters.append("distance > {}".format(dist))
        return newList


    @staticmethod
    def _distFilter(locs, dist):
        '''
        A helper function for performing distance
              filtering of a np matrix.
        Returns a list of bool with result[i] == true
              if too close to another point
        locs: np array of points[n][2]
        dist: the distance cutoff
        '''
        #number of points
        n = len(locs)
        #map between square form of row i, column j,
        #and the row vector from pdist
        q = lambda i,j,n: int(n*j - j*(j+1)/2+i-1-j)
        #initialize result
        result = [False] * n
        #calculate Euclidean distance
        dists = pdist(locs)
        #for each x,y pair
        for i in range(1,n):
            for j in range(i):
                #if distance between i and j is less than distance
                if dists[q(i,j,n)] < dist:
                    #both blobs fail, ie result = true
                    result[i] = True
                    result[j] = True
        return result

    def minimumDistances(self, subblocks = None, overlap = 250):
        '''
        Calculate the minimum distance between each blob.
        Similar algorithm to distFilter, but records the min distance
        returns a list of floats with the minimum distance
              between each point
        blobs: list of blobs
        subblocks: number of subdivisions of x and y dimension
            = None to dynamically choose number of subblocks
        overlap: the amount of overlap in pixels between subregions,
              in a sense defines the maximum, reliable distance reported
        '''

        if self.blobs is None or self.length() == 0:
            return None

        #initialize result
        result = [float("inf")] * len(self.blobs)
        #determine subblocks size
        if subblocks is None:
            subblocks = int(np.ceil(np.sqrt(len(self.blobs)/100)))
            subblocks = min(subblocks, 5)

        subLocs = self._groupBlobs(overlap, subblocks)

        #for each sub region list
        for i in range(subblocks+1):
            for j in range(subblocks+1):
                #number of points
                n = len(subLocs[i][j])
                #initialize temporary list of points
                locs = np.zeros((n,2))
                #add points into locs
                for ii,v in enumerate(subLocs[i][j]):
                    locs[ii,0] = self.blobs[v].X
                    locs[ii,1] = self.blobs[v].Y
                #calculate distances
                dists = blobList._minDists(locs)
                #record minimum of the distances and previous value
                for k, d in enumerate(dists):
                    result[subLocs[i][j][k]] =
                                min(result[subLocs[i][j][k]], d)

        temp = np.array(result)
        maxVal = max(temp[temp != float("inf")])
        maxVal = max(maxVal, overlap)
        for i, r in enumerate(result):
            if r == float("inf"):
                result[i] = maxVal
        return result

    @staticmethod
    def _minDists(locs):
        '''
        A helper function to calculate the closest
              neighbor of each blob
        returns a list of floats with result[i]
```

```
                indicating there exists another neighbor
            that distance away.
        locs: an np array of the x,y coordinates
        '''
        #get number of points
        n = len(locs)
        #lambda function to convert index in square and row form
        q = lambda i,j,n: int(n*j - j*(j+1)/2+i-1-j)
        #initialize result
        result = [float("inf")] * n
        #calculate Euclidean distance between each point
        dists = pdist(locs)
        #for each x,y pair
        for i in range(1,n):
            for j in range(i):
                #record the minimum of the current distance
                    #and the previous value
                result[i] = min(result[i], dists[q(i,j,n)])
                result[j] = min(result[j], dists[q(i,j,n)])
        return result

    def _groupBlobs(self, overlap, subblocks):
        '''
        A helper function for grouping blobs into subregions
              defined by the number of subblocks
        return a 2d list of indices split by the blob x and y coordinates
        overlap: amount of overlap for adding duplicate blobs
        subblocks: number of subdivisions in x and y
        '''
        #find min and max limits of x and y
        lowX = min(map(lambda x : x.X, self.blobs))
        highX = max(map(lambda x : x.X, self.blobs))
        lowY = min(map(lambda x : x.Y, self.blobs))
        highY = max(map(lambda x : x.Y, self.blobs))

        #find subblock size of x and y
        subX =  (highX - lowX)/ subblocks
        subY = (highY - lowY)/subblocks

        #initialize a 2d array of empty lists to hold each point
        result = [[[] for x in range(subblocks+1)]
                                for y in range(subblocks+1)]

        #place indices of points into subLocs list
        for i,v in enumerate(self.blobs):
            #get divisor and remainder
            (xd, xm) = (0,0) if subX == 0 else divmod(v.X-lowX, subX)
            (yd, ym) = (0,0) if subY == 0 else divmod(v.Y-lowY, subY)
            xd, yd = int(xd), int(yd)
            #put into 'normal block'
            result[xd][yd].append(i)
            #place into overlap region
            #in the top left corner
            if (xm <=2*overlap and ym <= 2*overlap) and
                          (xd-1 >=0 and yd -1 >=0):
                result[xd-1][yd-1].append(i)
            #in the left margin
            if xm <= 2*overlap and xd-1 >= 0:
                result[xd-1][yd].append(i)
            #in the top margin
            if ym <= 2*overlap and yd -1 >= 0:
                result[xd][yd-1].append(i)

        return result

    def circularPackPoints(self, spacing, maxSpots, offset, minSpots = 4,
                           r=GUIConstants.DEFAULT_PATTERN_RADIUS, c = 1):
        '''
        Expands each blob into several points surrounding the blob.
        blobs: list of blobs to expand
        spacing: minimum spacing between points
        maxSpots: max number of spots to expand for each blob
        offset: offset of circumference to space blobs
        minSpots: minimum number of spots for each blob.
              Ignores spacing with min spots
        r: radius of new blobs
        c: circumference of new blobs
        returns a list of blobs of the expanded positions
        '''
        #check maxspots to ensure less than min
        maxSpots = minSpots if maxSpots < minSpots else maxSpots
        #calculate min and max r:
        maxR = maxSpots*spacing/(2* np.pi)-offset
        #angles and unit vectors of max spots
        thetas = np.linspace(0,2*np.pi,maxSpots,False)
        maxUnits = np.vstack((np.cos(thetas),np.sin(thetas)))

        minR = minSpots*spacing/(2*np.pi)-offset
        #angles and unit vectors at min number of spots
        thetas = np.linspace(0,2*np.pi,minSpots,False)
        minUnits = np.vstack((np.cos(thetas),np.sin(thetas)))


        result = []
        ind = 0
        for blb in self.blobs:
            #check radius for min, max or between
            if(blb.radius > maxR):
                unitvec =  maxUnits
            elif(blb.radius < minR):
                unitvec = minUnits
          #between min and max, use max spots while retaining the spacing
            else:
                spots = np.floor(2*np.pi*(blb.radius + offset)/spacing)
                thetas = np.linspace(0,2*np.pi,spots,False)
                unitvec = np.vstack((np.cos(thetas),np.sin(thetas)))

            #expand each blob into a new x,y positions
            targetSpots = unitvec*(blb.radius + offset) + \
                np.matlib.repmat(np.array((blb.X, blb.Y))
                                 ,unitvec.shape[1],1).T
            #add targets to result
            for e in targetSpots.T:
                result.append(blob.blob(x = e[0],y = e[1],
                          radius = r, circularity = c, group = ind))
            #increment group number for next blob
            ind += 1

        result = self.partialDeepCopy(result)
        result.generateGroupLabels()
        return result

    def rectangularlyPackPoints(self, spacing, numLayers,
                               r = GUIConstants.DEFAULT_PATTERN_RADIUS,
                               c = 1,
                               dynamicLayering = False):
        '''
        Expands each blob into a grid of points,
              with regular rectangular spacing
        blobs: list of blob objects to expand
        spacing: spacing between new blobs
        numLayers: number of layers around each blob.
              1 generates a grid of 3x3 with the
            initial blob in the center.  This can be adjusted for radius
        r: radius to set new blobs to
        c: circularity of new blobs
        dynamicLayering: set to True to account for
              blob size in making pattern positions
        '''
       #marcher is a list of directions to move to for generating spacing
        #this starts at the right, moves down, left,
        #up, right, down, to spiral around the blob
        #additional layers are generated by
        #applying the marcher multiple times
        marcher = np.array([[0  ,  1],
                            [-1.   ,  0.   ],
                            [-1.   ,  0.   ],
                            [0.   ,  -1.   ],
                            [0.   ,  -1.   ],
                            [ 1.   ,  0.   ],
                            [ 1.   ,  0.   ],
                            [0  ,  1]])
        #use one mask each time
        if dynamicLayering == False:
            #start at center
            mask = np.array([[0,0]])
            for n in range(numLayers):
                #move to the right by n spaces
                current = np.array([[n+1.,0]])
                for i in range(8):
                    #add the marcher n times
                    direction = marcher[i,:]
                    for j in range(n+1):
                        current += direction
                        mask = np.append(mask,current,axis=0)
            #scale unit mask by spacing
            mask *= spacing

        result = []
        ind = 0
        for blb in self.blobs:
            #use new mask each blob,
              #with number of layers being size dependent
            if dynamicLayering == True:
                mask = np.array([[0,0]])
                #only change from above is the blb.radius/spacing
                for n in range(numLayers +
                                int(np.ceil(blb.radius / spacing))):
                    current = np.array([[n+1.,0]])
                    for i in range(8):
                        direction = marcher[i,:]
                        for j in range(n+1):
                            current += direction
                            mask = np.append(mask,current,axis=0)

                mask *= spacing
            #expand blb by mask
            for b in map(lambda x: blob.blob(x = x[0],
                                              y = x[1], radius = r,
                                              circularity=c,
                                              group=ind),
                          list(mask+(blb.X, blb.Y))
                          ):
                #add each point to result
                result.append(b)
            ind += 1

        result = self.partialDeepCopy(result)
        result.generateGroupLabels()
        return result

    def hexagonallyClosePackPoints(self, spacing, numLayers,
                                   r = GUIConstants.DEFAULT_PATTERN_RADIUS,
                                   c = 1,
                                   dynamicLayering = False):
        '''
        Expands each blob into a grid of points,
              with hexagonal close packed spacing
        blobs: list of blob objects to expand
        spacing: spacing between new blobs
        numLayers: number of layers around each blob.
              1 generates a grid of 7 with the
            initial blob in the center.  This can be adjusted for radius
        r: radius to set new blobs to
        c: circularity of new blobs
        dynamicLayering: set to True to account for blob size
              in making pattern positions
        '''
        #this may save some computation time to precompute
        sqrt3ov2 = np.sqrt(3)/2
        #list of directions to march along to generate a layer
        marcher = np.array([[-0.5  ,  sqrt3ov2],
                            [-1.   ,  0.   ],
                            [-0.5  , -sqrt3ov2],
                            [ 0.5  , -sqrt3ov2],
                            [ 1.   ,  0.   ],
                            [ 0.5  ,  sqrt3ov2]])
        #use one mask each time
        if dynamicLayering == False:
            #start at center
            mask = np.array([[0,0]])
            for n in range(numLayers):
                current = np.array([[n+1.,0]])
                for i in range(6):
                    direction = marcher[i,:]
                    for j in range(n+1):
                        current += direction
                        mask = np.append(mask,current,axis=0)

            mask *= spacing


        #strip off radius
        result = []
        ind = 0
        for blb in self.blobs:
            #use new mask each blob,
              #with number of layers being size dependent
            if dynamicLayering == True:
                mask = np.array([[0,0]])
                #change number of layers by blb radius
                for n in range(numLayers +
                          int(np.ceil(blb.radius / spacing))):
                    current = np.array([[n+1.,0]])
                    for i in range(6):
                        direction = marcher[i,:]
                        for j in range(n+1):
                            current += direction
                            mask = np.append(mask,current,axis=0)

                mask *= spacing
            #expand blb into points based on mask
            for b in map(lambda x: blob.blob(x = x[0], y = x[1],
                                             radius = r, circularity=c,
                                             group=ind),
                         list(mask+(blb.X, blb.Y))
                         ):
                result.append(b)
            ind += 1

        result = self.partialDeepCopy(result)
        result.generateGroupLabels()
        return result

    def generateGroupLabels(self):
        '''
        Populates groupLabels, a dict of NAME -> (x,y)
              for each group member.
        (x,y) is the top right corner (max X, min Y)
              in global coordinates
        '''

        self.groupLabels = dict()
        for b in self.blobs:
            if b.group is not None:
                #add in current blob
                if b.group not in self.groupLabels:
                    self.groupLabels[b.group] = (b.X, b.Y)
                else:
                    #update tuple
                    p = self.groupLabels[b.group]
                    self.groupLabels[b.group] =
                                 (max(b.X, p[0]), min(b.Y, p[1]))


    def getPatches(self, limitDraw, slideWrapper, blobColor):

        todraw = slideWrapper.getBlobsInBounds(self.blobs)

        if limitDraw and len(todraw) > GUIConstants.DRAW_LIMIT:

            todraw = [todraw[i]
                    for i in range(0, len(todraw),
                       len(todraw)//GUIConstants.DRAW_LIMIT)]

        return list(map(lambda el: plt.Circle((el[0],el[1]), el[2],
                                             color = blobColor,
                                             linewidth = 1,
                                             fill = False), todraw))
```

ImageUtilities/enumModule.py

```python
from enum import Enum, unique

@unique
class Direction(Enum):
    '''
    Enum class to encode directions for slide stepping
    '''
    left = 1
    right = 2
    up = 3
    down = 4

@unique
class StepSize(Enum):
    '''
    Enum class to encode sizes for slide stepping
    '''
    small = 1
    medium = 2
    large = 3
    giant = 4
```

ImageUtilities/slideWrapper.py

```
import openslide
from PIL import Image, TiffImagePlugin
import PIL.ImageOps
import numpy as np
import numpy.matlib
import os
import fnmatch
import matplotlib as mpl
from matplotlib.path import Path

from ImageUtilities.enumModule import Direction, StepSize
from ImageUtilities import blob

class SlideWrapper(object):
    '''
    Class to encapsulate interactions with microscopy experiments.
    Wraps the openslide package to support multiple
        channels/images and zoom levels.
    Keeps track of current view window so movement
        is called by step functions.
    '''
    def __init__(self, fileName, size = [1024,1024], startLvl = 0):
        '''
        Create a new slideWrapper instance with the fileName experiment
        Automatically looks for multiple images.
        Ndpi image pairs should end in Brightfield or Triple.
        Single Ndpi images are also supported.  Tif images
        should end in c#.tif (e.g. c1.tif) to be grouped together.
        The tif image set doesn't need to be consecutive.
        Single Tif images are also supported
        fileName: a tif or ndpi image
        size: The width and height of the image to load
        startLvl: the starting zoom level.  0 <= startLvl,
              with 0 being the max zoom
        '''

        (p,f) = os.path.split(fileName)
        (f,ex) = os.path.splitext(f)

        self.slides = []
        self.filetype = ex

        #nanozoomer, ends in triple or brightfield
        if ex == '.ndpi':
            #brightfield image selected
            if "Brightfield" == f[-11:]:
                self.slides.append([openslide.open_slide(fileName)])
                if os.path.exists(os.path.join(p,f[:-11]+'Triple'+ex)):
                    self.slides.append([openslide.open_slide(
                            os.path.join(p,f[:-11]+'Triple'+ex))])

            #fluorescence image selected
            elif "Triple" == f[-6:]:
                if os.path.exists(os.path.join(p,
                                    f[:-6]+'Brightfield'+ex)):
```

```
                        self.slides.append([openslide.open_slide(
                              os.path.join(p,f[:-6]+'Brightfield'+ex))])
                    self.slides.append([openslide.open_slide(fileName)])

                #single image selected
                else:
                    self.slides.append([openslide.open_slide(fileName)])

            #zeiss, ends in c#.tif
            elif ex == '.tif':
                #iterate through each number, 1-9
                if "c" == f[-2] and f[-1].isdigit():
                    for i in range(1,9):
                        if os.path.exists(os.path.join(
                                                  p,f[:-1]+str(i) + ex)):
                            self.slides.append(
                                    [openslide.open_slide(
                                    os.path.join(p,f[:-1]+str(i) + ex))])
                            if os.path.exists(os.path.join(p,
                                             '64x' + f[:-1]+str(i) + ex)):
                                self.slides[-1].append(openslide.open_slide(
                                                  os.path.join(p,
                                                  '8x' + f[:-1]+str(i) + ex)))
                                self.slides[-1].append(openslide.open_slide(
                                             os.path.join(p,
                                                  '64x' + f[:-1]+str(i) + ex)))
                        else:
                            self.slides.append(None)
                #single image
                else:
                    self.slides.append([openslide.open_slide(
                                    os.path.join(p,f + ex))])
                    #load decimated images if they exist
                    if os.path.exists(os.path.join(p,'64x' + f + ex)):
                        self.slides[-1].append(openslide.open_slide(
                                    os.path.join(p,'8x' + f + ex)))
                        self.slides[-1].append(openslide.open_slide(
                                          os.path.join(p,'64x' + f + ex)))
                #remove end until not empty
                while self.slides[-1] is None:
                    self.slides.pop()

            else:
                raise ValueError("Only tif and ndpi currently supported")

            ind = 0
            #get first non-blank channel
            while self.slides[ind] is None:
                ind += 1

            #initialize variables
            self.level_count = self.slides[ind][0].level_count
            self.dimensions = self.slides[ind][0].dimensions

            self.displaySlides = [True]*len(self.slides)
            self.brightInd = ind #index of brightfield image,
            #determines how channels are merged
```

```
        self.size = size
        self.lvl = startLvl
        self.lvl = 0 if self.lvl < 0 else self.lvl
        limit = self.level_count-1+2
        if len(self.slides[ind]) > 2:
            limit += 6
        self.lvl = limit if self.lvl > limit else self.lvl
        self.pos = [size[0]*2**(self.lvl-1), size[1]*2**(self.lvl-1)]

    def getImg(self):
        '''
        Reads the slide image from disk at the current position,
              zoom, and channels

        '''
        fluorImg = None
        brightImg = None
        for i,display in enumerate(self.displaySlides):
            #read in brightfield and fluorescence images
            if display == True:
                #only one image is designated as brightfield
                if i == self.brightInd:
                    brightImg = self._getImg(i)
                #fluorescence images are merged by summing the
                    #intensity in each channel. this can lead to overflow
                    #in images that are not 'pure' R,G,B
                else:
                    if fluorImg is None:
                        fluorImg = self._getImg(i)
                    else:
                        imgs = []
                        splitOld = fluorImg.split()
                        splitNew = self._getImg(i).split()
                        for j in range(4):#skip alpha
                            if np.max(splitOld[j]) < np.max(splitNew[j]):
                                imgs.append(splitNew[j])
                            else:
                                imgs.append(splitOld[j])
                        fluorImg = Image.merge('RGBA', imgs)

        #merge bright and fluorescence image, or return one of them
        if brightImg is None and fluorImg is None:
            slideImg = Image.new("RGB",self.size,"black")
        elif brightImg is not None and fluorImg is None:
            slideImg = brightImg
        elif fluorImg is not None and brightImg is None:
            slideImg = fluorImg
        else:
            slideImg =Image.blend(brightImg, fluorImg,0.5)

        return slideImg

    def _getImg(self, imageInd):
        '''
        Helper method to read in an image from a single channel.
              Uses instance position and zoom
        imageInd: the image index to read
        '''
        #have to convert the position to keep self.pos at the center
        #read_region take the top left point
        tempPos = list(map(lambda x, y: int(x-y*2**(self.lvl-1)),
                   self.pos, self.size))
        #if zoom level is in bounds for openslide
        if self.lvl < self.level_count:
            if self.slides[imageInd] is None:
                return None
            return self.slides[imageInd][0].read_region(tempPos,
                                                         self.lvl,
                                                         self.size)
        #decimate image to desired zoom level
        else:
            #read in larger area and resize down to desired size
            if self.lvl - self.level_count < 2:
                tempPos = list(map(lambda x, y: int(x-y*2**(self.lvl-1)),
                   self.pos, self.size))
                tempSize = list(map(lambda x:
                                int(x*2**(self.lvl-self.level_count+1)),
                                            self.size))
                return self.slides[imageInd][0].read_region(tempPos,
                                            self.level_count-1, tempSize)
                                                  .resize(self.size)

            #same as above, but with 8x decimated image
            elif self.lvl - self.level_count < 5 and
                          len(self.slides[imageInd]) > 1:
                tempPos[0] //= 8
                tempPos[1] //= 8
                tempSize = list(map(lambda x:
                                            int(x*2**(self.lvl-3)),
                                            self.size))
                return self.slides[imageInd][1]
                          .read_region(tempPos, 0, tempSize)
                          .resize(self.size)

            #same as above, but with 64x decimated image
            elif  len(self.slides[imageInd]) > 2 and
                          self.lvl-self.level_count < 8:
                tempPos[0] //= 64
                tempPos[1] //= 64
                tempSize = list(map(lambda x:
                                            int(x*2**(self.lvl-6)),
                                            self.size))
                return self.slides[imageInd][2]
                                .read_region(tempPos, 0, tempSize)
                                .resize(self.size)

            #zoom is outside of bounds for this channel
            else:
                return None

    def getMaxZoomImages(self, baseDir, positions,
                                size = None, prefix = '',
                                invert = False, imgInd = 1):
        '''
        Saves images of each position provided.
        baseDir: Directory to save all images
        positions: list of tuples with x,y positions of blobs
        size: size of images to save in pixels
        prefix: prefix of images to save
        invert: toggle color inversion.  Can be useful for printing
        imgInd: the image index to use
        '''
        if size is None:
            size = self.size
        for p in positions:
            imgInd = min(len(self.slides), imgInd)
            tempPos = list(map(lambda x, y: int(x-y/2), p, size))
            if invert:
                fp = os.path.join(baseDir, "{}{}_{}_inv.png"
                                      .format(prefix,p[0], p[1]))
                img = self.slides[imgInd][0]
                                      .read_region(tempPos, 0, size)
                img = Image.merge('RGB', img.split()[0:3])
                PIL.ImageOps.invert(img).save(fp)
            else:
                fp = os.path.join(baseDir, "{}{}_{}.png"
                                .format(prefix,p[0], p[1]))
                self.slides[imgInd][0]
                                .read_region(tempPos, 0, size).save(fp)

    def getMaxZoomImage(self, position = None, size = None, imgInd = 1):
        '''
        Get the image at the maximum zoom level.  Used in blob finding
        position: tuple of x,y position of image center.
              None to use self.position
        size: tuple of width and height, None for self.size
        imgInd: the image index to read
        '''
        imgInd = min(len(self.slides)-1, imgInd)
        if position is None:
            position = self.pos
        if size is None:
            size = self.size
        tempPos = list(map(lambda x, y: int(x-y/2),
                                    position, size))
        return (self.slides[imgInd][0]).read_region(tempPos, 0, size)

    def step(self, direction, stepSize):
        '''
        Step the position in the designated direction.
        direction: a slideWrapper.Direction enum
        stepSize: enum of step size. large = image size,
              medium 1/2, small 1/10 that size
        '''
        if stepSize == StepSize.large:
            factor = 1
        elif stepSize == StepSize.medium:
            factor = 2
        else:
            factor = 10
        dirMap = {
            Direction.left : [-1,0],
            Direction.right : [1,0],
            Direction.up : [0,-1],
            Direction.down : [0,1],
        }
        self._movePos(factor, dirMap[direction])

    def _movePos(self, factor, direction):
        '''
        Helper method to perform position movement.
        factor: division factor to step size
        direction: an x,y list of the step to perform
        '''
        #have to scale position movement by the current zoom level
        self.pos[0] += direction[0] * self.size[0]//factor*2**self.lvl
        self.pos[1] += direction[1] * self.size[1]//factor*2**self.lvl

    def _zoom(self, amt):
        '''
        Zoom helper method to bound self.lvl properly
        amt: integer change in zoom level.  <0 is zooming in
        '''
        self.lvl += amt
        #keep >= 0
        self.lvl = 0 if self.lvl < 0 else self.lvl
        #limit sets the amount of software decimation to use.
        #2 doesn't cause too much lag on GUI
        limit = self.level_count-1+2
        ind = 0
        while self.slides[ind] is None:
            ind += 1
        #if the images have 8 and 64x decimations available,
        #extra zoom levels are possible
        if len(self.slides[ind]) > 2:
            limit += 6
        self.lvl = limit if self.lvl > limit else self.lvl

    def zoomIn(self):
        '''
        Zoom the image in one step (2x smaller pixels)
        '''
        self._zoom(-1)

    def zoomOut(self):
        '''
        Zoom the image out one step (2x larger pixels)
        '''
        self._zoom(1)

    def resetView(self):
        '''
        Reset the position and zoom level
        Useful for debugging if the position gets far out of bounds
        '''
        self.lvl = 0
        self.pos = [self.size[0]/2, self.size[1]/2]

    def switchType(self):
        '''
        Cycle through image channels available,
              moving +1 from first true
        '''
        ind = 0
        for i in range(len(self.slides)):
            if self.displaySlides[i] == True:
                ind = i+1
                break
        ind %= len(self.slides)

        self.switchToChannel(ind)

    def switchToChannel(self, ind):
        '''
        Turn the target image channel on and the rest off
        ind: image channel to activate.
              Performs index out of bounds checks
        '''
        self.displaySlides = [False]*len(self.slides)
        if ind < len(self.slides):
            ind = 0 if ind < 0 else ind
            self.displaySlides[ind] = True

    def toggleChannel(self,ind):
        '''
        Toggle the supplied image channel on or off
        ind: the image channel to toggle
        '''
        ind = 0 if ind < 0 else ind
        if ind < len(self.slides):
            self.displaySlides[ind] = not self.displaySlides[ind]

    def setBrightfield(self,ind):
        '''
        Set the index of the brightfield image.
        If the supplied index is the current brightfield index,
              turns off the brightfield
        ind: the image channel to set as brightfield
        '''
        if ind == self.brightInd:
            self.brightInd = -1
        else:
            self.brightInd = ind

    def moveCenter(self, imgPos):
        '''
        Move self.pos to the supplied image position, in pixels.
        imgPos: the x,y pixel position to move to
        '''
        #have to modify by the current zoom level
        self.pos[0] += int((imgPos[0]-self.size[0]/2)*2**self.lvl)
        self.pos[1] += int((imgPos[1]-self.size[1]/2)*2**self.lvl)

    def getGlobalPoint(self, point):
        """
        Convert the local point in the image view to
              a slide global point
        point: the local pixel position as tuple
        returns the same point, relative to the top left
              of the image at max zoom
        """
        result = [0,0]
        result[0] = self.pos[0]
                    + round((point[0]-self.size[0]/2)*2**self.lvl)
        result[1] = self.pos[1]
                    + round((point[1]-self.size[1]/2)*2**self.lvl)
        return (result[0], result[1])

    def getLocalPoint(self, point):
        """
        Convert the global point in slide to
                    position in the current image
        point: the global pixel point
        returns the pixel position in the current image view
        """
        return [round((point[0] - self.pos[0])
                          /2**self.lvl + self.size[0]/2),
            round((point[1] - self.pos[1])
                          /2**self.lvl + self.size[1]/2)]

    def getPointsInBounds(self, points):
        """
        Test the supplied global points to see if they land
              in the current image.
        points: list of global slide pixel positions
        returns the points in bounds translated into local
              image coordinate system and the indices of those
              points in the input list
        """
        #get bounds of image in global coordinate
        xlow, ylow = self.getGlobalPoint((0,0))
        xhigh, yhigh = self.getGlobalPoint(self.size)
        zero = (xlow,ylow)
        result = []
        indices = []
        #for each point
        for i, p in enumerate(points):
            #if in bounds
            if p[0] >= xlow and p[0] <= xhigh and
                          p[1] >= ylow and p[1] <=yhigh:
                #add local point and the index of that point
                result.append(((p[0]-zero[0])/2**self.lvl,
                                            (p[1]-zero[1])/2**self.lvl))
                indices.append(i)
        return result, indices

    def getBlobsInBounds(self, blobs):
        """
        Test the supplied global points to see if they land
              in the current image.
        blobs: a list of blobs in global coordinates
        returns a list of (x,y,r) translated into local image
```

```
               coordinate system with radius scaled to zoom level
        """
        if len(blobs) == 0:
            return []
        #get bounds of image in global coordinate
        xlow, ylow = self.getGlobalPoint((0,0))
        xhigh, yhigh = self.getGlobalPoint(self.size)

        return [((b.X-xlow)/2**self.lvl,
                 (b.Y-ylow)/2**self.lvl,
                 b.radius/2**self.lvl)
                 for b in blobs
                 if b.X > xlow and b.X < xhigh and\
                     b.Y > ylow and b.Y < yhigh]


    def getSize(self):
        '''
        Returns the dimensions of the slide image
        '''
        return self.dimensions

    def getFluorInt(self, blobs, channel, imageInd,
                          offset = 0, reduceMax = False):
        '''
        Determines the intensity of pixels around each blob
        blobs: list of blob objects to analyze
        channel: the R,G,B channel to analyze (0, 1, 2)
        imageInd: the image channel to analyze
        offset: adjusts the blob radius to consider
              smaller or larger regions
        reduceMax: toggle between returning the average (False)
              or max (True) intensity
        '''
        result = []
        #use the max or mean intensity
        if reduceMax:
            reduction = lambda x: np.max(np.array(x.split()[channel]))
        else:
            reduction = lambda x: np.mean(np.array(x.split()[channel]))
        for i,b in enumerate(blobs):
            #note that this considers the square circumscribing the blob
            img = self.getMaxZoomImage((int(b.X),int(b.Y)),
                                       (int(b.radius+offset)*2,
                                         int(b.radius+offset)*2),
                                       imgInd=imageInd)
            #calc summed intens in area
            result.append(reduction(img))
            #report every 100 blobs
            if (i+1)%100 == 0:
                print(str(i+1) + ' blobs read')
        return result

    @staticmethod
    def decimateImg(img, factor):
        '''
        Static utility to decimate the image provided
        img: a openslide instance
        factor: integer factor to reduce size by
        returns a PIL.Image of img at the reduced size
        '''
        result = Image.new('RGB', tuple(map(lambda x:
                                   x//factor,img.dimensions)))
        import time
        start = time.time()

        #determines how large of a region to read in at once,
        #works in strips of image
        loadFac = 64

        #read in horizontal strips, this seems to be moderately faster
        for i in range(result.size[1]//loadFac):
            result.paste(img.read_region((0,i*factor*loadFac),
                         0,(result.size[0]*factor,factor*loadFac))
                         .resize((result.size[0],loadFac)),
                         0,i*loadFac,result.size[0],(i+1)*loadFac))
            if i!= 0 and i % 10 == 0 or i == 1:
                print("finished %d of %d subareas, %d seconds left" %
                      (i, result.size[1]//loadFac,
                        (time.time()-start)/i *
                                      (result.size[1]//loadFac-i)))

        #copy remainder
        if result.size[1] % loadFac != 0:
            result.paste(
                    img.read_region(
                          (0,result.size[1]//loadFac*loadFac*factor),
                          0,
                          (result.size[0]*factor,
                                result.size[1] % loadFac*factor))
                     .resize((result.size[0], result.size[1]%loadFac)),
                                (0,result.size[1]//loadFac*loadFac,
                                      .size[0], result.size[1]))

        return result

    @staticmethod
    def generateDecimatedImage(path, baseFile):
        '''
        Saves 8x and 64x image of the single file
        path: path containing image file.  New images written here
        baseFile: base file name with extension,
              8x and 64x will be prepended onto base name
        '''
        TiffImagePlugin.WRITE_LIBTIFF = True
        SlideWrapper.decimateImg(
              openslide.open_slide(
                    os.path.join(path,baseFile)),8)
        .save(os.path.join(path,'8x' + baseFile),
              compression='tiff_lzw')
        SlideWrapper.decimateImg(
              openslide.open_slide(
                    os.path.join(path,'8x' + baseFile)),8)
        .save(os.path.join(path,'64x' + baseFile),
                compression='tiff_lzw')
            TiffImagePlugin.WRITE_LIBTIFF = False

    @staticmethod
    def generateDecimatedImgs(filename):
        '''
        Saves 8x and 64x images of given image in filename as
              8xFILENAME and 64xFILENAME
        filename: Full path to tif image
        '''

        (p,f) = os.path.split(filename)
        (f,ex) = os.path.splitext(f)
        #zeiss, ends in c#.tif
        if ex == '.tif':
            #filename has c# form, decimate each
            if "c" == f[-2] and f[-1].isdigit():
                totimgs = 0
                for i in range(1,9):
                    if os.path.exists(os.path.join(
                                p,f[:-1]+str(i) + ex)):
                        totimgs += 1
                for i in range(1,9):
                    if os.path.exists(os.path.join(
                                p,f[:-1]+str(i) + ex)):
                        print("starting channel {} of {}"
                                            .format(i, totimgs))
                        SlideWrapper.generateDecimatedImage(p,
                                                  f[:-1]+str(i) + ex)
            #filename is a single tif image
            else:
                SlideWrapper.generateDecimatedImage(p,f + ex)

    @staticmethod
    def decimateDirectory(dirName):
        #get all dirs in parent dir
        for subd in [os.path.join(dirName,o)
              for o in os.listdir(dirName)
                    if os.path.isdir(os.path.join(dirName,o))]:
            targetFiles = [os.path.join(subd, o)
                    for o in os.listdir(subd)
                          if fnmatch.fnmatch(o, '*.tif')]
            for fname in targetFiles:
                (path, file) = os.path.split(fname)
                if (not os.path.exists(os.path.join(path, '8x' + file))
                          or not os.path.exists(
                                os.path.join(path, '64x' + file))) \
                    and file[0:2] != '8x' and file[0:3] != '64x':
                    print(fname)
                    SlideWrapper.generateDecimatedImage(path, file)
        print('Finished!')
```

ImageUtilities/TSPutil.py

```python
from scipy.spatial.distance import pdist
import numpy as np
import time
import random

def TSPRoute(inlist, optT1 = 30, optT2 = 60):
    '''
    optimizes the traversal order of a list of points
    should be close to optimal after completing all iterations
    returns a list of indices to visit in optimized order
    inlist: a list of the positions
    optT1: max optimization time for the initial,
        nearest neighbor optimization in seconds
    optT2: max optimization time for the 2-opt in seconds
    '''

    if inlist is None or len(inlist) == 0:
        return None

    dat = np.array([])

    #strip off just the x and y values
    for l in inlist:
        dat = np.append(dat,[l[0], l[1]])

    dat = np.reshape(dat,(np.size(dat)//2,2))
    #calculate pair-wise distances
    dists = pdist(dat)
    #map the i,j values of a square form matrix
  #to the flat form returned by pdist
    def sqr(i,j,n=dat.shape[0]):
        if i < j:
            return int(n*i-(i+1)*i/2 + (j-i-1))
        return int(n*j-(j+1)*j/2 + (i-j-1))

    #nearest neighbor traversal
    bestDist = float("inf")
    inds = list(range(dat.shape[0]))

    #randomize starting index
    random.seed(0)#for testing purposes
    random.shuffle(inds)

    iterat = 0
    start_time = time.time();
    #while not timed out
    while time.time()-start_time < optT1 and iterat < dat.shape[0]:
        #the current solution start point
        soln = [inds.pop()]
        #current iteration
        iterat += 1
        #remaining points to visit in this iteration
        remaining = [i for i in range(0,dat.shape[0])]
        remaining.pop(soln[0])

        #for each point
        for count in range(1,dat.shape[0]):
            #visit next closest neighbor
            nextI = np.argmin(list(map(lambda i:
                                    dists[sqr(i,soln[-1])],
                                    remaining)))
            #add to solution and remove from remaining
            soln.append(remaining.pop(nextI))

        #calculate the total traversed distance
        dist = sum(map(lambda i: dists[sqr(soln[i],soln[i+1])],
                                            range(dat.shape[0]-1)))
        #if better than previous nearest neighbor traversal
        if dist<bestDist:
            #update best path and print update
            bestDist = dist
            bestSoln = soln[:]
            print("{0} iterations of nearest neighbor in {1:.1f} seconds"
                          .format(iterat, time.time()-start_time))

    print("{} iterations of nearest neighbor".format(iterat))
    soln=bestSoln

    #add a blank node to end, won't move from the end
    #allows the start position to move around
    soln.append(len(soln))
    dists = np.append(dists, np.zeros(len(soln)))

    start_time = time.time();
    iterat=0

    #2-opt
    #while not timing out and still optimizing
    while time.time()-start_time < optT2:
        iterat += 1
        #keep track of if a switch in the path was made
        pathChanged = False
        #for each point
        for i in range(1,dat.shape[0]):
            #for each point between i and end
            for j in range(i+1,dat.shape[0]-1):
                #check if switching i-1 -> i to i-1 -> j is improvement
                if dists[sqr(soln[i-1],soln[i])]
                          + dists[sqr(soln[j],soln[j+1])] > \
                    dists[sqr(soln[i],soln[j+1])]
                                + dists[sqr(soln[j],soln[i-1])]:
                    #reverse the order of points visited
                    soln[i:j+1] = reversed(soln[i:j+1])
                    pathChanged = True
                    break #go to next i
        if pathChanged == False:
            break #stop if no switches were made over each for loop
        if iterat % 5 == 0:
            print("{0} iterations of TSP in {1:.1f} seconds"
                    .format(iterat, time.time()-start_time))
    print("{0} iterations of TSP in {1:.1f} seconds"
        .format(iterat, time.time()-start_time))

    del soln[-1] #remove last, dummy point
    print("TSP optimization finished!")
    return soln
```

CoordinateMappers/__init__.py

```
'''
Package with all coordinate system mappers and connected instruments
brukerMapper.py:      An abstract base class implementing coordinateMapper
                        specific for bruker type instruments using their
                        fractional distance.
connectedInstrument.py: An abstract base class specifying functions
                        required for interacting with a connected
                        instrument
coordinateMapper.py:    An abstract base class with some standard
                        methods for mapping pixel positions to physical
                        locations in an instrument.
flexImagingSolarix.py:  An extension of solarixMapper which generates
                        files suitable for acquisition with flexImaging.
oMaldiMapper.py:        Implementation of coordinate mapper for
                        acquisition with the AB Sciex oMaldi server.
                        Contains methods for slop correction
solarixMapper.py:       Implementation of brukerMapper for the solarix
                        FT-ICR  that generates xeo and xls files
                        for autoacquisition
supportedCoordSystems.py: A collection of coordinatemappers.  Only
                        mappers included here will be accessible
                        to the GUI
ultraflexMapper.py:     An implementation of brukerMapper for the
                        ultraflextreme tof/tof that generates xeo
                        files for autoexecute
zaber3axis.py           Concrete implementation of a connected instrument
                        for an XYZ stage for liquid microjunction
                        extraction.
zaberInterface.py:      An abstract base class with methods for
                        interacting with zaber linear actuators.
zaberMapper.py:         An implementation of coordinateMapper with a
                        connected zaber3axis used for the liquid
                        microjunction extraction.
'''
```

CoordinateMappers/brukerMapper.py

```
from CoordinateMappers import coordinateMapper
from GUICanvases import GUIConstants
from ImageUtilities import blob
import abc
import numpy as np
import itertools

class brukerMapper(coordinateMapper.CoordinateMapper,
                   metaclass=abc.ABCMeta):
    """
    A generic bruker mapper with constants for the slide II adaptor
        and xeo headers. all physical points are stored as motor
        coordinates so if a user enters C5 that has to be converted.
        Upon saving, the motor coordinates are finally converted
        in one step.
    """

    '''
    also need to define:
    self.motorCoordinateFilename: filename for intermediate
        mapping positions.  Should be unique
    '''

    def __init__(self):
        super().__init__()
        self.MTPMapY = {
            'C':0.478261,
            'D':0.391304,
            'E':0.304348,
            'F':0.217391,
            'G':0.130435,
            'J':-0.130435,
            'K':-0.217391,
            'L':-0.304348,
            'M':-0.391304,
            'N':-0.478261,
        }

        self.MTPMapX = {
            '5':-0.652174,
            '6':-0.565217,
            '7':-0.478261,
            '8':-0.391304,
            '9':-0.304348,
            '10':-0.217391,
            '11':-0.130435,
            '12':-0.043478,
            '13':0.043478,
            '14':0.130435,
            '15':0.217391,
            '16':0.304348,
            '17':0.391304,
            '18':0.478261,
            '19':0.565217,
            '20':0.652174,
        }

        self.header = ('<!-- $Revision: 1.5 $-->\n'
'<PlateType>\n'
'\t<GlobalParameters PlateTypeName="MTP Slide Adapter II"
                    ProbeType="MTP"\n'
'\t                  RowsNumber="100" ChipNumber="1" ChipsInRow="1"\n'
'\t                  X_ChipOffsetSize="0" Y_ChipOffsetSize="0"\n'
'\t                  HasDirectLabels="false" HasColRowLabels="true"\n'
'\t                  HasNearNeighbourCalibrants="false"\n'
'\t                  ProbeDiameterX="103.5" SampleDiameter="2"\n'
'\t                  SamplePixelRadius="5" ZoomFactor="1"\n'
'\t                  FirstCalibrant="TPX1" SecondCalibrant="TPX2"
                    ThirdCalibrant="TPX3"\n'
'\t                  />\n'
'\t<MappingParameters mox="56.239998" moy="42.635009"'
    ' sinphi="0.0" cosphi="1.0" '
'alpha="51.750000" beta="51.750000" tansigma="0.0"/>\n')

        self.footer = """\t</PlateSpots>
        <AutoTeachSpots>
            <PlateSpot PositionIndex="0" PositionName="TPX1"
                UnitCoord_X="-0.729469" UnitCoord_Y="0.550725"/>
            <PlateSpot PositionIndex="1" PositionName="TPX2"
                UnitCoord_X="0.729469" UnitCoord_Y="0.550725"/>
            <PlateSpot PositionIndex="2" PositionName="TPX3"
                UnitCoord_X="0.729469" UnitCoord_Y="0.057971"/>
            <PlateSpot PositionIndex="3" PositionName="TPX4"
                UnitCoord_X="-0.729469" UnitCoord_Y="0.057971"/>
            <PlateSpot PositionIndex="4" PositionName="TPY1"
                UnitCoord_X="-0.729469" UnitCoord_Y="-0.057971"/>
            <PlateSpot PositionIndex="5" PositionName="TPY2"
                UnitCoord_X="0.729469" UnitCoord_Y="-0.057971"/>
            <PlateSpot PositionIndex="6" PositionName="TPY3"
                UnitCoord_X="-0.729469" UnitCoord_Y="-0.550725"/>
            <PlateSpot PositionIndex="7" PositionName="TPY4"
                UnitCoord_X="0.729469" UnitCoord_Y="-0.550725"/>
        </AutoTeachSpots>
    </PlateType>"""

