



Publicly Accessible Penn Dissertations


1-1-2015

Biophysics of Human Neutrophil Haptokinesis

Steven Henry

University of Pennsylvania, sjhenry@ptd.net

Follow this and additional works at: <http://repository.upenn.edu/edissertations>

 Part of the [Allergy and Immunology Commons](#), [Biomedical Commons](#), [Biophysics Commons](#), [Immunology and Infectious Disease Commons](#), and the [Medical Immunology Commons](#)

Recommended Citation

Henry, Steven, "Biophysics of Human Neutrophil Haptokinesis" (2015). *Publicly Accessible Penn Dissertations*. 1061.
<http://repository.upenn.edu/edissertations/1061>

This paper is posted at Scholarly Commons. <http://repository.upenn.edu/edissertations/1061>
For more information, please contact libraryrepository@pobox.upenn.edu.

Biophysics of Human Neutrophil Haptokinesis

Abstract

Neutrophils are a type of white blood cell and first responders to tissue trauma and infection. This thesis explores the role of extracellular adhesivity in dictating neutrophil phenotype with respect to cell shape, motility, mechanical force generation, and the molecular constituents involved in these processes. The principle tool employed is microcontact printing, a powerful method to spatially organize a cell's adhesive environment. We demonstrate the capacity of neutrophils to sense adhesive density on stiff substrates and differentially respond to surfaces with low and high fibronectin content. On low and moderately adhesive surfaces neutrophils assume a highly spread, uropod-absent phenotype reminiscent of keratocytes. On highly adhesive surfaces neutrophils assume the classic amoeboid morphology with an elongated cell body, narrow lamellipodium, and knob-like trailing uropod. Our work reconciles conflicting observations of these two phenotypes previously attributed solely to the underlying stiffness of substrate. The spreading and motility quantified are haptokinetic, induced through the quiescent cell's interaction with immobilized adhesive ligand alone. Function blocking antibody studies implicated the promiscuous Mac-1 integrin receptor in supporting haptokinetic migration. We elucidate the density sensing length scale by presenting high and low adhesive cues to the cells simultaneously. Through rational design of the adhesive domains we conclude that neutrophils sense density at the whole cell length scale, integrating adhesive stimuli over their entire contact interface. Adhesion density sensitivity in stiff microenvironments has applicability to the study of cancer metastasis and particularly the epithelial-to-mesenchymal transition model. We also employ the microfabricated-Post-Array-Detectors (mPADs) traction platform to measure the forces associated with neutrophil spreading. We resolve with high spatial and temporal resolution a highly coordinated protrusive wave front of pN magnitude that propagates radially outwards from the cell center. Small molecule inhibitor studies establish that spreading was not analogous to lamellipodium formation but was sensitive to perturbations of actin cortical stiffness. Lastly, we apply the principles uncovered in neutrophils to the patterning of surface-active microfluidic vesicles by tuning vesicle-substrate adhesion and repulsion at the contact interface. The generation of ordered arrays of micron scale vesicles was a first of its kind.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

Graduate Group

Bioengineering

First Advisor

Daniel A. Hammer

Keywords

adhesion, microcontact printing, motility, neutrophil, outside-in integrin activation, traction force

Subject Categories

Allergy and Immunology | Biomedical | Biophysics | Immunology and Infectious Disease | Medical
Immunology

BIOPHYSICS OF HUMAN NEUTROPHIL HAPTOKINESIS

Steven J. Henry

A DISSERTATION

in

Bioengineering

Presented to the Faculties of the University of Pennsylvania in

Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2015

Supervisor of Dissertation

Daniel A. Hammer, PhD, Professor of Bioengineering

Graduate Group Chairperson

Jason A. Burdick, PhD, Professor of Bioengineering

Dissertation Committee:

Scott L. Diamond, PhD, Professor of Chemical and Biomolecular Engineering

John C. Crocker, PhD, Professor of Chemical and Biomolecular Engineering

Dongyun Huh, PhD, Assistant Professor of Bioengineering

BIOPHYSICS OF HUMAN NEUTROPHIL HAPTOKINESIS

COPYRIGHT © 2015 by Steven J. Henry

Acknowledgements

I thank Dan Hammer for inviting me to join his research group and giving me free reign to be curious and explore my interests. Dan has been a patient and enthusiastic advisor. In reflecting upon my graduate training, I recognize that Dan has placed an immense trust in me, and all of his graduate students, by allowing open and frequent collaboration with peers and other research laboratories. I thank my committee members for their guidance and time in serving as advisors to my thesis. Beyond his role as committee member, John Crocker has been an invaluable mentor. I am immensely thankful for John's energy and willingness to provide thoughtful reflection at every stage of my growth as a scientist. Numerous lab mates past and present were instrumental in my graduate training. Of truly exceptional note were my senior lab mates and collaborators Neha Kamat and Brendon Ricart. Neha was and remains an inspirational scientific role model. Brendon has been an appreciated voice of perspective and encouragement as I conclude my graduate work. Outside the lab, Mike Norton has been a steadfast friend and I will miss our lively and meandering conversations. Most importantly I thank my wife Kate who has equally shouldered the burden of this challenging journey. This body of work is as much a product of her enduring support as my own diligence.