        #list of all MTP points, used for drawing predicted points
        self.allPoints = list(itertools.product(self.MTPMapX.values(),
                                self.MTPMapY.values()))
        #a tuple of (R,s,t) for PBSR of motor coordinates to MTP
        self.motor2MTP = None
        #list of motor training coordinates
        self.motor = []
        #list of mtp training points
        self.mtp = []

        #load the stored training coordinates
        self.loadStagePoints()

    @abc.abstractmethod
    def loadStagePoints(self):
        '''
        read in or hard code the map from motor coordinates
```

```
            to MTP points. should populate the self.motor2MTP = (R,s,t)
        '''

    @abc.abstractmethod
    def isValidMotorCoord(self, inStr):
        '''
        tests if the user-entered string is a valid motor coordinate
        inStr: the string to test if it follows motor coordinate format
        returns true if inStr can be successfully
            parsed by extractMotorPoint
        '''

    def isValidEntry(self, inStr):
        '''
        Test if the string is a valid entry for a physical coordinate.
        Can be a motor coordinate or MTP string
        inStr: string to test
        returns true if inStr can be successfully parsed by extractPoint
        '''
        if inStr is None or len(inStr) < 2:
            return False
        return self.isValidMTP(inStr) or self.isValidMotorCoord(inStr)

    def isValidMTP(self, inStr):
        '''
        Test if the provided string is a valid MTP named coordinate
        inStr: String to test
        returns true if the point is encoded
        '''
        if inStr is None or len(inStr) < 2:
            return False
        Y = inStr[0].upper()
        X = inStr[1:]
        return X in self.MTPMapX and Y in self.MTPMapY

    def extractPoint(self, inStr):
        '''
        Extract motor coordinate of the supplied point.
        inStr: string to parse
        returns the motor coordinate or None if the string is not valid
        '''
        if self.isValidMTP(inStr):
            return self.extractMTPPoint(inStr)
        elif self.isValidMotorCoord(inStr):
            return self.extractMotorPoint(inStr)
        else:
            return None

    def extractMTPPoint(self, inStr, needMTP = False):
        '''
        From a provided, named MTP point, returns the motor position.
        inStr: the named position to try and parse
        needMTP: optional toggle to get the fractional distance
            instead of translating to a motor coordinate
        returns an (x,y) tuple of motor coordinate,
            fractional distance (needMTP == True)
            or None if instr is invalid
        '''
        if inStr is None or len(inStr) < 2:
            return None
        Y = inStr[0].upper()
        X = inStr[1:]

        if X in self.MTPMapX and Y in self.MTPMapY:
            if needMTP:
                return (self.MTPMapX[X], self.MTPMapY[Y])
            else:
                return self.MTPtoMotor((self.MTPMapX[X],
                                        self.MTPMapY[Y]))
        else:
            return None

    @abc.abstractmethod
    def extractMotorPoint(self, inStr):
        '''
        given a user-entered motor coordinate,
            parse out the x and y coordinates
        inStr format could change between instruments
            if copy/paste is supported
        '''

    def predictName(self, pixelPoint):
        '''
        predict the name of a given pixel position
        By default returns the named mtp coordinate
        '''
        if len(self.physPoints) < 2:
            return ''

        #convert pixel to motor
        motor = self.translate(pixelPoint)
        return self.predictLabel(motor)

    def mtpLabel(self, mtpCoord):
        '''
        From a given mtpCoordinate, returns the named position
            on an mtp slide II adapter
        mtpCoord: (x,y) tuple in fractional distance coordinates
        '''
        X,Y = mtpCoord

        #return MTP coordinate
        Ymin, Ykey = abs(Y-next(iter(self.MTPMapY.values()))), \
            next(iter(self.MTPMapY.keys()))
        Xmin, Xkey = abs(X-next(iter(self.MTPMapX.values()))), \
            next(iter(self.MTPMapX.keys()))
        for key, val in self.MTPMapY.items():
            if abs(Y-val) < Ymin:
                Ymin, Ykey = abs(Y-val), key
        for key, val in self.MTPMapX.items():
            if abs(X-val) < Xmin:
                Xmin, Xkey = abs(X-val), key
        return Ykey+Xkey

    def predictLabel(self, physPoint):
        '''
        Predicts the label of a registration mark
            based on the physical position
        physPoint: (x,y) in motor coordinate system
        returns the named, mtp point
        '''
        #motor to MTP
        return self.mtpLabel(self.motorToMTP(physPoint))

    def predictedPoints(self):
        '''
        Gets all the predicted points of the named, mtp positions
        '''
        if len(self.physPoints) < 2:
            return []
        result = []
        for p in self.allPoints:
            result.append(self.invert(self.MTPtoMotor(p)))

        return result

    def motorToMTP(self, motorCoord):
        '''
        Performs translation of the motor coordinate
            system to fractional distance
        with the self.motor2MTP map.
        motorCoord: (x,y) tuple the motor coordinate
        returns the (xy) tuple in fractional distance
        '''
        (R,s,t) = self.motor2MTP
        mtp = s * R * np.matrix([[motorCoord[0]],[motorCoord[1]]]) + t
        return (mtp[0,0], mtp[1,0])

    def MTPtoMotor(self, MTPcoord):
        '''
        Translates the fractional distance to a motor coordinate.
        MTPcoord: (x,y) tuple a fractional distance
        returns the (x,y) tuple in motor coordinate
        '''
        (R,s,t) = self.motor2MTP
        motor = np.linalg.inv(R)*\
            (np.matrix([[MTPcoord[0]],[MTPcoord[1]]]) - t)/s
        return (motor[0,0], motor[1,0])

    def writeXEO(self, filename, blobs):
        '''
        write an xeo file of the provided list of
            blobs with appropriate header
        and format to use as a Bruker geometry file.
        filename: the xeo file to save
        blobs: list of blobs to save
        '''
        if blobs is None or len(blobs) == 0:
            return
        output = open(filename, 'w')

        output.write(self.header)
        output.write('    <PlateSpots PositionNumber="{}">\n'
                            .format(len(blobs)))

        for i,p in enumerate(blobs):
            trans = self.motorToMTP(self.translate((p.X, p.Y)))
            if p.group is None:
                output.write('        <PlateSpot PositionIndex="{0}" \
                                PositionName="x_{1:.0f}y_{2:.0f}" \
                                UnitCoord_X="{3:.6f}" \
                                UnitCoord_Y="{4:.6f}"/>\n'.format(
                                i, p.X, p.Y, trans[0], trans[1]))
            else:
                output.write('        <PlateSpot PositionIndex="{0}" \
                        PositionName="s_{5:.0f}x_{1:.0f}y_{2:.0f}" \
                        UnitCoord_X="{3:.6f}" \
                        UnitCoord_Y="{4:.6f}"/>\n'.format(
                        i, p.X, p.Y, trans[0], trans[1], p.group))


        output.write(self.footer)
        output.close()

    def loadXEO(self,filename):
        '''
        From the provided xeo, parse a list of target positions
        filename: xeo file to parse
        '''
        infile = open(filename, 'r')
        lines = infile.readlines()
        result = []
        #ignore header and footer
        for l in lines[13:-12]:
            toks = l.split('"')
            pos = toks[3].split('_')
            #parse pixel position and group
            if len(pos) == 4:
                offset = 1
                x = int(pos[1+offset][:-1])
                y = int(pos[2+offset])
                s = int(pos[1][:-1])
                result.append(blob.blob(x=x, y=y, group =  s))
            else:
                offset= 0
                x = int(pos[1+offset][:-1])
                y = int(pos[2+offset])
                result.append(blob.blob(x = x, y = y))

        return result


    def getIntermediateMap(self):
        '''
        populates the intermediate map using the list of
            motor and mtp points
        '''
        result = []
        for i in range(len(self.motor)):
            result.append( (self.mtpLabel(self.mtp[i]),
                            self.motor[i][0], self.motor[i][1]))

        return result


    def loadStagePoints(self):
        '''
        loads in the intermediate map at self.motorCoordFilename
        '''
        #read in data file
        reader = open(self.motorCoordFilename, 'r')
        for l in reader.readlines():
            toks = l.split('\t')
            self.mtp.append(self.extractMTPPoint(toks[0],
                            needMTP=True));
            self.motor.append((int(toks[1]), int(toks[2])))

        #update map
        self._updateMotor2MTP()

    def setIntermediateMap(self, points):
        '''
        From the list of points, generate a new intermediate map
        points: list of (name, x, y) training points
        '''
        #parse returned points
        self.motor = []
        self.mtp = []
        writer = open(self.motorCoordFilename, 'w')
        for t in points:
            self.mtp.append(self.extractMTPPoint(t[0],
                            needMTP=True));
            self.motor.append((int(t[1]), int(t[2])))
            #save new file
            writer.write("{}\t{}\t{}\n".format(t[0], t[1], t[2]))

        writer.close()
        #update motor coordinate
        self._updateMotor2MTP()

    def _updateMotor2MTP(self):
        '''
        helper method to perform point based similarity registration
        from the motor coordinate system to the fractional distance
        '''
        self.motor2MTP = self._PBSR(self.motor, self.mtp, False)
```

CoordinateMappers/connectedInstrument.py

```python
import abc

class ConnectedInstrument(object, metaclass=abc.ABCMeta):
    """
    A abstract base class of a connected instrument which
    specifies the most basic set of functions to support for microMS
    to interface with the instrument
    """
    def __init__(self):
        '''
        Create a new connected instrument, with some important constants
        '''
        self.dwellTime = 1#seconds
        self.postAcqusitionWait = 0#seconds
        self.connected = False#has a connection been established
        super().__init__()

    @abc.abstractmethod
    def getPositionXY(self):
        '''
        Returns the current XY position of the connected instrument
        returns a tuple of (x,y)
        '''

    @abc.abstractmethod
    def moveToPositionXY(self, xypos):
        '''
        Move the stage to the specified (x,y) coordinate.
        xypos: (x,y) tuple in the instrument coordinate space
        '''

    @abc.abstractmethod
    def move(self, direction, stepSize):
        '''
        Move the stage in the specified direction.
            The direction the stage moves should match
            the probe movement.
        direction: a enumModule.Direction enum
        stepSize: enumModule.StepSize specifying
            if the step should be large
        '''

    @abc.abstractmethod
    def moveProbe(self, direction, stepSize):
        '''
        Move the probe in the specified direction.
            This may not be general  but is necessary
            for 3-axis collection
        direction: a enumModule.Direction enum
        stepSize: enumModule.StepSize specifying
            if the step should be large
        '''

    @abc.abstractmethod
    def setProbePosition(self):
        '''
        Signal the instrument that the probe
            is in the optimized position
        '''

    @abc.abstractmethod
    def getProbePosition(self):
        '''
        Get the current position of the probe
            for queries on probe position
        '''

    @abc.abstractmethod
    def collect(self):
        '''
        Perform a single collection at the current position
            for self.dwellTime (in seconds)
        '''

    @abc.abstractmethod
    def collectAll(self, positions):
        '''
        Perform sequential collections at each point specified.
        After collection, the probe should return
            to self.finalPosition() for self.postAcqusitionWait seconds.
            if postAcqusitionWait == -1 stay at finalPosition.
        positions: list of (x,y) tuples
        '''

    @abc.abstractmethod
    def initialize(self, portname):
        '''
        Begin connection with an instrument.
        portname: port to connect two
        '''

    @abc.abstractmethod
    def finalPosition(self):
        '''
        Move the probe to the final "resting" position
        '''
```

CoordinateMappers/coordinateMapper.py

```python
import abc
import numpy as np

from ImageUtilities import blob

class CoordinateMapper(object, metaclass=abc.ABCMeta):
    """
    An abstract interface of a coordinate mapper object.
    Used to translate between coordinate systems for each
        supported instrument
    """

    def __init__(self):
        self.physPoints = []
        self.pixelPoints = []
        self.update = False

        self.isConnectedToInstrument = False
        self.connectedInstrument = None
        self.reflectCoordinates = False

        '''
        also need to define the strings
        self.instrumentExtension: extension of file used by instrument
            for target positions
        self.instrumentName: Name of instrument for display and logic.
            Should be unique

        finally supportedCoordSystems needs to be updated with
            the import and supportedMappers
        '''

    @abc.abstractmethod
    def isValidEntry(self, inStr):
        '''
        checks if the user-entered coordinate is valid
        inStr: the user entry
        returns a bool, true if the entry was valid
        '''

    @abc.abstractmethod
    def extractPoint(self, inStr):
        '''
        converts the entered point to a physical coordinate
        in simplest case can just parse the input,
            could require other conversions
        inStr: user entry, should be validated prior to passing in
        returns a tuple in physical coordinates
        '''

    @abc.abstractmethod
    def predictName(self, pixelPoint):
        '''
        used to predict the user entry, is frequently overwritten when
            setting a registration point. Can be a "named" coordinate
            to match instrumentation or just the predicted
            physical coordinate
        pixelPoint: a tuple of the x,y pixel position
        returns a string with the predicted input
        '''

    @abc.abstractmethod
    def predictLabel(self, physPoint):
        '''
        given a physical position of a registration point,
            the label to draw on the image
        can be a 'named' coordinate or a short position
        '''

    @abc.abstractmethod
    def predictedPoints(self):
        '''
        returns a list of pixel points of predicted 'named' positions
        used to show the predicted grid if the instrument has preset
            positions
        Returns [] if not implemented or not enough training points set
        '''

    @abc.abstractmethod
    def loadInstrumentFile(self, filename):
        '''
        read in an instrument file produced by saveInstrumentFile
        filename: the name of the instrument file
        should return a list of blobs used in blob finding,
        radius and circularity can be hard coded as
            GUIConstants.DEFAULT_BLOB_RADIUS and 1 for display purposes
        '''

    @abc.abstractmethod
    def saveInstrumentFile(self, filename, blobs):
        '''
        write the file used by the instrument to profile each position
        typically requires special formatting.  It is also possible to
        write meta data in a separate file to simplify loading later
        filename: output file name
        blobs: list of blobs
        '''

    def saveInstrumentRegFile(self, filename):
        '''
        Similar to saveInstrumentFile, but saves the positions of the
        pixelPoints used for registration
        '''
        regBlobs = [blob.blob(p[0], p[1]) for p in self.pixelPoints]
        self.saveInstrumentFile(filename, regBlobs)

    @abc.abstractmethod
    def getIntermediateMap(self):
        '''
        return the coordinates needed to construct the intermediate map,
            where necessary the intermediate map is used to convert from
            physical positions to a coordinate system used by the
            instrument.  These are typically set points on the instrument
            which could change.
        the output should be a list of tuples which are used to
            populate a table in the GUI.
        Subclasses should save and load these points as needed,
            preferably to a txt file with the class name
        '''

    @abc.abstractmethod
    def setIntermediateMap(self, points):
        '''
        Update the intermediate map based on the points supplied.
            The user will likely update some of the points so format
            should be similar to the structure returned by
            getIntermediateMap
        points: a list of tuples
        '''

    def PBSR(self):
        '''
        calculates R, t and s for point based registration from
            pixels (x) to physical positions (y)
        '''
        (self.R, self.s, self.t) = self._PBSR(self.pixelPoints,
                                              self.physPoints,
                                              self.reflectCoordinates)

    def translate(self, pixelPoint):
        '''
        Translate a provided pixel point to physical coordinate
        pixelPoint: a x,y tuple in pixel space
        '''
        return self._translate(pixelPoint, self.reflectCoordinates)

    def invert(self, physPoint):
        '''
        Translate a provided physical point to pixel coordinate
        physPoint: a x,y tuple in physical space
        '''
        return self._invert(physPoint, self.reflectCoordinates)

    def addPoints(self, pixelPoint, physPoint):
        '''
        Adds the provided x,y tuples to the appropriate lists
        Does some type checking and signals the need for a pbsr update
        pixelPoint: (x,y) tuple in global pixel space
        physPoint: (x,y) tuple of physical coordinate
        '''
        #check if tuples of list two
        if isinstance(pixelPoint, tuple) and \
                isinstance(physPoint, tuple) and \
                len(pixelPoint) == 2 and \
                len(physPoint) == 2:

            self.physPoints.append(physPoint)
            self.pixelPoints.append(pixelPoint)
            self.update = True

    def clearPoints(self):
        '''
        Resets all physical and pixel points
        '''
        self.physPoints = []
        self.pixelPoints = []
        self.update = True

    def _translate(self, pixelPoint, reflected):
        '''
        a helper method of translate that has the extra variable
            for reflection
        pixelPoint: (x,y) tuple in global pixel space
        reflected: boolean to signal if the two coordinate
            spaces are reflected
        returns an (x,y) tuple in physical space
        '''
        #can't perform transformation
        if len(self.physPoints) < 2:
            raise KeyError('Not enough training points')

        #update if needed
        if self.update == True:
            self.PBSR()
            self.update = False

        #if reflecting, negate the y axis
        if reflected:
            result = self.s *
                    self.R *
                    np.matrix([[pixelPoint[0]],[-pixelPoint[1]]])
                    + self.t
        else:
            result = self.s *
                    self.R *
                    np.matrix([[pixelPoint[0]],[pixelPoint[1]]])
                    + self.t

        return (result[0,0], result[1,0])

    def _invert(self, physPoint, reflected):
        '''
        helper method for inverting a physical point to a pixel position
        physPoint: (x,y) coordinate of physical position
        reflected: boolean toggle to indicate if the coordinate
            spaces are reflections
        return (x,y) in pixel positions
        '''
        #not enough training points
        if len(self.physPoints) < 2:
            raise KeyError('Not enough training points')

        #update transformation as needed
        if self.update == True:
            self.PBSR()
            self.update = False

        #calculate inverse transformation
        result = np.linalg.inv(self.R)*\
            (np.matrix([[physPoint[0]],[physPoint[1]]]) - self.t)/self.s

        #negate y axis if reflected
        if reflected:
            return (result[0,0], -result[1,0])
        else:
            return (result[0,0], result[1,0])

    def _PBSR(self, X, Y, reflected = False):
        '''
        calculate R, t and s for point based registration from
        pixel (x) to  physical (y)
        X: list of tuples of pixel coordinates
        Y: list of tuples of physical coordinates
        reflected: boolean switch to indicate if the coordinates
            are related by a reflection
        returns (R, s, t)
        y ~ s*R*x+t
        '''
        flip = -1 if reflected else 1

        xbar = [0,0]
        ybar = [0,0]
        n = len(X)
        for i in range(n):
            xbar[0] += X[i][0]
            xbar[1] += flip*X[i][1]
            ybar[0] += Y[i][0]
            ybar[1] += Y[i][1]
        xbar[0] /= n
        xbar[1] /= n
        ybar[0] /= n
        ybar[1] /= n

        xtilde = list(map(lambda x:
                        (x[0]-xbar[0], (flip*x[1])-xbar[1]), X))
        ytilde = list(map(lambda x: (x[0]-ybar[0], x[1]-ybar[1]), Y))

        H = np.matrix('0 0; 0 0')

        for s,p in zip(xtilde, ytilde):
            H = H + np.outer(s,p)

        U,s,V = np.linalg.svd(H)

        R = np.dot(
                np.dot(V,
                       np.matrix([[1,0],
                                  [0,np.linalg.det(np.dot(V,U))]])),
                np.transpose(U))

        sTop = 0
        sBot = 0
        for s,p in zip(xtilde, ytilde):
            sTop += np.dot(np.dot(R,s), p)
            sBot += np.dot(s,s)
        s = sTop/sBot
        s = s[0,0]

        if s < 0:
            s = -s
            R = -R

        sbar = np.matrix([[xbar[0]],[xbar[1]]])
        pbar = np.matrix([[ybar[0]],[ybar[1]]])
        t = pbar-s*R*sbar
        return (R,s,t)

    def removeClosest(self, pixelPoint):
        '''
        remove the closest fiducial pair to the provided pixel point
        pixelPoint: (x,y) tuple to remove
        '''
        closestI = 0
        if self.pixelPoints:
            #start with distance to fist point
            closestDist = (self.pixelPoints[0][0]-pixelPoint[0])**2
                            +(self.pixelPoints[0][1]-pixelPoint[1])**2
            for i,p in enumerate(self.pixelPoints):
                #update if p is closer
                if (p[0]-pixelPoint[0])**2
                        +(p[1]-pixelPoint[1])**2 < closestDist:
                    closestDist = (p[0]-pixelPoint[0])**2
                                    +(p[1]-pixelPoint[1])**2
                    closestI = i
            #remove points and signal the need to update
            self.pixelPoints.pop(closestI)
            self.physPoints.pop(closestI)
            self.update = True


    def highestDeviation(self):
        '''
        returns the index of pixelPoints with the highest deviation
        in target registration error
        '''
        if len(self.physPoints) < 2:
            raise KeyError('Not enough training points')

        #update as needed
        if self.update == True:
            self.PBSR()
            self.update = False

        #get all predicted pixel positions
        predPixPoints = list(map(lambda x: self.invert(x),
                                            self.physPoints))
        dists = []
        #calculate deviations (dist squared)
        for i in range(len(predPixPoints)):
            dists.append((self.pixelPoints[i][0]
                              - predPixPoints[i][0])**2
                         +(self.pixelPoints[i][1]
                              - predPixPoints[i][1])**2)
        #return max deviation
        return np.argmax(dists)

    def loadRegistration(self, filename):
        '''
        load the msreg file by populating the physical and pixel lists
        filename: the msreg file to load
        '''
        #clear list
        self.physPoints = []
        self.pixelPoints = []

        infile = open(filename, 'r')

        #toss lines until hitting 'image x'
        l = infile.readline()
        while 'image x' not in l:
            l = infile.readline()

        #then get next
        l = infile.readline()

        #read while lines are not none
        while l:
            #parse out the points
            toks = l.rstrip().split('\t')
            self.pixelPoints.append((int(float(toks[0])),
                                     int(float(toks[1]))))
            self.physPoints.append((float(toks[2]), float(toks[3])))
            l = infile.readline()
        #update pbsr if possible
        if len(self.physPoints) > 2:
            self.PBSR()

    def saveRegistration(self, filename):
        '''
        save the registration file
        filename: the msreg file to write
        '''
        #update if needed and enough points
        if self.update == True and len(self.physPoints) > 2:
            self.PBSR()
            self.update = False
        output = open(filename, 'w')
        output.write(self.instrumentName + '\n')
        #write the registration transformation
        if len(self.physPoints) > 2:
            output.write("S:{}\nR:{}\nT:{}\n"
                         .format(self.s, self.R, self.t))
        #write the coordinates
        output.write("image x\timage y\tphysical coordinate\n")
        for i,s in enumerate(self.pixelPoints):
            output.write("{}\t{}\t{}\t{}\n".format(s[0], s[1],
                          self.physPoints[i][0],self.physPoints[i][1]))

        output.close()
```

CoordinateMappers/flexImagingSolarix.py

```
from CoordinateMappers.solarixMapper import solarixMapper
from ImageUtilities.blob import blob
import os

class flexImagingSolarix(solarixMapper):
    '''
    This is another implementation for the solarix which uses
    flexImaging to perform profiling, instead of autoexecute.
    Most functions are directly inherited from the solarix mapper
    '''

    def __init__(self):
        '''
        Create a new solarix mapper
        Only overwriting is the instrument extension and name
        '''
        super().__init__()
        self.instrumentExtension = '.txt'
        self.instrumentName = 'flexImagingSolarix'

    def saveInstrumentFile(self, filename, blobs):
        '''
        Save the instrument file of the provided list of blobs
        filename: the file to write to
        blobs: list of blob targets to save
        file format is a space delineated x, y, name, region
        '''
        if blobs is None or len(blobs) == 0:
            return
        output = open(filename, 'w')
        output.write('# X-pos Y-pos spot-name region\n')
        #write out the fiducial locations for registration
        for i in range(len(self.physPoints)):
            phys = self.physPoints[i]
            pix = self.pixelPoints[i]
            output.write('{0:.0f} {1:.0f} fiducial{2} 01\n'
                                .format(phys[0], -phys[1], i))

        for b in blobs:
            phys = self.translate((b.X, b.Y))
            if b.group is not None:
                output.write('{0:.0f} {1:.0f} s{4}_x{2:.0f}_y{3:.0f} 01\n'
                                .format(phys[0], -phys[1], b.X, b.Y, b.group))
            else:
                output.write('{0:.0f} {1:.0f} x{2:.0f}_y{3:.0f} 01\n'
                                .format(phys[0], -phys[1], b.X, b.Y))

        output.close()

    def saveInstrumentRegFile(self, filename):
        if self.pixelPoints is None or len(self.pixelPoints) == 0:
            return
        output = open(filename, 'w')
        output.write('# X-pos Y-pos spot-name region\n')

        for p in self.pixelPoints:
            phys = self.translate((p[0], p[1]))
            output.write('{0:.0f} {1:.0f} x{2:.0f}_y{3:.0f} 01\n'
                         .format(phys[0], -phys[1], p[0], p[1]))


    def loadInstrumentFile(self, filename):
        '''
        Loads target locations from a target file
        filename: the file to read
        returns a list of blobs
        '''
        input = open(filename, 'r')
        result = []
        for l in input.readlines()[1:]:
            toks = l.split(' ')
            toks = toks[2].split('_')
            if len(toks) == 3:
                result.append(blob(int(toks[1][1:]), int(toks[2][1:]),
                                   group = int(toks[0][1:])))
            elif len(toks) == 2:
                result.append(blob(int(toks[0][1:]), int(toks[1][1:])))
        return result
```

CoordinateMappers/oMaldiMapper.py

```
from CoordinateMappers import coordinateMapper
import os
import numpy as np
import itertools

from ImageUtilities import blob

class oMaldiMapper(coordinateMapper.CoordinateMapper):
    '''
    coordinate mapper of an ab sciex oMaldi server.
    Tries to account for slide slop in generation of instrument files
    '''

    def __init__(self):
        '''
        creates a new oMaldiMapper and sets some constant values
        '''
        super().__init__()

        self.instrumentExtension = '.ptn'
        self.instrumentName = 'oMALDI'
        self.reflectCoordinates = True

        #these are rough estimates, but shouldn't be used
            #beyond predicting labels
        self.SIMSMapY = {
            0:30400,
            10:27200,
            20:24000,
            30:20800,
            40:17600,
            50:14400,
            60:11200,
            70:8000,
            80:4800,
            90:1600
        }

        self.SIMSMapX = {
            10:30400,
            9:27200,
            8:24000,
            7:20800,
            6:17600,
            5:14400,
            4:11200,
            3:8000,
            2:4800,
            1:1600
        }

        self.allPoints = list(itertools.product(self.SIMSMapX.values(),
                                                self.SIMSMapY.values()))

    def isValidEntry(self, inStr):
        '''
        Tests if the string is a valid motor entry.
        Sample entry is two ints separated by a space
        inStr: string to test
        returns true if the string can be successfully parsed
        '''
        if " " in inStr:
            toks = inStr.split(" ")
            try:
                int(toks[0])
                int(toks[1])
                return True
            except:
                return False
        else:
            return False

    def extractPoint(self, inStr):
        '''
        Extracts a motor coordinate from the provided string.
        inStr: the string to parse
        returns an (x,y) tuple in physical coordinate space
        '''
        if not self.isValidEntry(inStr):
            return None
        toks = inStr.split(" ")
        return( (int(toks[0]), int(toks[1])) )

    def predictName(self, pixelPoint):
        '''
        Predicts the motor coordinate from a pixel position.
        pixelPoint: (x,y) tuple
        returns a blank string
        '''
        return ""

    def predictLabel(self, physicalPoint):
        '''
        Predict the label of a physical point.
        physicalPoint: (x,y) tuple of the position to predict
        returns a string of the position of a standard 100 spot plate
        '''
        Y = physicalPoint[1]
        X = physicalPoint[0]
        Ymin, Ykey = abs(Y-next(iter(self.SIMSMapY.values()))), \
            next(iter(self.SIMSMapY.keys()))
        Xmin, Xkey = abs(X-next(iter(self.SIMSMapX.values()))), \
            next(iter(self.SIMSMapX.keys()))
        #find closest position
        for key, val in self.SIMSMapY.items():
            if abs(Y-val) < Ymin:
                Ymin, Ykey = abs(Y-val), key
        for key, val in self.SIMSMapX.items():
            if abs(X-val) < Xmin:
                Xmin, Xkey = abs(X-val), key
        return Ykey+Xkey

    def predictedPoints(self):
        '''
        Returns a list of predicted points of the standard 100 spot plate
        '''
        if len(self.physPoints) < 2:
            return []
        result = []
        for p in self.allPoints:
            result.append(self.invert(p))
        return result

    def loadInstrumentFile(self, filename):
        '''
        Loads all the targets associated with a given instrument file.
        filename: the ptn file to load.  Actually gets information
            from a partner text file.
        returns a list of blobs of the target locations
        '''
        result = []
        if os.path.exists(filename[0:-4] + '.txt'):
            reader = open(filename[0:-4] + '.txt', 'r')
            for l in reader.readlines():
                toks = l.split('\t')
                if len(toks) == 3:
                    result.append(blob.blob(float(toks[0]),
                                            float(toks[1]),
                                            group = int(toks[2])))
                else:
                    result.append(blob.blob(float(toks[0]),
                                            float(toks[1])))
        else:
            print('{} containing pixel positions not found!'
                  .format(filename[0:-4] + '.txt'))
        return result

    def saveInstrumentFile(self, filename, blobs):
        '''
        Save the list of blobs to the provided filename
        filename: the base ptn filename to save
        blobs: list of blobs to save
        '''
        if blobs is None or len(blobs) == 0:
            return
        slop = 0.1
        #assuming start at spot 43:
        scale = 0.0031249999999999984;
        rot = np.matrix([[ -1.000e+00,   2.77555756e-16],
        [  2.22044605e-16,  -1.000e+00]]);
        transl = np.matrix([[ 30.],
        [-50.]]);

        points = [];

        output = open(filename[0:-4] + '.txt', 'w')
        for b in blobs:
            trans = self.translate((b.X,b.Y))
            result = scale
```

```
                    * rot
                    * np.matrix([[trans[0]],[-trans[1]]])
                    + transl
            points.append((result[0,0], result[1,0]))
            if b.group is not None:
                output.write('{0:.0f}\t{1:.0f}\t{2}\n'
                             .format(b.X, b.Y, b.group))
            else:
                output.write('{0:.0f}\t{1:.0f}\n'.format(b.X, b.Y))
        output.close()

        points = self.SlopCorrection(points, slop, slop)

        output = open(filename, 'w')
        for p in points:
            output.write('{0:.3f}\t{1:.3f}\n'.format(p[0], p[1]))
        output.close()

    def SlopCorrection(self, points, xcorr, ycorr):
        '''
        Attempts to correct for linear actuator motor slop in the stage.
        As the stage changes direction from positive to negative
            direction in either axis, a constant value is added or
            subtracted to make the stage move a little further or less
            depending on slop in drive screw.
        Assume start at spot 43, apply when changing directions
            pattern value at start is 5,5 coming in direction of 6,4
        points: list of points to visit
        xcorr: x slop correction value
        ycorr: y slop correction value
        returns a new list of points with slop correction
        '''
        path = []
        output = []
        #path contains the last "two points visited" in x and y
        #to determine what kind of slope correction, if any, to apply
        path.append((6,4))
        path.append((5,5))
        xslop = 0
        yslop = 0
        for p in points:
            path.append(p)
            #update xslop
            # +-
            if path[0][0] < path[1][0] and path[1][0] > path[2][0]:
                xslop -= xcorr
            #-+
            elif path[0][0] > path[1][0] and path[1][0] < path[2][0]:
                xslop += xcorr
            #no change, forward point
            elif path[1][0] == path[2][0]:
                path[1] = (path[0][0], path[1][1])
            #update yslop
            # +-
            if path[0][1] < path[1][1] and path[1][1] > path[2][1]:
                yslop -= ycorr
            #-+
            elif path[0][1] > path[1][1] and path[1][1] < path[2][1]:
                yslop += ycorr
            #no change, forward point
            elif path[1][1] == path[2][1]:
                path[1] = (path[1][0], path[0][1])

            output.append((p[0]+xslop,p[1]+yslop))
            path.pop(0)
        return output

    def getIntermediateMap(self):
        '''
        The intermediate map is hard coded
        '''
        return [('Not in use', 0, 0)]

    def setIntermediateMap(self, points):
        '''
        This is unused
        '''
        pass
```

CoordinateMappers/solarixMapper.py

```
from CoordinateMappers import brukerMapper
import xlsxwriter
from PyQt5 import QtCore, QtGui, QtWidgets
import os

class solarixMapper(brukerMapper.brukerMapper):
    """
    coordinate mapper for the solarix
    noticeable changes include encoding of motor coordinates,
    requirement of xls for auto acquisition, and limiting number of blobs
    """

    def __init__(self):
        '''
        initialize a new solarix mapper with some specified constants
        '''
        d, f = os.path.split(__file__)
        self.motorCoordFilename = os.path.join(d,
                                    'solarixMapperCoords.txt')
        self.instrumentExtension = '.xeo'
        self.instrumentName = 'solariX'
        super().__init__()
        self.reflectCoordinates = True

    def isValidMotorCoord(self,instr):
        '''
        Checks if the supplied string is a valid motor coordinate.
        Solarix motor coordinates are delimited by a '/'
        instr: the string to test
        returns true if the string can be successfully parsed
        '''
        if instr is None:
            return False
        if "/" in instr:
            toks = instr.split("/")
            try:
                int(toks[0])
                int(toks[1])
                return True
            except:
                return False
        else:
            return False

    def extractMotorPoint(self,inStr):
        '''
        Parse the suppled string to generate a motor point
        inStr: the string to parse
        returns an (x,y) tuple of the motor coordinate
        '''
        if not self.isValidMotorCoord(inStr):
            return None
        toks = inStr.split("/")
        return( (int(toks[0]), int(toks[1])) )
```

```python

    def loadInstrumentFile(self, filename):
        '''
        Load target locations of a given XEO
        filename: the file to load
        returns a list of blobs of the targets in the file
        '''
        return self.loadXEO(filename)

    def saveInstrumentFile(self, filename, blobs):
        '''
        saves an instrument file of the provided list of blobs
        filename: the file to write to
            if more than 900 points, uses filename as a base name
        blobs: list of target positions
        '''
        if blobs is None or len(blobs) == 0:
            return
        maxPoints = 400
        if len(blobs) > maxPoints:
            fn = filename[:-4]
            for i in range(len(blobs) // maxPoints):
                self.writeXEO(fn + '_' + str(i) + '.xeo',
                                blobs[i*maxPoints:(i+1)*maxPoints])
                self.writeAutoXlsx(fn + '_' + str(i) + '.xlsx',
                                blobs[i*maxPoints:(i+1)*maxPoints])
            #get the remainder
            self.writeXEO(fn + '_' + str(len(blobs)//maxPoints) + '.xeo',
                            blobs[-(len(blobs) % maxPoints):])
            self.writeAutoXlsx(fn + '_' + str(len(blobs)//maxPoints)
                                + '.xlsx',
                                blobs[-(len(blobs) % maxPoints):])

        else:#write a single xeo
            self.writeXEO(filename,blobs)
            filename = filename[:-3] + 'xlsx'
            self.writeAutoXlsx(filename, blobs)

    def writeAutoXlsx(self, filename, blobs):
        '''
        Write the xlsx file required for autoexecute
        filename: the xlsx name
        blobs: list of blobs to save
        '''
        workbook =  xlsxwriter.Workbook(filename)
        ws = workbook.add_worksheet()

        header = ['Spot Number', 'Chip Number', 'Data Directory',
                    'Data File Name', 'Method Name', 'Sample Name',
                    'Comment']
        for i,h in enumerate(header):
            ws.write(0, i, h)

        for i,p in enumerate(blobs):
            ws.write(i+1, 0, "x_{0:.0f}y_{1:.0f}".format(p.X, p.Y))
            ws.write(i+1, 1, "0")
            ws.write(i+1, 5, "x_{0:.0f}y_{1:.0f}".format(p.X, p.Y))

        workbook.close()

    def predictName(self, pixelPoint):
        '''
        predict the motor coordinate or name from the pixel point
        Tries to read the coordinate from the clipboard of a QT GUI
        pixelPoint: (x,y) tuple of global pixel space
        returns the predicted string
        '''
        clipboard = QtWidgets.QApplication.clipboard()
        if clipboard is not None and \
            clipboard.text() is not None and \
            clipboard.text() != '':
            return clipboard.text()
        return super().predictName(pixelPoint)
```

CoordinateMappers/supportedCoordSystems.py

```
'''
Contains all the supported coordinate systems and a list of
instances of each type.
'''

###add new import here
from CoordinateMappers import ultraflexMapper
from CoordinateMappers import solarixMapper
from CoordinateMappers import oMaldiMapper
from CoordinateMappers import zaberMapper
from CoordinateMappers import flexImagingSolarix

###add new mapper instance here
supportedMappers = [ultraflexMapper.ultraflexMapper(),
                    solarixMapper.solarixMapper(),
                    flexImagingSolarix.flexImagingSolarix(),
                    oMaldiMapper.oMaldiMapper(),
                    zaberMapper.zaberMapper()]


#check for defined names here
supportedNames = list(map(lambda x: x.instrumentName, supportedMappers))
list(map(lambda x: x.instrumentExtension, supportedMappers))
```

CoordinateMappers/ultraflexMapper.py

```
from CoordinateMappers import brukerMapper
import os

class ultraflexMapper(brukerMapper.brukerMapper):
    """
    coordinate mapper for the ultrafleXtreme
    """

    def __init__(self):
       '''
       set up a new ultraflex mapper and set some constants
       '''
       #the intermediate map coordinates
       d, f = os.path.split(__file__)
       self.motorCoordFilename = os.path.join(d,
                                       'ultraflexMapperCoords.txt')
       self.instrumentExtension = '.xeo'
       self.instrumentName = 'ultrafleXtreme'
       super().__init__()
       self.reflectCoordinates = True


    def isValidMotorCoord(self, instr):
        '''
        Test if the supplied string is a valid coordinate.
        Valid strings are separated by a space and contain two ints
        instr: string to test
        returns true if the string is able to be parsed
        '''
        if instr is None:
            return False
        if " " in instr:
            toks = instr.split(" ")
            try:
                int(toks[0])
                int(toks[1])
                return True
            except:
                return False
        else:
            return False

    def extractMotorPoint(self,inStr):
        '''
        Parses the string to generate a motor coordinate
        inStr: the string to parse
        returns an (x,y) tuple if the string successfully parses
        '''
        if not self.isValidMotorCoord(inStr):
            return None
        toks = inStr.split(" ")
        return( (int(toks[0]), int(toks[1])) )

    def loadInstrumentFile(self, filename):
        '''
        Loads an xeo file and returns a list of blobs.
        filename: the xeo file to read
        returns a list of blobs representing the target coordinates
        '''
        return self.loadXEO(filename)

    def saveInstrumentFile(self, filename, blobs):
        '''
        Save the provided list of blobs as an xeo file
        filename: the file to write to
        blobs: list of target blobs
        '''
        self.writeXEO(filename, blobs)
```

CoordinateMappers/zaber3axis.py

```
from CoordinateMappers import zaberInterface
from CoordinateMappers import connectedInstrument
from ImageUtilities.enumModule import Direction, StepSize
import time

class Zaber3Axis(zaberInterface.ZaberInterface,
                 connectedInstrument.ConnectedInstrument):
    '''
    A connected zaber linear stage with XYZ axes.
    Note the multiple inheritance
    '''

    def __init__(self):
        '''
        Setup, but not connect the stage
        Initializes a few constants that may need changing
            and some instance variables
        '''
        super().__init__()
        self.xdev = 2
        self.ydev = 1
        self.zdev = 3

        self.smallStep = 500 #microsteps

        self.smallZstep = 50 #microsteps

        self.mediumFactor = 10
        self.largeFactor = 100
        self.giantFactor = 1000

        #the position of the z axis when the probe is at the surface
        self.bottomPosition = 0;
        #true if the probe is at the surface, else false
        self.atBottom = False


    def initialize(self, portName, timeout = None):
        '''
        Attempt to connect to the specified port and initialize stage
        portName: the port to connect to
        timeout: how long to listen to the port, None for blocking calls
        '''
        try:
            self._openPort(portName,timeout)
        except:
            self.connected = False
            return
        #set the connection to true
        self.connected = True
        #renumber to make sure all ids are unique and as expected
        self.renumber()
        #clear replies from renumber
        self.checkReplies(3)
        #home all stages
        self.homeAll()
        #move xyz to 100,000 to engage each stage
        self.moveToPositionXY((1, 1))
        self._send(self.zdev, self.COMMANDS['MOVE_ABS'], 1)
        self._receive()

    def homeAll(self):
        '''
        home all stages and clear replies.
        Homes the z axis first to help protect the probe
        '''
        if not self.connected:
            return
        self.home(3)
        self._receive()
        self.home(2)
        self.home(1)
        self._receive()
        self._receive()
        self.atBottom = False

    def checkReplies(self, numreads):
        '''
        Performs multiple receive calls and checks for rejections
        numreads: the number of reads to perform
        returns true if no errors or rejections occurred
        '''
        if not self.connected:
            return
        result = True
        for i in range(numreads):
            (device, command, data) = self._receive()
            if command == 255:
                print('Rejected command')
                result = False

        return result

    def getPositionXY(self):
        '''
        get the current x,y position
        returns (x,y) in stage coordinates
        '''
        if not self.connected:
            return None
        x = self.getPosition(self.xdev)
        y = self.getPosition(self.ydev)
        return (x,y)

    def moveToPositionXY(self,  xypos):
        '''
        Move the stage to the specified xy position.  A blocking call.
        Will also retract the probe if it is at the sample surface.
        xypos: (x,y) tuple to move to
        '''
        if not self.connected:
            return
        if self.atBottom:
            self.toggleProbe()
        x,y = xypos
        self._send(self.xdev, self.COMMANDS['MOVE_ABS'], x)
        self._send(self.ydev, self.COMMANDS['MOVE_ABS'], y)
        self._receive()
        self._receive()

    def move(self, direction, stepSize):
        '''
        performs a relative move in the specified direction
        direction: a enumModule.Direction enum
        stepSize: enumModule.StepSize specifying the step
        '''
        #if not connected, do nothing
        if not self.connected:
            return
        #retract probe if it's at the surface
        if self.atBottom:
            self.toggleProbe()

        #calculate the steps to perform
        step = self.smallStep
        if stepSize == StepSize.large:
            step *= self.largeFactor
        elif stepSize == StepSize.medium:
            step *= self.mediumFactor

        #change device for each direction
        if direction == Direction.left or \
            direction == Direction.right:
            dev = self.xdev
        elif direction == Direction.down or \
            direction == Direction.up:
            dev = self.ydev
        else:
            raise ValueError('Invalid direction')
        #this is inverted relative to the stage,
        #but correct relative to the probe
        #retract for these directions
        if direction == Direction.down or \
            direction == Direction.right:
            step = -1 * step

        #blocking call
        self._send(dev, self.COMMANDS['MOVE_REL'], step)
        self._receive()

    def _collect(self, position):
        '''
        Perform a single collection at the specified position
        position: (x,y) of stage coordinate to sample
        '''
        #do nothing if not connected
        if not self.connected:
            return

```

```python
        #move the probe into place (is blocking)
        self.moveToPositionXY(position)
        #collect at the current position
        self.collect(finish = False)

    def collect(self, finish = True):
        '''
        Collect at the current position for self.dwellTime
        finish: call self.finishCollection at end of collection
        '''
        #do nothing if not connected
        if not self.connected:
            return
        #if probe is not at the bottom
        if not self.atBottom:
            self.toggleProbe()#lower, otherwise do nothing
        #wait for dwellTime
        time.sleep(self.dwellTime)
        self.toggleProbe()#raise
        if finish == True:
            self.finishCollection(forceHome = False)

    def collectAll(self, positions):
        '''
        Collect from each position specified
        positions: a list of (x,y) coordinates in motor positions
        '''
        #do nothing if not connected
        if not self.connected or self.bottomPosition == 0:
            return
        #start by homing all
        self.homeAll()
        #collected from each position
        for i, p in enumerate(positions):
            print("Collecting from sample {}".format(i+1))
            self._collect(p)
        print("Finished collection")
        self.finishCollection(forceHome = True)

    def finishCollection(self, forceHome):
        #if self.postAcqusitionWait is not 0,
        #have to move to final position
        if self.postAcqusitionWait != 0:
            self.finalPosition()
            if self.postAcqusitionWait != -1:
                time.sleep(self.postAcqusitionWait)
                self.homeAll()
        elif forceHome == True:
            #finish homing all
            self.homeAll()

    def moveProbe(self, direction, stepSize):
        '''
        Move the probe relative to the current position
        direction: a valid connectedInstrument.Direction
        stepSize: enum for step size
        '''
        #do nothing without a connection
        if not self.connected:
            return
        #find step size
        step = self.smallZstep
        if stepSize == StepSize.medium:
            step *= self.mediumFactor
        elif stepSize == StepSize.large:
            step *= self.largeFactor
        elif stepSize == StepSize.giant:
            step *= self.giantFactor
        if direction == Direction.up:
            step = -step
        self._send(self.zdev, self.COMMANDS['MOVE_REL'], step)
        self._receive()
        #regardless of position, the probe is no longer at the bottom
        self.atBottom = False

    def setProbePosition(self):
        '''
        set the probe position as at the surface
        '''
        if not self.connected:
            return
        #store the current position
        self.bottomPosition = self.getPosition(self.zdev)
        #probe is now at the bottom
        self.atBottom = True
        #automatically retract
        self.toggleProbe()

    def getProbePosition(self):
        if not self.connected:
            return None
        return self.getPosition(self.zdev)

    def toggleProbe(self):
        '''
        toggle the current probe position.
            If at bottom, raise, else lower
        '''
        if self.bottomPosition == 0 or not self.connected:
            return
        if self.atBottom:
            pos = self.bottomPosition
                    - self.smallZstep * self.largeFactor*5
            pos = 0 if pos < 0 else pos
        else:
            pos = self.bottomPosition

        self._send(self.zdev, self.COMMANDS['MOVE_ABS'], pos)
        self._receive()
        self.atBottom = not self.atBottom

    def finalPosition(self):
        '''
        Move the probe to the washing position,
            which is 10000 above the slide position
        '''
        if self.bottomPosition == 0 or not self.connected:
            return
        self.homeAll()
        pos = self.bottomPosition - 10000#NOTE CONSTANT VALUE

        self._send(self.zdev, self.COMMANDS['MOVE_ABS'], pos)
        self._receive()
```

CoordinateMappers/zaberInterface.py

```python
import serial, struct, abc

class ZaberInterface(object, metaclass=abc.ABCMeta):
    '''
    An abstract base class for interacting with zaber linear stages
    Encodes methods for basic IO with a stage
    '''
    def __init__(self):
        #the serial object to talk to
        self.stage = None

        #standard commands, more at
        #http://www.zaber.com/wiki/Manuals/
                #Binary_Protocol_Manual#Quick_Command_Reference
        self.COMMANDS = {
            'HOME'      :   1,
            'RENUMBER'  :   2,
            'MOVE_ABS'  :   20,
            'MOVE_REL'  :   21,
            'CUR_POS'   :   60
            }
        super().__init__()

    '''
    home and renumber don't receive
    as they don't know the number of connected devices
    '''
    def home(self, device = 0):
        '''
        home the specified device or all
        device: the device to home
        '''
        self._send(device, self.COMMANDS['HOME'])

    def renumber(self):
        '''
        renumber all devices
        '''
        self._send(0, self.COMMANDS['RENUMBER'])

    def getPosition(self, device):
        '''
        provides the current location of the requested device
        device: the device to query
        '''
        self._send(device, self.COMMANDS['CUR_POS'])
        (deviceOut, command, data) = self._receive()
        #sometimes the receives can get misaligned
        if device == deviceOut:
            return data
        else:
            return -1

    def _openPort(self, portName, timeout=None):
        '''
        begin communication with a serial stage
        portName: the name of the port to communicate with
        timeout: the time to wait for a reply.
            Set to None for blocking calls
        '''
        try:
            self.stage = serial.Serial(portName, 9600, 8,
                                        'N', 1, timeout=timeout)
        except Exception as ext:
            print(ext)
            print("Error initializing {}!".format(portName))
            raise ValueError("stage not initialized!")

    def _send(self, device, command, data=0):
        '''
        send a packet using the specified device number,
            command number, and data
        The data argument is optional and defaults to zero
        device: the id of the connected device
        command: a command, using the dictionary in init
        data: the optional data to send as well
        '''
        if self.stage == None:
            raise ValueError("stage not initialized!")
        data = int(data)
        packet = struct.pack('<BBl', device, command, data)
        self.stage.write(packet)

    def _receive(self):
        '''
        reads the serial port
        there must be 6 bytes to receive (no error checking)
        returns the (device, command, data)
        '''
        if self.stage == None:
            raise ValueError("stage not initialized!")
        r = [0,0,0,0,0,0]
        for i in range (6):
            r[i] = ord(self.stage.read(1))

        data =  (256.0**3.0*r[5])
              + (256.0**2.0*r[4])
              + (256.0*r[3])
              + (r[2])
        if r[5] > 127:
            data -= 256.0**4

        device = r[0]
        command = r[1]

        return (device, command, data)

    def findPorts(self):
        '''
        Query each COM port for a possible connection.
        Will work for windows only
        '''
        result = []
        for i in range(256):
            try:
                name = 'COM{}'.format(i)
                s = serial.Serial(name)
                result.append(name)
                s.close()
            except serial.SerialException:
                pass
        return result
```

CoordinateMappers/zaberMapper.py

```
from CoordinateMappers import coordinateMapper
from CoordinateMappers import zaber3axis

from ImageUtilities import blob

class zaberMapper(coordinateMapper.CoordinateMapper):
    '''
    A coordinate mapper of the zaber XYZ stage.
    Has a connected instrument, but otherwise the coordinate
    mapping is fairly simple.
    '''

    def __init__(self):
        '''
        Set up a new instance of zaberMapper
        '''
        super().__init__()
        #note there is a connected instrument
        self.isConnectedToInstrument = True
        self.instrumentExtension = '.txt'
        self.instrumentName = 'Zaber LMJ'
        self.reflectCoordinates = False
        #set up the instrument as a 3axis zaber stage
        self.connectedInstrument = zaber3axis.Zaber3Axis()

    def isValidEntry(self, inStr):
        '''
        Validate the possible coordinate
        inStr: the string to test, expects two floats separated
            by a space
        returns true if extract point will successfully parse
        '''
        if " " in inStr:
            toks = inStr.split(" ")
            try:
                float(toks[0])
                float(toks[1])
                return True
            except:
                return False
        else:
            return False

    def extractPoint(self, inStr):
        '''
        Parse the physical coordinate from the provided string
        inStr: the input string
        returns an (x,y) tuple of the physical coordinate,
            or None if string is not valid
        '''
        if not self.isValidEntry(inStr):
            return None
        toks = inStr.split(" ")
        return( (float(toks[0]), float(toks[1])) )

    def predictName(self, pixelPoint):
        '''
        Predicts the physical location from the pixel position.
        When the instrument is connected, reads in the
            actual physical point
        pixelPoint: (x,y) tuple in global coordinate space
        '''
        #read position if instrument is initialized
        if self.connectedInstrument.connected:
            xy = self.connectedInstrument.getPositionXY()
            return '{} {}'.format(xy[0], xy[1])
        #else return blank string
        return ''

    def predictLabel(self, physPoint):
        '''
        Predict the label of a registration point based on
            the physical location. Since there are no set, named points
            for the stage this always returns a blank string
        physPoint: (x,y) tuple in physical coordinate space
        '''
        return ''

    def predictedPoints(self):
        '''
        Returns a list of predicted, set points in the pixel
            coordinate space. In this particular implementation,
            only returns the current position of the probe when the
            instrument is connected and enough training points
            are provided.
        '''
        if len(self.physPoints) < 2 or
            not self.connectedInstrument.connected:
            return []
        else:
            phys = self.connectedInstrument.getPositionXY()
            return [self.invert(phys)]

    def loadInstrumentFile(self, filename):
        '''
        Loads a zaberMapper instrument file and returns a list of blobs
        with the target locations.
        filename: the file to load
        returns a list of blob objects
        '''
        result = []
        reader = open(filename, 'r')

        for l in reader.readlines():
            toks = l.split('\t')
            if len(toks) == 3:
                #group is encoded
                result.append(blob.blob(float(toks[0]), float(toks[1]),
                                        group = int(toks[2])))
            else:
                #no group
                result.append(blob.blob(float(toks[0]), float(toks[1])))
```

```python

        return result

    def saveInstrumentFile(self, filename, blobs):
        '''
        Save the list of target locations as an instrument file
        filename: the file to save
        blobs: the list of target blob locations
        '''
        if blobs is None or len(blobs) == 0:
            return
        output = open(filename, 'w')
        for p in blobs:
            if p.group is not None:
                output.write('{0:.0f}\t{1:.0f}\t{2}\n'
                                .format(p.X, p.Y, p.group))
            else:
                output.write('{0:.0f}\t{1:.0f}\n'.format(p.X, p.Y))
        output.close()

    def getIntermediateMap(self):
        '''
        This is ignored as no intermediate map is required
        '''
        return [('Not in use', 0, 0)]

    def setIntermediateMap(self, points):
        '''
        This is ignored as no intermediate map is required
        '''
        pass
```

Single Cell Profiling on the C_{60} SIMS

Motivation, Overview and Extensions

The lab-built, hybrid C_{60} SIMS/MALDI mass spectrometer is a prototype instrument for performing SIMS analysis on biological samples. Utilizing the cluster ion beam facilitates desorption/ionization of intact molecules < 1000 Da. With the Q-Star mass analyzer, tandem MS could provide structural information on unknown constituents. Single cell profiling with SIMS would complement the established MALDI-MS analysis by providing spectral information on low-mass compounds with minimal sample damage. However, several limitations in instrumentation prevented direct application of existing workflows for optically-guided MS. Spectra could not be acquired at specific, discrete locations and stored into separate data files. Instead, a "chromatogram" was acquired during the entire experiment. To parse the continuous stream of spectra, the time and location of the sample stage had to be monitored. This was accomplished with an Arduino microcontroller which output time and position to a separate computer. The board could further regulate the primary ion beam by detecting stage motion and modulating the gate voltage accordingly. The file containing stage information was utilized along with the mass spectra to analyze spectra from single cells. This section presents the Arduino and Matlab code for performing these experiments. The Matlab scripts should be run in order of ConvertAllWifftoMZ, LoadData, DataCleanup, RemoveSplits. Afterwards, the data variable will contain *m/z* and intensity values suitable for a variety of multivariate analyses. Five helper functions for LoadData are also included. Additional details may be found in Chapter 7. Clearly improvements in usability are possible, but would require finer control of the mass analyzer or sample stage including hardware modifications, which were avoided.

SIMS_linear_encoder_tracker.ino

```
//keep track of current count of steps
volatile long countAB = 0;
volatile long countCD = 0;
//previous value of count
volatile long countABp = 0;
volatile long countCDp = 0;
//additional variables for times
volatile unsigned long timep, time, etime, times, lastMoveTime;
//the current state of each encoder
boolean A, B, C, D;
//state of MS and if acquisition has started
boolean I, started;
//State of encoders and their previous values
byte stateAB, stateABp, indexAB, stateCD, stateCDp, indexCD;
//convert state to a step movement
volatile int QEM[16]={0,-1,0,1,1,0,-1,0,0,1,0,-1,-1,0,1,0};
//control when the bean is turned on and off
long beamStart = 30; //microseconds
long beamStop = 40; //microseconds

void setup()
{
  //start serial communication at 9600 baud
  Serial.begin(9600);
  //signal recording computer
  Serial.println("CLEARDATA");
  //header of output file
  Serial.println("LABEL,Time,t,AB,CD,C60status");
  pinMode(19,INPUT);//Mass spec state
  //handle change in instrument state
  attachInterrupt(4,Trigger,CHANGE);
  I = digitalRead(19);
  //check if instrument is already reading
  if ((I==HIGH))
    started = 1;
  else
    started = 0;
  //initialize input pins for encoder and handle changes
  pinMode(2, INPUT);//Channel A of encoder 1 blue
  pinMode(3, INPUT);//Channel B of encoder 1 black
  pinMode(21, INPUT);//Channel A of encoder 2 purple
  pinMode(20, INPUT);//Channel B of encoder 2 green
  attachInterrupt(0,ABchange,CHANGE);
  attachInterrupt(1,ABchange,CHANGE);
  attachInterrupt(2,CDchange,CHANGE);
  attachInterrupt(3,CDchange,CHANGE);

  pinMode(13,OUTPUT);//LED PIN and beam control
  digitalWrite(13,HIGH);//send 5V to Relay, beam OFF
  timep = micros(); //set initial time
  lastMoveTime = micros();
  //read the initial value of A,B,C,D encoders
  A = digitalRead(2);//Y axis
  B = digitalRead(3);
  C = digitalRead(21);//X axis
  D = digitalRead(20);
  //set initial state value
  if ((A==HIGH)&&(B==HIGH)) stateABp = 0;
  if ((A==HIGH)&&(B==LOW)) stateABp = 1;
  if ((A==LOW)&&(B==LOW)) stateABp = 2;
  if ((A==LOW)&&(B=HIGH)) stateABp = 3;

  if ((C==HIGH)&&(D==HIGH)) stateCDp = 0;
  if ((C==HIGH)&&(D==LOW)) stateCDp = 1;
  if ((C==LOW)&&(D==LOW)) stateCDp = 2;
  if ((C==LOW)&&(D=HIGH)) stateCDp = 3;
  }

//main control loop
void loop()
{
  I = digitalRead(19);
  //only perform checks if MS is acquiring
  if ((I==HIGH))
  {
      //current time
      time = micros();
      //elapsed time
      etime =  time - timep;
      //update if > 100 microseconds elapsed
      if (etime > 1) //0.1s
        {
          //print time delay (time - start time)
          Serial.print("DATA,TIME,");
          Serial.print(time-times);
          Serial.print(",");
          //print counts of encoders
          Serial.print(countAB);
          Serial.print(",");
          Serial.print(countCD);
          Serial.print(",");
          //print if the stage is moving
          Serial.println((countAB == countABp) &&
                                  (countCD == countCDp) ?
                                  "stopped" : "moving");
          //record time of previous update
          timep = time;

          //if the stage is stopped (no change from previous values)
          if ((countAB == countABp) && (countCD == countCDp))
            {
              //determine how long since last movement
              long tempTime = micros() - lastMoveTime;
              //if within the beam start and stop times, turn on beam
              if (tempTime > beamStart && tempTime < beamStop)
                  digitalWrite(13, LOW);
              else
                  digitalWrite(13, HIGH);
            }
          //beam off while moving
          else
            {
              digitalWrite(13, HIGH);
              //update last move time
              lastMoveTime = micros();
            }
          //record previous counts
          countABp = countAB;
          countCDp = countCD;

  }

}
}

//handle state change on the MS status
void Trigger ()
{
  I = digitalRead(19);
  //if analysis has not started
  if ((started==0)&&(I==HIGH))
  {
    //record start time and initialize counts
    times = micros();
    started=1;
    countAB=0;
    countCD=0;
    noInterrupts ();
  }
}

//handle Y encoder change
void ABchange()
{
  A = digitalRead(2);
  B = digitalRead(3);
  //determine state value
  if ((A==HIGH)&&(B==HIGH)) stateAB = 0;
  if ((A==HIGH)&&(B==LOW)) stateAB = 1;
  if ((A==LOW)&&(B==LOW)) stateAB = 2;
  if ((A==LOW)&&(B==HIGH)) stateAB = 3;
  indexAB = 4*stateAB + stateABp;
  byte oldSREG = SREG;   // remember if interrupts are on or off
  noInterrupts ();
  //update count based on state
  countAB = countAB + QEM[indexAB];
  SREG = oldSREG;    // turn interrupts back on, if they were on
  //record previous state
  stateABp = stateAB;
}

void CDchange()
{
  C = digitalRead(21);
  D = digitalRead(20);
  //determine state value
  if ((C==HIGH)&&(D==HIGH)) stateCD = 0;
  if ((C==HIGH)&&(D==LOW)) stateCD = 1;
  if ((C==LOW)&&(D==LOW)) stateCD = 2;
  if ((C==LOW)&&(D==HIGH)) stateCD = 3;
  indexCD = 4*stateCD + stateCDp;
  byte oldSREG = SREG;   // remember if interrupts are on or off
  noInterrupts ();
  //update count based on state
  countCD = countCD + QEM[indexCD];
  SREG = oldSREG;    // turn interrupts back on, if they were on
  //record previous state
  stateCDp = stateCD;
}
```

ConvertAllWifftoMZ.m

```
function [] = ConvertAllWifftoMZ(dirpath, ptrn)
    %%Generates mzXML files for each wiff in dirpath
    %dirpath: string of full path to directory containing files
    %ptrn: string for regex matching to export just a subset

    t = cd(dirpath);

    files = dir([ptrn '*.wiff']);

    for i = 1:length(files)
        filename = files(i).name;
        %only convert if the mzXML does not exist.  Assumes -Sample 1 is
        %appended to the file
        if exist([filename(1:end-5) '-Sample 1.mzXML'],'file') ~= 2
            %call msconvert from proteowizard. also must add binary
            %to path variable in windows!
            system(['msconvert ' filename ' --mzXML'])
        end
    end
    cd(t);
end
```

LoadData.m

```
clear

%name of directory containing wiffs (and therefore mzxmls)
wiffLocation = 'wiff';
%name of directory containing positions (xlsx files)
posLocation = 'arduino';
%the dwell time on the stage
dwellTime = 6; %seconds
%mz range to resample to
mzrange = [100 850];
%number of mz values for binning
resampleN = 10000;

parentDirs = {'../../SIMS/20160524_DRG_TRIPCHCA/',...
        %list additional dirs, relative to current path
    };
mzxmls = {'slide15b-Sample 1.mzXML',...
        %list additional files here, relative to parentDirs
    };
%directory with cell locations (cell find files from microMS)
locDir = '../../SIMS/20160524_mixCHCA/images';
%specific cell find files relative to locDir
celllocs = {'slide15b/slide15bcells.txt',...
        %list additional files here
    };

for i = 1:length(mzxmls)
    %should be -sample, get first token
    %will break if '-' is in file name
    t = strsplit(mzxmls{i}, '-');
    %record name and directory
    data(i).name = t{1};
    data(i).dirname = parentDirs{i};

    %read location file. the 8 is for header lines and is 9 in current
    %version of microMS
    t = dlmread(fullfile(locDir, celllocs{i}), '\t', 8,0);

    %read in arduino-generated file
    t = xlsread(fullfile(parentDirs{i}, posLocation,
            [data(i).name '.xlsx']));

    %correct time for overflow on long experiments
    if sum(diff(t(:,2)) < 0) == 1
        ind = find(diff(t(:,2)) < 0)+1;
        t(ind:end,2) = t(ind:end,2) + t(ind-1,2);
    elseif sum(diff(t(:,2)) < 0) > 1
        error('too many overflows!')
    end

    %correct for slight time shift between wall time and arduino time
    lm = fitlm(t(:,2), t(:,1)*24*60*60, 'linear');
    startT = t(1,2);
    t(:,2) = predict(lm, t(:,2));
    t(:,2) = t(:,2) - t(1,2)+startT/1e6;

    %record corrected time and x/y positions
    data(i).pos = t(:,2:4);

    %determine when stopping occurred with helper function
    data(i).stops = detectStopping(data(i).pos(:,2:3));

    %parse stop times from stops and dwell times
    [data(i).stepStart, data(i).stepEnd] = ...
        parseStopTimes(data(i).stops, data(i).pos(:,1), dwellTime);

    %read in mzxml
    t = mzxmlread(fullfile(parentDirs{i}, wiffLocation,mzxmls{i}), ...
        'Verbose', false);

    %parse time from string in retentionTime
    data(i).MStime = cellfun(@(a) str2double(a(3:end-1)), ...
        {t.scan(:).retentionTime});

    %get 184 and 760 intensity
    cur184 = arrayfun(@(a) extractIntens(184.09, a.peaks.mz), t.scan);
    cur760 = arrayfun(@(a) extractIntens(760.6, a.peaks.mz), t.scan);

    %determine which scans contain MS for a single cell using findInd
    data(i).MSscans = arrayfun(@(a,b) ...
        findInd(a, b, data(i).MStime, cur184), ...
        data(i).stepStart, data(i).stepEnd);
    data(i).MSscans760 = arrayfun(@(a,b) ...
        findInd(a, b, data(i).MStime, cur760), ...
        data(i).stepStart, data(i).stepEnd);

    %report number of 'missing' scans before removing
    sum(data(i).MSscans == 0)
    data(i).MSscans = data(i).MSscans(data(i).MSscans ~= 0);

    %resample just the scans containing data as separate temp variables
    data(i).mzs = linspace(mzrange(1), mzrange(2), resampleN);
    t3 = arrayfun(@(a) ...
        parseAndResample(a.peaks.mz, resampleN, mzrange), ...
        t.scan(data(i).MSscans), 'UniformOutput', false);
    t2 = arrayfun(@(a) ...
        parseAndResample(a.peaks.mz, resampleN, mzrange), ...
        t.scan(data(i).MSscans760), 'UniformOutput', false);

    %record intensity matrices for 184 and 760
    data(i).intens = vertcat(t3{:});
    data(i).intens760 = vertcat(t2{:});
    data(i).sumIntens = (data(i).intens + data(i).intens760);

end
%clear temp variables before saving
clear i t posLocation wiffLocation mzxmls tic t2 t3 t4 t5 ans
save loadedData184_760
```

detectStopping.m

```
function [ stopped ] = detectStopping( positions )
    %takes a nx2 matrix of x,y positions
    %returns true if there was a stop
    %output(1) is always false
    dists = diff(positions);
    stopped = [dists(:,1) == 0 & dists(:,2) == 0;0];
end
```

parseStopTimes.m

```
function [ startTimes, endTimes ] = ...
    parseStopTimes( stopped, time, dwellTime )
    %takes the boolean array stopped, the time in seconds,
    %and nominal dwell time and determines the start and end times
    %of analysis

    %good way to find when state changes in boolean array
    steps = [0; diff(stopped)];
    startTimes = time(steps==1);
    endTimes = time(steps==-1);

    %remove first stop if < first start (stage stopped at beginning)
    if startTimes(1) > endTimes(1)
        endTimes(1) = [];
    end

    %remove points larger than end times, for stage stopping at end
    if length(startTimes) > length(endTimes)
        startTimes(length(endTimes):end) = [];
    end

    %remove stops < 1/2 dwell or > dwell + 1
    durations = endTimes - startTimes;

    startTimes(durations < dwellTime / 2 | ...
                durations > dwellTime +1) = [];
    endTimes(durations < dwellTime / 2 | ...
                durations > dwellTime +1) = [];

end
```

extractIntens.m

```
function [ intens ] = extractIntens( mzval, input )
    %given a target mzval and input from readmzxml, return intensity
    %closest to mzval.  A helper function for vectorizing processing

    %separate intercalated data
    intens = input(2:2:end);
    mzs = input(1:2:end);

    %find closest mz value and return intens
    [~, ind] = min(abs(mzval - mzs));
    intens = intens(ind);
end
```

findInd.m

```
function [ ind ] = findInd( start, stop, times, tic)
    %helper function for vectorizing processing
    %given a start and stop time along with time array and
    %tic of the ion of interest, returns which index had max intens

    %find difference between start and time
    dif = times-start;

    %if no times are after start time, return 0
    if isempty(min(dif(dif >= 0)))
        ind = 0;
    else
        %find start index, smallest dif > 0
        indStart = find(dif == min(dif(dif >= 0)));
        %same thing for stop index
        dif = times - stop;
        if isempty(max(dif(dif <= 0)))
            ind = 0;
        else
            indEnd = find(dif == max(dif(dif <= 0)));
            %find max index between start and end
            [~, ind] = max(tic(indStart:indEnd));
            %offset by indstart so it matches the whole time series
            ind = ind + indStart - 1;
        end
    end

end

```

parseAndResample.m

```
function [ intens, mzsout ] = parseAndResample( inputmz, N, mzrange )
    %helper function for resampling data from readmzxml
    %given input data, number of mz values, and mz range
    %returns the resampled intensity and output mz values

    %separate mz and intens from intercalated data
    mzs = inputmz(1:2:end);
    intens = inputmz(2:2:end);

    %calculate linearly spaced mz values in range
    mzsout = linspace(mzrange(1), mzrange(2), N);
    %determine bin size
    bin = (mzsout(2) - mzsout(1))/2;

    %calculate intensity as sum intensity within bin width
    intens = arrayfun(@(a) ...
        sum(intens(mzs < a + bin & mzs > a - bin)),...
        mzsout);

end
```

DataCleanup.m

```matlab
clear
load loadedData184_760

%remove end position which corresponds to the stage moving back to start
for i = 1:length(data)
    data(i).stepStart(end) = [];
    data(i).stepEnd(end) = [];
    data(i).MSscans(end) = [];
    data(i).MSscans760(end) = [];
    data(i).intens(end,:) = [];
    data(i).intens760(end,:) = [];
    data(i).sumIntens(end,:) = [];
end

%any additional modifications go here. When two cells are too close
%the stage doesn't move enough to count as a separate event
i = 1;
data(i).celllocs(59:60,:) = [];
data(i).celllocs(74:75,:) = [];
data(i).celllocs(321,:) = [];

%this code can be uncommented and run with F9 to find discrepancies
%between the MS and movement.

% i = 9; len = 15;
% figure
% for j = 15:5:400
%     subplot(2,1,1)
%     plot(data(i).celllocs(j:j+len,1), data(i).celllocs(j:j+len,2),'o-')
%     for k = j:j+len
%         text(data(i).celllocs(k,1), data(i).celllocs(k,2), num2str(k));
%     end
%     tt = data(i).pos(:,1) > data(i).stepStart(j) & ...
%        data(i).pos(:,1) < data(i).stepEnd(j);
%     for k = j+1:j+len
%         tt = tt | (data(i).pos(:,1) > data(i).stepStart(k) & ...
%        data(i).pos(:,1) < data(i).stepEnd(k));
%     end
%     subplot(2,1,2)
%         plot(data(i).pos(tt,3), -data(i).pos(tt,2), 'o-')
%     title(num2str(j))
%     pause
% end

%this is for data quality
%remove first 10 cells
for i = 1:length(data)
   data(i).stepStart(1:10) = [];
   data(i).stepEnd(1:10) = [];
   data(i).MSscans(1:10) = [];
   data(i).MSscans760(1:10) = [];
   data(i).intens(1:10,:) = [];
   data(i).intens760(1:10,:) = [];
   data(i).sumIntens(1:10,:) = [];
   data(i).celllocs(1:10,:) = [];
```

```
end


%remove low signal (m/z 184 < 250)
for i = 1:length(data)
   [~, ind184] = min(abs(data(i).mzs - 184.07));
   toremove = max(data(i).intens(:,ind184-1:ind184+1), [], 2) < 250;
   sum(toremove)/length(toremove)
   mz184stats(i).startCells = length(data(i).stepStart);
   mz184stats(i).removedCells = toremove;
   mz184stats(i).celllocs = data(i).celllocs;
   data(i).stepStart(toremove) = [];
   data(i).stepEnd(toremove) = [];
   data(i).MSscans(toremove) = [];
   data(i).MSscans760(toremove) = [];
   data(i).intens(toremove,:) = [];
   data(i).intens760(toremove,:) = [];
   data(i).sumIntens(toremove,:) = [];
   data(i).celllocs(toremove,:) = [];
   mz184stats(i).remainCells = length(data(i).stepStart);
end

save cleanDataLow184
```

RemoveSplits.m

```
clear
load cleanData

%remove data where the 184 and 760 intensity
%do not occur in the same scan
for i = 1:length(data)
    toRemove = data(i).MSscans ~= data(i).MSscans760;
    disp(sum(toRemove)/length(data(i).stepStart));
    data(i).stepStart(toRemove) = [];
    data(i).stepEnd(toRemove) = [];
    data(i).MSscans(toRemove) = [];
    data(i).MSscans760(toRemove) = [];
    data(i).intens(toRemove,:) = [];
    data(i).intens760(toRemove,:) = [];
    data(i).sumIntens(toRemove,:) = [];
    data(i).celllocs(toRemove,:) = [];
end

save cleanNoSplitData
```

Determination of Removal Efficiency from Radiographic Images

Motivation, Overview and Extensions

For characterization of the liquid microjunction extraction probe (Chapter 8), one parameter which had to be determined was the removal efficiency from a MALDI target plate. The experimental details are described more fully in Chapter 8, but for the discussion here the output data consisted of two images, before and after extraction, where the intensity at each pixel corresponded to the radiographic intensity at the position. To estimate the removal efficiency, the normalized intensity was compared between each image to determine the spatial distribution of the fraction removed during extraction. Since the resulting distribution contained random noise, the removal efficiency was estimated by fitting the distribution to a general, 2-dimensional Gaussian function. Details on the fitting equation are found in Chapter 8. The code below utilizes hard coded values which could be adapted for future analyses. The output is a series of images as seen in Figure 8.8 and the fitting parameters, reported in Table 8.1.

RemovalFit.m

```
close all
%input images
pre=imread('1.tif','tiff');
post=imread('2.tif','tiff');

%cropping points
x1=15;
x2=425;
y1=30;
y2=100;

%crop and convert to double arrays for fitting
pre=double(pre(y1:y2,x1:x2));
post=double(post(y1:y2,x1:x2));

%the approximate centers of the extraction points, from data cursor
approxCents = [46 26
    132 26
    214 26
    292 26
    379 26
    ];

%center of background region, not on a spot
bkgrd = [80 50];
%size of subregion to examine
window = 10;

%center of region, on a spot but not extracted
offEx = [33 40];
%account for transpose with matlab images
x = offEx(2);
y = offEx(1);
%crop out off-spot image
preOff = pre(x-window:x+window,y-window:y+window);
postOff = post(x-window:x+window,y-window:y+window);

%normalize images to mean in offExtraction area
%to account for exposure differences
pre = pre/mean(preOff(:));
post = post/mean(postOff(:));

%repeat extraction with background area
x = bkgrd(2);
y = bkgrd(1);
preBack = pre(x-window:x+window,y-window:y+window);
postBack = post(x-window:x+window,y-window:y+window);

%temporary image for estimated intensities
temp = zeros(size(preBack));

%functions for fitting
aval = @(theta, sigx, sigy) ...
    cos(theta)^2/(2*sigx^2) + sin(theta)^2/(2*sigy^2);
bval = @(theta, sigx, sigy)...
```

```
    -sin(2*theta)/(4*sigx^2) + sin(2*theta)/(4*sigy^2);
cval = @(theta, sigx, sigy)...
    sin(theta)^2/(2*sigx^2) + cos(theta)^2/(2*sigy^2);

%gaussian 2d functions
gaus2d = @(A, mux, muy, theta, sigx, sigy, x, y)...
    A*exp(-(aval(theta, sigx, sigy) .* (x-mux).^2 ...
                -2*bval(theta, sigx, sigy) .* (x-mux) .* (y-muy)...
                +cval(theta, sigx, sigy) .*(y-muy).^2));

%open new figure, initialize fits (fts)
figure;
fts = [];

%keep x,y coordinates as they are reused
[xs, ys] = meshgrid(-window:window,-window:window);
xs = xs(:);
ys = ys(:);

%for each extraction
for i = 1:size(approxCents,1)
    %cut out region around center
    %transpose the centers
    x = approxCents(i,2);
    y = approxCents(i,1);
    %subtract background intensity
    preImg = pre(x-window:x+window,y-window:y+window) ...
                - mean(preBack(:));
    postImg = post(x-window:x+window,y-window:y+window) ...
                - mean(postBack(:));
    %show previous image
    subplot(5,5, 5*(i-1)+1);
    imshow(mat2gray(preImg, [0 1.5]));
    %show post image
    subplot(5,5, 5*(i-1)+2);
    imshow(mat2gray(postImg, [0 1.5]));

    %show removal distribution (1-post/pre)
    subplot(5,5, 5*(i-1)+3);
    diff = 1-(postImg)./(preImg);
    %absolute is to make the residuals match here
    imshow(mat2gray(abs(diff), [0 1]));

    %determine fit, A is in [0, 2], x,y in [+/- window], theta 0,2pi,
    %sigma in 0,window size
    fts{i} = fit([xs ys], diff(:), gaus2d,...
        'StartPoint', [.6, 0, 0, pi, 5, 5], ...
        'Lower', [0, -window, -window, 0, 0, 0],...
        'Upper', [2, window, window, 2*pi, window, window]);

    %plot estimated extraction profile
    subplot(5,5,5*(i-1)+4);
    temp(:) = fts{i}([xs, ys]);
    imshow(mat2gray(temp, [0 1]));

    %show absolute residuals
    subplot(5,5,5*(i-1)+5);
    imshow(mat2gray(abs(diff-temp), [0 1]));
end;

%draw titles on just the first row
titles = {'Before', 'After', 'Fraction Removed', 'Fit', 'Residuals'};
for i = 1:5
   subplot(5,5,i)
   title(titles{i});
end
colormap hot

%display fit information in console
for i = 1:5
    fts{i}
end
```

Migration Time Alignment for CE MS

Motivation, Overview and Extensions

A challenge in any separation is the comparison of migration times between runs. For CE, numerous variables are difficult to control due to sample or environmental variation, leading to variance in migration times. In the simplest case, as assumed here, the change is a linear relationship between samples. To correct and align migration times, all that is required is the determination of migration time for several compounds found in each sample, followed by mapping the two migration times onto each other by linear regression.

First, extract ion electropherograms are exported from Bruker DataAnalysis using the method replicated below for each *m/z* value of interest. The resulting text files are then read into Matlab with the provided extractEIE function for analysis. While code is present for attempting to normalize to the migration time "standard" peaks, the data was not utilized in a quantitative way. A clear improvement would be the direct analysis of raw electropherogram data instead of exporting the intermediate values. Addition of internal standards could facilitate more quantitative multivariate analysis.

Export EIC Method

```
Option Explicit

'Remove previous analyses
Analysis.Chromatograms.Clear
Analysis.Compounds.Clear

Dim i

'initialize array with m/z values from bigList
Dim vaMyArray
vaMyArray = Array(155.0344,179.1446,175.1195,203.223571,146.165722,
    89.107873,112.087472,131.129671,189.135151,161.129,104.1075,
    241.1301,114.0660,133.097703,90.0543,147.112,123.0558,136.06232,
    146.1181,104.0712,156.0773,138.091889,138.0919,132.1009,162.113019,
    152.057235,139.050753,170.081719,154.0868,137.0594,136.0841,
    177.1028,191.118438,168.102454,170.0817,209.092618,219.149738,
    204.1236,184.0974,76.0399,132.0768,161.092618,244.092,242.125315,
    268.1046,239.1066,325.043696,90.0543,130.086804,138.055,291.13046,
    104.071154,295.129398,104.071154,90.055504,118.0858,120.0661,
    106.0504,132.102454,132.1009,223.075255,124.039854,132.102454,
    120.066069,133.0646,205.0977,150.058876,147.076968,176.103517,
    148.060984,247.140631,166.086804,235.108268,268.10458,182.0817,
    100.039854,139.050753,137.046336,116.071154,196.097369,198.0766,
    104.071154,122.027576,134.045334,118.0862,144.102,306.145382,
    132.066069,255.085834,208.0971,279.08034,221.0926,235.108268,
    269.088596,142.026922,348.070913,324.05968,184.073872,86.060589,
    364.0653,233.129003,265.112308,126.022491,245.077363,192.102454,
    243.098098,162.0889,178.0868,134.060589,279.08034,206.081719,
    192.0665,220.118,308.091634,330.060348,249.145048,164.038141,
    130.050419,190.0499,244.0928)

Dim nArraySize
nArraySize = UBound(vaMyArray)  - LBound(vaMyArray)
ReDim vaEICDefinitions(nArraySize)

'add new EIC definition for each m/z value
For i = LBound(vaMyArray) To UBound(vaMyArray)
    Call AddEICDefinition(i, CStr(vaMyArray(i)))
Next

'add all definitions to the current EICs and smooth
Analysis.Chromatograms.AddChromatograms(vaEICDefinitions)
Analysis.Chromatograms.Smooth

Form.close

'helper function for defining an EIC
Sub AddEICDefinition(i, sRange)
    Dim EIC
    Set EIC = CreateObject("DataAnalysis.EICChromatogramDefinition")
    EIC.MSFilter.Type = daMSFilterMS
    EIC.ScanMode = daScanModeFullScan
    EIC.Polarity = daPositive
    EIC.WidthLeft = "0.005"
    EIC.WidthRight = "0.005"
    EIC.Range = sRange

    Set vaEICDefinitions(i) = EIC
End Sub

'save each chromatogram as a separate txt file
dim chrom
For i = LBound(vaMyArray) To UBound(vaMyArray)
    set chrom = Analysis.Chromatograms(i - LBound(vaMyArray)+1)
    chrom.Export "D:\Data\" +CStr(vaMyArray(i))+".txt", daXY
Next

Form.close
```

extractEIE.m

```
function [masses, migrationTime, intensities] = extractEIE(dirname)
    %from a directory of text files, extracts the masses, migrationTime
    %and intensities for each EIE

    %get all files in directory
    filenames = dir(dirname);
    %remove directories from filenames
    filenames = filenames([filenames(:).isdir] == false);

    %read in first file to determine sizes of migration times and masses
    data = dlmread(fullfile(dirname, filenames(1).name));

    %initialize variable sizes
    migrationTime = data(:,1);%seconds
    masses = zeros(1, length(filenames));
    intensities = zeros(length(migrationTime), length(masses));

    %for each text file
    for i = 1:length(filenames)
        %read file
        data = dlmread(fullfile(dirname, filenames(i).name));
        %record mass from file name and intensities
        masses(i) = str2double(filenames(i).name(1:end-4));
        intensities(:,i) = data(:,2);
    end
end
```

ceAlignAndAnalyze.m

```
clear; close all; clc;

%directories for sample sets
sampleDirs = {
    '../../Tube 2'
    '../../Tube 5'
    };%add more here

%since reading the raw data is slow, try to read the mat file
if ~exist('dataraw.mat', 'file')

    %for each directory
    for i = 1:length(sampleDirs)
       %read in text files
       [mass, time, intens] = ExtractEIE(sampleDirs{i});
       %record name
       [~,name,~] = fileparts(sampleDirs{i});

       %perform average smoothing
       windowSize = 7;
       intens = filter(ones(1,windowSize)/windowSize, 1, intens);
       %background adjust
       intens = msbackadj(time, intens);

       %record all values
       data(i).mass = mass;
       data(i).rt = time;
       data(i).intens = intens;
       data(i).name = name;
    end
    %save for next run
    save dataraw data

else
    load dataraw
end

%read in name/mz values from big list
[num, txt] = xlsread('../../Big List.xlsx', 1, '', 'basic');
mzs = num(:,2);
names = txt(2:end,2);

%specify mz values to use for alignment
mzAlign = [120.0661 132.1025 156.0773 166.0868 76.0399 90.0543];
nameAlign = {'Threonine', 'Leucine', 'Histidine', ...
             'Phenylalanine', 'Glycine', 'Alanine'};

mass = data(1).mass;

%determine alignment times and areas
for i = 1:length(data)
   %migration times to use for calibration
   data(i).calTimes = zeros(size(mzAlign'));
   %total peak area of each MT standard
   data(i).calPA = 0;
   intens = data(i).intens;
   time = data(i).rt;
   name = data(i).name;
   %for each peak in MT standard
    for ii = 1:length(mzAlign)
        %find nearest mass
        %if one, use the intensity
        if(sum(abs(mass-mzAlign(ii)) < 0.001) == 1)
            inten = intens(:, abs(mass-mzAlign(ii)) < 0.001);
        %else for more, use the average intensity
        else
            inten = mean(intens(:, abs(mass-mzAlign(ii)) < 0.001),2);
        end
        %find peak in EIE
        [pks, locs, w, p] = findpeaks(inten, time, ...
                                    %find one peak
                                    'NPeaks', 1, 'MinPeakDistance', ...
                              time(end)-time(2));%find largest peak
        %record time
        data(i).calTimes(ii) = locs;
        %calculate peak area for norm
        data(i).calPA = data(i).calPA + ...
            trapz(time(abs(time-locs)<w),inten(abs(time-locs)<w));
    end
end

%determine corrected migration times
%first is used as standard
data(1).corrRT = data(1).rt;
%for all others
for i = 2:length(data)
    %fit to first sample
    fit2one = fit(data(i).calTimes, data(1).calTimes, 'poly1');
    %calculate corrected time
    data(i).corrRT = fit2one(data(i).rt);
end