ABSTRACT

BIOPHYSICS OF HUMAN NEUTROPHIL HAPTOKINESIS

Steven J. Henry

Professor Daniel A. Hammer

Neutrophils are a type of white blood cell and first responders to tissue trauma and infection. This thesis explores the role of extracellular adhesivity in dictating neutrophil phenotype with respect to cell shape, motility, mechanical force generation, and the molecular constituents involved in these processes. The principle tool employed is microcontact printing, a powerful method to spatially organize a cell's adhesive environment. We demonstrate the capacity of neutrophils to sense adhesive density on stiff substrates and differentially respond to surfaces with low and high fibronectin content. On low and moderately adhesive surfaces neutrophils assume a highly spread, uropod-absent phenotype reminiscent of keratocytes. On highly adhesive surfaces neutrophils assume the classic amoeboid morphology with an elongated cell body, narrow lamellipodium, and knob-like trailing uropod. Our work reconciles conflicting observations of these two phenotypes previously attributed solely to the underlying stiffness of substrate. The spreading and motility quantified are haptokinetic, induced through the quiescent cell's interaction with immobilized adhesive ligand alone. Function blocking antibody studies implicated the promiscuous Mac-1 integrin receptor in supporting haptokinetic migration. We elucidate the density sensing length scale by presenting high and low adhesive cues to the cells simultaneously. Through rational design of the adhesive domains we conclude that neutrophils sense density at the whole cell length scale, integrating adhesive stimuli over their entire contact interface. Adhesion density sensitivity in stiff microenvironments has applicability to the study of cancer metastasis and particularly the epithelial-to-mesenchymal transition model. We also employ the microfabricated-Post-Array-Detectors (mPADs) traction platform to measure the forces associated with neutrophil spreading. We resolve with high spatial and temporal resolution a highly coordinated protrusive wave front of pN magnitude that propagates radially outwards from the cell center. Small molecule inhibitor studies establish that spreading was not analogous to lamellipodium formation but was sensitive to perturbations of actin cortical stiffness. Lastly, we apply the principles uncovered in neutrophils to the patterning of surface-active microfluidic vesicles by tuning vesicle-substrate adhesion and repulsion at the contact interface. The generation of ordered arrays of micron scale vesicles was a first of its kind.

Table of Contents

| | |
|---|-----|
| Acknowledgements..... | iii |
| Abstract..... | iv |
| List of Figures..... | ix |
| Chapter 1: Overview of the Thesis | 1 |
| Organization..... | 1 |
| Specific Aims..... | 2 |
| Aim 1: Quantify effect of adhesion density on neutrophil shape and motility | 3 |
| Aim 2: Elucidate the length scale of neutrophil density sensing | 4 |
| Aim 3: Measure the forces associated with adhesion-driven spreading of neutrophils..... | 4 |
| Aim 4: Spatially organize acellular microfluidic vesicles into arrays using surface adhesion | 5 |
| Chapter 2: Background | 7 |
| The Human Neutrophil | 7 |
| Microcontact Printing | 11 |
| microfabricated-Post-Array-Detectors..... | 13 |
| Models of Cell Spreading | 16 |
| References..... | 21 |
| Chapter 3: Ligand Density Elicits a Phenotypic Switch in Human Neutrophils | 27 |
| Preface..... | 27 |
| Abstract..... | 27 |
| Introduction..... | 28 |
| Materials and Methods..... | 29 |
| Media | 29 |
| Substrates | 30 |
| Protein Deposition and Blocking | 31 |
| Neutrophil Isolation | 32 |
| Cell Motility Experiments..... | 32 |
| Integrin Blocking | 34 |
| Results and Discussion | 34 |
| Observation of Neutrophil Phenotypes on Two Different Substrates . | 34 |
| Quantifying Motility of Amoeboid and Keratocyte-Like Phenotypes | 44 |
| Flow Cytometry to Assess Activation State | 46 |
| Effect of Chemoattractant on Keratocyte-Like Motility..... | 48 |
| Identifying Integrin Chains Responsible for Adhesion | 53 |
| Conclusions..... | 57 |
| Acknowledgements..... | 57 |

| | |
|--|-----|
| References..... | 58 |
| Chapter 4: Human Neutrophil Adhesion Density Sensing at the Whole Cell | |
| Length Scale..... | 61 |
| Preface..... | 61 |
| Abstract..... | 61 |
| Introduction..... | 62 |
| Materials and Methods..... | 64 |
| Media and Reagents..... | 64 |
| Substrate Production..... | 65 |
| Neutrophil Isolation..... | 66 |
| Quantitative Fluorescence Microscopy..... | 67 |
| Cell Motility Experiments and Data Analysis..... | 67 |
| Results..... | 68 |
| Engineering Substrates to Present Neutrophils with Two Adhesive Length Scales..... | 68 |
| Neutrophils Integrate Adhesive Stimulation Across Entire Contact Interface..... | 72 |
| Comparable Neutrophil Motility on Discrete Island and Continuous Fields..... | 77 |
| Discussion..... | 80 |
| Acknowledgements..... | 83 |
| References..... | 84 |
| Chapter 5: Protrusive and Contractile Forces of Spreading Human Neutrophils..... | |
| Preface..... | 86 |
| Abstract..... | 86 |
| Introduction..... | 87 |
| Materials and Methods..... | 89 |
| Media and Reagents..... | 89 |
| microfabricated-Post-Array-Detectors (mPADs) and Microcontact Printing..... | 90 |
| Calculation of Post Spring Constant..... | 91 |
| Neutrophil Isolation..... | 93 |
| Spreading Experiments..... | 93 |
| Antibody Blocking and Cytoskeletal Inhibitor Studies..... | 94 |
| Cell Profile Imaging..... | 94 |
| Data Analysis..... | 95 |
| Identification of Cell-Engaged Posts..... | 95 |
| Cell Reference Frame Coordinate System..... | 95 |
| Computing a Spreading Velocity..... | 96 |
| Results and Discussion..... | 96 |
| Neutrophil Spreading on mPADs..... | 96 |
| Metrics of Spreading and Contractility..... | 102 |
| Biochemical Perturbations of the Cell Cytoskeleton..... | 106 |

| | |
|---|-----|
| Spreading is Haptokinetically Induced | 111 |
| Estimating Extent of Post Sidewall Printing..... | 115 |
| Estimating Energy of Neutrophil-FN Interaction | 117 |
| Origin of the Protrusive Signal | 121 |
| Conclusions..... | 124 |
| Acknowledgements..... | 128 |
| References..... | 129 |
| | |
| Chapter 6: Single Vesicle Patterning of Uniform, Giant Polymersomes Into Microarrays | 132 |
| Preface..... | 132 |
| Abstract | 132 |
| Introduction..... | 133 |
| Materials and Methods..... | 135 |
| Reagents | 135 |
| Polymersome Preparation | 136 |
| Polymersome Functionalization..... | 136 |
| Substrate Fabrication | 137 |
| Vesicle Patterning | 137 |
| Results and Discussion | 138 |
| Producing Monodisperse Populations of Micron-Scale Microfluidic Vesicles | 138 |
| Controlling Microfluidic Payload Encapsulation | 141 |
| Surface Functionalizing Vesicles for Controlled Adhesion..... | 141 |
| Spatially Patterning Arrays of Functionalized Vesicles and an Application..... | 142 |
| Conclusions..... | 153 |
| Acknowledgements..... | 153 |
| References..... | 154 |
| | |
| Chapter 7: Future Directions..... | 156 |
| Visualizing the Cytoskeleton | 156 |
| Fixation and Vinculin Staining Method Notes | 161 |
| Neutrophil Motility on mPADs | 162 |
| Additional Small Molecule Inhibitor Work..... | 170 |
| Vesicle Haptokinesis..... | 171 |
| References..... | 175 |
| | |
| Appendix A: Custom MATLAB Code for Analysis of Neutrophil Motility..... | 177 |
| Introduction..... | 177 |
| Methodology | 177 |
| Code | 184 |
| | |
| Appendix B: Custom MATLAB Code for Analysis of Neutrophil Spreading..... | 307 |
| Introduction..... | 307 |

| | |
|---------------------------------|-----|
| Methodology | 309 |
| Areas for Code Improvement..... | 314 |
| Code | 314 |
| References..... | 386 |

List of Figures

| | | |
|-------------|---|----|
| Figure 3.1 | Two neutrophil morphologies..... | 35 |
| Figure 3.2 | Phenotype does not follow method of protein deposition | 37 |
| Figure 3.3 | Keratocyte-like phenotype recapitulated on Pluronic-blocked glass..... | 38 |
| Figure 3.4 | Exquisite cell-ligand specificity on Pluronic-blocked substrates | 39 |
| Figure 3.5 | Microcontact printing overview and sub-saturating density quantification | 40 |
| Figure 3.6 | Neutrophil phenotype on increasing densities of fibronectin (FN) ... | 42 |
| Figure 3.7 | Mean squared displacements (MSDs) of two motility modes | 45 |
| Figure 3.8 | Quantification of L-selectin expression levels <i>via</i> flow cytometry ... | 47 |
| Figure 3.9 | Quantification of neutrophil haptokinesis and chemokinesis of keratocyte-like phenotype..... | 50 |
| Figure 3.10 | Sample sizes per condition and complete results of model-independent significance testing..... | 51 |
| Figure 3.11 | MSDs of all independent observations of FN/fMLF experimental conditions tested..... | 52 |
| Figure 3.12 | Integrin blocking on FN..... | 54 |
| Figure 3.13 | Neutrophil adhesion to BSA | 56 |
| Figure 4.1 | Stamp-off method of microcontact printing to generate island arrays..... | 69 |
| Figure 4.2 | Stamp-off method of microcontact printing to generate island arrays: no contrast enhancement..... | 70 |
| Figure 4.3 | Measurement of mean printed island diameter and pitch | 71 |
| Figure 4.4 | Engineering islands with two adhesive length scales | 73 |
| Figure 4.5 | Neutrophils sense density at whole cell length scale | 75 |
| Figure 4.6 | Observation of each phenotype per experimental condition..... | 76 |
| Figure 4.7 | Neutrophil motility on islands is comparable to low density continuous fields | 78 |

| | | |
|-------------|--|-----|
| Figure 4.8 | MSD analysis of neutrophil motility on islands and fields..... | 79 |
| Figure 5.1 | Human neutrophil spreading on fibronectin printed mPADs | 97 |
| Figure 5.2 | Identification of cell-engaged posts by variance analysis..... | 98 |
| Figure 5.3 | Cell reference frame coordinate transformation schematic | 100 |
| Figure 5.4 | Spatial dichotomization of force trajectories | 101 |
| Figure 5.5 | Characterizing protrusion and contraction using the ensemble of neutrophil spreading events | 104 |
| Figure 5.6 | Per-cell spreading velocity analysis..... | 105 |
| Figure 5.7 | Cytoskeletal perturbation <i>via</i> small molecule inhibitors | 109 |
| Figure 5.8 | XZ kymograph of neutrophil spreading on stiff mPADs..... | 110 |
| Figure 5.9 | Vertical profiles of neutrophils imaged <i>via</i> confocal microscopy | 113 |
| Figure 5.10 | Metrics of XZ and XY cell profiles | 114 |
| Figure 5.11 | Confocal images of 0.49 μm diameter green fluorescent beads using the same magnification and acquisition settings as post arrays..... | 118 |
| Figure 5.12 | Confocal measurements of FN-AlexaFluor488 printed post arrays in a solution of 90% glycerol | 119 |
| Figure 5.13 | Post invagination as origin of protrusive signature..... | 122 |
| Figure 6.1 | Controlling vesicle size and loading..... | 139 |
| Figure 6.2 | Controlling vesicle diameter by adjusting the continuous phase flow rate | 140 |
| Figure 6.3 | Functionalizing polymersomes <i>via</i> biotinylation..... | 143 |
| Figure 6.4 | NeutrAvidin (NAv) functionalization of polymersomes made from biotin-modified polymer | 144 |
| Figure 6.5 | Patterning single polymersomes | 146 |
| Figure 6.6 | Polymersome capture on NAv-printed surfaces | 147 |
| Figure 6.7 | Polymersomes patterned on NAv islands of 50 μm diameter and 100 μm pitch | 148 |
| Figure 6.8 | Effect of NAv surface area on non-specific binding | 149 |