%draw each m/z value, pausing between
figure;
for j = 1:length(mass)
    for i = 1:length(data)
       subplot(5,4,i); hold on;
       plot(data(i).corrRT, data(i).intens(:,j))
       title(data(i).name)
       xlim([0 2000])
    end
    set(gcf, 'Name', num2str(mass(j)))
    pause
end
```

APPENDIX B

microMS User Manual

Notes and Acknowledgements

The following is a detailed description of the operation of microMS for performing microscopy-guided mass spectrometry profiling. First, the capabilities are explained and demonstrated to enable novice users to quickly begin performing experiments. The second half of the guide details how to add support for new instrumentation through modifying the source code, which is found in Appendix A.

microMS is a feature-rich GUI for performing basic image analysis and correlation of image positions into physical coordinate spaces. The overall goal of this package is to image and locate a field of cells or other objects dispersed across a microscope slide, allow subpopulations to be selected based on flexible and user defined criteria, convert the cell / object locations to a platform dependent set of positions to be used for follow-up assays such as selected cell collections or mass spectrometry profiling. While developed for single cell analysis by mass spectrometry, with few modifications, the underlying code is versatile enough for a variety of targets, image modalities, and follow-up analytical systems. microMS aims to simplify cell finding, improve coordinate registration, and provide an interface suitable for novice users. Several additional features are added to expand the repertoire of profiling experiments. For more advanced users, the addition of new, off-line instrument platforms and even direct instrument control are possible. This guide will introduce the most common usage of microMS, detail additional features, and provide a starting point for adding new instrument coordinate systems. The operation and use of this code has been described in Chapter 5, including an application for high throughput single cell profiling via MALDI (Chapter 6), SIMS (Chapter 7), and follow-up CE-MS analysis (Chapter 8).

Installation and Startup Instructions

Refer to http://neuroproteomics.scs.illinois.edu/microMS.htm for the most recent instructions for installation and startup.

Windows Installation and Execution

Most dependencies of microMS are included in standard distribution packages of python 3. We have had success using anaconda which also includes an IDE (https://www.continuum.io/downloads). A python 3.X (X >= 5) version is required to be installed. After installation, pyserial and openslide require separate installations. Pyserial is installed by opening Windows PowerShell and entering `pip install pyserial`. This should automatically install the most recent version. Openslide requires additional binaries, located here: http://openslide.org/download/ under Windows Binaries. Download and extract the folders, placing them in your documents or program files. The contained bin folder also must be added to the operating system path (e.g. C:\Program Files\Openslide\bin\): http://www.howtogeek.com/118594/how-to-edit-your-system-path-for-easy-command-line-access/ Once added correctly, the openslide python wrapper is simply installed by entering `pip install openslide-python` in PowerShell.

With these dependencies installed, microMS is installed by downloading and extracting the ZIP file. The main script is run with `python microMS.py` in a command prompt in the script directory or by double clicking the microMS.bat file which runs the above command and waits for a key press.

Linux Installation and Execution

With an installation of python 3, the required packages are installed in the terminal with `pip3 install <package>`. Openslide is installed with:

```
apt-get install python-openslide
pip3 install openslide-python
```

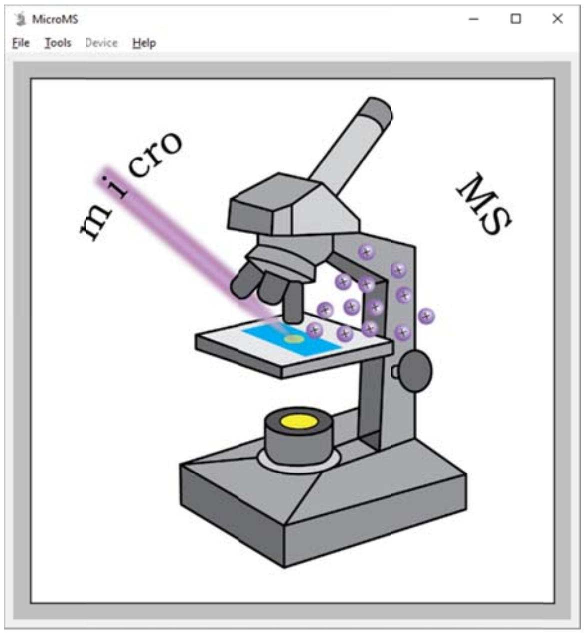
After unzipping the microMS file, the main method is run with either `python microMS.py` or ./`microMS.py` after executable permission is granted (i.e. `chmod +x microMS.py`). Adding the microMS directory to the PATH variable will allow execution from any directory.

Image File Types

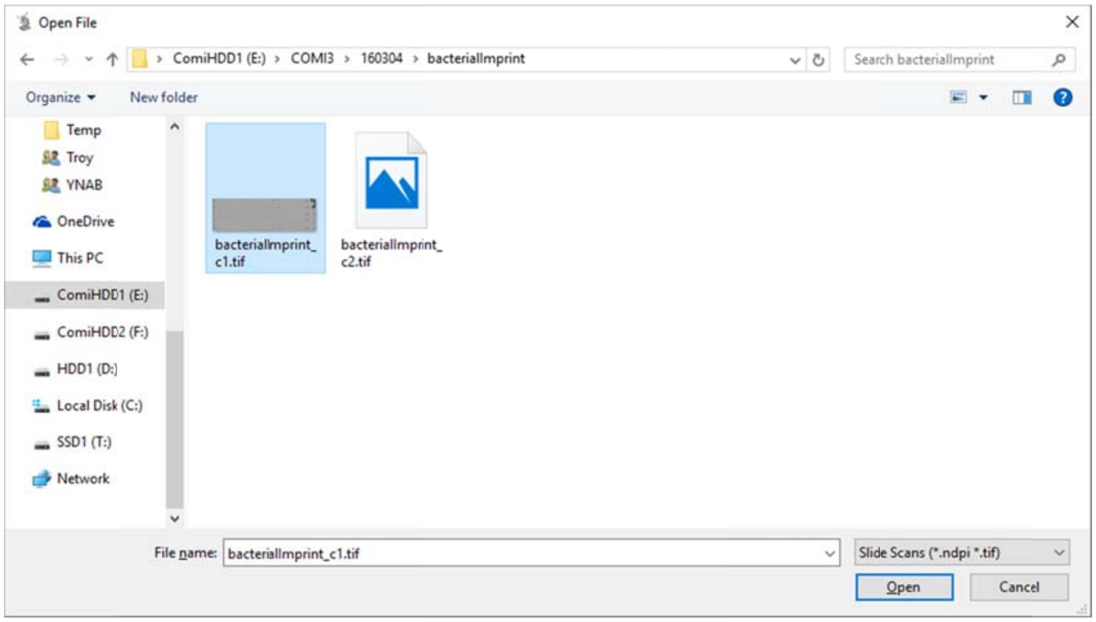
microMS utilizes openslide to read sections of whole slide images. Any openslide-supported format should be a suitable input, though only Hamamatsu ndpi and bigTiff images have been thoroughly tested and are accepted by default. For multichannel images, ndpi files should end with 'Brightfield' (for channel 1, brightfield) and 'Triple' (for channel 2, RGB fluorescence). Tiff images should *not* start with '8x' or '64x'. Multichannel tiff images should be indicated with the suffix 'c#.tif' where '#' is a digit between 1 and 9. By default, tiff images can only be zoomed out by 4x. Tiff images may be decimated to 8x and 64x, generating the files '8x<image name>' and '64x<image name>' with <image name> being the original file name.

Opening an Image

On startup, the main GUI window is displayed with the program icon:



An image is opened by selecting *Open* in the *File* menu (or Ctrl + O). This will launch a file selection dialog which prompts the user for the desired file which will subsequently be loaded. Selecting any image of a multichannel-data image set will open all channels with matching filenames as described above.



File name: bacterialImprint_c1.tif

Slide Scans (*.ndpi *.tif)

Multiple image channels (if they exist) are loaded at the same time and overlaid on top of each other:

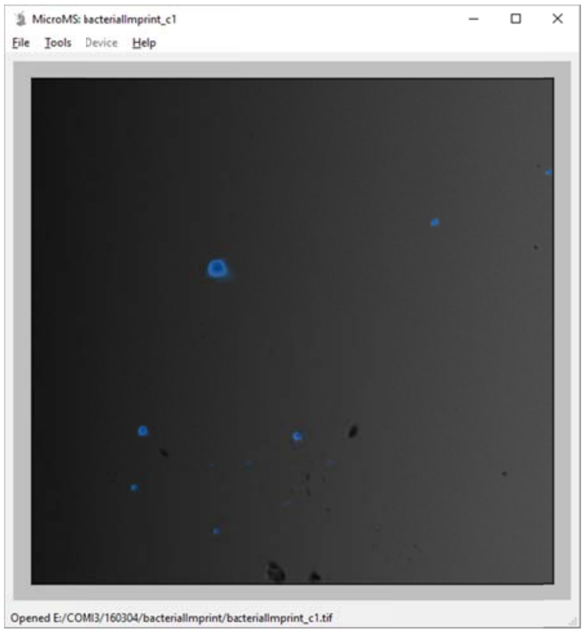
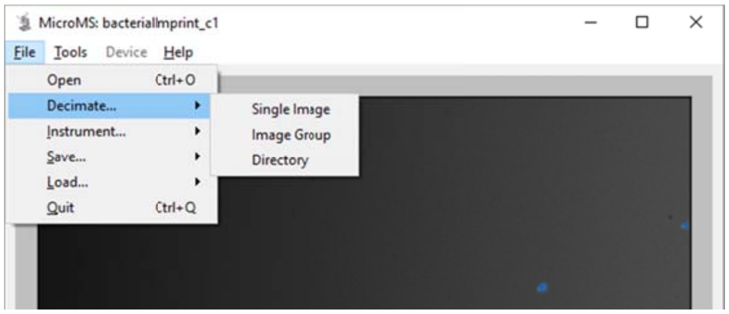

MicroMS: bacterialImprint_c1
File Tools Device Help
Opened E:/COMI3/160304/bacterialImprint/bacterialImprint_c1.tif

Image Decimation

Non-pyramidal tiff images will by default only be able to display at up to 1/4x. To enable further zoom levels from 1X to 1/256x (which is recommended), tiff image sets must be decimated by selecting the *Decimate* option in the *File* menu. Several options are available:



Both 'Single Image' and 'Image Group' options are for the selection of one tiff image. When an image of a multichannel-data image set is selected with the 'Image Group' option, all images of the set will be decimated. The 'Directory' option allows the selection of a single directory. All images cotained in sub-directories of the selected directory be decimated as shown schematically below:

Single Image

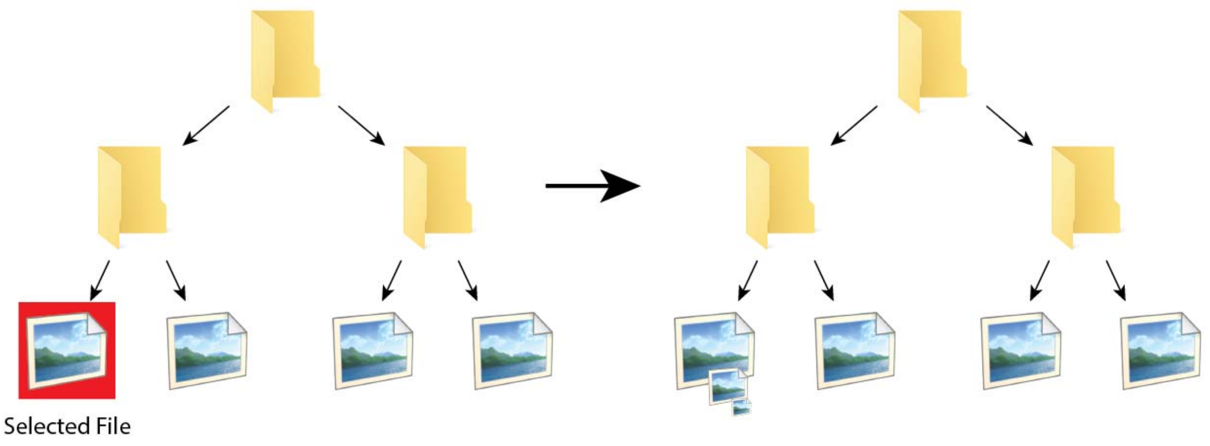
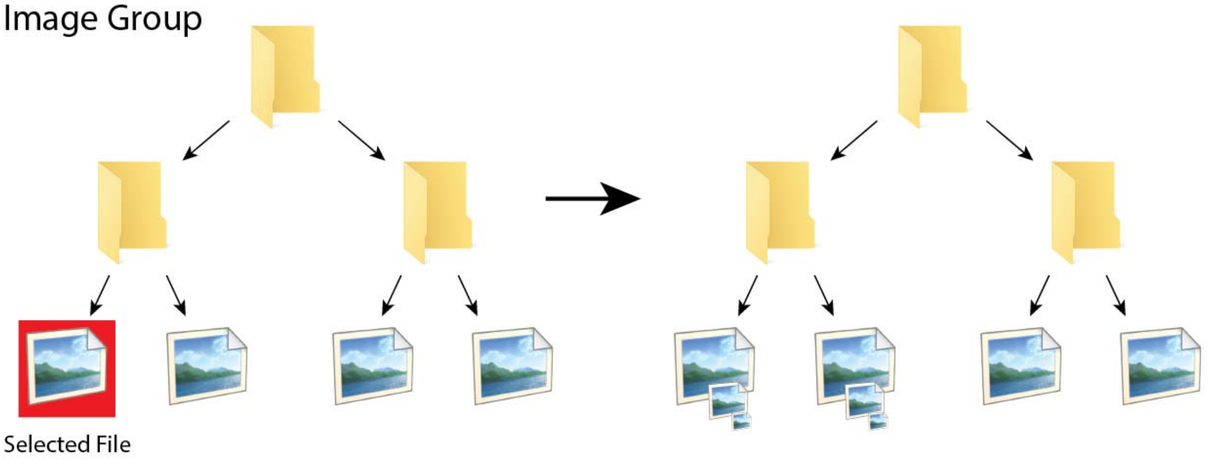
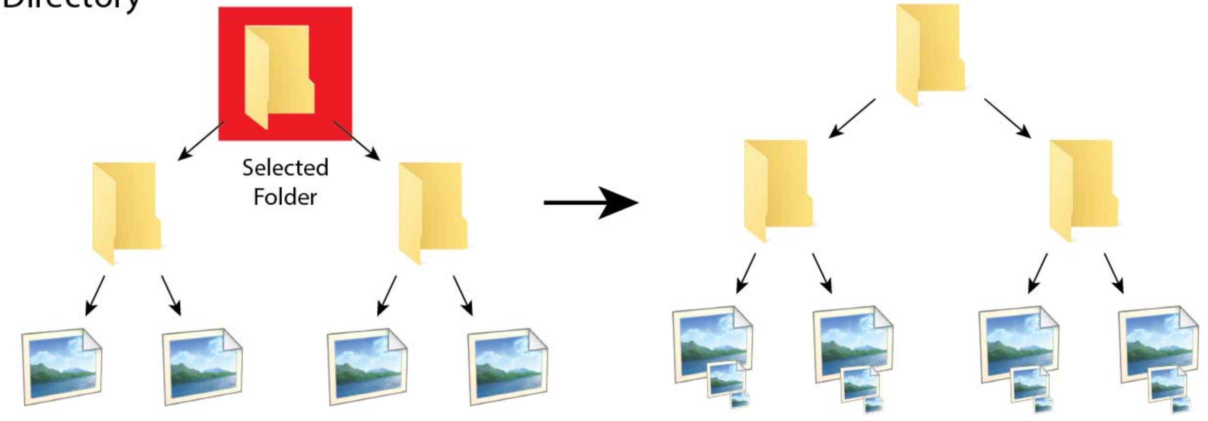


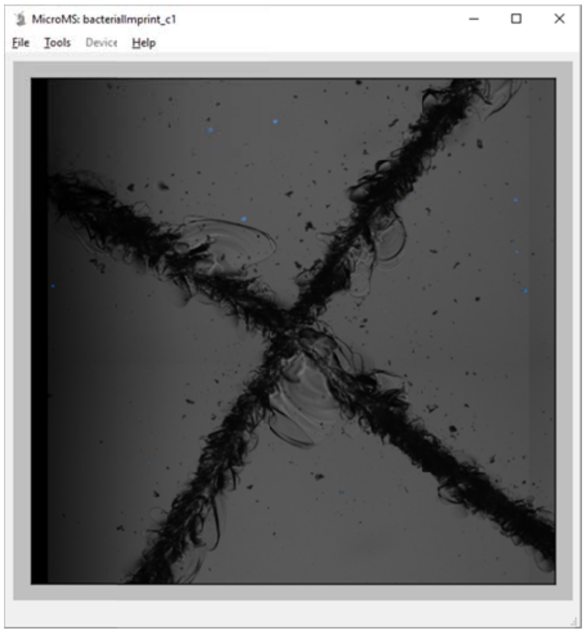
Image Group



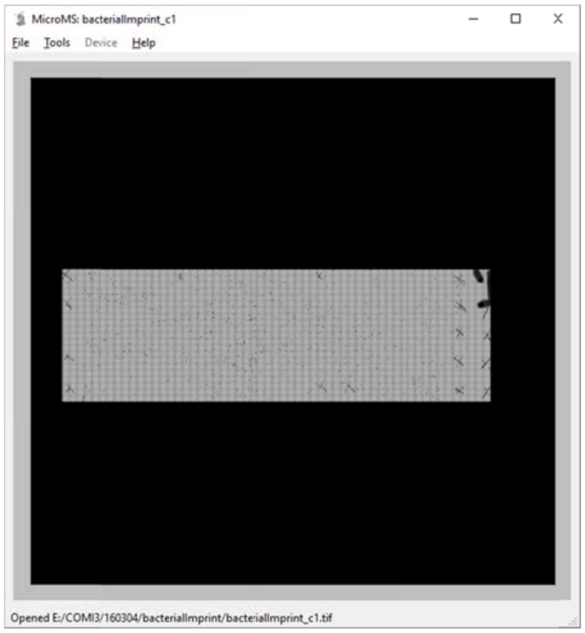
Directory



Following decimation of a single image or a group of multichannel images, the image(s) will be automatically opened. The max zoom level increases from:



to:

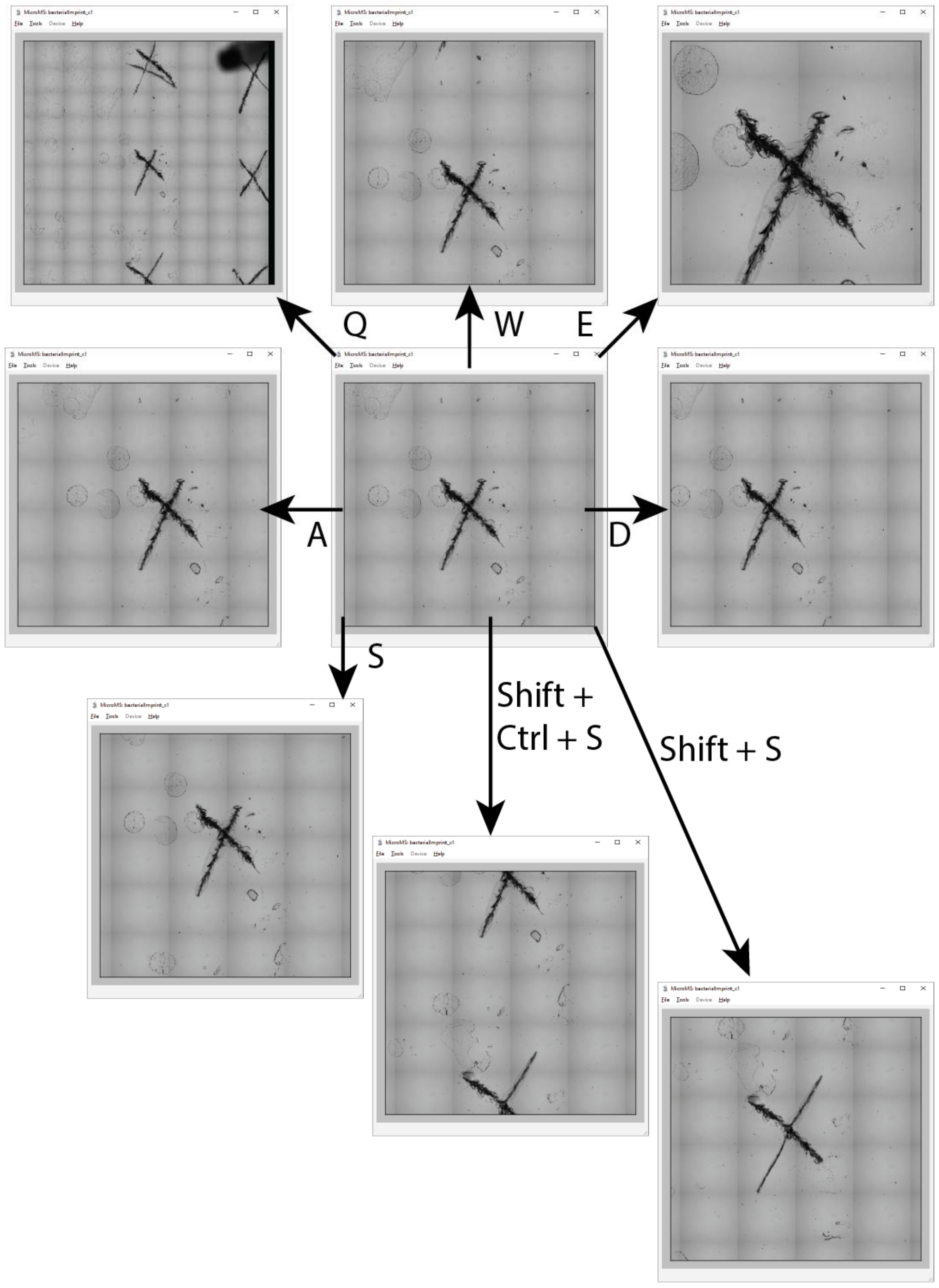


Opened E:/COMI3/160304/bacterialImprint/bacterialImprint_c1.tif

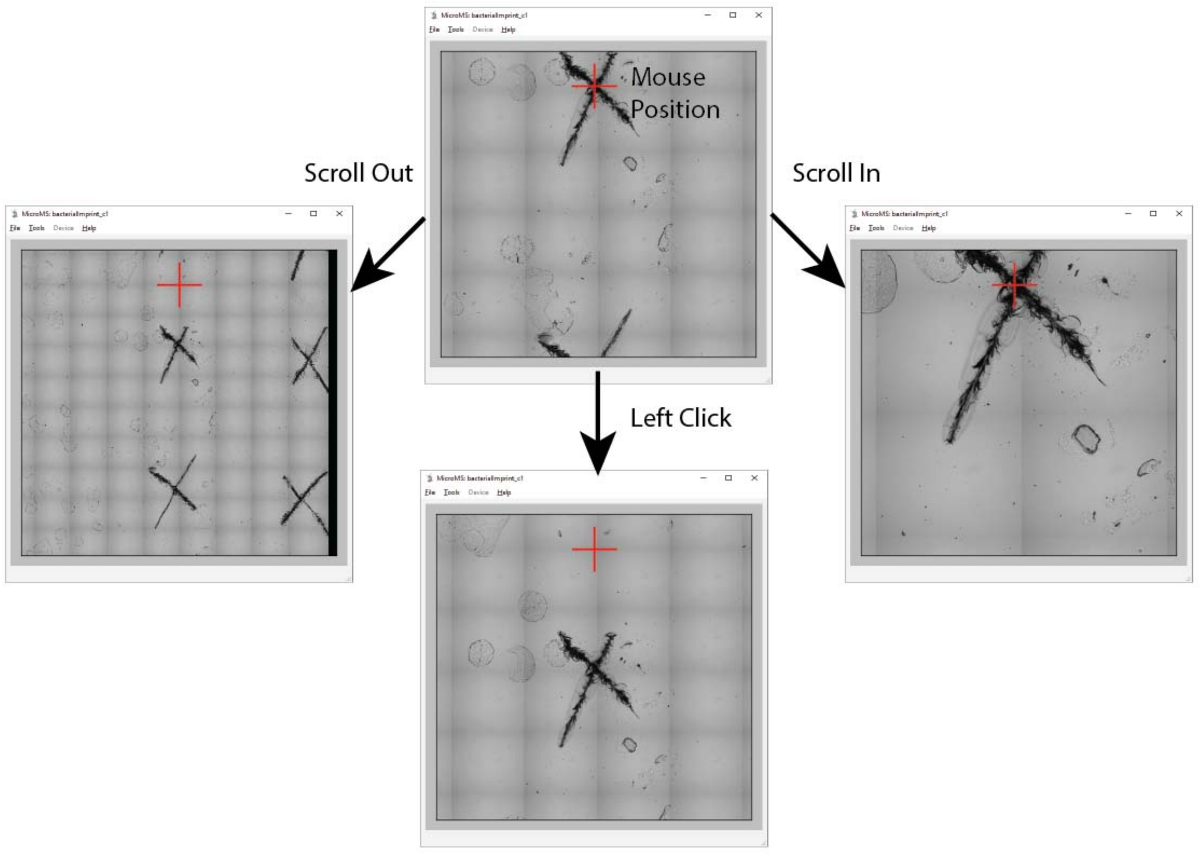
Notice that since a single image (brightfield) was decimated, only the specified channel is displayed at higher zoom levels. After decimation is complete, the process does not need to be repeated the next time the image set is opened. Selecting the non-decimated image will automatically load the full image stack.

Image Navigation

Images may be navigated with a combination of keyboard and mouse controls. With keyboard movements, the keys W, A, S, D move the view frame up, left, down, and right, respectively. Each step moves 1/10 of the frame size. Combining each key with *Ctrl*+*Shift* moves 1/2 the frame, and *Shift* moves a full frame. The keys Q and E zoom out and in:


Q
W
E
A
D
S
Shift + Ctrl + S
Shift + S
MicroMS: bacterialImprint_c1
File Tools Device Help

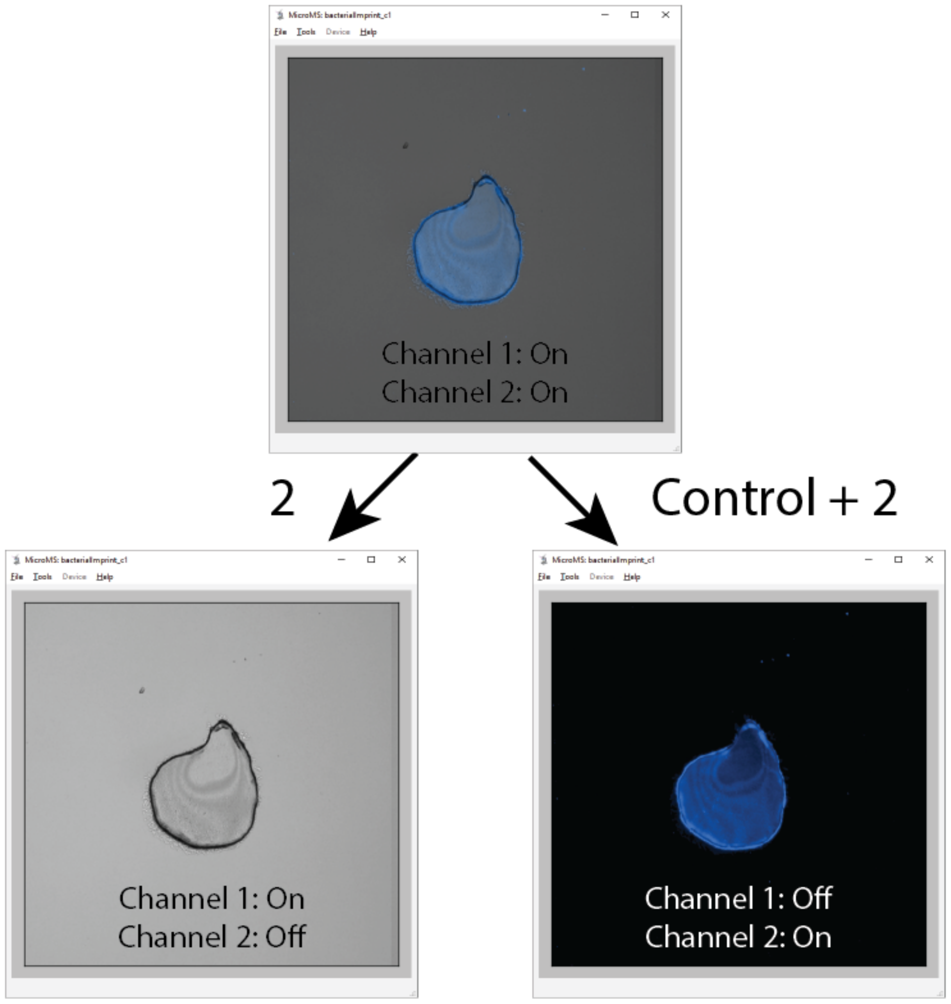
With the mouse, the scroll wheel is used to zoom in (wheel up) and out (wheel down). When zooming in, the center of the frame also moves to keep the mouse location in the same position. Left clicking a position moves that spot to the center of the frame:



The key R will reset the field of view to the top left corner at full zoom. This operation is useful if the field of view moves too far from the sample area. Moving around an image data set should be smooth, but due to the size of images, some lag between input and display may occur. Also note that images are read from disk (not stored into memory/RAM). This design choice was made to allow microMS to run on a variety of systems without requiring large amounts of memory. However, reading images on a network drive or external hard drive (USB 2) is very slow!

Switching Image Channels

Displaying different image channels is controlled through the numeric keys and is cycled with the T or Z key. Each image is enumerated based on its name. For ndpi files, "Brightfield" is number 1 and "Triple" is number 2. With multichannel tiff images, the number in "c#.tif" is parsed and assigned to the corresponding image number. Pressing the number of an image will toggle that corresponding channel on and off. The combination of *Crtl* + 'image number' will turn all channels off except for the selected channel. It is possible to turn off all image channels, which will display a black background.



Up to 9 different channels may be utilized in one experiment, but only one color (red, green or blue) is taken from each fluorescent channel during image overlay. The color channel with the highest intensity is utilized as the "color" for that channel. The brightfield image must be channel one. Since each image is read from disk, displaying more channels will slow the response time proportionally.

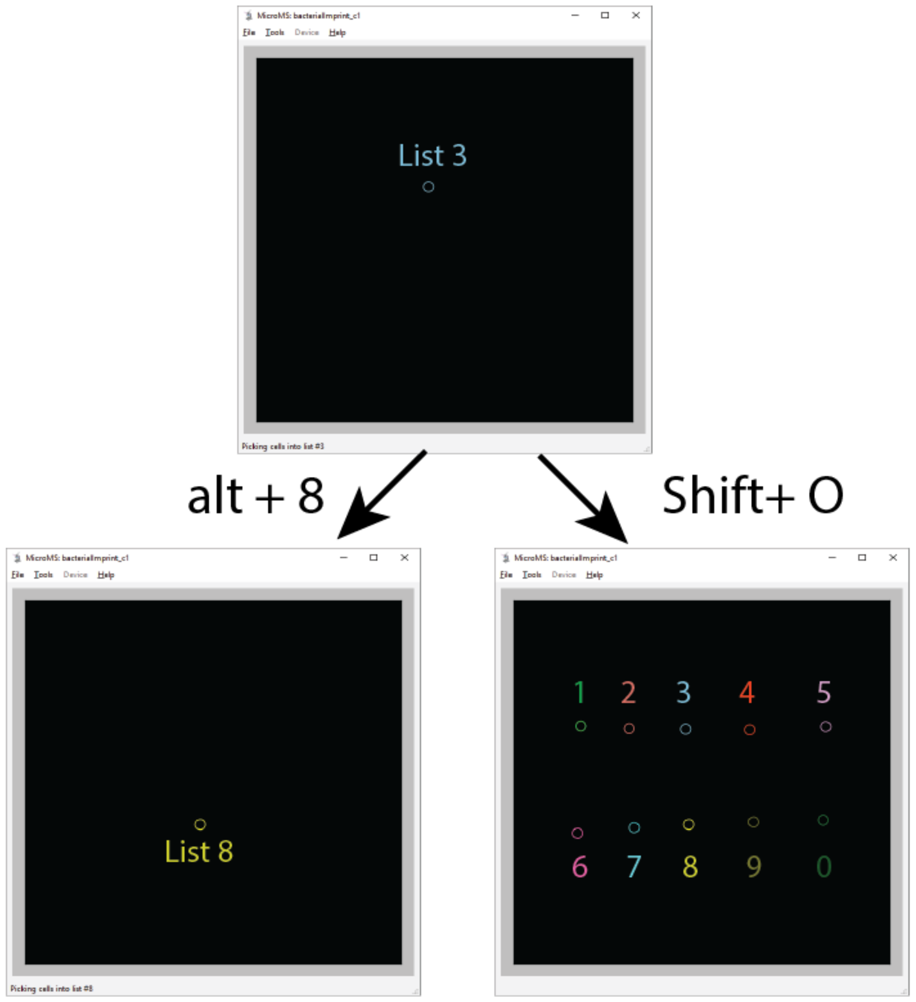
Blobs

All points of interest in microMS are represented as objects called blobs. Each blob consists of an x and y coordinate, in pixels, an effective radius and the object's circularity. The blob area (used to determine size and circularity) is taken as the number of pixels above the fluorescence threshold. A blob's effective radius is then $\sqrt{\frac{area}{\pi}}$ and it's circularity is $\frac{4\pi\times area}{perimeter} \in [0,1]$. Due to the calculation of perimeter with pixels, the circularity is generally larger than would be expected for a non-pixelated object. The x and y position is the center of mass for the Boolean image of pixels above the specified threshold. Note that this does not account for the intensity of the underlying image. When blobs are added manually, the resulting circularity is always 1. Blobs may optionally have a group assigned to them as part of packing routines explained below.

Collections of blobs are stored in lists. Up to 10 different blob lists can be utilized in one image set. Each list is treated as an independent set of targets and by default only one list is displayed at a time. Lists spawn new child lists during packing and filtering procedures. Generally, the first empty list (in increasing order) will be filled with automatically generated target sets. Loading found cells or instrument positions will populate the currently selected list, possibly overwriting its contents.

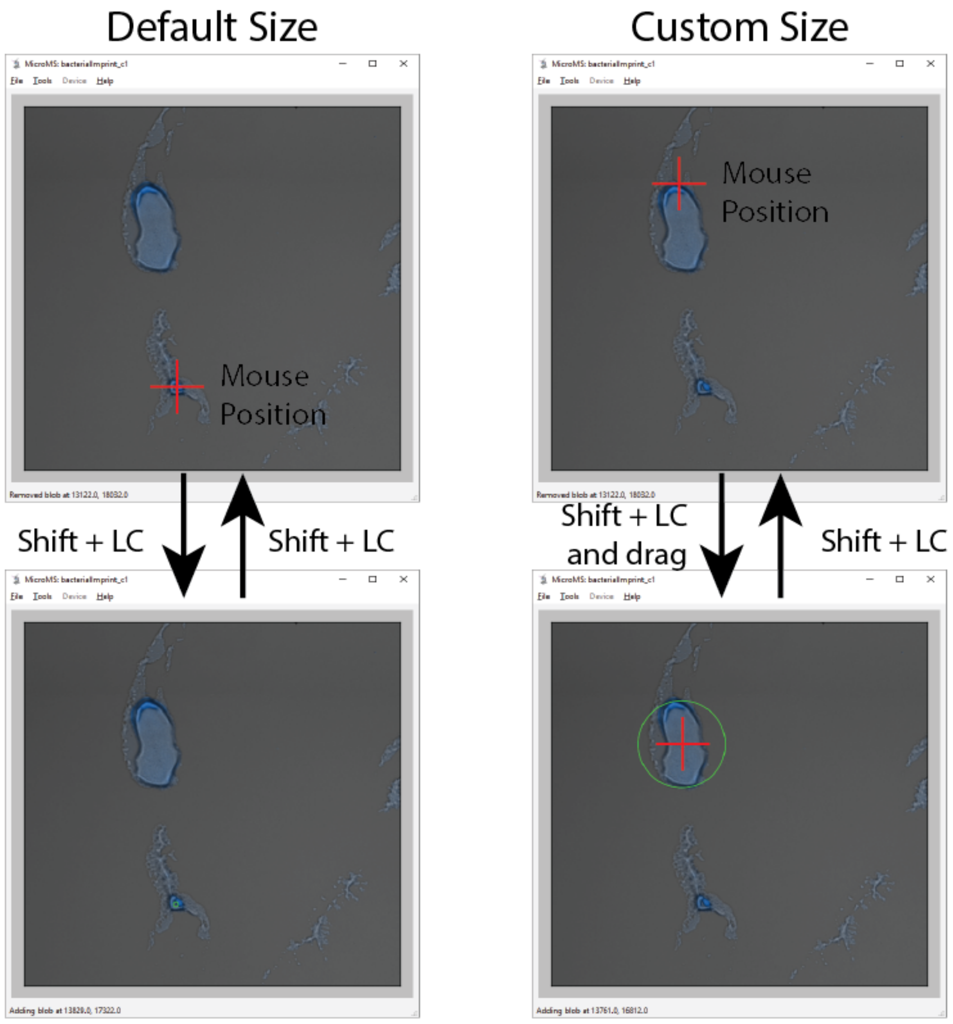
Displaying target collections

Each list of blobs is designated with a unique color. The currently selected list is displayed and is changed by the combination of *Alt* + ‘the list number’. By default, only the currently selected list will be displayed. All subsequent discussions regarding changes to the blob list will only affect the current list. It is occasionally useful to see multiple lists simultaneously. Drawing all blob lists is toggled with *Shift* + O. Blobs which overlap will show a blended color, though generally it is difficult to see which combination is overlaid.


MicroMS: bacterialImprint_c1
File Tools Device Help
List 3
Picking cells into list #3
alt + 8
Shift+ O
List 8
Picking cells into list #8
1 2 3 4 5
6 7 8 9 0

Manual addition of targets

New blobs are added to the current list with a left mouse button click with *Shift* held. The default, minimum radius of a manually added blob is 10 pixels. Larger blobs can also be added by performing a left click and drag. The circumference is set when the mouse button is depressed and the center upon its release. During the dragging, all other features will disappear and the resulting blob will be dynamically drawn. Blobs are removed by holding *Shift* and left mouse click anywhere inside of them. Releasing a custom drawn blob inside of an existing blob will remove the existing blob without drawing the custom blob.



Removed blob at 13122.0, 18032.0

Removed blob at 13122.0, 18032.0

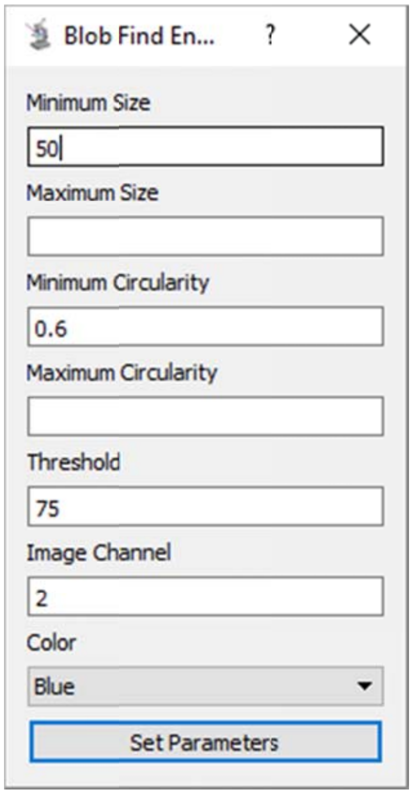
Adding blob at 13829.0, 17322.0

Adding blob at 13761.0, 16812.0

Automatic blob finding

Input blob finding parameters

Automatic blob finding is at the core of performing high throughput analysis. Currently, blobs are found with a threshold and group algorithm, which is sufficient to locate bright objects which have high contrast and low background. Briefly, the intensity of each pixel in the image is compared with the supplied intensity threshold value. Adjacent pixels with intensities above the


Blob Find En...

Minimum Size: 50

Maximum Size:

Minimum Circularity: 0.6

Maximum Circularity:

Threshold: 75

Image Channel: 2

Color: Blue

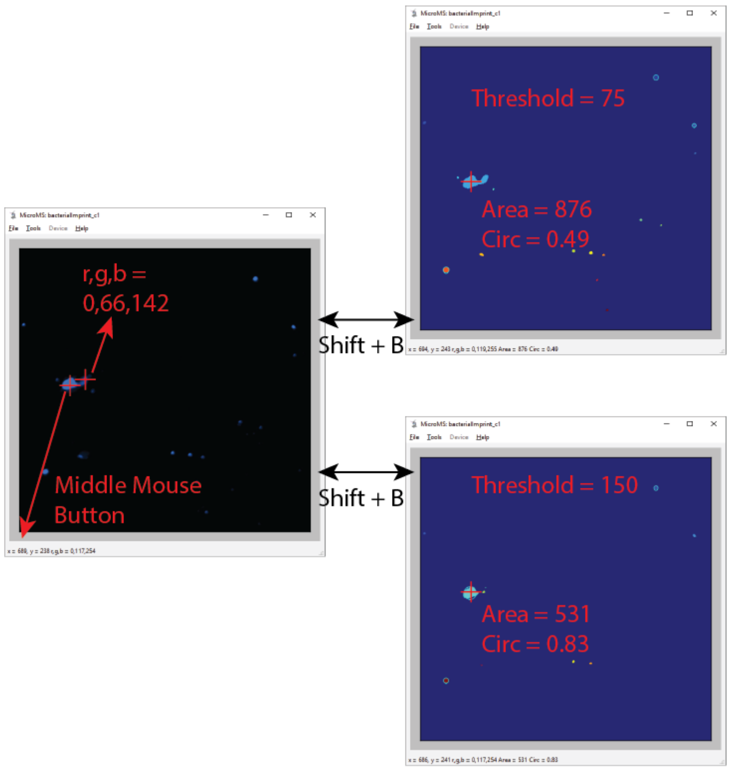
threshold are part of the same, putative blob. Next, the number of pixels in a putative blob (size) and its shape (circularity) are examined. If these characteristics are within the supplied ranges the collection of pixels is considered a blob.

The parameters for cell finding must be chosen carefully to ensure only targets of interest are selected, though filtering after blob finding is possible as discussed later. The parameters for blob finding are shown to the left and are accessed by selecting *Blob Options* under the *Tools* menu (or *Ctrl* + B). The size corresponds to the number of adjacent pixels which are above the intensity threshold. Circularity is defined above; more circular objects will have circularity closer to 1. Threshold is the intensity threshold to discriminate objects from background. Image channel is the image within the dataset to consider for cell finding, with image channel 1 corresponding to brightfield. Finally, color is the RGB channel to extract for performing thresholding. The screenshot above is of an analysis of blue fluorescence in the image channel 2.

Size, threshold and channel should be integers, circularity can be floating point. Maximum size and circularity may be left blank to indicate there is no upper boundary to these values. Any invalid inputs will be reverted to the last valid input on clicking "*Set Parameters".* In addition to setting the cell finding parameters, "*Set Parameters*" performs an initial cell finding of the current position at the maximum zoom level. If the current field of view is zoomed out, it will be automatically set to the max zoom level. Any found objects will be highlighted with a turquoise circle. Performing any other action will clear these found blobs. Test blob finding can also be performed by pressing B.

Pixel Information and Threshold View

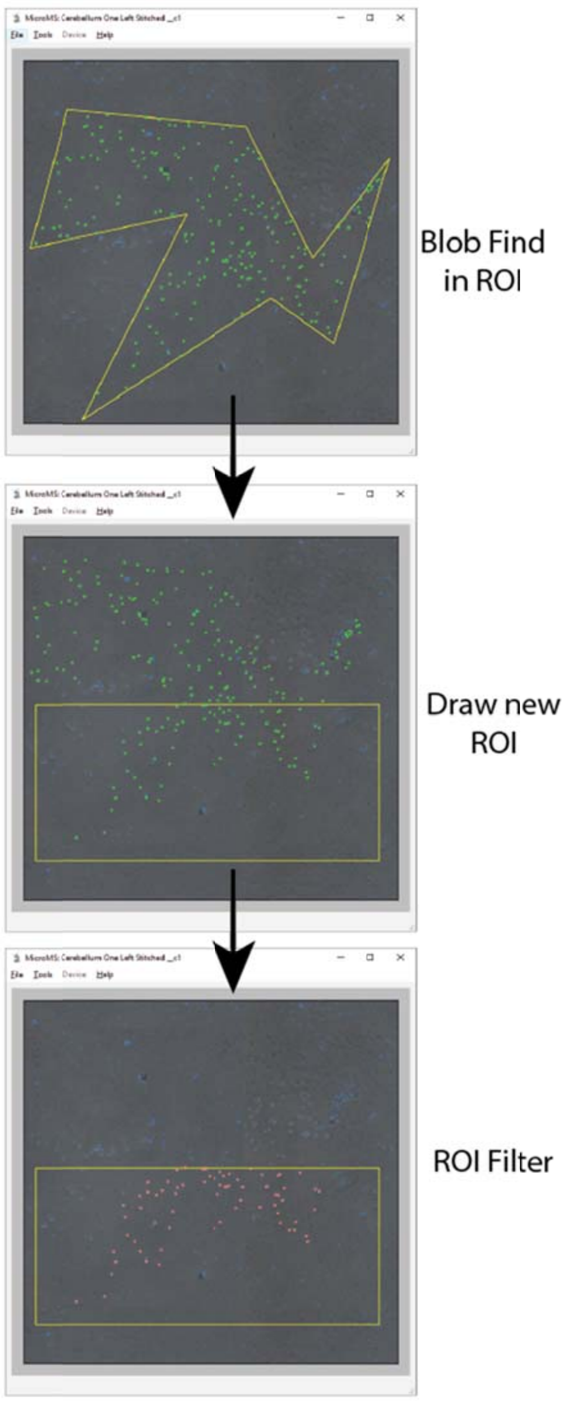
An easy way to determine suitable blob finding parameters, or identify why some blobs are excluded, is by examining the Pixel Information and utilizing Threshold View. At any time, clicking the middle mouse button will display information about the current pixel. This includes the pixel position relative to the maximum zoom image and the RGB values of the displayed image. The RGB values help establish a suitable threshold for positive pixels. *Shift* + B toggles the threshold view for the current image set. The current threshold and color channel are utilized to visualize blobs that pass the threshold. The background is displayed in a dark blue color. Pixels passing the threshold are then grouped and displayed as a unique color. Changing the blob parameters updates the threshold view and shows the currently found cells. Additionally, while in threshold view, performing a middle mouse button click on a blob will also provide its area and circularity in the status bar. This quickly provides feedback on why some blobs are ignored and help with selecting blob finding parameters.



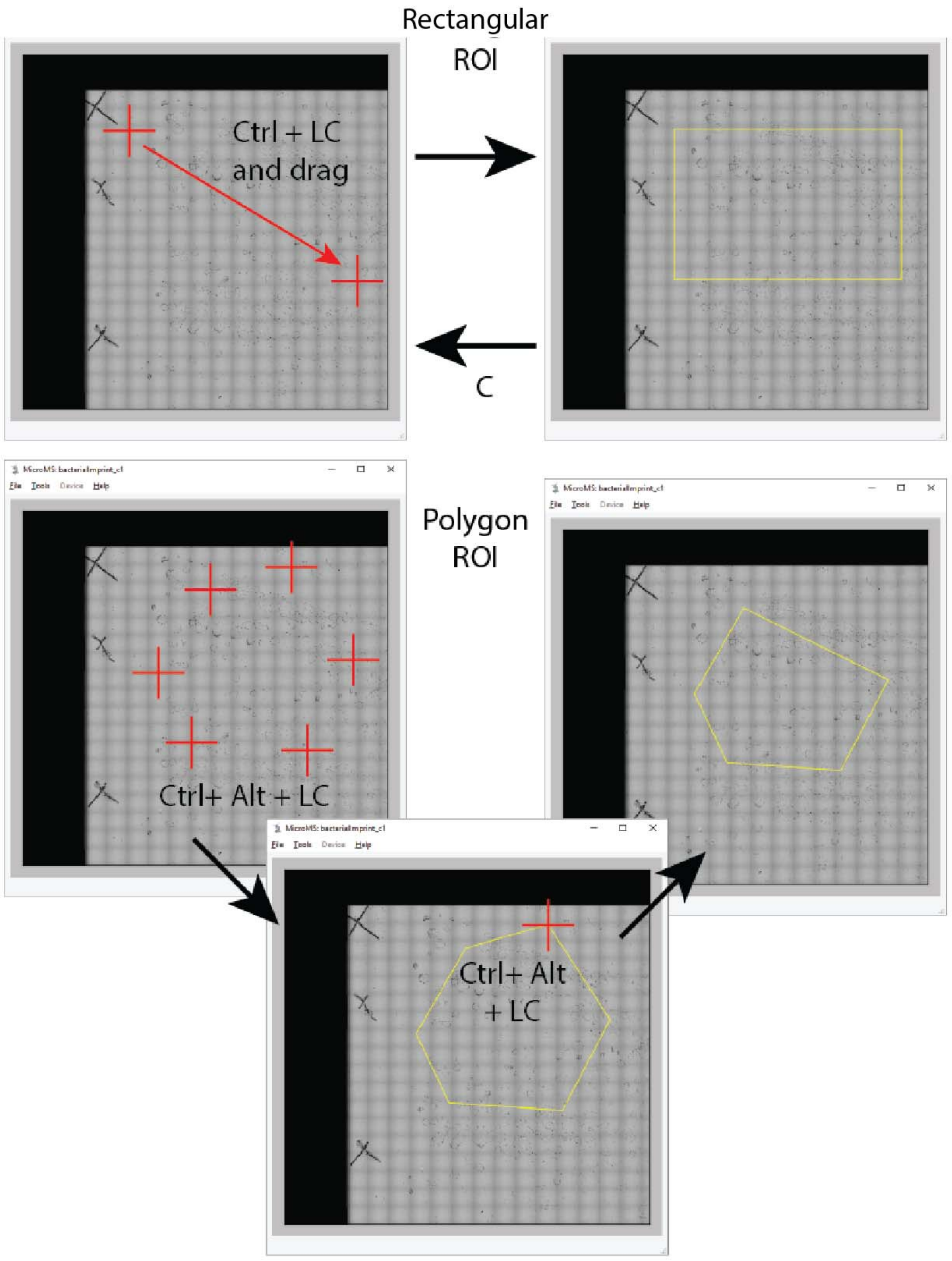
As seen above, at a threshold of 75, the selected blob is not included in found blobs because its circularity is below the set point of 0.6. Increasing the threshold removes the right feature of the blob and its circularity increases to 0.83, hence passing the circularity threshold.

Regions of Interest

Once blob finding parameters are chosen, automatic blob finding is performed by selecting the Blob *Find* option under the *Tools* menu. By default, this will perform blob finding on the entire slide. Note that this operation can take several minutes for large areas. Upon completion, the user is prompted for a base filename which is used to write a <BASE>.txt cell finding file and <BASE>.msreg file in the image file directory. These will be described in further detail below. All found blobs will be stored in the current cell finding list. By default, only a random set of 150 blobs will be drawn in a field of view. This significantly speeds up drawing speed when moving around a slide. This option may be toggled by the *Limit Drawn Blobs* option under the *Tools* menu.



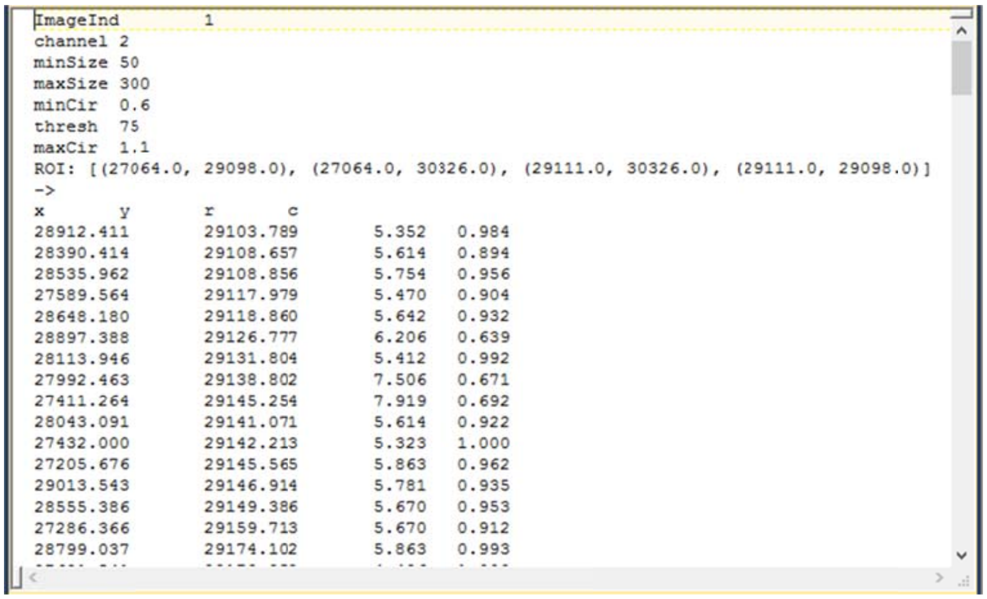
Since multiple sample populations may be present in one image and fiducials should be excluded, it is frequently useful to draw an ROI to restrict blob finding. An ROI may be set up prior to blob finding or ROI filtering can be performed on an existing blob list. Blob finding is restricted to an ROI, so it is faster to perform blob finding when the fiducials and exterior areas are excluded. ROIs are drawn as either rectangles or polygons. To draw a rectangle, hold *Ctrl* and left click and drag between two diagonal vertices of the rectangle. As the mouse moves, the ROI will update to show how the ROI would look if the mouse button was released. To reduce latency, only the slide image is displayed during this process (no blobs are shown). Once drawn, a rectangular ROI will behave identically to a hand-drawn polygon ROI. To interact with ROIs on a vertex level, hold *Ctrl* and *Alt*, and move the mouse around the ROI. Again, the ROI will update as the mouse moves, showing the ROI that would result upon a mouse click. The ROI cannot have overlapping edges and its vertices are removed by clicking on or near them. Alternatively, vertices can be added in order, by holding *Ctrl* and *Shift*. This simplifies drawing complex shapes, but is more difficult to modify. Finally, ROIs are cleared by pressing the C key.



ROI filtering can also be performed at any time after blob finding. Simply draw an ROI for the area to keep blobs, and select either *ROI Filter Retain* or *ROI Filter Remove* under the *Tools* menu. The former operation removes all blobs outside of the specified ROI and moves the resulting list to the next open blob list. The latter operation performs the opposite filtering, placing all blobs outside the ROI into a new list. The original blob list is retained in its list number in either case.

Saving and loading blob lists

Once blobs have been found automatically or manually placed, their positions and finding parameters can be saved for later examination or for use in other analysis steps. The file is in plain text format and human readable, as shown below:



```
ImageInd        1
channel 2
minSize 50
maxSize 300
minCir  0.6
thresh  75
maxCir  1.1
ROI: [(27064.0, 29098.0), (27064.0, 30326.0), (29111.0, 30326.0), (29111.0, 29098.0)]
->
x       y       r       c
28912.411       29103.789       5.352   0.984
28390.414       29108.657       5.614   0.894
28535.962       29108.856       5.754   0.956
27589.564       29117.979       5.470   0.904
28648.180       29118.860       5.642   0.932
28897.388       29126.777       6.206   0.639
28113.946       29131.804       5.412   0.992
27992.463       29138.802       7.506   0.671
27411.264       29145.254       7.919   0.692
28043.091       29141.071       5.614   0.922
27432.000       29142.213       5.323   1.000
27205.676       29145.565       5.863   0.962
29013.543       29146.914       5.781   0.935
28555.386       29149.386       5.670   0.953
27286.366       29159.713       5.670   0.912
28799.037       29174.102       5.863   0.993
```

The first 7 lines correspond to the blob finding parameters used for automatic cell finding. This will still be present for manually found cells. Also note the 'ImageInd' is zero based (i.e. the first image is 0) and channel is encoded so that [0,1,2] -> [Red, Green, Blue]. Next, the ROI is stored as a list of xy pixel coordinates for the polygon. Since the ROI may change after blob finding, this may not represent the correct ROI used throughout the operation, but rather the most recent ROI. However, after automatic cell finding the current blobs are saved which will include the ROI. Next, the arrow symbol “->” is used to list the “filters” used during the generation of the blob list. Entries include distance filtering and histogram filtering. These are not used in the software logic but provide a limited record of parameters used for generating a list of blobs. Finally, the x,y coordinate, radius and circularity of each blob are recorded.

There are three options for saving the blob lists: (1) The *Save Current Blobs* option under the *File* menu, (2) *Histogram Divisions*, and (3) *All lists of blobs*. The first option will launch a save file dialog, allowing the user to specify the filename to save the current blob list. This is useful if only the last step of filtering and processing is needed or if only one list was utilized. The second option, histogram divisions, will be covered in more detail later, but this option saves the low and high intensity populations as separate files with encoded filenames. When the user selects a file, it is used as a base file name with additional information added to the end. Note that because no actual filtering was performed, the final divisions are not included in the filters list. Finally, the last option, saving all lists of blobs, provides a quick way to export all blob lists. Again, the user-specified file is utilized as a base file name and each list number is saved as <BASE>_<LIST NUMBER>.txt in the specified directory. The list number is again zero based so that the first list is saved as <BASE>_0.txt.

Previously generated blob list files can be loaded back into microMS. This restores the blob finder parameters from the file, the ROI, the filter list and the collection of blobs. To open a blob file, select the *Load/Found Blobs* under the *File* menu. The contained blobs will be used to populate the currently selected blob list, overwriting any existing information.

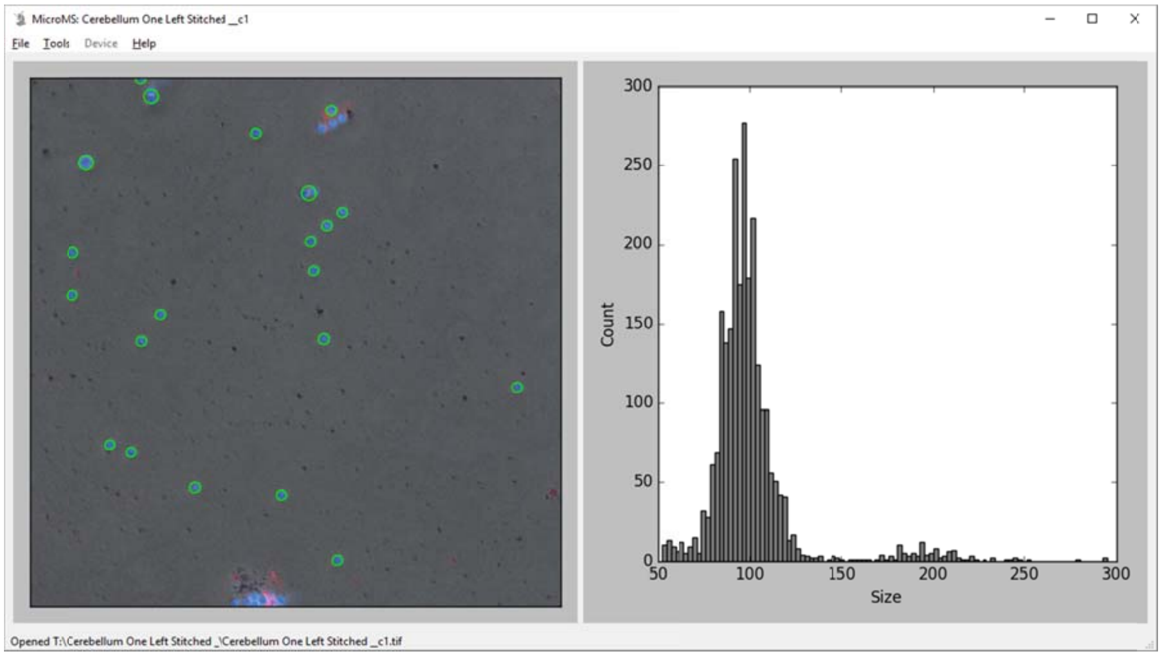
Filtering and Stratifying Blobs

After all blobs have been found or manually selected, it is frequently useful to begin segregating different classes and remove uninteresting blobs. In addition to refining the ROI as mentioned above, microMS provides filtering based on size, circularity, pairwise distance, and fluorescence channels. All of these metrics can be examined and filtered by interacting with the population level histogram. Additionally, pairwise distance filtering is applied by selecting the *Distance Filter* under the *Tools* menu.

Introduction to the histogram

The histogram interface of microMS provides an interactive method for examining population-level statistics of the blob list. Details of each population metric are described in more detail below. This section will discuss how to set, interact with and utilize the histogram.

Once a list of blobs is generated, the histogram is activated by selecting the *Histogram Window* option under the *Tools* menu or *Ctrl* + H. Changing blob lists, opening a new blob list, or performing blob finding will have the histogram be recalculated. Opening a new image, manually adding a blob, or performing blob patterning will close the histogram window. By default, the distribution of sizes of the current blob list will be shown when the histogram is initially opened.


MicroMS: Cerebellum One Left Stitched _c1

Opened T:\Cerebellum One Left Stitched _\Cerebellum One Left Stitched _c1.tif

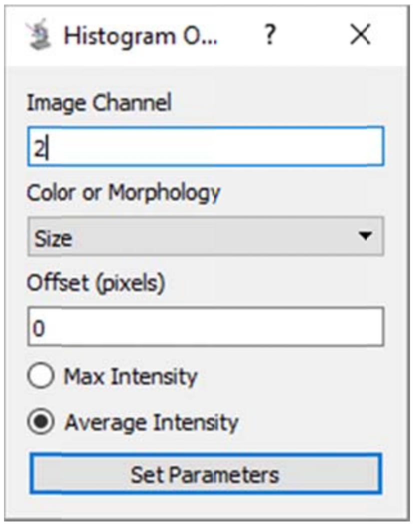

Histogram O...

Image Channel: 2

Color or Morphology: Size

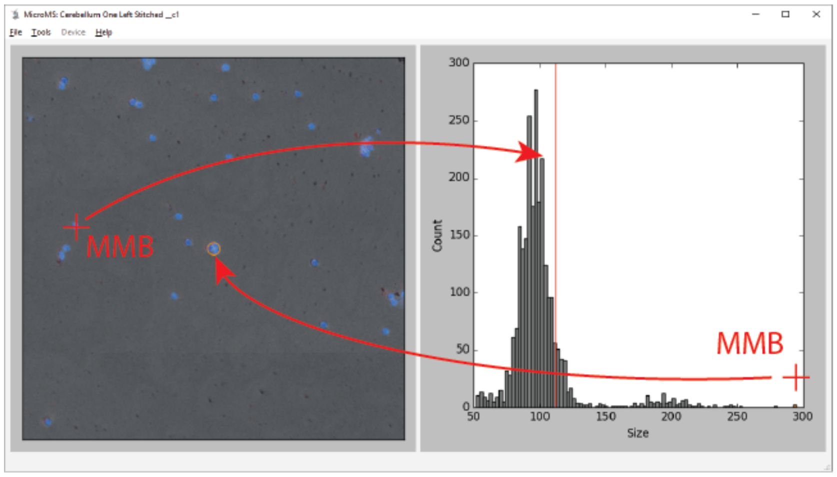
Offset (pixels): 0

Max Intensity [✓ Average Intensity]

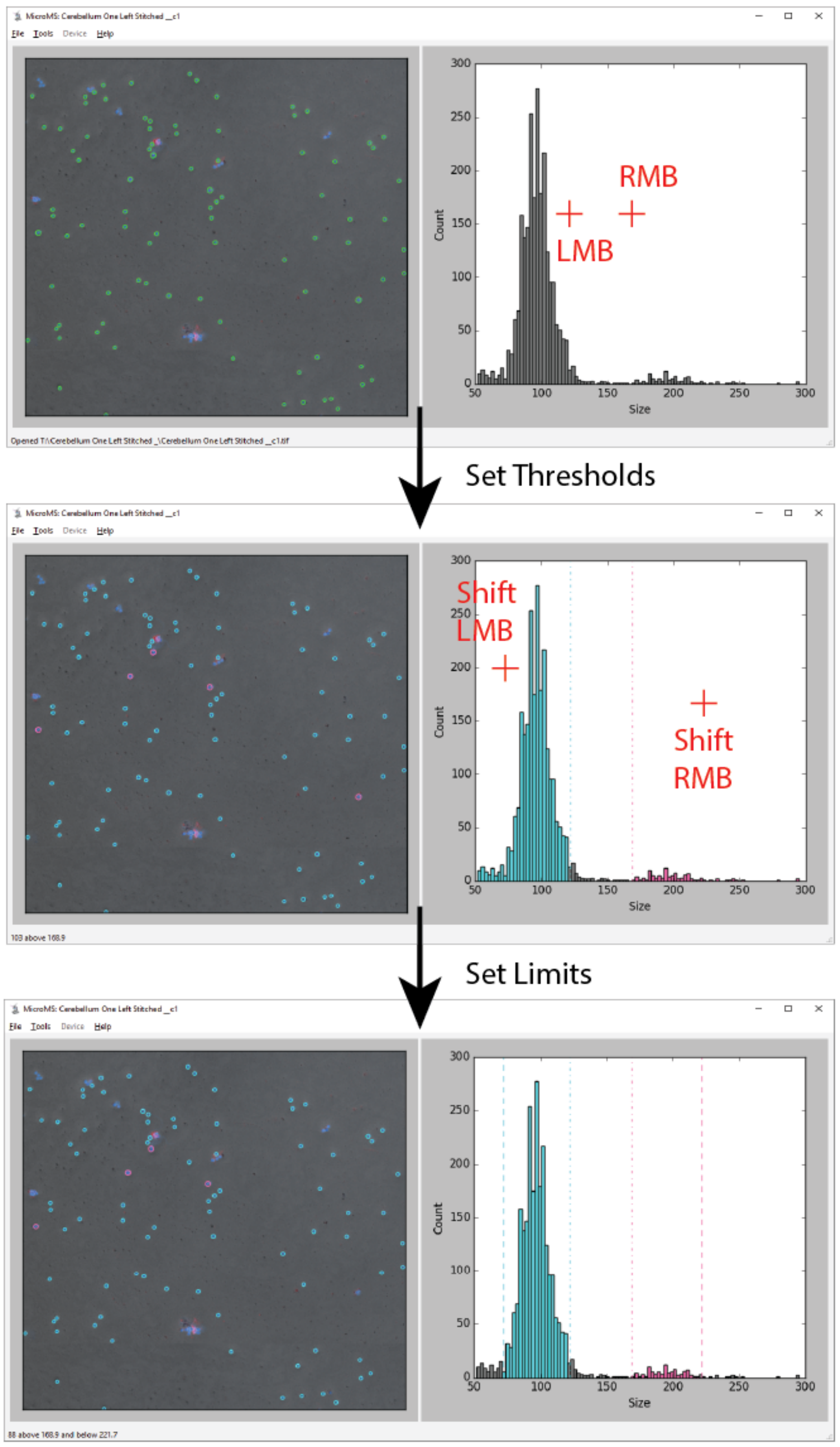
The above histogram shows a sample distribution where cells were found with a size less than 300 pixels (in area). Clearly, there are two populations present, those with areas ~100 pixels and some approximately twice as large. Using the mouse scroll wheel zooms the x axis in and out. The histogram settings are accessed in *Histogram Options* under the *Tools* menu. Here the metrics used for generating the histogram and some useful parameters are selected. Under *Color or Morphology* the fluorescence channel, size, circularity or distance may be selected. The remaining options relate solely to fluorescence intensity. Image channel is the image to examine for parsing fluorescence intensities, where 1 refers to the brightfield image. The corresponding color channel must also be specified. For example, if you wanted to examine the red intensity, collected in sample_c3.tif, image channel should be "3" and 'Color' must be red. Offset corresponds to the amount to increase (> 0) or decrease (< 0) the blob radius when examining the blob region. Finally, the intensity to display may correspond to the maximum or mean intensity within the blob region. To improve speed of analysis, the intensity corresponds to the entire circumscribed *square* for each blob. As such, neighbors at the “corners” of each region may skew results. Furthermore, the mean intensity has a dependence on the size of circular objects.

At any time, the values displayed in the histogram and the histrogram image may be saved by selecting *Save/ Histogram Image* or *Save/Histogram Values* option under the *File* menu. The image is a png of the current figure with all markup described below. Values are tab deliniated text files with the name (x,y coordinates) and corresponding metric value for each blob.

The histogram and slide image interact with each other to assist with picking values for filtering the population. A middle mouse button (MMB) click selects a single blob in the image or bar in the histogram. When a blob is MMB-selected, its population metric (e.g. size) is shown on the histogram as a red, vertical line. This is helpful to assess where in the histogram certain blobs are located. MMB clicking a bar on the histogram shows just the blobs falling in that range of values. The bar and blob are both colored orange. Additionally, the image view is centered on the first blob falling in that range. Showing a single bar is helpful to see what blobs look like which fall in a specific bin in the histogram. To clear this, and any histogram filters, press C.



Examining single blobs and bars are useful for determining threshold values, but cannot be used as filters. Two independent filters are provided to partition the blob populations. Nominally they correspond to high and low pass filters, but they can overlap and function similarly. The low pass filter is activated with the left mouse button, high pass with the right mouse button. These set a high or low pass threshold for the histogram and cause the image to redraw, showing the blob locations satisfying the filter. The threshold values are shown as vertical lines and bins of the histogram within range are colored to match the corresponding blobs. The value selected and the number of blobs within range are also displayed on the status bar. High and low *limits* are applied for the low and high pass filters, respectively, by performing a *Shift*+ Left or Right Mouse Click at the desired location.

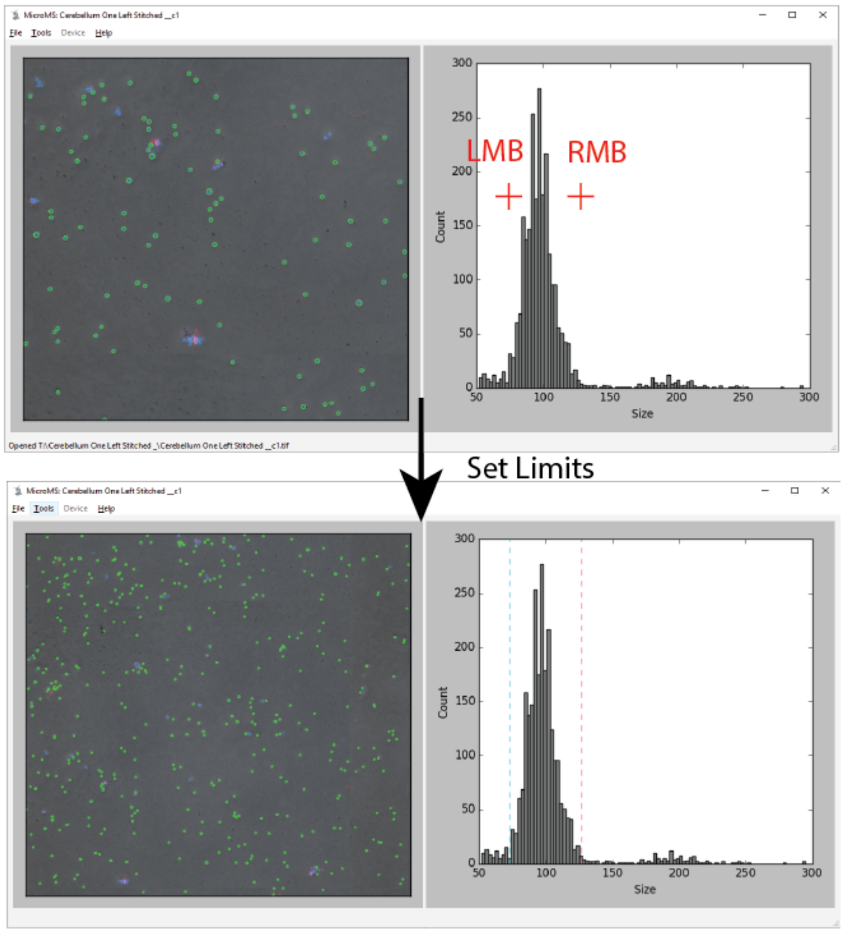


```
Opened T:\Cerebellum One Left Stitched _\Cerebellum One Left Stitched __c1.tif
```

```
103 above 168.9
```

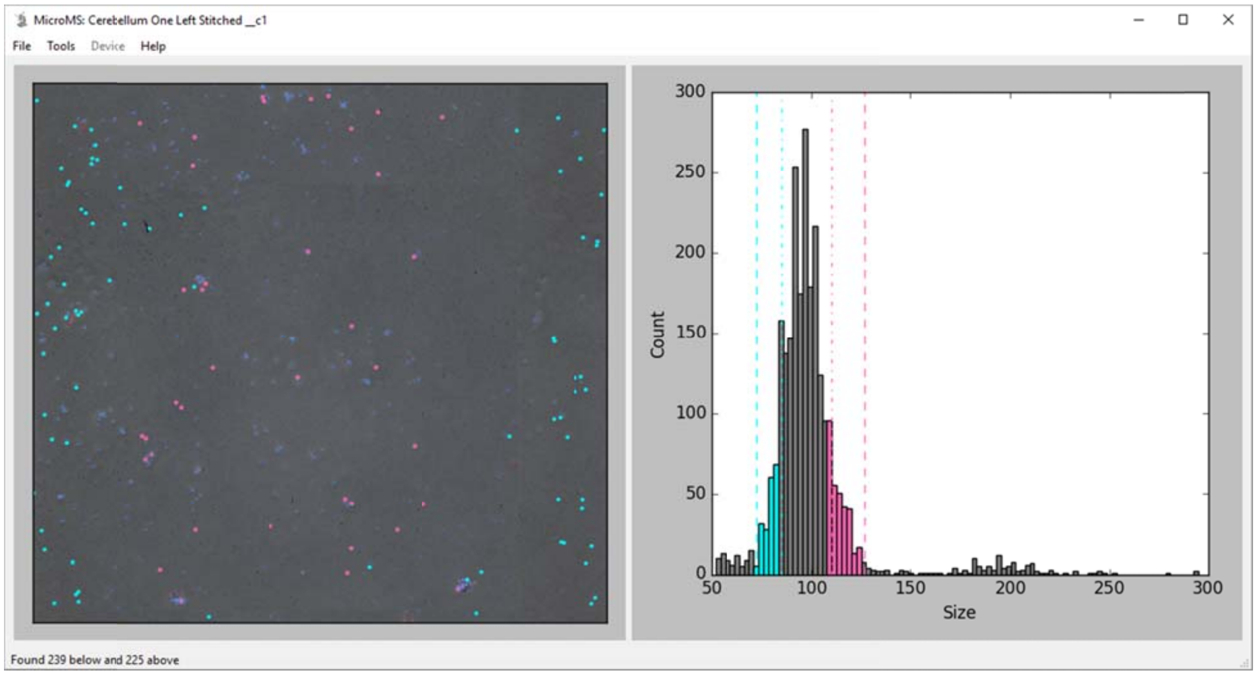
```
88 above 168.9 and below 221.7
```

Filters can also be generated automatically, which is useful for examining a set number of extreme members within a population. In the example above, there appear to be two populations. The following steps illustrate how the largest and smallest cells (250 cells for each category) within the population centered on 100 pixels are selected. First, select high and low *limits* with *Shift* clicking (LMB = left mouse button, RMB = right mouse button).


MicroMS: Cerebellum One Left Stitched __c1
File
Tools
Device
Help
LMB
RMB
Count
Size
Set Limits
MicroMS: Cerebellum One Left Stitched __c1
File
Tools
Device
Help
Count
Size

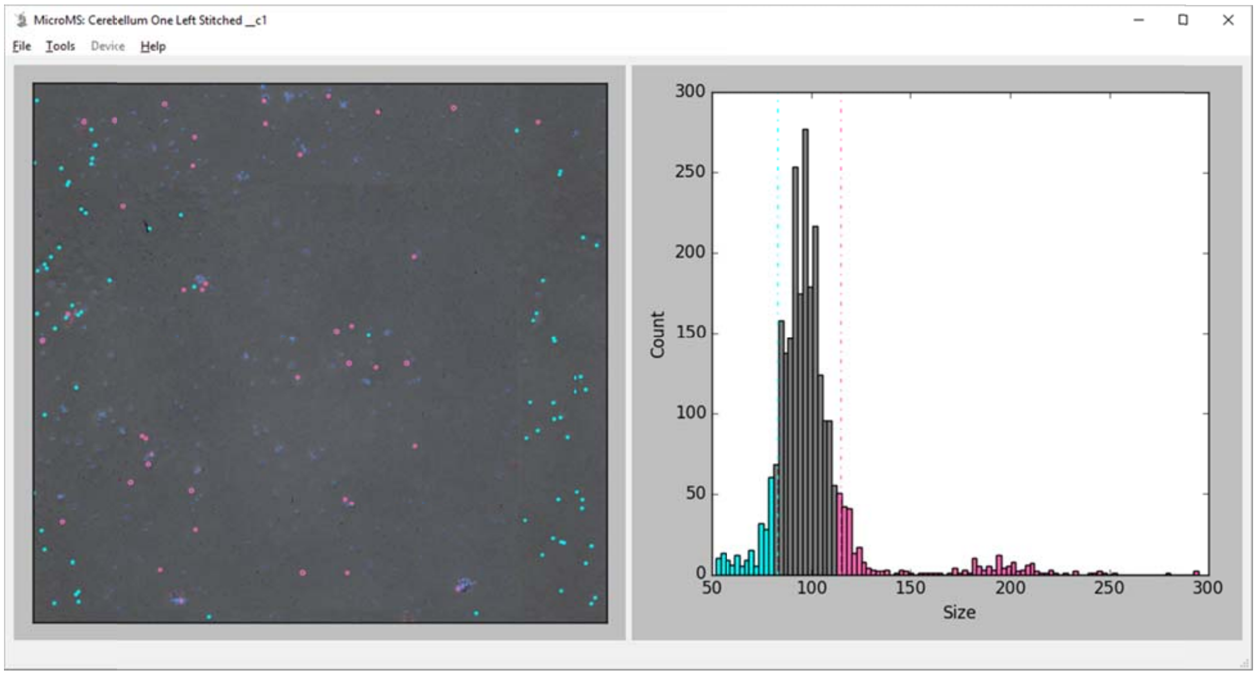
Opened T:\Cerebellum One Left Stitched _\Cerebellum One Left Stitched __c1.tif

Note this step does not set any filters, only defines limits. The extremes within these limits (or the entire population, if no limits are set) are found by selecting the *Pick Extremes* option under the *Tools* menu. This launches a popup box requesting the number of blobs to try and find for each filter; in this example, '250' should be entered. Next, microMS attempts to find the histogram divisions that provide approximately the requested number of cells in the high and low range of the population. The actual numbers in each filter are displayed in the status bar.



Found 239 below and 225 above

Because this distribution is fairly symmetric, the locations of the thresholds are approximately the same distance from the limits. However, repeating the request without set limits produces the following:

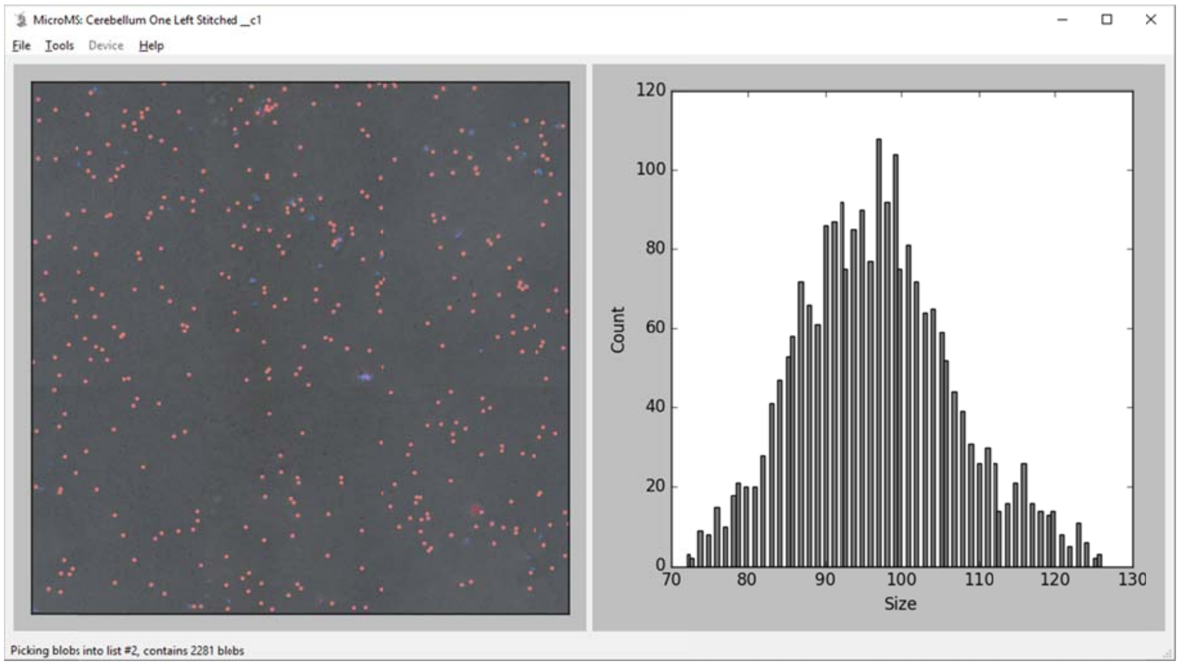


Where the range of values included in the high threshold is much larger to obtain the same number of blobs as the low threshold.

Using the ranges: saving and filtering regions

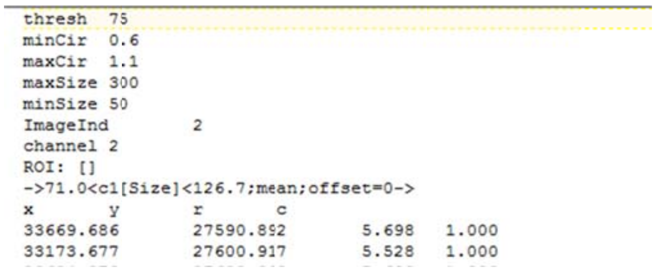
With the appropriate filters in place, their output is either saved as separate blob finding files or used to generate additional blob lists for further filtering. To save the current histogram filters, select *Save/Histogram Divisions* under the *File* menu. This prompts the user for a base name to save the resulting text files. For each histogram division, a text file is saved with the name <BASE>_<high or low>_<VALUE>.txt where BASE is supplied by the user, high or low depends on the filter division, and VALUE is the high or low *threshold* value defining the region. To utilize the output for performing measurements, the generated cell lists must be loaded as separate lists to generate an instrument file.

Alternatively, the filters can generate new blob lists for further filtering or generation of separate instrument files. To utilize this feature, select the ranges of interest and select the *Apply Filter* option under the *Tools* menu.



Picking blobs into list #2, contains 2281 blobs

Each filter generates its own blob list and fills an empty list. The new list allows the application of additional filters prior to saving. Note that unlike saving histogram divisions, the filter set used to generate the blob list *is* stored in the blob finding file. For example, the list above looks like:

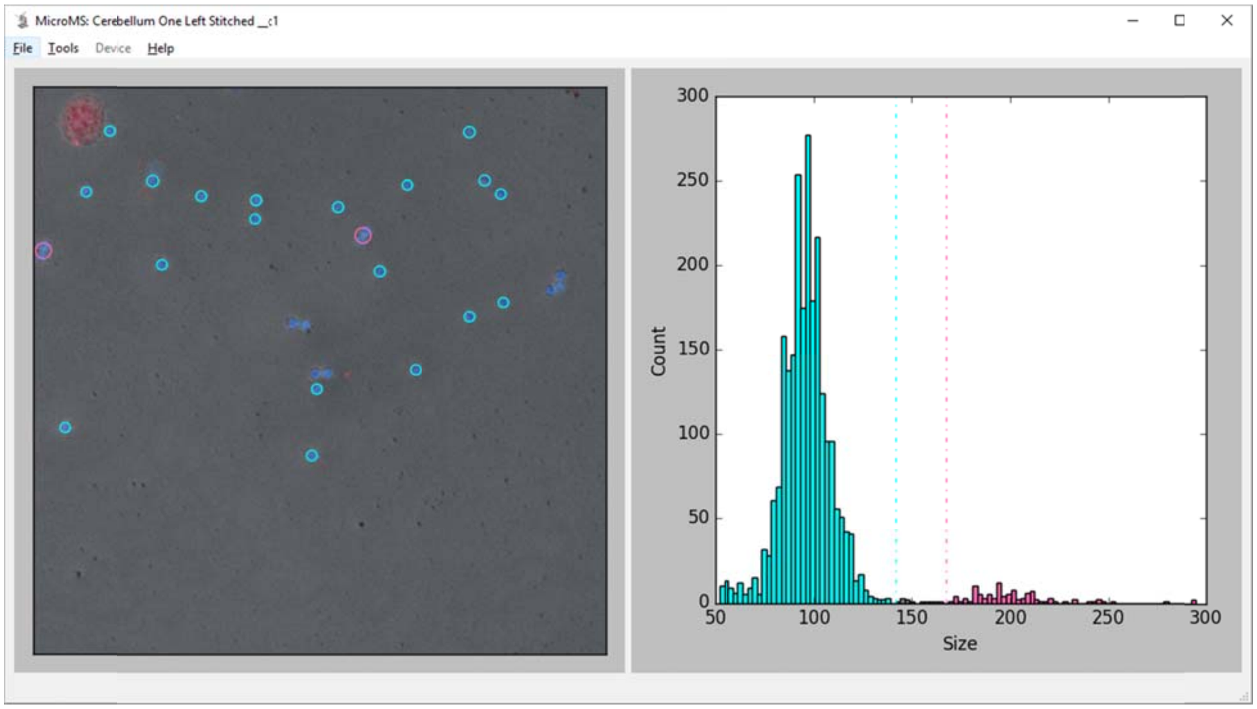