| | | |
|-------------|---|-----|
| Figure 6.9 | Polymersomes patterned on smaller NAv islands of 10 μm diameter and 50 μm pitch | 151 |
| Figure 6.10 | Creating sensor arrays..... | 152 |
| Figure 7.1 | Loss of phenotype after paraformaldehyde fixation | 157 |
| Figure 7.2 | Improved phenotype stability and vinculin plaque detection with microtubule stabilizing buffer fixation | 159 |
| Figure 7.3 | Summary of mPADs specifications | 163 |
| Figure 7.4 | Neutrophil motility on post arrays is stiffness dependent..... | 165 |
| Figure 7.5 | Neutrophil traction generation is stiffness dependent..... | 168 |
| Figure 7.6 | Attempted vesicle haptokinesis experiment..... | 172 |
| Figure A.1 | Neutrophil motility data analysis workflow | 178 |
| Figure B.1 | Resting lattice bitmap reconstruction..... | 308 |

Chapter 1

Overview of the Thesis

Organization

Broadly speaking, the unifying theme of this thesis work was biological adhesion. The majority of investigations contained herein explored how a particular human immune cell called the neutrophil responded to environmental adhesive cues in terms of motility, mechanical force generation, and the molecular constituents involved in those two processes. Additionally, studies on the application of biological adhesion and repulsion to acellular vesicle patterning were also pursued. In all studies, the unifying methodology employed was microcontact printing. This is a powerful process for spatially organizing the adhesive environment of cellular and acellular systems. Using microcontact printing we engineered environments to study the response of neutrophils and surface active microfluidic vesicles to extracellular adhesive cues.

The thesis is organized into seven chapters and two appendices. Chapter 2 presents the reader with background sufficient to contextualize the experimental results. It includes a general overview of the biological role of neutrophils in the body and the evolution of microcontact printing as a technological platform. Chapter 3 explores adhesion density as a controller of neutrophil shape and migratory phenotype. Elucidating the receptor responsible for mediating this density-sensitive adhesion is also presented. The custom MATLAB codes developed to analyze the motility data are provided in Appendix A. Chapter 4 addresses an outstanding question that arises from the prior

chapter, namely what is the length scale of the neutrophil's sensitivity to adhesive density? By rational design of hybrid surfaces in which high and low adhesive stimulation is presented to the cells simultaneously we address this question. In Chapter 5 we consider the mechanics associated with adhesion-induced spreading prior to the onset of the motility observed in Chapters 3 and 4. Here we employed a traction platform to measure the protrusive and contractile forces associated with spreading. A substantial portion of this chapter is also devoted to elucidating the cytoskeletal components involved in the protrusive and contractile phases of spreading. The custom MATLAB codes developed to analyze the mechanical spreading data are provided in Appendix B. Chapter 6 transitions to acellular microfluidic vesicles which have been rendered surface active through biotinylation. The role of interface adhesion and repulsion is explored in patterning these vesicles into regular arrays. Chapter 7 explores future directions for further inquiry. Importantly, preliminary and pilot experimental observations accompany these recommendations.

The work contained herein is the product of years of collaborative effort with researchers inside and outside the Hammer laboratory. In that spirit, a preface to each experimental chapter is included which lists the co-authors involved in that chapter's studies as well as their individual contributions. The preface also details if the content of that chapter has been published in a peer-reviewed journal or is under review.

Specific Aims

The following specific aims are provided to help organize the thesis content and articulate the motivations for pursuing each aim as well as explicitly state the hypotheses

being tested. Because in multiple cases the initial hypotheses were rejected, the principal experimental outcomes of the aims are also summarized.

Aim 1: Quantify effect of adhesion density on neutrophil shape and motility

Motivation: Two characteristic migratory phenotypes have been reported of neutrophils on planar (unconstrained two dimensional) surfaces: amoeboid and keratocyte-like. Observations of keratocyte-like phenotype were previously ascribed uniquely to the underlying stiffness of the migratory surface. However, scattered observations of both phenotypes were present on substrates of equivalent stiffness throughout the literature. The goal of this aim was to reconcile these disparate observations by considering the effect of surface adhesivity on neutrophil shape and mode of migration.

Hypotheses: Neutrophil shape and motile phenotype are strongly controlled by the underlying adhesivity of the extracellular environment. Integrin receptors will mediate this adhesion.

Outcomes: Neutrophil shape and motile phenotype were dictated by adhesion density on equivalently stiff substrates. The differences were qualitatively and quantitatively distinct. On highly adhesive surfaces, neutrophils assumed the classic amoeboid phenotype having a narrow elongated cell body, moving quickly, and performing frequent directional changes. On low and moderately adhesive surfaces neutrophils assumed a phenotype reminiscent of fish epithelial keratocyte cells featuring a highly spread lamellipodium, moving slowly, but in a directionally persistent manner. Adhesion was found to be mediated by the promiscuous MAC-1 ($\alpha_M\beta_2$) receptor. We demonstrated this promiscuity by recapitulating the findings on a second adhesive ligand.

Aim 2: Elucidate the length scale of neutrophil density sensing

Motivation: In the previous aim we found that neutrophils assumed the amoeboid phenotype on highly adhesive surfaces and the keratocyte-like phenotype on low and moderately adhesive surfaces. The goal of this aim was to consider the length scale of this density sensing and present neutrophils with hybrid environments in which high and low density cues were presented to the cells simultaneously.

Hypothesis: If neutrophils sense density on the submicron length scale they will assume the amoeboid phenotype on discrete islands of high density protein, despite the total protein content across the cell-substrate interface being low. Conversely, if neutrophils integrate adhesive stimulation across their cell bodies they will assume the keratocyte-like phenotype on the same surfaces despite the submicron, on-island protein density being high.

Outcomes: Neutrophils assume the keratocyte-like phenotype on discrete islands in which the on-island (local) protein density was high but the area average (global) protein density was low. By careful design and validation of these hybrid surfaces we were able to conclude that neutrophils integrate adhesive stimulation over their entire cell-substrate contact interface and respond to discrete islands as if they were a continuous field of low density protein.

Aim 3: Measure the forces associated with adhesion-driven spreading of neutrophils

Motivation: In the previous two aims the focus was on quantifying long time (i.e. minutes and hours) motility elicited by surface adhesivity. However neutrophil spreading was a prerequisite to the onset of motility and was known to be a temporally fast (i.e.

seconds) phenomenon. Here the goal was to quantify the forces associated with neutrophil spreading and identify the molecular components involved.

Hypotheses: Neutrophil spreading will be an active process analogous to lamellipodium formation in which actin polymerization and branching protrude the cell membrane outwards.

Outcomes: Adhesion driven spreading was sufficiently forceful to generate detectable deflections on the order of pN. However, using inhibitors of various cytoskeletal components, we demonstrated that this protrusion was not the result of lamellipodium formation. Rather we showed that adhesion-driven spreading was a competition between surface energy at the cell-substrate interface and resistance to shape change imparted by the cell cortical tension. Protrusion was observed because a small degree of adhesive protein on the sidewall of pillar tips induces the cell to spread through a finite volume of pillar tips.

Aim 4: Spatially organize acellular microfluidic vesicles into arrays using surface adhesion

Motivation: Regular arrays of pay-load capable micron scale acellular vesicles were lacking as an experimental platform for the study of vesicle sensing and inter-vesicle communication. Our goal was to realize such an array by merging the technological platforms of microcontact printing and microfluidic vesicle generation.

Hypotheses: Adhesive islands via microcontact printing will stabilize micron-scale microfluidic vesicles rendered surface active through biotinylation. Interstitial pluronic will prevent non-specific vesicle adhesion.

Outcomes: We demonstrated a first of its kind organization of micron scale microfluidic vesicles into regular arrays by adhesion stabilization. Although we initially hypothesized that the role of between-island PEGylation was to inhibit non-specific binding we actually found that the PEG brush in these interstitial spaces interacted with the PEG brush on the vesicle membrane and induced vesicle mobility through steric repulsion. Once on an adhesive island, biotin-avidin ligation stabilized the vesicle against further transit provided the adhesive plaque was sufficiently large.

Chapter 2

Background

The Human Neutrophil

The complex and distributed organ that is the mammalian immune system is often dichotomized in terms of the innate and adaptive branches for the sake of pedagogy. The innate branch consists of physical barriers to pathogen challenges (e.g. epidermis and mucosal films) as well as the family of terminally differentiated granulocytes (e.g. neutrophils and eosinophils) that are capable of executing their response to infection and trauma on the timescale of seconds and minutes. While fast, the response is not highly specific. Conversely the adaptive branch is comprised of the family of T-cells and B-cells that mount a highly specific pathogen response and confer long term immunological memory to the host organism. However, this exquisitely specific response requires a latency period of hours and days to fully develop. Thus, the two branches are complementary and collectively ensure the temporal continuity of the host organism's immunity. In reality the distinction is entirely pedagogical as the constituent elements are intimately and continuously engaged in a dynamic cross talk (1).

Neutrophils, as do all blood cells, originate in the bone marrow from hematopoietic precursor cells. They represent the largest fraction of blood cells continuously generated at $\sim 10^{11}$ neutrophils per day in healthy adults. Once matured these cells are equipped with a variety of terminal functions including pathogen engulfment (phagocytosis), secretion of soluble chemical cues (cytokines), production of

reactive oxygen intermediates (ROIs), and the controlled release of nuclear DNA to form nuclear-extracellular-traps (NETs) (2).

A prerequisite to the execution of any of these terminal functions is the cell's arrival at the locus of trauma (3) or infection (4) *via* vascular rolling, extravasation, and extravascular migration (5). This spatially and temporally controlled sequence of events is collectively known as the leukocyte adhesion cascade. Our detailed molecular understanding of this sequence is the result of decades of empirical observations *in vivo* and *in vitro*.