```
thresh  75
minCir  0.6
maxCir  1.1
maxSize 300
minSize 50
ImageInd        2
channel 2
ROI: []
->71.0<c1[Size]<126.7;mean;offset=0->
x       y       r       c
33669.686       27590.892       5.698   1.000
33173.677       27600.917       5.528   1.000
```

Where the line "`->71.0<c1[Size]<126.7;mean;offset=0->`" indicates image channel 1 (c1) was used to filter using Size between 71.0 and 126.7, with a mean reduction and offset of 0. Additional filter steps will be listed in the order they were performed to show what actions produced the current blob list.

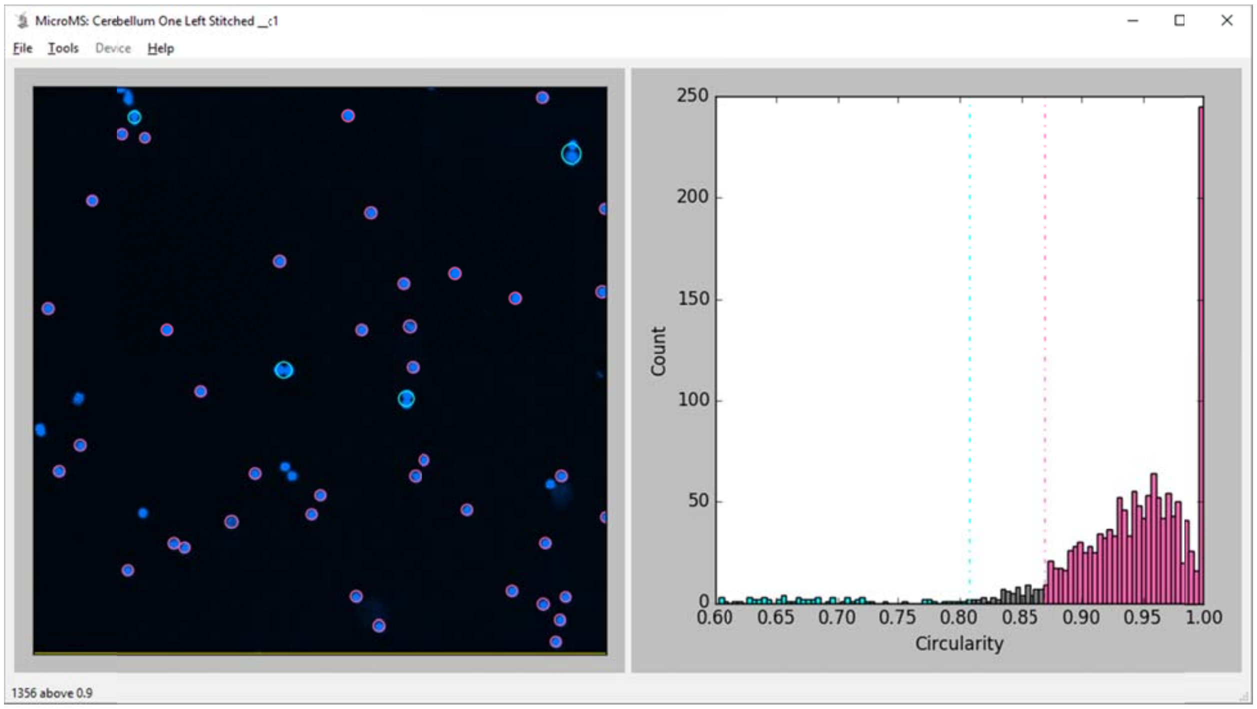
Morphology: Size and circularity

Note that size and circularity effectively provide the same filtering as blob finding. However, it is frequently useful to perform automatic blob finding with less stringent metrics and refine the population later. Following this protocol ensures that distance filtering does not skip neighbors that are clustered together. Using the same population as above:



Notice that dividing the population in half on size allows the identification of cells too close to resolve during cell finding (shown in pink) while individual cells are highlighted in teal. However, the cells near the center of the image are not identified because *they were not located during blob finding*. As such, they will not be considered during distance filtering.

Circularity is also helpful to identify unresolved cells or other debris:



Now the unresolved cells are shown in the low pass filter because they have low circularity. Examining the size and circularity helps ensure data quality by removing unresolved features and other imaging artifacts.

Distance filtering

Proper distance filtering is crucial to ensure acquired spectra belong to a single blob. The exact value will depend on the instrument position accuracy, probe size, and any suspected analyte delocalization during sample preparation (e.g. MALDI matrix application). Generally, the distance filter should be larger than the probe size to prevent contamination during acquisition. Only found blobs will be considered with distance filtering. At any time, a set value can be used for distance filtering in the *Distance Filter* under the *Tools* menu. Upon completion, the blobs passing the filter will be copied to the next open blob list. Alternatively, an appropriate distance filter may be investigated by using the histogram window. When selected, the minimum pairwise distance is calculated. Generally, the distribution should be Poisson. Large populations of blobs very close together indicate a high seeding density. By selecting high pass filters, microMS will report the number of blobs passing the filter in the status bar and display their locations.

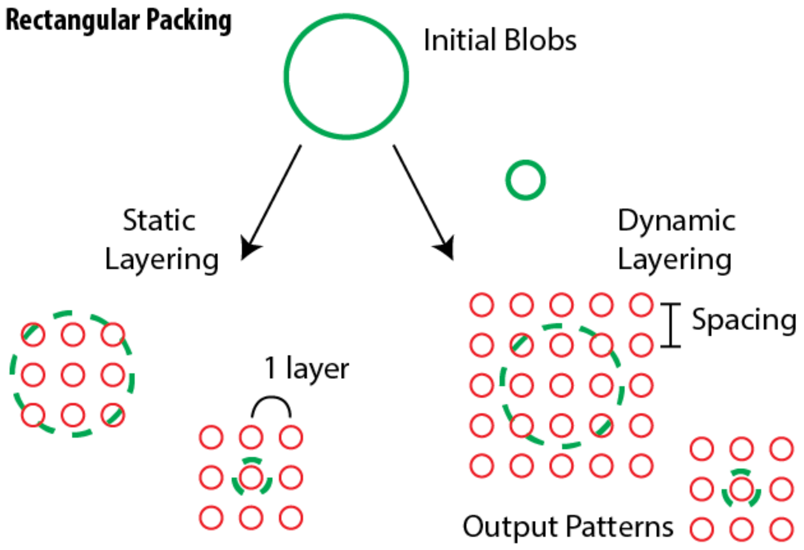
Fluorescence intensity

While the previous metrics provided sample quality checks, fluorescence intensity helps generate orthogonal "labeling" of blobs even before mass spectral acquisition. Additionally, examining the same fluorescence channel as used for blob finding helps remove dim blobs from consideration. Generating the intensity histograms will be slower than morphology due to the required image analysis steps.

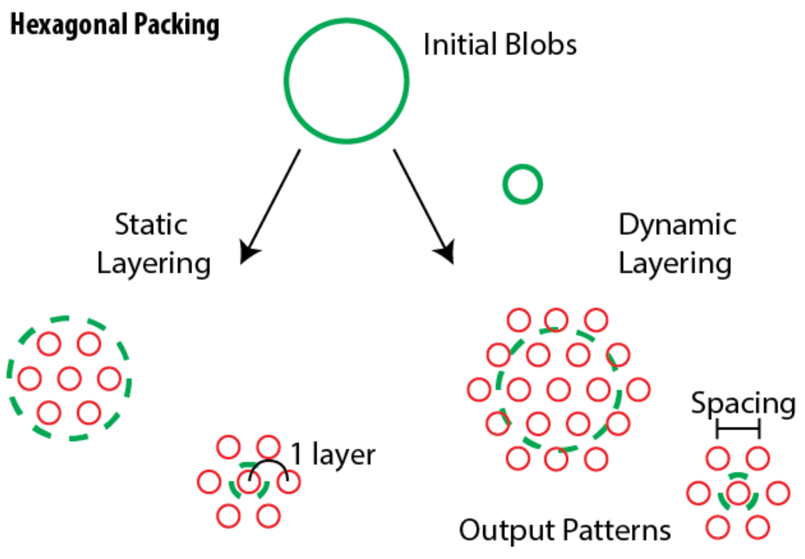
Blob patterning

Once blobs have been found and filtered, it is occasionally useful to further pattern target positions, either to acquire an "image" of the area surrounding each blob or to more effectively sample blobs much larger than the probe size. Three different packing patterns are available for performing blob patterning: circular, rectangular, and hexagonally close packed. Each pattern is accessed through the *Tools* menu and prompts the user for additional packing parameters. The resulting patterns are stored in a new blob list which consists of x,y points, a single radius, circularity of 1, and a group number which uniquely ties each pattern to a parent blob. Note that generated patterns may cause overlap of target positions.

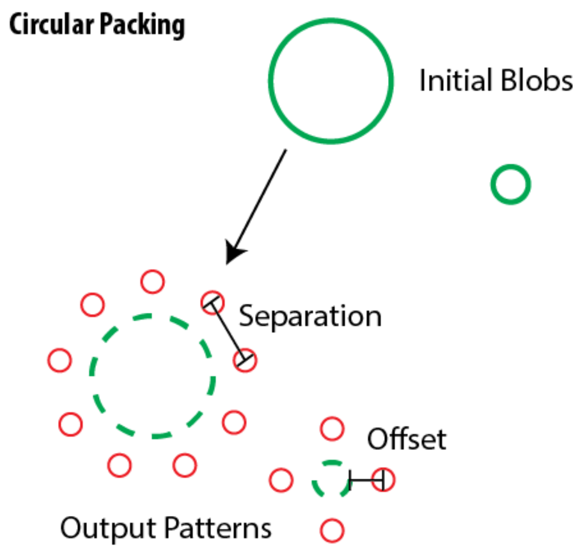
Rectangular packing generates even x,y spacing in a grid of target positions, effectively allowing the generation of mass spectral images over each blob. Rectangular packing requires three parameters: spot to spot distance, number of layers, and whether to perform dynamic layering. The spacing dictates the pitch, or pixel size, of the resulting image. The number of layers corresponds to the number of horizontal/vertical pixels from the center to generate. A value of 0 results in a single target per blob, 1 results in 9 targets, 3 in 25 targets, etc. Due to the way the patterns are generated, acquisition proceeds by spiraling out from the center position. With static layering, all resulting patterns will be the same size, regardless of blob shape. If dynamic layering is chosen, the number of layers will be adjusted for each blob to ensure that the entire area is covered in targets. In this case, the input number of layers controls the *extra* layers to include during patterning.



The same set of parameters is required for hexagonal and rectangular packing, except instead of having constant x,y spacing, targets are positioned in a hexagonally close packed arrangement to maximize the number of acquisitions per area. Resulting data can provide mass spectral images, but most analysis software assumes rectangular packing.



Finally, circular packing allows the analysis of the circumference of blobs. This allows an efficient sampling of compounds which migrate from blob locations or are only found around the exterior of blobs. Unlike the previous packing, circular packing requires the minimum separation, maximum number of targets per blob and an offset from the circumference. There is also a minimum number of targets which is set to 4 by default and must be adjusted in the source code.



In circular packing, the number of targets is always dynamically adjusted based on the blob radius plus the user supplied offset; negative values of offset place targets in the blob interior. Very large blobs will have the maximum number of targets placed evenly around their circumference. Smaller blobs will have fewer targets, but will maintain the minimum separation between targets. Very small blobs will have the minimum 4 targets, regardless of the resulting target separation. This strategy provides several replicates per blob, but prevents repetitive acquisition for larger blobs.

Instrument correlation

Once all targets are found, filtered and patterned, the blob information needs to be translated to instrument input. MicroMS provides an interface for performing instrument integration either offline or with direct instrument control. Full instrument control requires significant extensions of the connected Instrument interface and will differ dramatically between instruments. As such, this section will only cover aspects applicable to all instruments.

Point-based similarity registration and fiducials

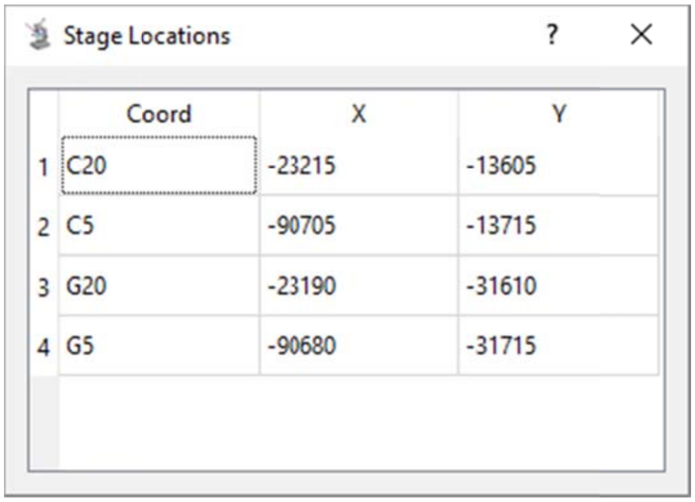
At its core, microMS utilizes a point-based similarity registration to map a set of fiducials between physical space and an image coordinate system. The target locations in physical space are then inferred from their locations in the image using a linear coordinate transformation. The specific registration accounts for translation, rotation and scaling. Some limited support is available for reflections, but no corrections are made for skewed perspectives.

Accurately analyzing target locations depends on the precision of the sample stage, the microprobe size, correct stitching of optical images, and accurate estimations of the fiducial locations. The location of fiducials has drastic effects on accuracy and therefore requires care. We have successfully utilized etched fiducial markers in the shape of an X. Location of the intersection of the two lines can be performed accurately and is less susceptible to distortion between image systems, particularly after MALDI matrix application. Other options include placement of dyes/paints, selective laser ablation, or beads. Generally, smaller fiducials are located with higher precision, but they must be large enough to locate on the instrument camera system. Ideally, the fiducials would fluoresce in the same wavelength as the blobs, otherwise multiple image channels must accurately overlay.

No assumptions are made regarding the instrument besides the basic requirements that fiducial locations must be found and recorded, and the instrument must be directed to arbitrary target locations. Frequently, several intermediate steps are required to accomplish these goals, even in the simplest cases. Each instrument has its own coordinateMapper which defines instrument-specific functions for interacting with microMS. Some instruments have multiple coordinateMappers if different instrument control software is targeted (e.g. the solariX targets positions through autoexecute or flexImaging functions). The current instrument is displayed and changed under the *Instrument* option under the *File* menu.

Instrument settings and intermediate coordinates

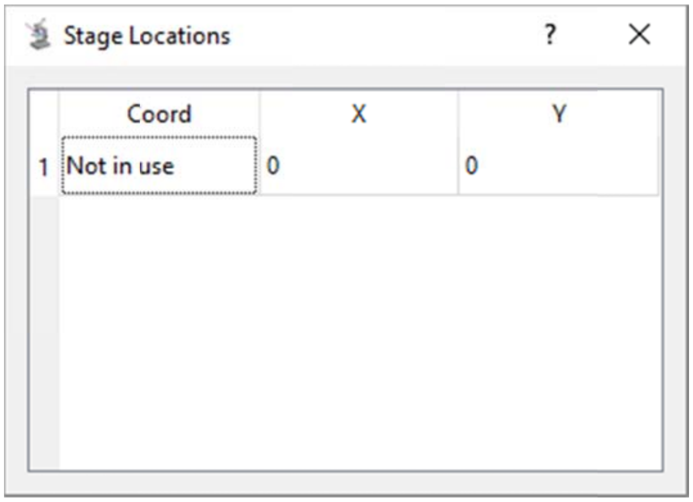
Occasionally instruments utilize more than one coordinate system for physical locations. For example, Bruker MALDI instruments provide direct output of the 2D linear stage positions. However, the coordinates used to direct motion during automatic acquisition are scaled fractions of the entire sample plate. In this case, microMS must utilize an additional coordinate transformation to map between the intermediate, motor position and the final, fractional distance position. To adjust or calibrate these positions, microMS provides a simple interface displaying set coordinates:


Stage Locations

	Coord	X	Y
1	C20	-23215	-13605
2	C5	-90705	-13715
3	G20	-23190	-31610
4	G5	-90680	-31715

Here C20 refers to a set position in the ultrafleX software, and when the stage is directed to that point, the motor coordinate has the x,y position of -23215, -13605. The coordinate mapper then incorporates this information to generate a new intermediate map between the motor coordinates and the final output. These values rarely change, but adjusting the motor stage or using a different set of teaching points cause them to move and should be calibrated occasionally. Closing the window updates the information of the coordinate mapper and a txt file in the source folder.

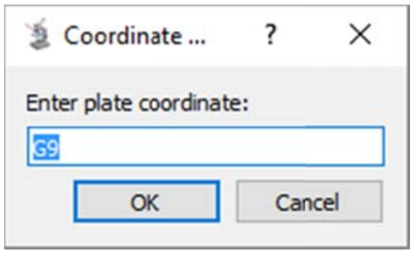
Some coordinate mappers do not require this intermediate map and will display that these values are not used:


Stage Locations

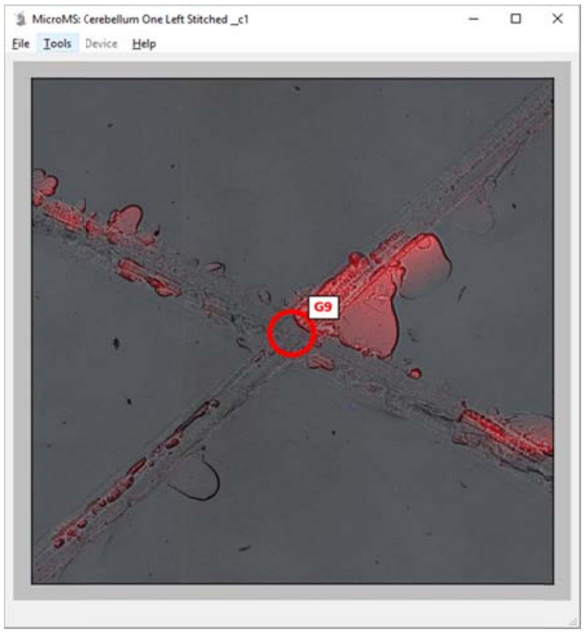
	Coord	X	Y
1	Not in use	0	0

Interacting with fiducial markers

Accurate fiducial training requires precise location of the fiducial on the instrument and the image. Once found in the instrument, the physical position is input into microMS by RMB on the image. This causes a popup window to appear to input the physical coordinate:

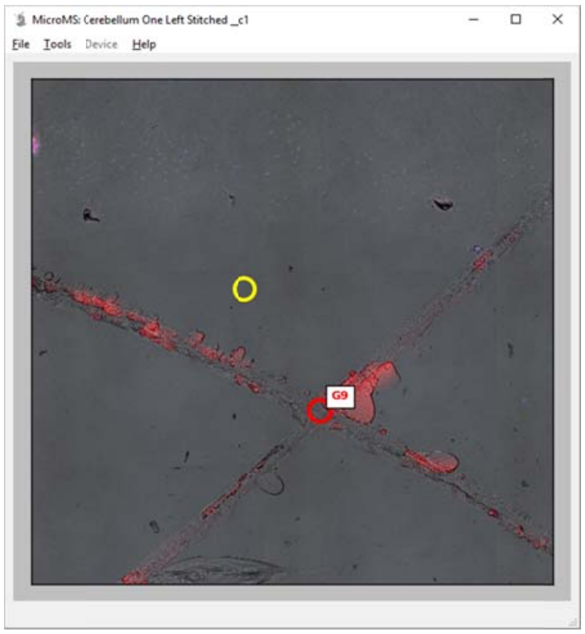


A predicted location is displayed for some instruments (G9 above) but should only be used as a quick way to assess if the registration is drastically incorrect. This will only be available after at least 2 fiducials are trained. Entering the coordinate and clicking "*OK*", or pressing enter, will record the fiducial:

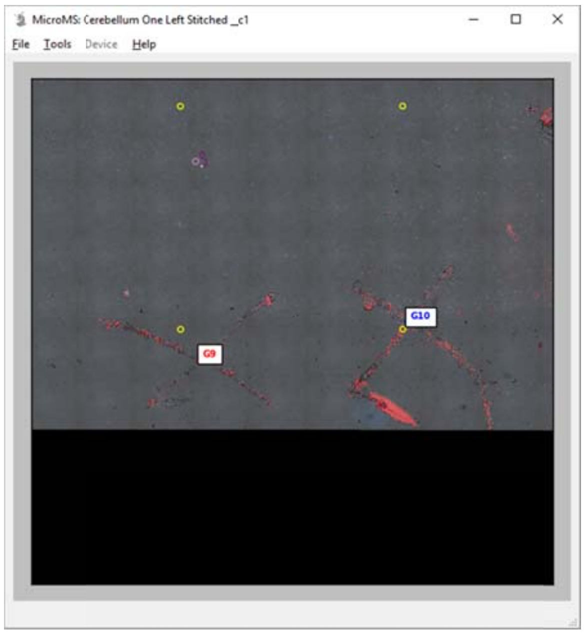


Fiducials are displayed as either red or blue circles with text indicating the closest predicted point. If the input value cannot be parsed correctly, an error message will appear on the status bar and the fiducial will not be recorded. Fiducials are removed by holding *Shift* and RMB.

To help locate fiducials in the instrument, microMS displays predicted plate locations for some instruments. These represent set points on the stage which are encoded positions and easily located. Here the predicted location of G9 is shown in yellow:



The quality of the fiducial set is assessed interactively through predicted points. As new fiducials are supplied, the predicted points are updated. Large movements of predicted locations or deviations between actual positions indicate poor accuracy. microMS also tries to detect fiducial training errors by highlighting the fiducial with the worst fiducial localization error in red, e.g. G9 in the following image:

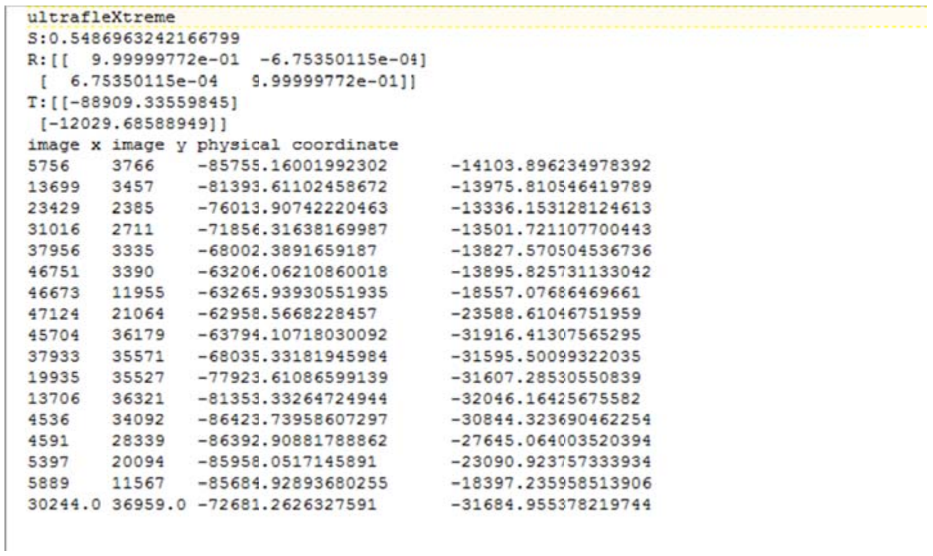


The fiducial localization error is estimated by using the current registration to predict the physical location of each fiducial based on its pixel position. The fiducial with the largest deviation is then drawn in red. While this is the "worst" fiducial mark, it is not necessarily bad or an unacceptable amount of error. Generally, the worst fiducial should be removed and reselected until it is no longer the worst. This process is repeated until the fiducial continues to be the worst, which indicates the registration can no longer be improved.

Two fiducials are required to utilize predictions, worst fiducials, and save instrument positions. Generally the error decreases by 1/sqrt(number of fiducials), so more fiducials result in higher accuracy. Accuracy also depends on fiducial location relative to the target positions. We have found surrounding the target area with fiducials produces high accuracy while minimizing possible interference of chemical information between fiducial and sample.

Saving and loading registration

The current registration information, including the instrument name and fiducial set, is saved by selecting *Save/Registration* under the *File* menu. This generates an msreg (mass spectral registration) file, which is a human readable text file.



```
ultrafleXtreme
S:0.5486963242166799
R:[[  9.99999772e-01  -6.75350115e-04]
 [  6.75350115e-04   9.99999772e-01]]
T:[[-88909.33559845]
 [-12029.68588949]]
image x image y physical coordinate
5756    3766    -85755.16001992302      -14103.896234978392
13699   3457    -81393.61102458672      -13975.810546419789
23429   2385    -76013.90742220463      -13336.153128124613
31016   2711    -71856.31638169987      -13501.721107700443
37956   3335    -68002.3891659187       -13827.570504536736
46751   3390    -63206.06210860018      -13895.825731133042
46673   11955   -63265.93930551935      -18557.07686469661
47124   21064   -62958.5668228457       -23588.61046751959
45704   36179   -63794.10718030092      -31916.41307565295
37933   35571   -68035.33181945984      -31595.50099322035
19935   35527   -77923.61086599139      -31607.28530550839
13706   36321   -81353.33264724944      -32046.16425675582
4536    34092   -86423.73958607297      -30844.323690462254
4591    28339   -86392.90881788862      -27645.064003520394
5397    20094   -85958.0517145891       -23090.923757333934
5889    11567   -85684.92893680255      -18397.235958513906
30244.0 36959.0 -72681.2626327591       -31684.955378219744
```

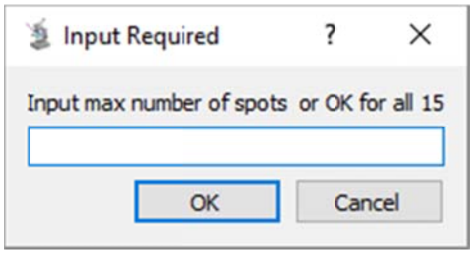
The first line has the instrument name. The next 5 lines contain variables of the linear transformation for mapping pixel positions to the physical coordinates. The remainder of the file records the pixel and physical location of each fiducial mark.

Previous registrations are loaded by selecting *Load/ Registration* under the *File* menu. This populates the fiducial training set and changes the current instrument if needed. Generally, it is not advised to reuse registration if the sample has been removed and inserted into the instrument as small changes in sample positioning manifest as systematic errors, casing a missed acquisition at every target location.

Saving and loading instrument files

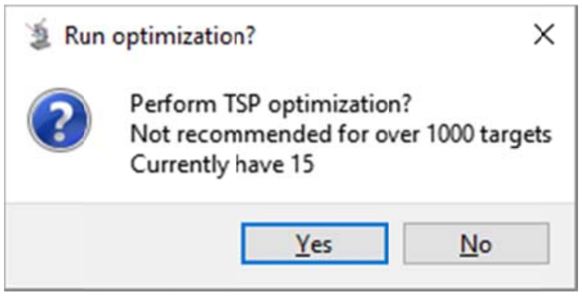
With a good set of fiducials trained, the blob positions for the instrument may be generated. Again, there is an opportunity to assess fiducial training by selecting *Save/Fiducial Positions* under the *File* menu. This generates an instrument-specific file of target locations for the fiducials. By using the fiducial file as input for the instrument, the difference between expected and actual fiducial location helps determine if accuracy is sufficient for the given experiment.

To generate target points (blob positions), select *Save/Instrument Positions* under the *File* menu. Only the current blob list will be used for generating target positions. Again, an instrument-specific file type is saved, but there are several more options during export. First, the user is prompted for the number of blob positions should be exported:


Input Required

Input max number of spots or OK for all 15

This function provides a method to randomly select a subset of the blobs for analysis, especially useful to conserve instrument time. Clicking "*OK*" with a valid number in the text box selects the specified number of blobs for analysis. Leaving the box blank uses all spots and clicking "*Cancel*" stops the export operation. Next, the user is asked about path optimization:


Run optimization?

Perform TSP optimization?
Not recommended for over 1000 targets
Currently have 15

microMS uses a simplified version of traveling salesperson path (TSP) optimization which optimizes the travel path, but bounds the optimization run time to at most 3 minutes, so the resulting path will not be fully optimal. The optimization determines the order to visit each target to minimize the total distance traveled. However, the computation requires calculating each pairwise distance between targets, effectively consuming RAM on the order of the number of points squared. Clicking "*No*" orders the points from top to bottom, left to right, and generally causes the stage to move about twice as far as the optimized path.

Sample positions can also be loaded from instrument files. Targets will retain their x,y position and group number, but will lose their size and circularity measurements. As such, if these values are important they should be loaded from a found cell file.

Advanced topics

The above should be sufficient for most users. However, the real power of microMS comes from the design choice to make instruments an abstract base class, greatly simplifying the work required to support new and diverse instruments. microMS also offers more advanced operations including direct instrument control.

Customizing GUI Settings

Several settings for the GUI are set in the file GUICanvases/GUIConstants.py. This file is fully commented with brief descriptions. The top section defines several colors for blob lists, predicted points, and fiducials. Of note, `MULTI_BLOB` contains the colors for all blob lists, in order from List 1 to 10. The `ROI_DIST` is the minimum distance between two vertices before the current vertex is removed. Lowering the value will allow drawing more complex shapes, but make deleting points more difficult. Next, some constant values are provided for the default blob radius and default fiducial radius. The default blob and fiducial values were chosen for a particular application which may not be suitable for all purposes. Setting the `DEFAULT_RADIUS` to the probe size simplifies detection of targets too close together. Fiducial radius should be approximately the size of a given fiducial mark to help assess if the fiducial was placed in the correct position in the image. Next, the `DRAW_LIMIT` and `TSP_LIMIT` define limits on the maximum number of blobs for computationally expensive operations. More powerful computers can increase these values as needed, or if higher performance is required they may be decreased. `DRAW_LIMIT` defines the maximum number of blobs to draw from each list. This is overridden in the menu bar option. `TSP_LIMIT` defines how long a blob list to consider for TSP optimization be default. Again, this can be bypassed in the GUI when saving instrument positions, but this acts as a simple guard to consuming too much memory.

The next section lists colors utilized in drawing the population level histogram. These are all aesthetic changes and do not affect function of the histogram. The constants for blob shapes allows further customization of the default blob size, for either manually drawn targets or automatically generated patterns of targets.

Next, several files and directories are defined for performing standard debugging loads. These assist in opening a "standard" image data set for testing new features or replicating bugs. Once all files are defined, the debug data set is opened by pressing *Ctrl* + D. This is only operational when microMS first opens and if all files exist on the current machine.

Supporting new instruments

The goal is that a user with moderate python experience will be able to support new instruments in the future while maintaining the image analysis functionality of microMS. Hopefully, this section will act as a template for generating offline instrument coordinate mappers with arbitrary systems. The details are not important for a general reader, but should help guide more advanced users to support their own instruments. This section will demonstrate how to support two new instruments, a hypothetical Generic XYsampler and equally-absurd Bruker flexArmstrong.

microMS coordinate mapper organization and requirements

The main GUI of microMS interacts with coordinate mappers through the supportedCoordSystems.py module. This initializes new instances of each supported mapper and generates their names for display. Each member in supportedMappers must inherit from the abstract base class defined in coordinateMapper.py. This module defines all necessary functions to fully leverage microMS and implements some basic functions. CoordinateMappers may also have an instance of connectedInstrument.py (another abstract base class) to interface with instruments. However, implementing direct control will not be covered here as it is specific for each instrument.

Due to the shared characteristics between Bruker MALDI mass spectrometers, another abstract base class is included with microMS, brukerMapper.py. This contains implementations for many of the functions specified in coordinateMapper and defines a smaller set of functions required for off-line analysis with Bruker instruments.

Implementing coordinateMapper for the Generic XYsampler

coordinateMapper.py contains the required methods and information on how to implement them. To reemphasize, inheriting classes must:

- Define self.instrumentExtension and self.instrumentName
- Implement isValidEntry, extractPoint, predictName, predictLabel, predictedPoints, loadInstrumentFile, saveInstrumentFile, getIntermediateMap and setIntermeidateMap.
- Add an import and initialize an instance in supportedCoordSystems.

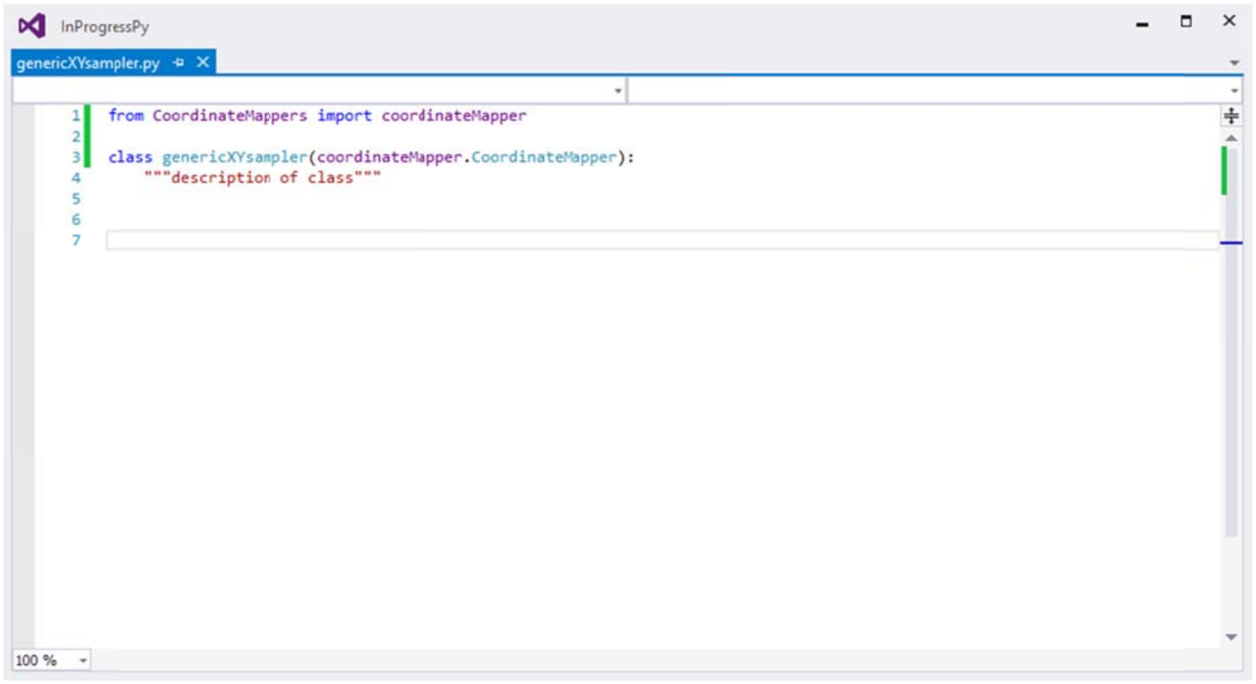
Note that any methods not overridden will cause an error immediately upon running microMS.

The Generic XYsampler is a fictional, new mass analyzer with some interesting requirements:

- Motor positions are read from the instrument, and are of the form <Xcoordinate>_<Ycoordinate> as floating point numbers. When moving down in the image, the Y coordinate increases.
- There are 25 set points in a 5x5 grid at motor positions 0..100..400 labeled A-Y from left to right, top to bottom.

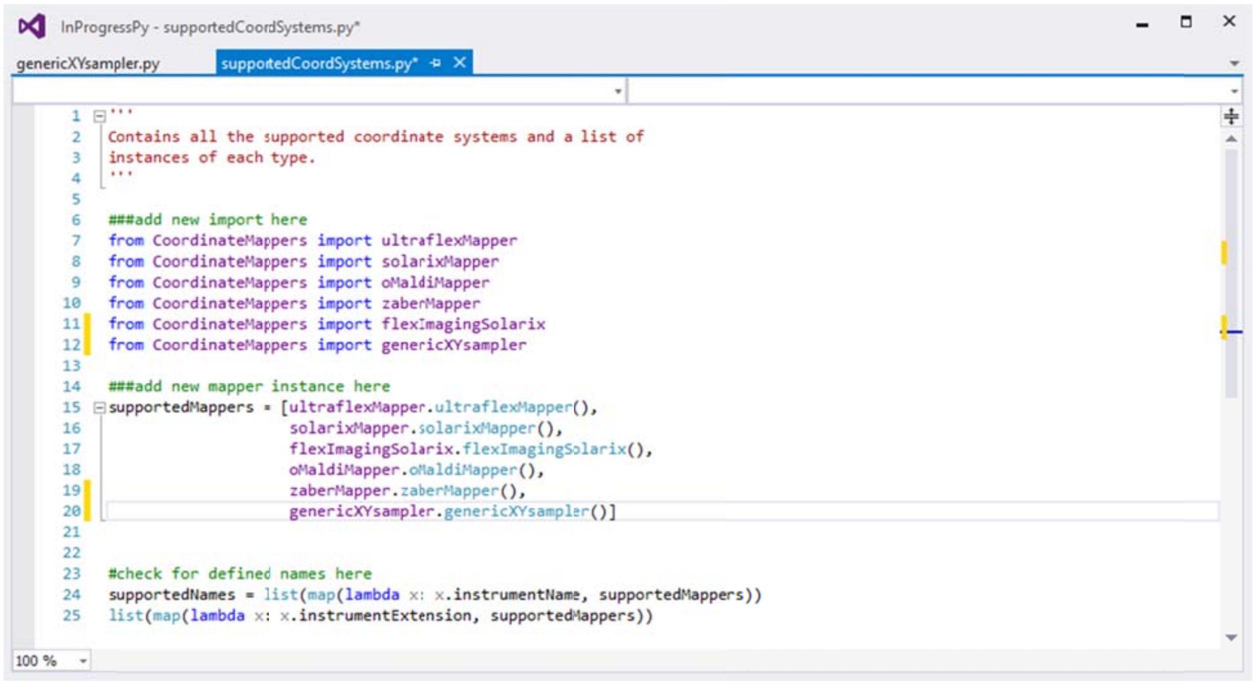
- Instrument files are in .csv format, with the first column the sample name, the next two x and y coordinates in “GenericCoordinates”.
- GenericCoordinates are an offset of +100 in both directions, but could change on leap years.

Start by making a new python class in the coordinateMappers package called genericXYsampler.py that inherits from coordinateMapper.



```python
from CoordinateMappers import coordinateMapper

class genericXYsampler(coordinateMapper.CoordinateMapper):
    """description of class"""
```

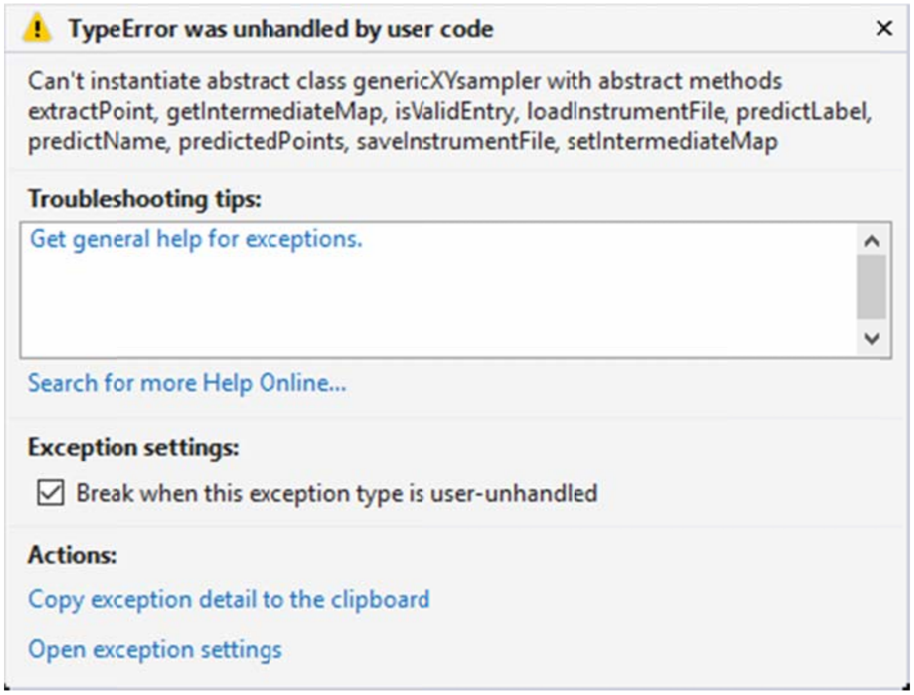
And add the genericXYsampler to the supportedCoordSystems


InProgressPy - supportedCoordSystems.py*

genericXYsampler.py | supportedCoordSystems.py*

```
'''
Contains all the supported coordinate systems and a list of
instances of each type.
'''

###add new import here
from CoordinateMappers import ultraflexMapper
from CoordinateMappers import solarixMapper
from CoordinateMappers import oMaldiMapper
from CoordinateMappers import zaberMapper
from CoordinateMappers import flexImagingSolarix
from CoordinateMappers import genericXYsampler

###add new mapper instance here
supportedMappers = [ultraflexMapper.ultraflexMapper(),
                    solarixMapper.solarixMapper(),
                    flexImagingSolarix.flexImagingSolarix(),
                    oMaldiMapper.oMaldiMapper(),
                    zaberMapper.zaberMapper(),
                    genericXYsampler.genericXYsampler()]


#check for defined names here
supportedNames = list(map(lambda x: x.instrumentName, supportedMappers))
list(map(lambda x: x.instrumentExtension, supportedMappers))
```

Running microMS causes the following error:


TypeError was unhandled by user code

Can't instantiate abstract class genericXYsampler with abstract methods extractPoint, getIntermediateMap, isValidEntry, loadInstrumentFile, predictLabel, predictName, predictedPoints, saveInstrumentFile, setIntermediateMap

Troubleshooting tips:

Get general help for exceptions.

Search for more Help Online...

Exception settings:

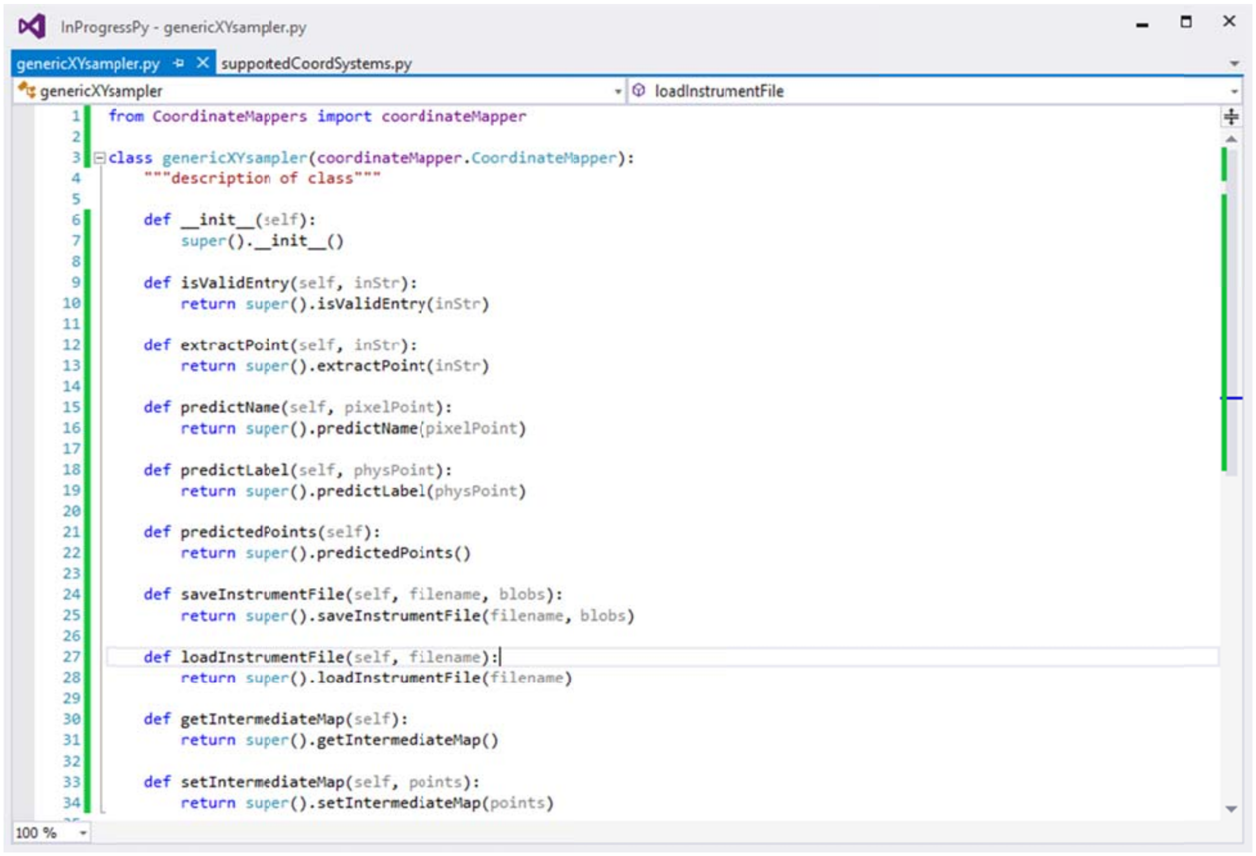
☑ Break when this exception type is user-unhandled

Actions:

Copy exception detail to the clipboard

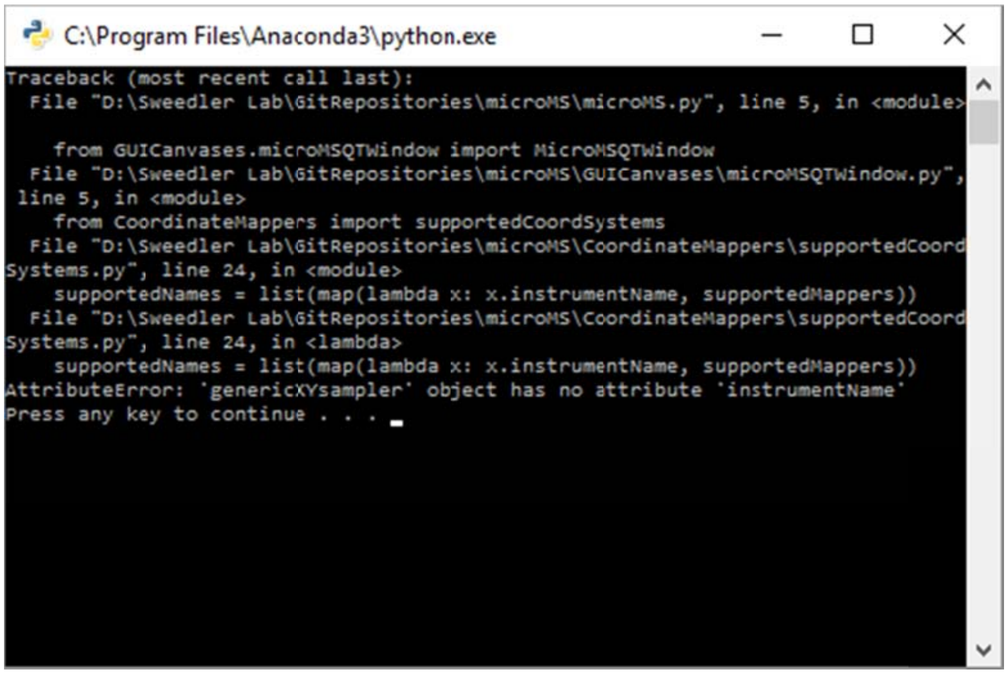
Open exception settings

Next, fill in genericXYsampler with some unimplemented methods for each of the abstract methods:



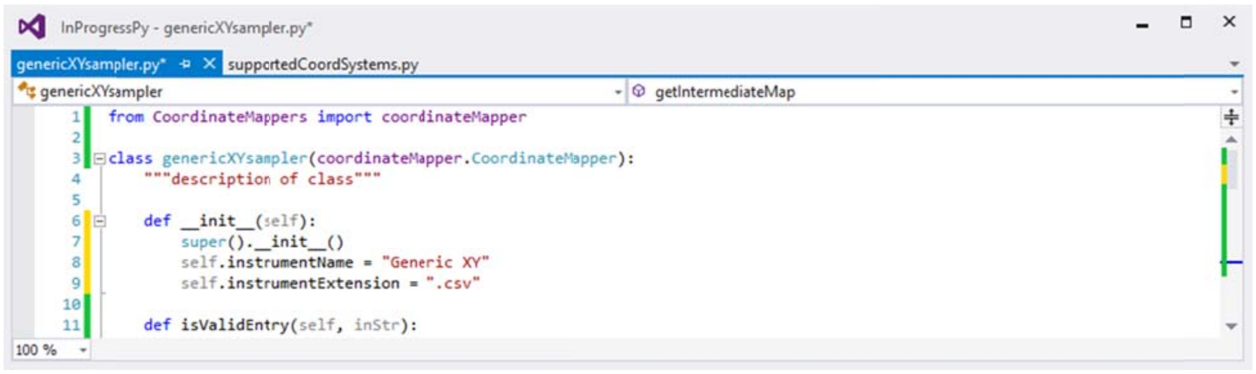
```python
from CoordinateMappers import coordinateMapper

class genericXYsampler(coordinateMapper.CoordinateMapper):
    """description of class"""

    def __init__(self):
        super().__init__()

    def isValidEntry(self, inStr):
        return super().isValidEntry(inStr)

    def extractPoint(self, inStr):
        return super().extractPoint(inStr)

    def predictName(self, pixelPoint):
        return super().predictName(pixelPoint)

    def predictLabel(self, physPoint):
        return super().predictLabel(physPoint)

    def predictedPoints(self):
        return super().predictedPoints()

    def saveInstrumentFile(self, filename, blobs):
        return super().saveInstrumentFile(filename, blobs)

    def loadInstrumentFile(self, filename):
        return super().loadInstrumentFile(filename)

    def getIntermediateMap(self):
        return super().getIntermediateMap()

    def setIntermediateMap(self, points):
        return super().setIntermediateMap(points)
```

Running now generates the following error:



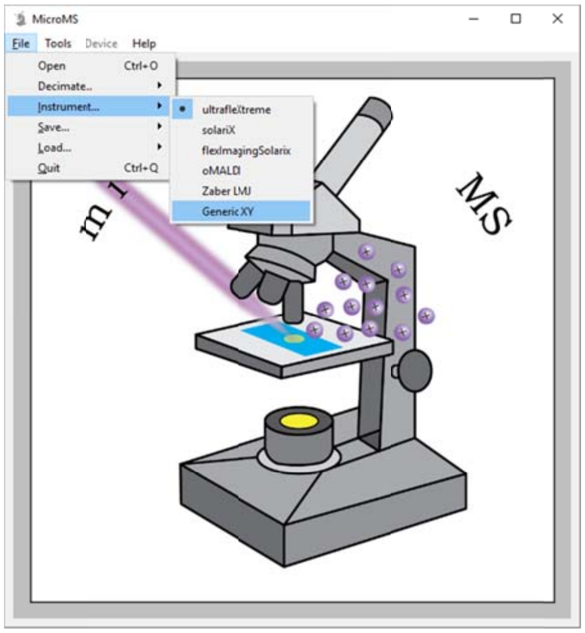
```
Traceback (most recent call last):
  File "D:\Sweedler Lab\GitRepositories\microMS\microMS.py", line 5, in <module>
    from GUICanvases.microMSQTWindow import MicroMSQTWindow
  File "D:\Sweedler Lab\GitRepositories\microMS\GUICanvases\microMSQTWindow.py",
 line 5, in <module>
    from CoordinateMappers import supportedCoordSystems
  File "D:\Sweedler Lab\GitRepositories\microMS\CoordinateMappers\supportedCoord
Systems.py", line 24, in <module>
    supportedNames = list(map(lambda x: x.instrumentName, supportedMappers))
  File "D:\Sweedler Lab\GitRepositories\microMS\CoordinateMappers\supportedCoord
Systems.py", line 24, in <lambda>
    supportedNames = list(map(lambda x: x.instrumentName, supportedMappers))
AttributeError: 'genericXYsampler' object has no attribute 'instrumentName'
Press any key to continue . . .
```

Because instrumentName hasn't been defined. Define that variable name as well as instrumentExtension in the __init__ method:



```
from CoordinateMappers import coordinateMapper

class genericXYsampler(coordinateMapper.CoordinateMapper):
    """description of class"""

    def __init__(self):
        super().__init__()
        self.instrumentName = "Generic XY"
        self.instrumentExtension = ".csv"

    def isValidEntry(self, inStr):
```

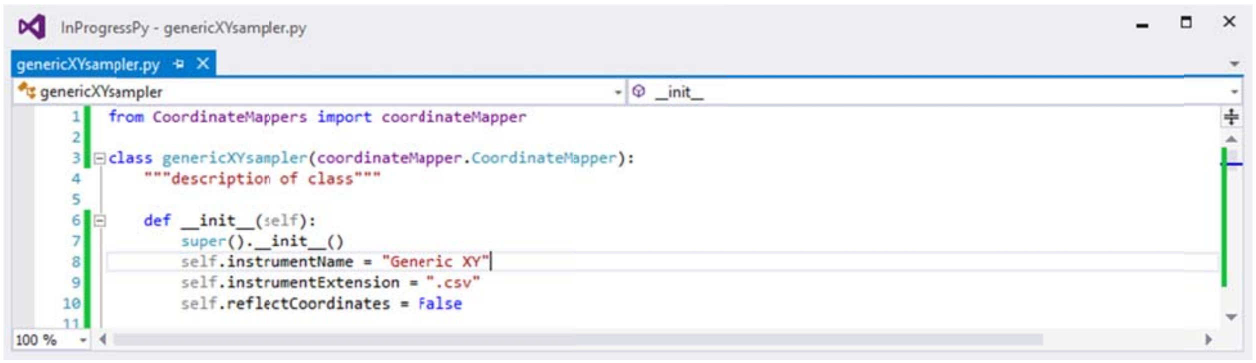
The instrumentName is used to display the instrument in the menubar and is saved for the registration file. As such it should be fairly short and unique among supported instruments. Adding these variables results in a running, but completely nonfunctional version of the Generic XY sampler:



IMPORTANT AND CONFUSING! The last parameter to set in __init__ is reflectCoordinates. In computer graphics (microMS is no exception), the top left of an image is defined as (0,0).

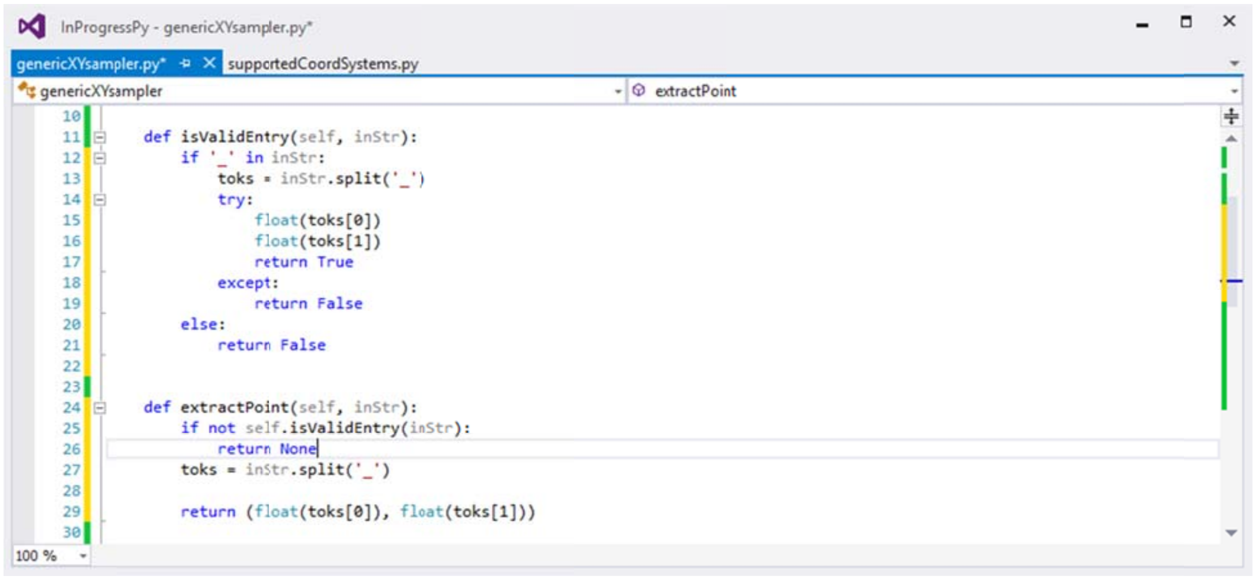
The x coordinate increases moving right, the y coordinate increases moving down. Because microMS utilizes a similarity registration, it will not handle reflections properly by default. The reflectCoordinates Boolean controls if the instrument coordinate system needs to be inverted to register properly. Setting this incorrectly will produce inaccurate positioning! For the instrument, if the x axis increases and the y axis increases moving down and to the right (relative to the image/sample) reflectCoordinates should be false. If both decrease it should also be false. If only one increases, it should be true. Another way to assess this is generate a scatter plot of the x and y coordinates of a triangle in both coordinate spaces. If the positions orient properly with just a rotation, reflectCoordinates is false.

In this case, moving down the image causes the Y coordinate to increase, so:



```
from CoordinateMappers import coordinateMapper

class genericXYsampler(coordinateMapper.CoordinateMapper):
    """description of class"""

    def __init__(self):
        super().__init__()
        self.instrumentName = "Generic XY"
        self.instrumentExtension = ".csv"
        self.reflectCoordinates = False
```

Next, implement the remaining required methods. First, for handling motor positions, isValidEntry and extractPoint need to be implemented. IsValidEntry checks if the supplied string is a valid motor coordinate and returns true if it is. extractPoint should take a string, validate it with isValidEntry and return a new tuple with x,y coordinates as numerics:



InProgressPy - genericXYsampler.py*

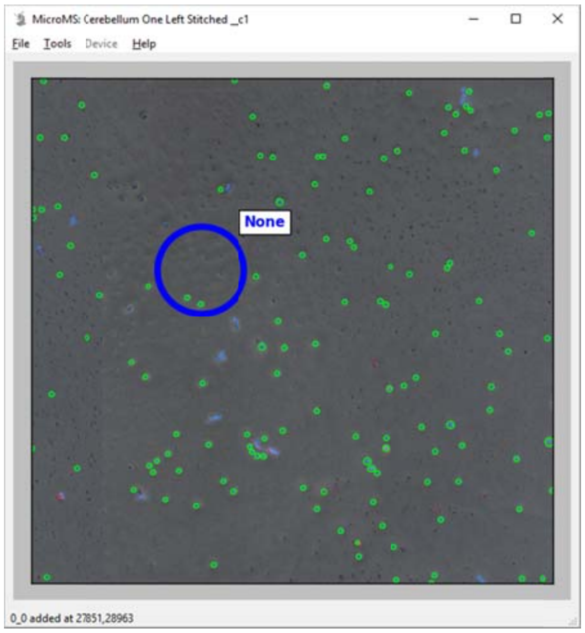
genericXYsampler.py* supportedCoordSystems.py

genericXYsampler extractPoint

```
    def isValidEntry(self, inStr):
        if '_' in inStr:
            toks = inStr.split('_')
            try:
                float(toks[0])
                float(toks[1])
                return True
            except:
                return False
        else:
            return False


    def extractPoint(self, inStr):
        if not self.isValidEntry(inStr):
            return None
        toks = inStr.split('_')

        return (float(toks[0]), float(toks[1]))
```

100 %

Now new points may be validated and added to the coordinate mapper:

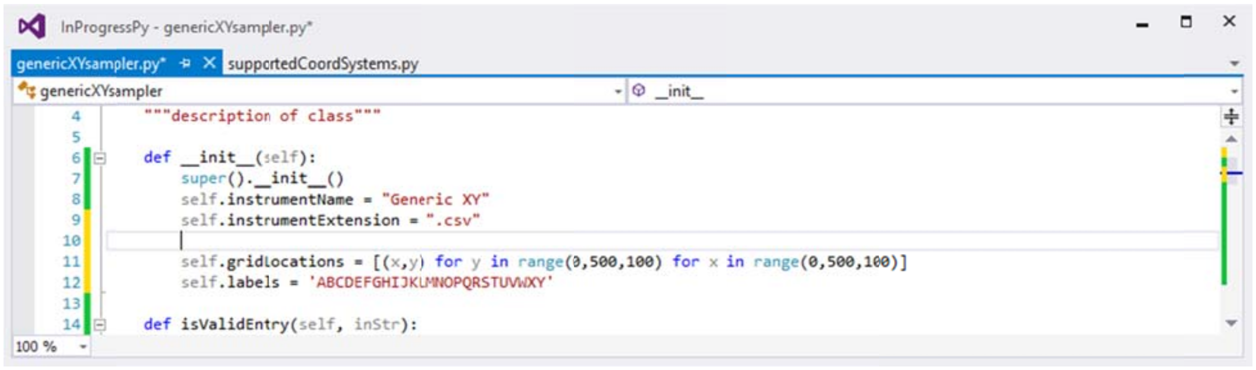


0_0 added at 27851,28963

The next methods handle predictions. PredictName is used to fill in the popup text box when a user adds a fiducial mark. Here, it should return the closest label in A-Y. PredictLabel is shown next to the fiducial mark (currently None). Again, this should be the closest label in A-Y. Finally, predictPoints is a set of pixel positions to show when toggled with P. These are the grid positions of 0..100..400 in a 5x5 array mentioned above for Generic XY. While PredictName and

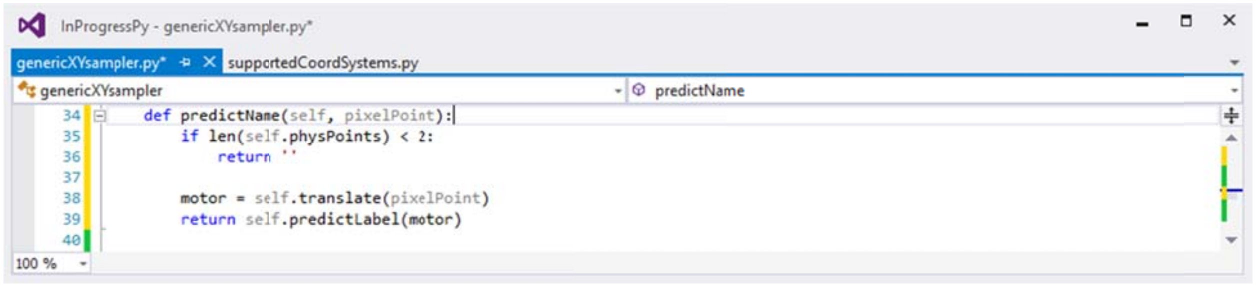
PredictLabel appear to perform the same function here, there are important differences. PredictName uses a *pixel* position while PredictLabel takes a *physical* position. When the user first adds a fiducial, the pixel position is known, so PredictName is used for the text. After the fiducial is entered, the physical position is known and PredictLabel is called. There are also cases when the return values should differ. The solarixMapper returns the system clipboard during PredictName, but the closest MTP point in PredictLabel.

First, generate the grid of locations and a list of labels in the same order (in this case a string).



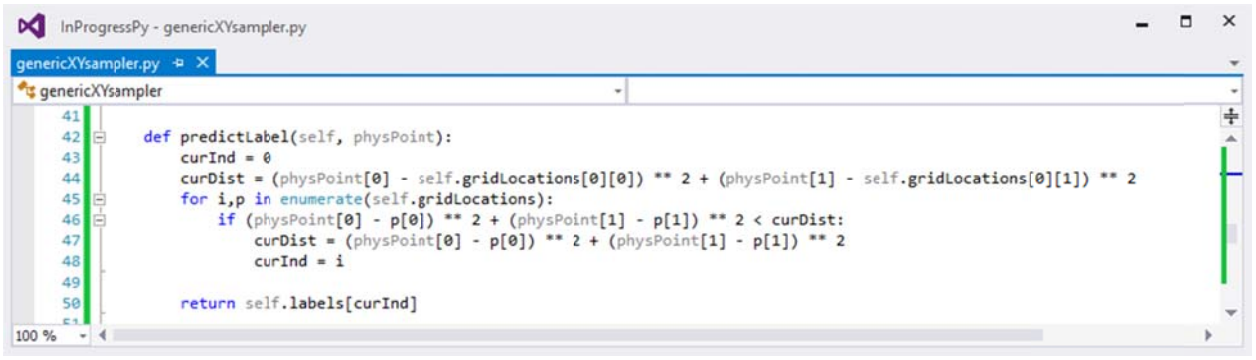
```
    """description of class"""

    def __init__(self):
        super().__init__()
        self.instrumentName = "Generic XY"
        self.instrumentExtension = ".csv"

        self.gridLocations = [(x,y) for y in range(0,500,100) for x in range(0,500,100)]
        self.labels = 'ABCDEFGHIJKLMNOPQRSTUVWXY'

    def isValidEntry(self, inStr):
```

PredictName will take a pixelPosition and try to predict the label by translating the pixel to a physical location:



```
    def predictName(self, pixelPoint):
        if len(self.physPoints) < 2:
            return ''

        motor = self.translate(pixelPoint)
        return self.predictLabel(motor)
```

Note that self.translate is a method for converting pixel coordinates to motor coordinates and is implemented in the base class. predictLabel is implemented by iteratively searching for the closest motor position:


InProgressPy - genericXYsampler.py

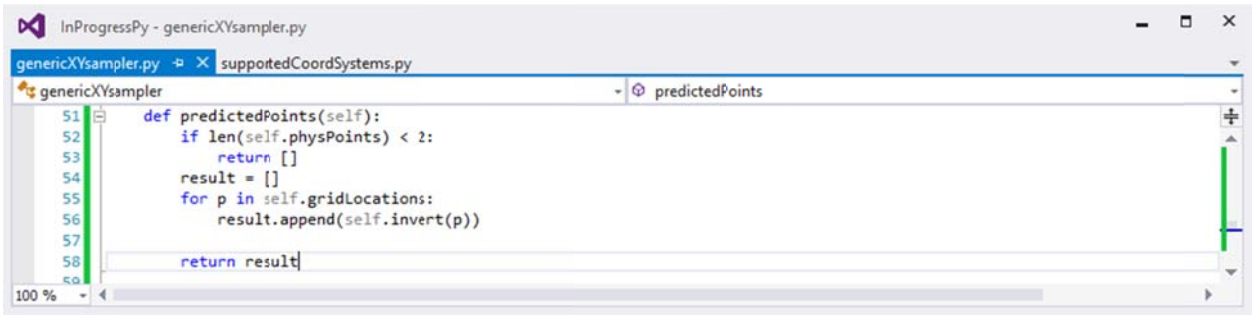
genericXYsampler.py

genericXYsampler

```
    def predictLabel(self, physPoint):
        curInd = 0
        curDist = (physPoint[0] - self.gridLocations[0][0]) ** 2 + (physPoint[1] - self.gridLocations[0][1]) ** 2
        for i,p in enumerate(self.gridLocations):
            if (physPoint[0] - p[0]) ** 2 + (physPoint[1] - p[1]) ** 2 < curDist:
                curDist = (physPoint[0] - p[0]) ** 2 + (physPoint[1] - p[1]) ** 2
                curInd = i

        return self.labels[curInd]
```

100 %

PredictedPoints is self.gridLocations when inverted to pixel positions (another base method):


InProgressPy - genericXYsampler.py

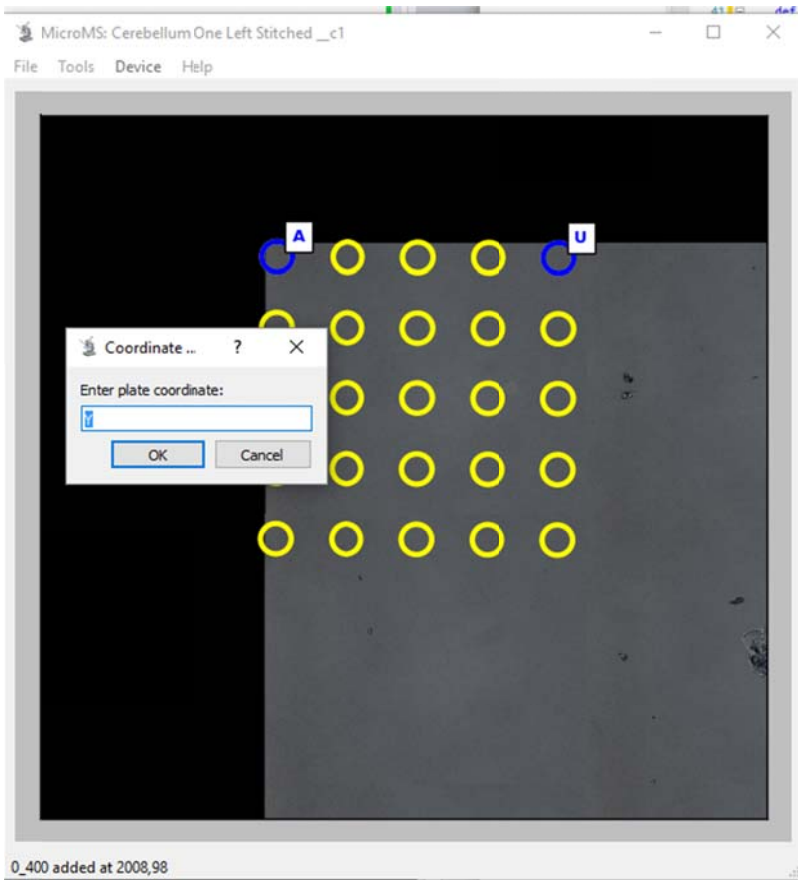
genericXYsampler.py supportedCoordSystems.py

genericXYsampler predictedPoints

```
    def predictedPoints(self):
        if len(self.physPoints) < 2:
            return []
        result = []
        for p in self.gridLocations:
            result.append(self.invert(p))

        return result
```

100 %

Executing microMS now shows predicted points in yellow, labeled fiducials, and a prediction of fiducial input (bottom right in this case):

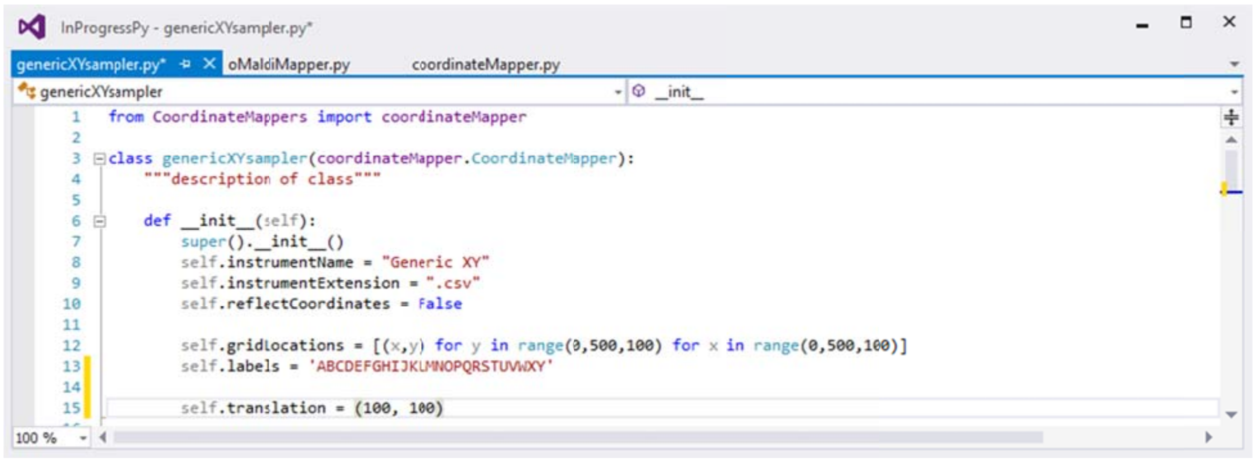


Before moving to the intermediate mapper, note that entering 'Y' is not a valid entry as it isn't in the form of <Xcoordinate>_<Ycoordinate>. To correct this, isValidEntry and extractPoint would need to change to accept letters and map a letter to its motor coordinate. brukerMapper uses a similar method for predicting locations.

Intermediate maps are a way to move between a coordinate system readily accessible to the user (e.g. motor coordinates) and a system used by the instrument (e.g. GenericCoordinates). microMS passes intermediate training points as lists of tuples, with the first element being a set point (A-Y, here) and the next two being an X and Y coordinate. The list populates the table in the *Grid Settings* in the *Tools* menu. In most applications, the coordinates would need to be stored as a separate file to save intermediate maps between executions. Here, the values will initialize as a static map and updates will last only during execution.

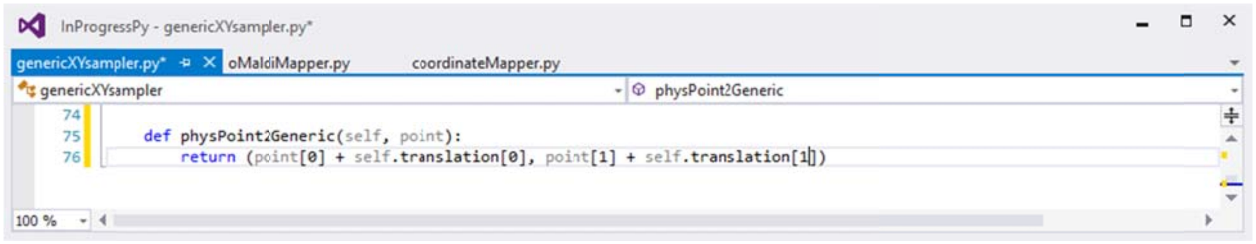
In coordinateMapper, the intermediate map could immediately convert physical coordinates to the second coordinate system and store them in self.physPoints. This would simplify saving instrument positions, but if the intermediate map changes, the current registration will be invalid as it is based on a previous intermediate map. Also, since the user typically cannot access the intermediate coordinate system it is difficult to examine the registration file later to assess its accuracy. For these reasons, microMS utilizes the motor coordinates for physical points and converts them only on saving instrument files.

To further simplify genericXYsampler, we will assume only translation is possible and is exactly specified with one point. In practice a point based registration is required, see oMaldiMapper or brukerMapper for examples. A new variable is needed to store the translation as well as a method to map from motor to GenericCoordinates:



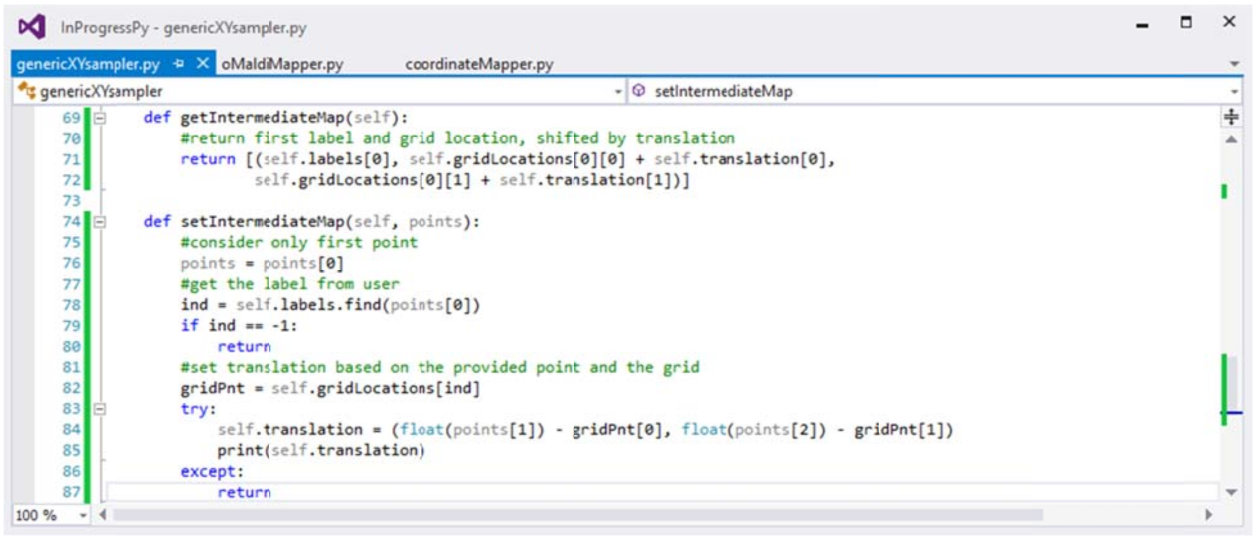
```
from CoordinateMappers import coordinateMapper

class genericXYsampler(coordinateMapper.CoordinateMapper):
    """description of class"""

    def __init__(self):
        super().__init__()
        self.instrumentName = "Generic XY"
        self.instrumentExtension = ".csv"
        self.reflectCoordinates = False

        self.gridLocations = [(x,y) for y in range(0,500,100) for x in range(0,500,100)]
        self.labels = 'ABCDEFGHIJKLMNOPQRSTUVWXY'

        self.translation = (100, 100)
```

Where (100,100) is from the specification.



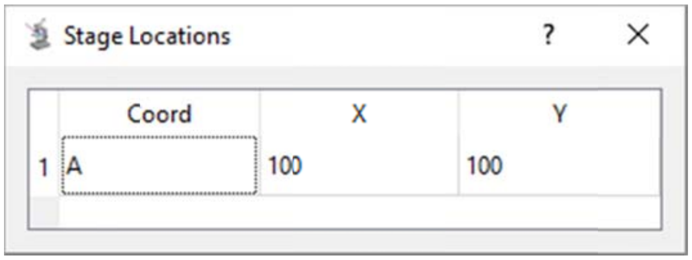
```

    def physPoint2Generic(self, point):
        return (point[0] + self.translation[0], point[1] + self.translation[1])
```

Now the get and set intermediateMap implementations:



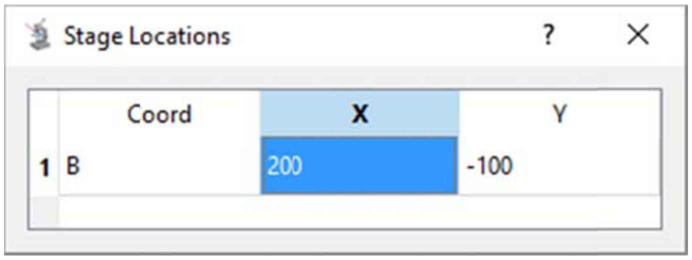
```
    def getIntermediateMap(self):
        #return first label and grid location, shifted by translation
        return [(self.labels[0], self.gridLocations[0][0] + self.translation[0],
                self.gridLocations[0][1] + self.translation[1])]

    def setIntermediateMap(self, points):
        #consider only first point
        points = points[0]
        #get the label from user
        ind = self.labels.find(points[0])
        if ind == -1:
            return
        #set translation based on the provided point and the grid
        gridPnt = self.gridLocations[ind]
        try:
            self.translation = (float(points[1]) - gridPnt[0], float(points[2]) - gridPnt[1])
            print(self.translation)
        except:
            return
```

'Get' populates the single table row from the first point in label/grid. Note the return type is a <u>list</u> of triples. Set utilizes the first point to recalculate translation, with minimal error checking on the user input. Upon running microMS, selecting the generic instrument, and opening the instrument settings:



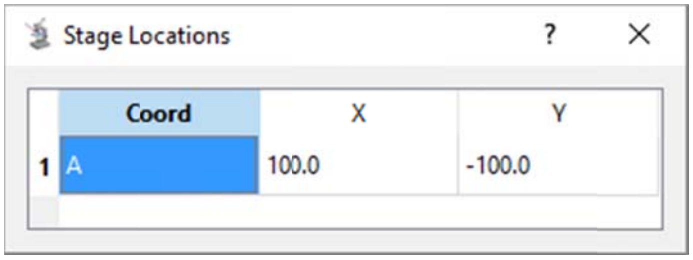
	Coord	X	Y
1	A	100	100

Each entry can be modified, for example:



	Coord	X	Y
1	B	200	-100

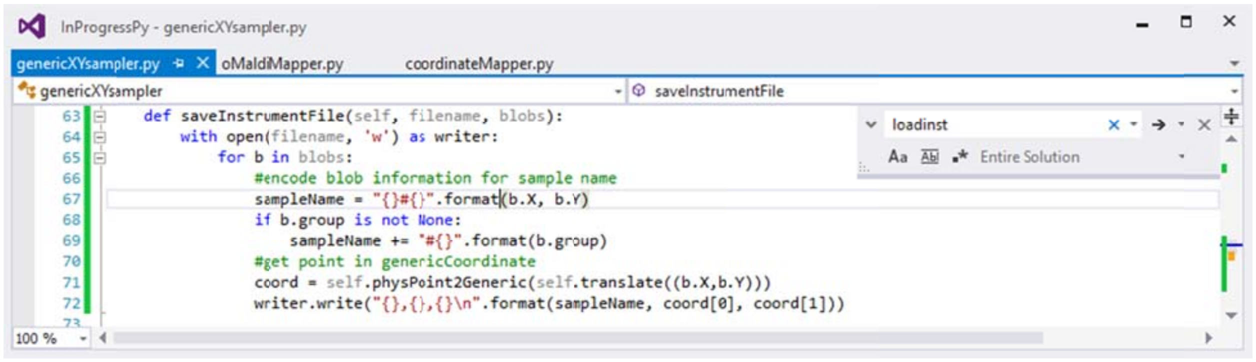
Which should set the translation to (100, -100) once the window is closed. The result is verified by the print statement. Opening the instrument settings again shows the new value of coordinate A:



	Coord	X	Y
1	A	100.0	-100.0

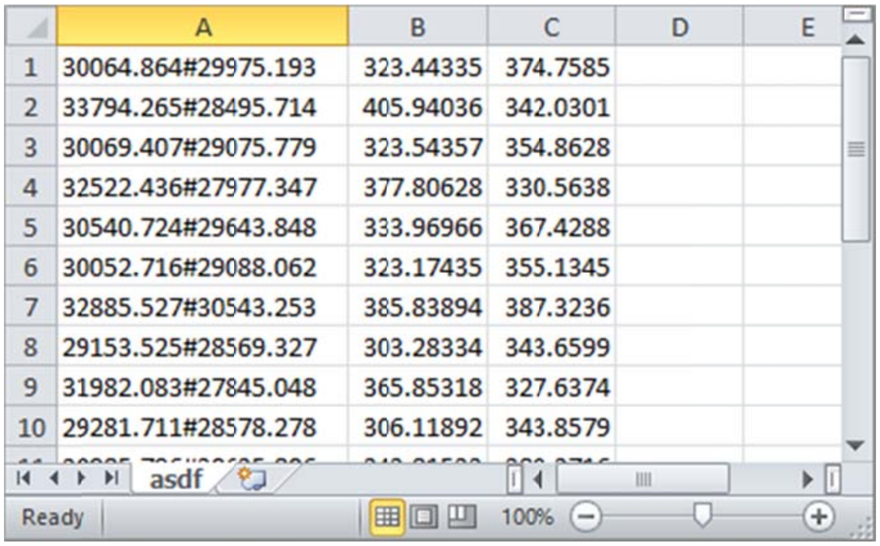
Finally, save and load instrumentFile need implementations. There are few requirements from microMS on saving an instrument file, allowing arbitrary format specific for each instrument. Loading an instrument file requires the method to return a new list of blobs (not a blobList) with the original x,y pixel position and group (if relevant). MicroMS makes no attempt to record blob radius and circularity in instrument files as instrument files specify target *positions* instead of actual blobs. However, the pixel positions and groups are required to correlate those targets back to an image and pattern. Where possible, microMS encodes the coordinates as sample names. For instruments where the instrument file cannot have a sample name, it may be useful to save the x, y, group data as a separate metadata file (see oMaldiMapper).

For genericXYsampler, the blob information will be encoded in the sample name as <X>#<Y> or <X>#<Y>#<Group> as appropriate. The X and Y coordinates in GenericCoordinates require the blob *pixel* positions to be transformed by self.translate to motor coordinates, and then shifted by self.translation, before writing the contents. Here is an implementation of saveInstrumentFile:



```python
    def saveInstrumentFile(self, filename, blobs):
        with open(filename, 'w') as writer:
            for b in blobs:
                #encode blob information for sample name
                sampleName = "{}#{}".format(b.X, b.Y)
                if b.group is not None:
                    sampleName += "#{}".format(b.group)
                #get point in genericCoordinate
                coord = self.physPoint2Generic(self.translate((b.X,b.Y)))
                writer.write("{},{},{}\n".format(sampleName, coord[0], coord[1]))
```

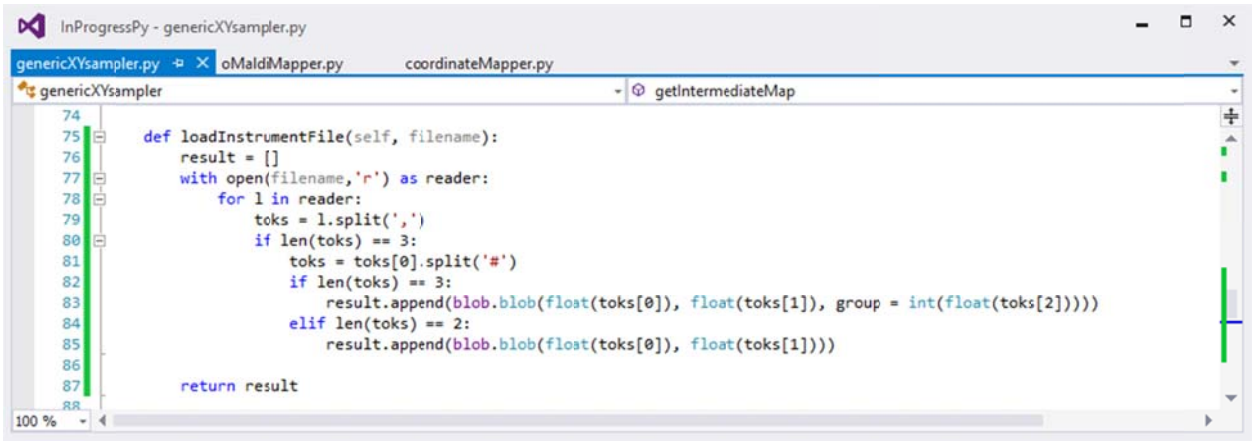
Running the code generates a csv like this:



	A	B	C	D	E
1	30064.864#29975.193	323.44335	374.7585		
2	33794.265#28495.714	405.94036	342.0301		
3	30069.407#29075.779	323.54357	354.8628		
4	32522.436#27977.347	377.80628	330.5638		
5	30540.724#29643.848	333.96966	367.4288		
6	30052.716#29088.062	323.17435	355.1345		
7	32885.527#30543.253	385.83894	387.3236		
8	29153.525#28569.327	303.28334	343.6599		
9	31982.083#27845.048	365.85318	327.6374		
10	29281.711#28578.278	306.11892	343.8579		

The sample name formatting is ugly as it displays the entire floating point number, but this is adjustable in the format call.

Loading the instrument file requires parsing just the sample name to populate a new blob list (don’t forget to import blob from ImageUtilities):



```
    def loadInstrumentFile(self, filename):
        result = []
        with open(filename,'r') as reader:
            for l in reader:
                toks = l.split(',')
                if len(toks) == 3:
                    toks = toks[0].split('#')
                    if len(toks) == 3:
                        result.append(blob.blob(float(toks[0]), float(toks[1]), group = int(float(toks[2]))))
                    elif len(toks) == 2:
                        result.append(blob.blob(float(toks[0]), float(toks[1])))

        return result
```

Now genericXYsampler is completely functional!

Implementing brukerMapper for the Bruker flexArmstrong

To combat the rising popularity of the genericXYsampler, Bruker has introduced its equally fictional flexArmstrong mass analyzer. As mentioned earlier, brukerMapper is another abstract base class within microMS that handles many common functions of Bruker instruments, including handling xeo files, fractional distances, and an intermediate map between motor coordinates and xeo positions. The slideIIadapter xeo geometry file defines a set of coordinates for a microscope slides which are referenced by brukerMapper. The brukerMapper interface greatly simplifies the addition of a new Bruker instrument, assuming the xeo files are identical. The solarixMapper and ultraflexMapper are both implementations of brukerMapper and can be used as further examples. ultraflexMapper is a simple implementation that uses motor coordinates separated by a space and saves a single XEO file for each blob list. solarixMapper is slightly more complicated as it populates fiducial locations from the clipboard, automatically splits lists into xeo files of 400 points (a software maximum), and generates xlsx files for starting autoexecute in the instrument control software. flexImagingSolarix is another mapper that inherits from solarixMapper, but defines a different set of instrumentFiles for use in flexImaging.

Classes inheriting from brukerMapper must:

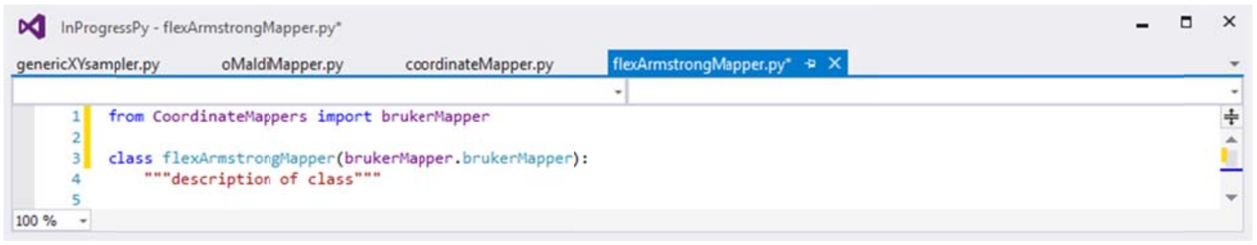
- Define motorCoordFilename to store intermediate maps between runs of microMS, an instrumentExtension, instrumentName and reflectCoordinates
- Implement isValidMotorCoord, extractMotorPoint, loadInstrumentFile and saveInstrumentFile.
- Add an import and initialize an instance in supportedCoordSystems.

The flexArmstrong is a new mass analyzer with control software and autoexecute functions similar to the ultraflex:

- It handles arbitrarily long xeo files and requires no additional files for performing automatic acquisition.
- Motor coordinates are input as <X>$<Y>.
- Motor coordinates are equal to the fractional distances multiplied by 1000. The Y coordinate *decreases* as the stages moves down the slide.
- The slideIIadaptor is supported and unchanged from previous version.

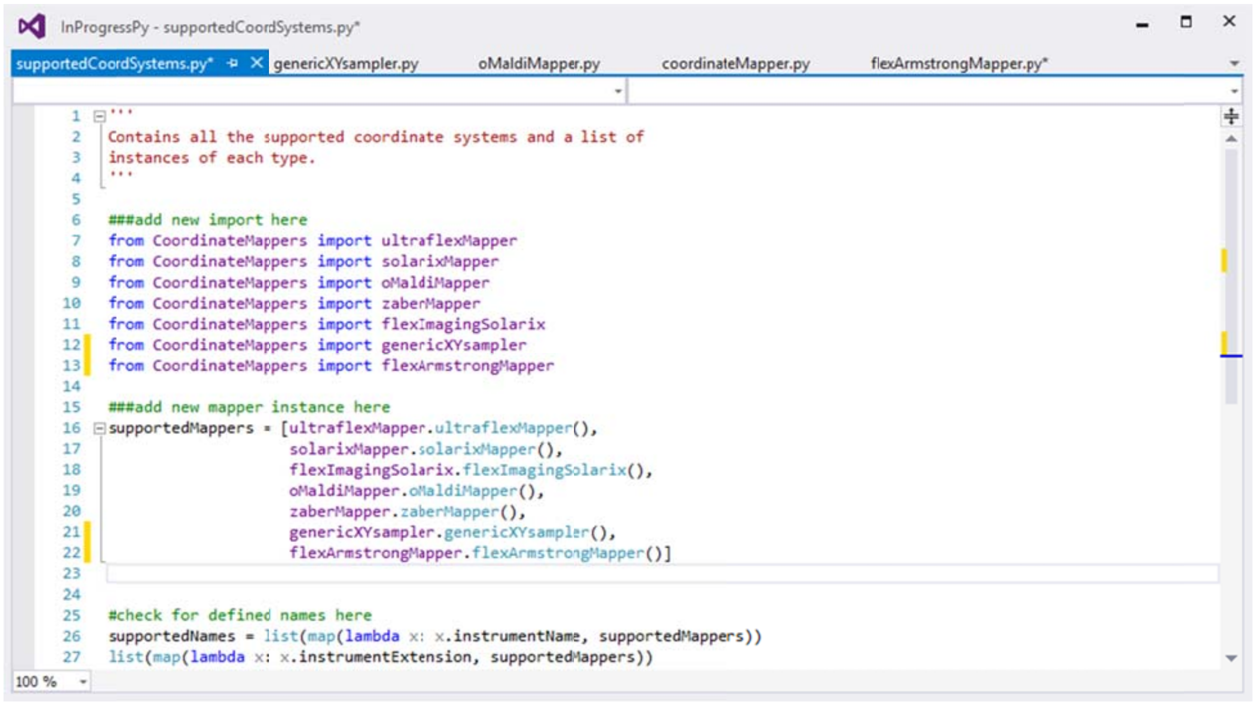
Note that the last feature is the only requirement for inheriting from brukerMapper.

Similar to the genericXYsampler, create a new class in coordinate mappers called flexArmstrongMapper which inherits from brukerMapper:



```
from CoordinateMappers import brukerMapper

class flexArmstrongMapper(brukerMapper.brukerMapper):
    """description of class"""

```

And add the mapper to supportedCoordSystems

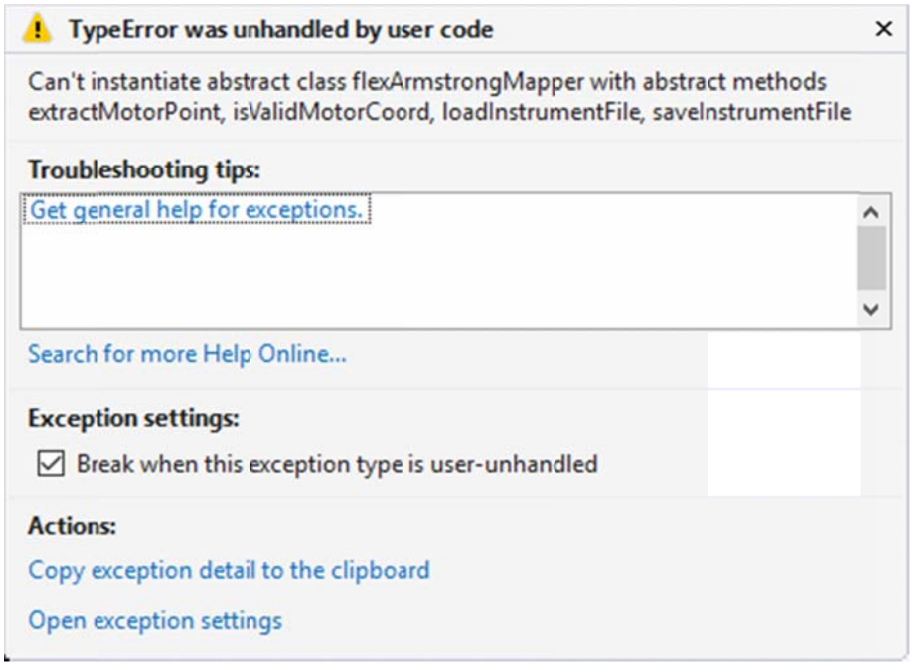

InProgressPy - supportedCoordSystems.py*

supportedCoordSystems.py* | genericXYsampler.py | oMaldiMapper.py | coordinateMapper.py | flexArmstrongMapper.py*

```
'''
Contains all the supported coordinate systems and a list of
instances of each type.
'''

###add new import here
from CoordinateMappers import ultraflexMapper
from CoordinateMappers import solarixMapper
from CoordinateMappers import oMaldiMapper
from CoordinateMappers import zaberMapper
from CoordinateMappers import flexImagingSolarix
from CoordinateMappers import genericXYsampler
from CoordinateMappers import flexArmstrongMapper

###add new mapper instance here
supportedMappers = [ultraflexMapper.ultraflexMapper(),
                    solarixMapper.solarixMapper(),
                    flexImagingSolarix.flexImagingSolarix(),
                    oMaldiMapper.oMaldiMapper(),
                    zaberMapper.zaberMapper(),
                    genericXYsampler.genericXYsampler(),
                    flexArmstrongMapper.flexArmstrongMapper()]


#check for defined names here
supportedNames = list(map(lambda x: x.instrumentName, supportedMappers))
list(map(lambda x: x.instrumentExtension, supportedMappers))
```

100 %

Running again generates errors for missing methods and missing variables:


TypeError was unhandled by user code

Can't instantiate abstract class flexArmstrongMapper with abstract methods extractMotorPoint, isValidMotorCoord, loadInstrumentFile, saveInstrumentFile

Troubleshooting tips:

Get general help for exceptions.

Search for more Help Online...

Exception settings:

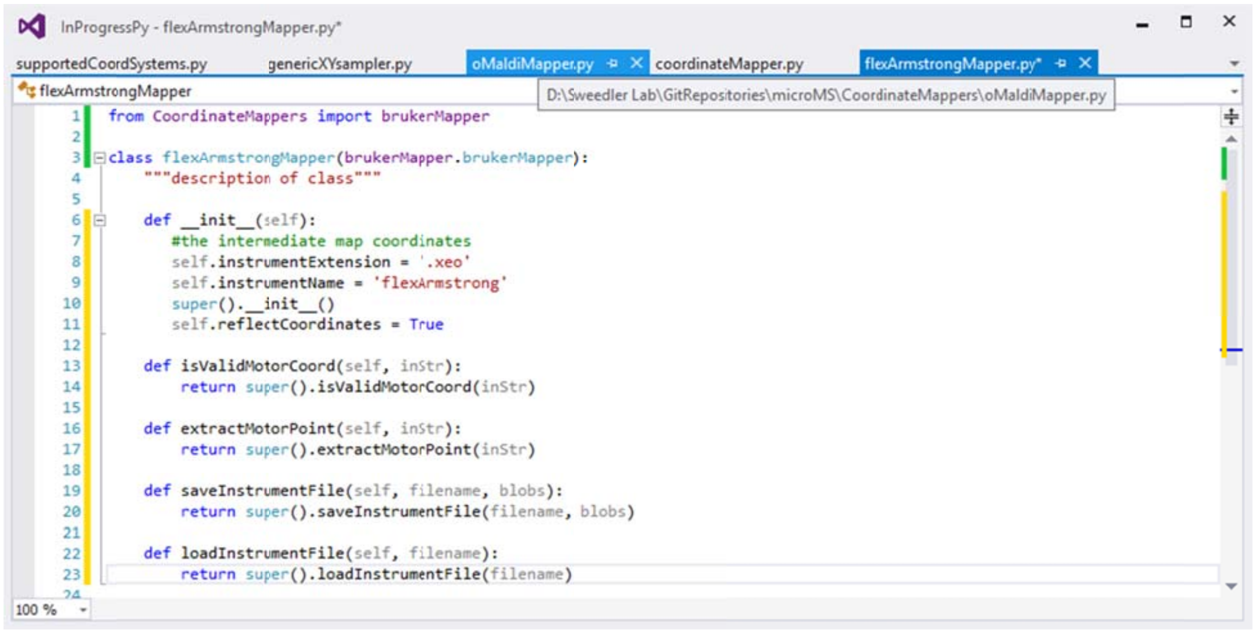
☑ Break when this exception type is user-unhandled

Actions:

Copy exception detail to the clipboard

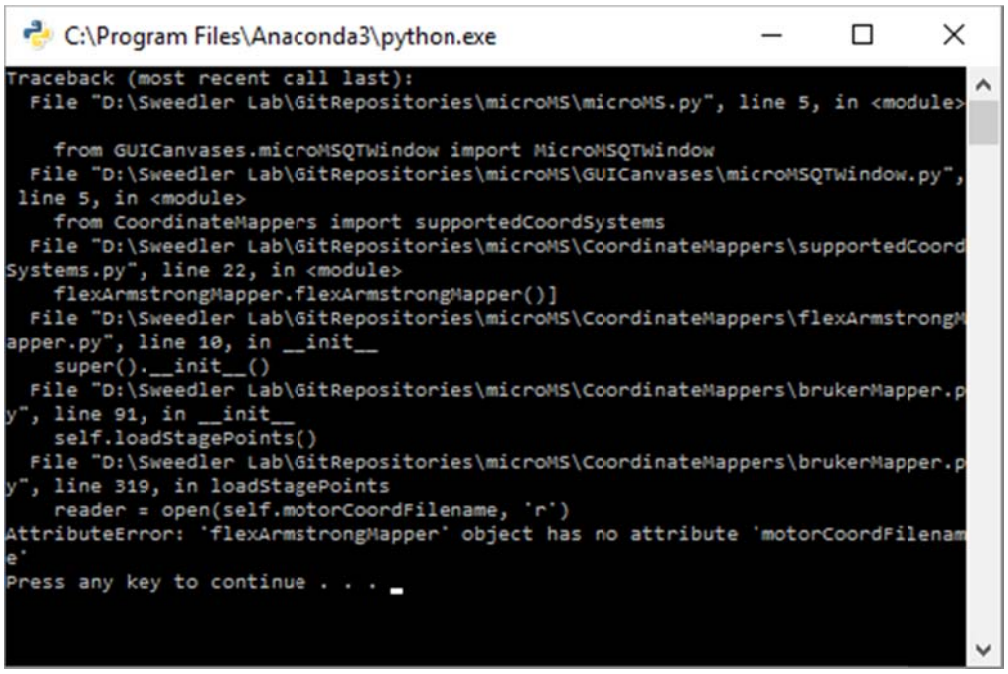
Open exception settings

Adding in method stubs and instrumentExtension, name and reflectCoordinates:



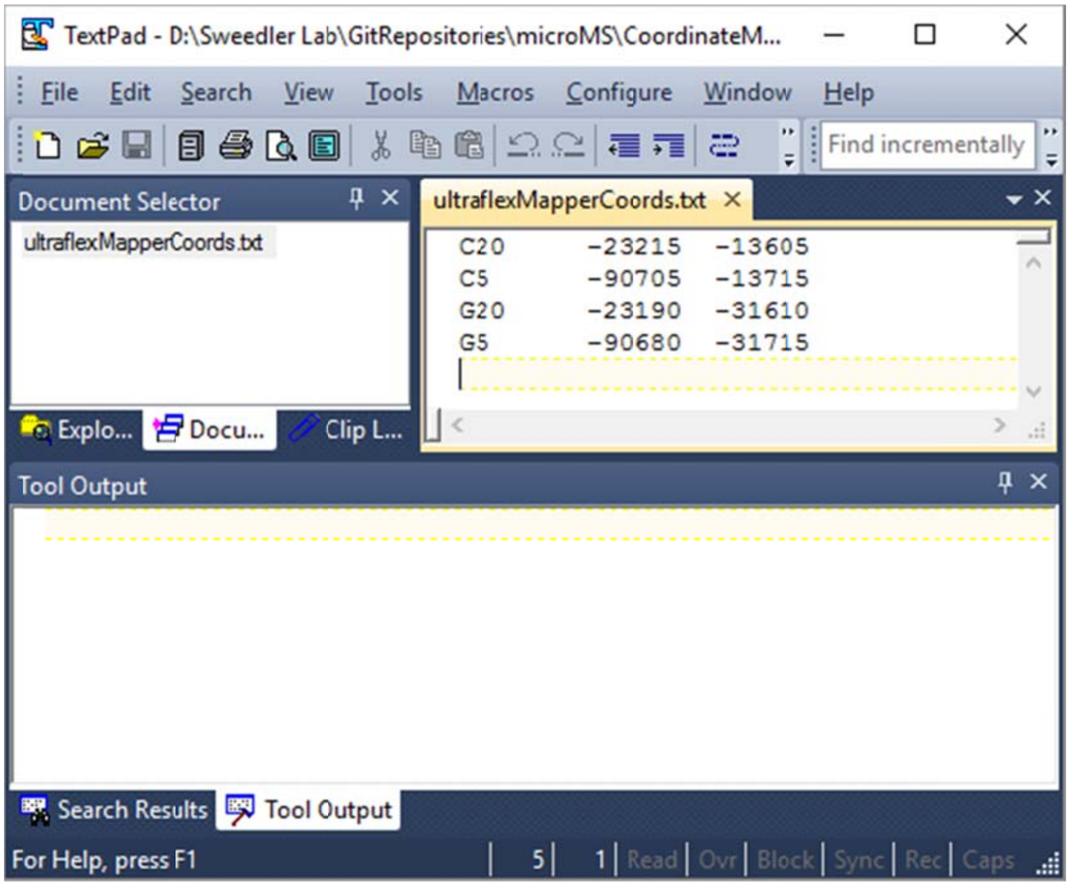
```python
from CoordinateMappers import brukerMapper

class flexArmstrongMapper(brukerMapper.brukerMapper):
    """description of class"""

    def __init__(self):
        #the intermediate map coordinates
        self.instrumentExtension = '.xeo'
        self.instrumentName = 'flexArmstrong'
        super().__init__()
        self.reflectCoordinates = True

    def isValidMotorCoord(self, inStr):
        return super().isValidMotorCoord(inStr)

    def extractMotorPoint(self, inStr):
        return super().extractMotorPoint(inStr)

    def saveInstrumentFile(self, filename, blobs):
        return super().saveInstrumentFile(filename, blobs)

    def loadInstrumentFile(self, filename):
        return super().loadInstrumentFile(filename)
```

Where reflectCorodinates is True since the Y motor coordinate *decreases* as the stage moves down the image (and pixel positions increase). Running again produces a new error:



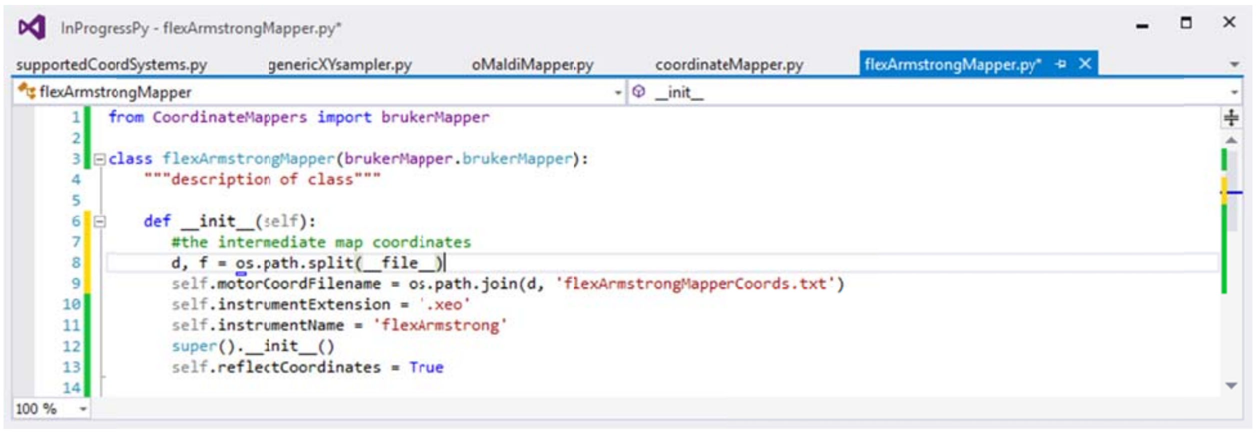
```
Traceback (most recent call last):
  File "D:\Sweedler Lab\GitRepositories\microMS\microMS.py", line 5, in <module>

    from GUICanvases.microMSQTWindow import MicroMSQTWindow
  File "D:\Sweedler Lab\GitRepositories\microMS\GUICanvases\microMSQTWindow.py",
 line 5, in <module>
    from CoordinateMappers import supportedCoordSystems
  File "D:\Sweedler Lab\GitRepositories\microMS\CoordinateMappers\supportedCoord
Systems.py", line 22, in <module>
    flexArmstrongMapper.flexArmstrongMapper()]
  File "D:\Sweedler Lab\GitRepositories\microMS\CoordinateMappers\flexArmstrongM
apper.py", line 10, in __init__
    super().__init__()
  File "D:\Sweedler Lab\GitRepositories\microMS\CoordinateMappers\brukerMapper.p
y", line 91, in __init__
    self.loadStagePoints()
  File "D:\Sweedler Lab\GitRepositories\microMS\CoordinateMappers\brukerMapper.p
y", line 319, in loadStagePoints
    reader = open(self.motorCoordFilename, 'r')
AttributeError: 'flexArmstrongMapper' object has no attribute 'motorCoordFilenam
e'
Press any key to continue . . .
```

Which complains about the lack of a motorCoordFilename. The filename can be anywhere on the operating system, though for simplicity microMS uses the directory containing the coordinateMappers. The actual file should be tab delimited text, similar to the following for ultraflex:

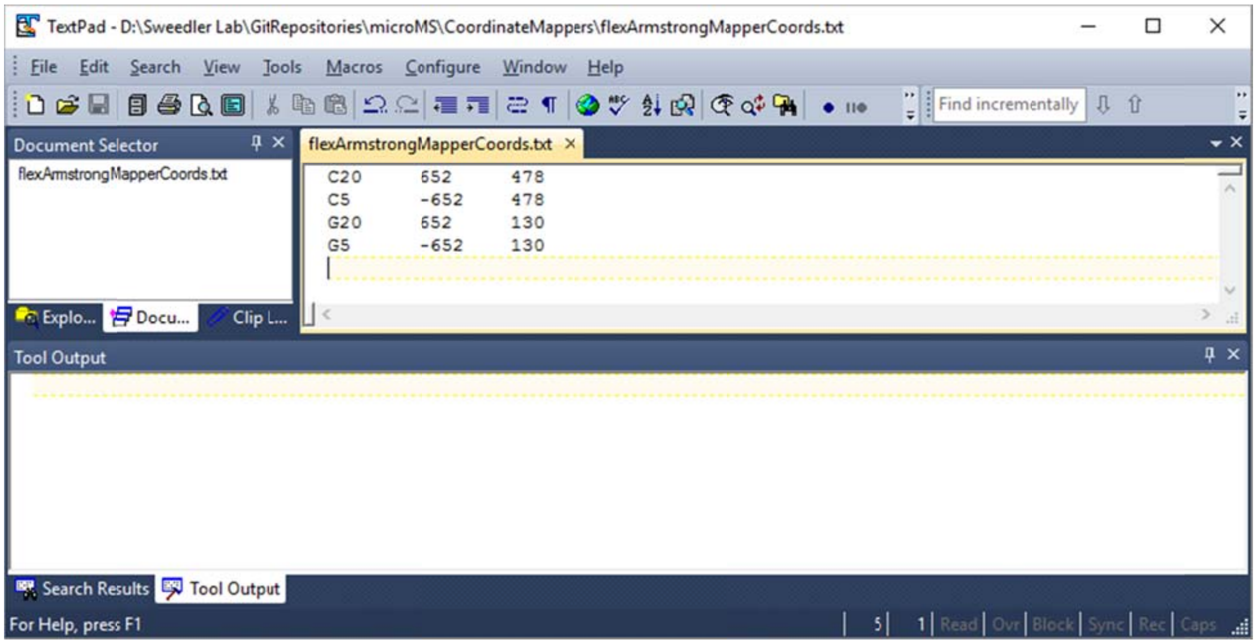

TextPad - D:\Sweedler Lab\GitRepositories\microMS\CoordinateM...
File Edit Search View Tools Macros Configure Window Help
Find incrementally
Document Selector
ultraflexMapperCoords.txt
Explo... Docu... Clip L...
Tool Output
Search Results Tool Output
For Help, press F1
5 1 Read Ovr Block Sync Rec Caps

```
C20     -23215  -13605
C5      -90705  -13715
G20     -23190  -31610
G5      -90680  -31715
```

Where the first column corresponds to an xeo position on the slideIIadapter and the next two columns are the x and y coordinates. brukerMapper takes care of reading this file, presenting it to the user and applying changes to set the intermediate mapper. Write xeo also handles performing a similarity registration on the intermediate mapper to generate fractional distances. What is needed is a unique filename and corresponding text file with the initial coordinates:

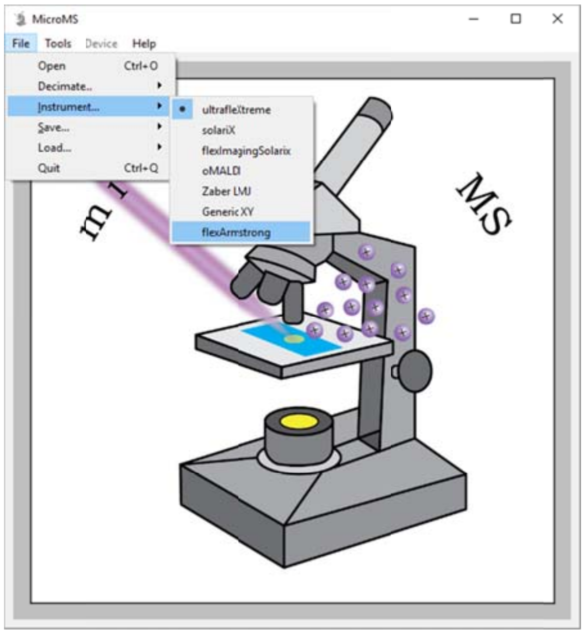


```python
from CoordinateMappers import brukerMapper

class flexArmstrongMapper(brukerMapper.brukerMapper):
    """description of class"""

    def __init__(self):
        #the intermediate map coordinates
        d, f = os.path.split(__file__)
        self.motorCoordFilename = os.path.join(d, 'flexArmstrongMapperCoords.txt')
        self.instrumentExtension = '.xeo'
        self.instrumentName = 'flexArmstrong'
        super().__init__()
        self.reflectCoordinates = True
```

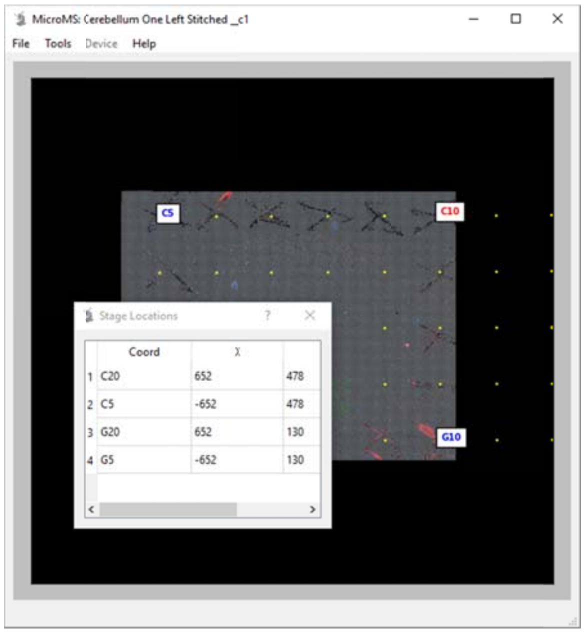

TextPad - D:\Sweedler Lab\GitRepositories\microMS\CoordinateMappers\flexArmstrongMapperCoords.txt
File Edit Search View Tools Macros Configure Window Help
Find incrementally
Document Selector
flexArmstrongMapperCoords.txt
Explo... Docu... Clip L...
Tool Output
Search Results Tool Output
For Help, press F1
5 1 Read Ovr Block Sync Rec Caps

```
C20     652     478
C5      -652    478
G20     652     130
G5      -652    130
```

Note that motorCoordFilename must be defined prior to calling super().__init__(), which will try to read the file. The values in the Coords file were chosen based on the third specification and are the fractional distances multiplied by 1000. Now microMS is running, though the mapper is nonfunctional:

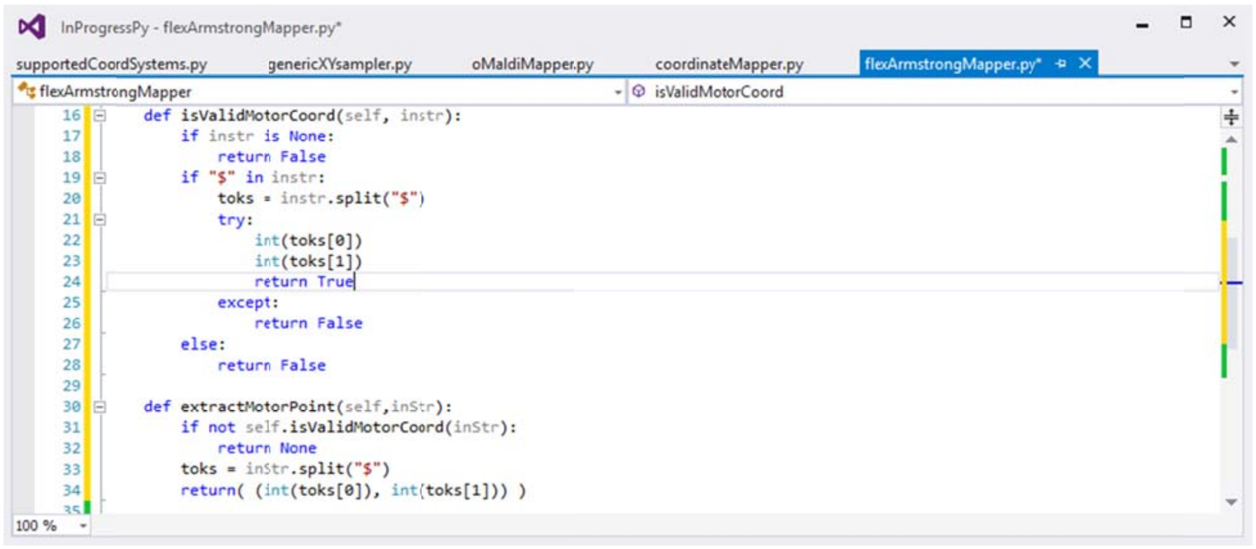


But it is closer to functional than you might think based on the amount of work for the genericXYsampler. Already the flexArmstrongMapper handles named MTP coordinates (like C5), generate predicted names, labels and points, and get and set intermediate maps!



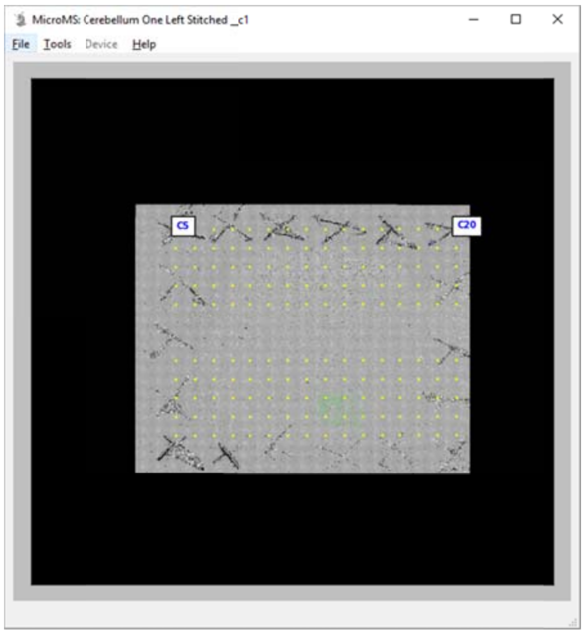
	Coord	X	
1	C20	652	478
2	C5	-652	478
3	G20	652	130
4	G5	-652	130

What it *cannot* do yet is utilize motor coordinates as fiducial locations or save/load instrument files. Motor coordinate methods are fairly simple and are almost copied directly from the genericXYsampler for extracting entries:

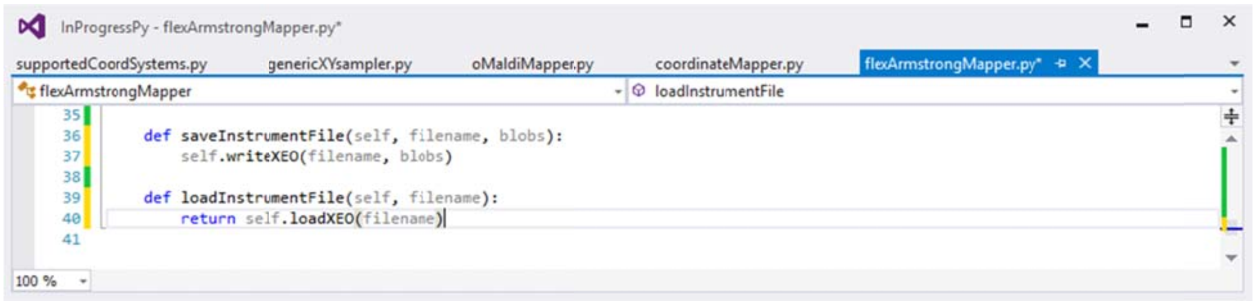


```
    def isValidMotorCoord(self, instr):
        if instr is None:
            return False
        if "$" in instr:
            toks = instr.split("$")
            try:
                int(toks[0])
                int(toks[1])
                return True
            except:
                return False
        else:
            return False

    def extractMotorPoint(self,inStr):
        if not self.isValidMotorCoord(inStr):
            return None
        toks = inStr.split("$")
        return( (int(toks[0]), int(toks[1])) )
```

Since brukerMapper accepts named positions (C5) in addition to motor coordinates, extractPoint must check for a valid MTP or motor coordinate and call the appropriate extraction method. Classes inheriting from brukerMapper only have to handle motor coordinates as the MTP cases are identical. Now the flexArmstrongMapper properly handles motor coordinates:



And finally loading and saving instrument files. As nothing special is required of the xeo files, the base methods of brukerMapper are suitable to load and write XEO files:



```
    def saveInstrumentFile(self, filename, blobs):
        self.writeXEO(filename, blobs)

    def loadInstrumentFile(self, filename):
        return self.loadXEO(filename)
```

And that's it! When testing initial accuracy, it is important to save the Fiducial Positions and check that their locations are accurate. Large deviations indicate an issue with some aspect of the mapper code. While it is impossible to predict future tweaks in new instruments, this should act as a starting guide to getting things up and running.

Direct instrument control

This section describes the organization of code and operation of microMS when used for direct instrument control. As the only supported instrument is a lab-built xyz stage prototype, it is highly unlikely to be useful to general readers, but can assist with attempts to integrate another instrument.

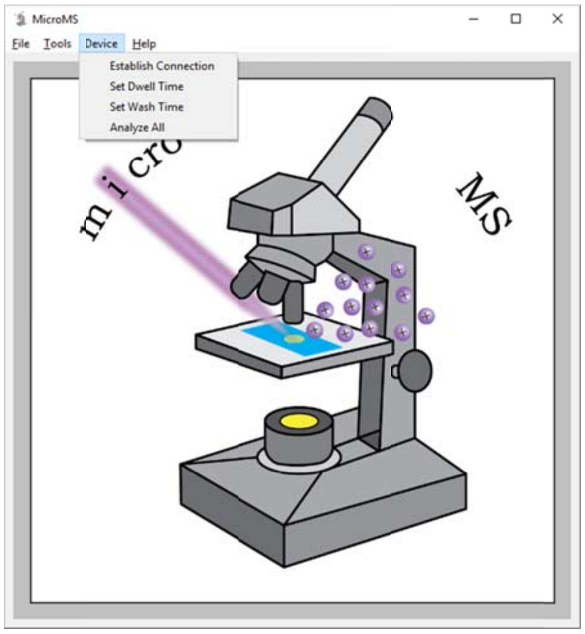
Code Organization

Instrument control with microMS is performed through a coordinateMapper with an instance of a connectedInstrument. ConnectedInstrument is another abstract base class that defines a set of required methods for interacting with connected instruments, such as moving, getting a position, and collecting from positions. The implemented instrument is a Zaber xyz stage used for liquid extraction. microMS interacts with the zaberMapper implementation of the coordinateMapper. This is a fairly simple coordinateMapper as there is no intermediate map or predicted labels. The novel aspects are a connectedInstrument, discussed more below, and directly reading the stage position for predicting the name of a fiducial. The predicted name is then directly read as a fiducial location. Predicted points are also generated from the current stage position.

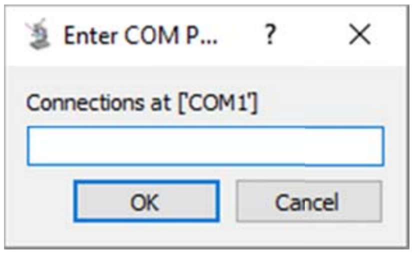
The connectedInstrument of zaberMapper is a zaber3axis object, which inherits from connectedInstrument and zaberInterface. The zaberInterface is another abstract base class which has a number of wrapper methods and a dictionary of commands to simplify communication with Zaber linear stages. zaber3axis implements communication with the stage including device renumbering, stage initialization, movement and collection. The main GUI interacts with the connectedInstrument of a coordinateMapper, when the instrument is not None and instrument.connected is true.

Direct instrument operation

To begin direct instrument control, first select an instrument which has a non-None value for connectedInstrument, currently ZaberLMJ is the only such mapper. With a valid mapper, the *Device* tab in the menu bar becomes enabled, providing additional options for interacting with connected instruments:



Selecting *Establish Connection* causes the following window to popup:



With a list of valid connections. This is only functional with a Windows computer, but adjusting serial communication would enable it with other operating systems. Entering a valid COM port and clicking OK causes microMS to attempt communication at the port and calls homeAll(). All calls to zaber3axis are blocking so the GUI will appear to freeze while the stage is actively moving.

The stage is moved with the following hotkeys:

- i, j, k, l moves the stage up, left, down, right a small amount, respectively.
- *Shift* and i, j, k, l moves the position 100 times farther than the small step.
- *Shift* and *Ctrl* with i, j, k, l moves the stage 10 times farther than a small step.
- + moves the probe up, or sets a focus
- - moves the probe down. *Shift* and *Ctrl* function similar to i, j, k, l. Additionally, *Shift*+*Ctrl*+*Alt* causes the probe to take a giant step, equal to 1000 times a small step.

Once a fiducial is located on the stage and image, its position is trained by pressing the right mouse button on the image location. The current stage position is read as a predicted point which is directly used in training. As before, *Shift* + RMB removes the closest fiducial and the worst fiducial is shown in red.

With at least 2 fiducials, the stage is moved to a position on the image by pressing *Alt*+LMB. Pressing P to toggle predicted points will cause the current stage position to be read during GUI redraws and displays the probe location as a yellow circle. Note that this function causes lag during stage interactions as each movement triggers a redraw and additional stage communication. Alternatively, pressing *Ctrl*+F will display the stage x,y and z position in motor coordinates in the status bar.

To perform a collection or measurements, the probe position must be set. With the probe in the correct position, press *Shift* + V to set the position. This also causes the probe to retract. After the position is set, the probe is moved in and out of acquisition position by pressing V. To collect at an arbitrary location, press X. This moves the probe into position and collects for the amount set in *Set Dwell Time* in the *Device* menu. If the wash time (in *Set Wash Time*) is not 0, the probe will then move into its final position for the specified amount of time. After washing, all stages are homed. Setting Wash Time to -1 causes the probe to stay in its final position until the user homes the stages. If wash time is 0, the probe will simply retract, staying in the same x,y location. Pressing H causes all stages to home, *Shift* + H moves the probe to the final position and stays there.

The final available function is to collect all, in *Analyze All*. This causes the stage to first home, then visit each target blob location for the dwell time. After visiting all blobs, the stage will either home, move to final position, or move to final position for "wash time" and then home if wash time is 0, -1 or a positive value respectively.