In the vasculature, quiescent neutrophils transiently roll along the endothelial cell wall in a selectin-mediated capacity and can be induced to arrest after encountering endothelial-immobilized chemokines and chemoattractants (6). Arrest is achieved through integrins, a family of heterodimeric receptors expressed on the cell surface, which ligate a variety of extracellular adhesive ligands and enable cell anchorage. The two integrins of predominate importance in the leukocyte adhesion cascade are LFA-1 ($\alpha_L\beta_2$) and MAC-1 ($\alpha_M\beta_2$), both of which ligate the ICAM-1 adhesive ligand which is also expressed on the endothelial cell surface (7). LFA-1 and MAC-1 work cooperatively to enable neutrophil firm arrest with the later being particularly sensitive to chemoattractant stimulation (8). In a process known as inside-out integrin activation, chemokine ligation by cell-surface G-protein coupled receptors induces increased MAC-1 affinity and, consequently, cell adhesiveness (9). Following firm arrest, neutrophils exit the vasculature between or through endothelial cells and migrate to the tissue wound site (5).

In addition to soluble and immobilized chemical cues that direct immune cell response and function (10-15), cells encounter numerous physical cues in the body (e.g. stiffness, dimensionality, adhesivity, and roughness) that are strong determinants of shape, force generation, and gene expression (16-17). Blood cell response to physical cues such as substrate rigidity (15, 18-21), confinement (22-23), shear force (24), and adhesion density (25-26) have been areas of on-going investigation. The focus of this thesis is the role of adhesive ligand density on directing neutrophil phenotype in terms of cell shape, motility, mechanical force generation, and the molecular constituents involved in these processes. The motivation for considering this particular environmental factor stemmed from the desire to reconcile two conflicting morphological observations of neutrophil shape and motility in the literature.

On a majority of two-dimensional *in vitro* substrates, neutrophils are reported to exhibit an amoeboid morphology (27-33). The distinguishing features of this phenotype are a narrow, elongated cell body with a frontward ruffled-lamellipodium and rearward knob-like uropod (27). Detailed images of this morphology have been captured with high resolution scanning electron microscopy (SEM) (34-35). However, there have also been observations of neutrophils assuming a very different, well-spread, uropod-absent, phenotype on two-dimensional substrates (15, 18-19). In those instances the alternative phenotype was attributed exclusively to the underlying stiffness of the material, as neutrophil spread area was shown to increase with increasing substrate rigidity. Yet, the amoeboid phenotype was also observed on stiff substrates such as those in the previously mentioned SEM studies. This suggested to us that substrate stiffness was not a unique controller of neutrophil morphology. Hypothesizing that another factor was involved in

modulating these two phenotypes, our work focused on the role of adhesion ligand density. This hypothesis was motivated by observed adhesion density sensitivity in fish keratocytes (36) and computational predictions of adhesion density effects on cell motility (37).

It is important to comment on the nature of the neutrophil-surface interaction we are exploring in this thesis as it differs in considerable ways from the adhesive interactions just discussed in the context of the leukocyte adhesion cascade. The basis of the empirical work presented in subsequent chapters is neutrophil adhesion, motility, and force generation induced by haptokinetic interaction of the cell with fibronectin under static (no flow) conditions. Fibronectin (FN) is a large glycosylated adhesive ligand present in blood and extracellular matrices (38). Through function blocking antibody studies we ultimately attribute our observed neutrophil adhesion and motility to MAC-1 (Fig. 3.12). We suggest that the promiscuity of this receptor for multiple adhesive ligands demonstrated by us (Fig. 3.13) and others (39-40) confers the neutrophil with the ability to mount an immunologic response even the absence of canonical selectin-mediated rolling and chemoattractant-induced firm arrest.

To validate that our experimental platform was a valid model of haptokinetic stimulation, we used L-selectin as a marker of neutrophil quiescence. L-selectin expression levels are a sensitive indicator of a neutrophil's transition from quiescence (high expression) to a phenotype primed for integrin-mediated binding (low expression) (41). Our series of flow cytometry control experiments revealed that while neutrophils were capable of chemoattractant-stimulated (inside-out) integrin activation, they were not primed for binding prior to FN exposure (Fig. 3.8). As such our model of neutrophil

haptokinesis is itself a model of the alternative integrin activation pathway called outside-in stimulation. In contrast to inside-out activation, the outside-in pathway is whereby low affinity integrin-ligand interactions induce a high affinity conformational change in the integrin receptor (42).

Microcontact Printing

To probe neutrophil response to adhesive ligand density and organization we employed microcontact printing (43). The ability to pattern molecules on the micron scale was pioneered by the Whitesides group and initially demonstrated by organizing self-assembled monolayers of alkanethiols on gold substrates using elastomeric stamps to achieve transfer (44). Elastomeric stamps of poly(dimethylsiloxane) (PDMS), sold under the trade name Sylgard 184 by Dow Corning, are themselves patterned by casting against silicon wafers etched by standard photolithographic techniques (45). After the PDMS stamps are cured, the peeled stamp has the complimentary topography of the silicon master against which the stamp was initially cast.

This engineering approach to spatially organizing a cell's adhesive environment has been widely used to probe integrin clustering (46-47), effect of cell shape on viability (48) and focal adhesion architecture (49), and the role of extracellular matrix distribution on cell spreading (50) in the context of mesenchymal cells. In this thesis we apply the same principles to elucidate the effect of adhesive ligand density distribution on human neutrophils. Only recently has microcontact printing been applied in studies of hematopoietic-derived cells (12, 14, 51-52).

The form of microcontact printing employed in this thesis is the direct patterning of protein molecules onto PDMS surfaces. This is facilitated by the fact that PDMS,

natively hydrophobic, can be rendered hydrophilic by plasma oxidation (53-54). The differential hydrophobicity of the stamp and substrate are critical in mediating protein transfer at the interface (55). In our experiments a thin layer of PDMS, spun on glass coverslips and cured, is rendered hydrophilic by treatment in ultraviolet ozone (UVO) (47). When a protein-inked PDMS stamp (hydrophobic) contacts a UVO-exposed PDMS coverslip (hydrophilic), a preferential transfer of protein from the hydrophobic stamp to the hydrophilic coverslip occurs. In the case of a flat stamp a continuous field of protein is transferred to the substrate. In the case of a stamp with topographical features a discretized pattern of protein is transferred. A schematic of this process is provided in Figure 3.5.

A variation of microcontact printing we also employ to organize the spatial adhesive environment of neutrophils is referred to as “stamp-off”. Stamp-off was developed by Desai and coworkers and exploits the tunable hydrophobicity of PDMS by inserting an additional step in the normal microcontact printing work flow (47). Prior to contacting a flat, uniformly inked PDMS stamp (hydrophobic) to a plasma-treated PDMS coverslip (hydrophilic), the original stamp is itself contacted by a plasma-treated PDMS stamp (hydrophilic) with topographical features. The result is selective removal (i.e. “stamp-off”) of protein from the uniformly inked stamp. A schematic of this process is provided in Figure 4.1. Successive iterations of stamp-off and re-inking can result in complex patterns of multiple adhesive ligands on a flat stamp (47). The advantage stamp-off affords is the elimination of failure modes like feature collapse and ceiling sag that direct stamp-on methods can succumb (43).

The final step in microcontact printing on PDMS substrates is passivation or blocking of the bare regions not occupied by protein. This is accomplished by incubation of the printed substrate in a dilute solution of the non-ionic triblock copolymer Pluronic F-127 (poly(ethylene oxide)-poly(propylene oxide)-poly(ethylene oxide)). The PPO core is hydrophobic and organizes along the PDMS interface while the PEO termini are hydrophilic and project into the aqueous environment. The result is a passivated surface resistant to neutrophil binding (Fig. 3.4). Pluronic mediated resistance is critical as conventional passivation strategies (e.g. albumin incubation) induce non-specific neutrophil adhesion. In this thesis the neutrophil adhesion and motility are entirely attributable to the printed adhesive ligand with no off-ligand effects. Pluronic passivation is stable for days (56) and possible on a variety of tissue culture amenable substrates including glass and tissue culture plastic (57). Lastly, Pluronic is amenable to fluorescence tagging which can facilitate quantitative measurements of the extent of surface loading (58).

microfabricated-Post-Array-Detectors

A number of traction platforms have been developed to measure a cell's mechanical interaction with its surrounding microenvironment. These include wrinkling silicone films (59), two-dimensional (60-61) and three-dimensional (62) bead-in-gel systems, fluorescent markers in PDMS sheets (63), micromachined cantilevers (64), as well as moderately dense (65) and ultra-dense (66) arrays of vertical polymeric posts (microfabricated-Post-Array-Detectors or "mPADs"). In this thesis we employ mPADs to quantify the forces associated with neutrophil spreading as the system represents a natural extension of the PDMS-based work done to assess adhesive ligand density sensitivity in

the same cells. A note on nomenclature, the descriptors “mPADs”, “post arrays”, and “pillar arrays” are used interchangeably throughout this thesis.

The same basic fabrication scheme previously described to achieve topographical stamps for microcontact-printing (i.e. photolithographic etching of a silicon master) is utilized to generate mPADs. However, in the case of mPADs, cells are in a sense induced to spread on the topographical “stamp” itself. This is a subtle over-simplification as the topographical stamp actually serves as a negative relief of the silicon master, used to cast a replica of the silicon master features in PDMS prepolymer onto thin glass coverslips. Extensively annotated protocols for the fabrication of mPADs, their functionalization, and the associated analyses of cell deflections have been made publically available by the Chen group (67-68).

There are a variety of stiffness definitions to describe the discretized environment a cell experiences on mPADs. Figure 7.3 summarizes the specifications of each post array discussed in this thesis as well as a number of stiffness metrics. On the simplest level, each pillar can be modeled as a cantilever subjected to a load at its unconstrained terminus (see derivation of Eq. 5.4) (69) in which case the material spring constant (k_{spring}) is a natural description of pillar stiffness. Pillar stiffness is altered by changing the diameter, length, or modulus of the polymer used for casting. In this thesis stiffness is tuned by varying diameter and length parameters. Important theoretical work done by Schoen and coworkers established a series of corrections to these spring constants as a function of post aspect ratio. The corrections account for the contribution of pillar tilting and base warping to the measured free terminus deflection (70). The magnitude of the correction decreases nonlinearly with increasing post aspect ratio (i.e. the ratio of the post

diameter to its width). While the spring constant is a natural description of single pillar stiffness, it is ambiguous with respect to the macroscopic or ensemble (multi-post) stiffness perceived by a cell spanning many such posts. In a macroscopic context it is more natural to describe the arrays in terms of a Young's modulus (E) or shear modulus (G) (71). Alternatively, Ladoux and coworkers developed a theoretical description of effective array stiffness by solution of the Green's function for a discretized substrate (under certain governing assumptions) (72). The Ladoux model estimates the Young's moduli of post arrays as being substantially softer than anticipated by a local pure shear model. While different definitions of stiffness yield different stiffness values, relative differences within a given experiment can yield insights into the effect of microenvironment stiffness on biological function.

The nature of the cell-pillar interaction has been an active area of study. Through confocal microscopy and reconstruction of the vertical profile of cell-engaged pillars, Lemmon and coworkers established that shear is a greater contribution to post deflection than torque (73). A criticism sometimes leveled against mPADs as a traction platform is the discretized nature of the cell-substrate interaction. Work by Lenhart and coworkers in mesenchymal cells (50) as well as our own findings in neutrophils (Chapter 4) demonstrate that this discretization is not prohibitive and does not produce anomalous biological phenotypes. In fact the integrated response of local adhesive stimuli over the cell body to induce a coordinated whole cell response, yields important insight into the nature of the intracellular signaling at play during cell spreading and traction generation.

Models of Cell Spreading

In Chapter 5 we report the first measurements of neutrophil protrusive force during adhesion ligand induced spreading. The treatment of cell spreading as a physical phenomenon, a thermodynamic competition between the energy of the adhesive environment driving the cell to spread and the cell's cohesive forces resisting shape change, has a long history. S. B. Carter first observed a preference in mouse fibroblasts for adhesion to high density palladium-shaded glass over low density (74). Carter dubbed fibroblast motility up a gradient of adhesivity "haptotaxis." Importantly he drew a physical analogy with the wetting of a liquid droplet on a surface. The equilibrium shape of such a droplet in an aqueous medium is described by Young's equation relating the angle of the droplet-substrate interface to the substrate-medium, droplet-medium, and substrate-droplet interfacial energies. In the same manuscript as Carter's haptotaxis observations, J. L. Moilliet modeled the fibroblast as a liquid droplet subject to Young's equation and then extended the model to include a third liquid interface (a thin shell surrounding the liquid droplet) in an attempt to recapitulate the effect of the cell membrane enclosing the cell cytoplasm (74). The incorporation of this additional boundary interface had the effect of increasing the observed contact angle. The conclusion was that the cell membrane imparts the cell with a greater resistance to spreading than a purely liquid model alone would predict.

Single cell micropipette aspiration of quiescent (i.e. not spread) neutrophils has been reliably used to measure neutrophil cortical tension and cytoplasmic viscosity. In nearly all cases, passive neutrophils are modeled as simple viscous fluid drops with constant surface tension, a model also referred to as the Newtonian liquid drop model

(75-79). While the Waugh group has demonstrated some non-Newtonian characteristic in the neutrophil cytoplasm, namely a power law relation between viscosity and shear rate (80), the prevalent treatment is of the resting neutrophil as a passive viscous liquid drop with apparent surface tension.

Actin in a quiescent neutrophil is confined to a thin cortical shell proximal to the cytoplasmic membrane (81). Disruption of actin polymerization kinetics with the small molecule inhibitors jasplakinolide and cytochalasin B has been found to alter the measured surface tension of resting neutrophils subjected to micropipette aspiration. Jasplakinolide induces actin polymerization and stabilizes filamentous actin (82). In neutrophils, pretreatment with jasplakinolide has been shown to increase the rigidity of the cortex as measured by micropipette aspiration (83). Conversely, cytochalasin B is known to dramatically reduce the rate of actin polymerization and simultaneously interfere with filament-filament interactions that stabilize the actin network (84). In contrast to jasplakinolide stiffening in neutrophils, pretreatment with cytochalasin B has been shown to decrease cortical rigidity as measured by micropipette aspiration (78). In this thesis we extend these observations made in passive, non-adherent neutrophils to neutrophils undergoing adhesion-induced spreading. We find that spreading is completely abrogated following cortical stiffening (*via* jasplakinolide) while the velocity of spreading is dramatically reduced following cortical softening (*via* cytochalasin B).

More recent work by Cuvelier and coworkers considers the dynamics of cell spreading (in terms of contact area radius as a function of time) when modeled as a viscous shell that encloses a liquid droplet (85). McGrath provides a concise summary of Cuvelier's model in reference (86). The model predicts two spreading regimes. At short

times contact radius evolves as $R \sim t^{0.5}$. At long times the adhesive patch is comparable to the size of the cell and contact radius evolves as $R \sim t^{0.25}$. The Cuvelier group considers this model of a viscous cortex encasing a liquid droplet to be broadly applicable as they empirically demonstrate its relevance to the spreading of mesenchymal carcinoma cells (HeLa and S180) and biotinylated red blood cells.

However, the appropriateness of the Cuvelier model in recapitulating neutrophil spreading dynamics is debatable. Recently, Waugh and coworkers observed that neutrophil spreading on fields of the chemoattractant IL-8 exhibit a linear increase in contact radius with time and a logarithmic deceleration in spreading velocity (87). We estimate from their published data that contact radius evolves as roughly $R \sim t^{0.8}$, a spreading rate significantly faster than the short time ($R \sim t^{0.5}$) and long time ($R \sim t^{0.25}$) regimes predicted by the viscous shell surrounding a liquid drop model. Additionally, Waugh and coworkers observe that the neutrophil contact area grows so quickly its diameter exceeds the equatorial diameter of the quiescent neutrophil for the majority of the experimental observation. This implies that if a comparison is to be made it must be made within the long time viscous dissipation regime of Cuvelier's model. A regime where the discrepancy between Waugh's empirical $R \sim t^{0.8}$ and Cuvelier's predicted $R \sim t^{0.25}$ is even more conspicuous.

However, in this thesis we report neutrophil spreading velocities of adhesion-driven spreading on fibronectin printed mPADs, a model of haptokinetic spreading. While we have limited resolution of the evolution of the spreading neutrophil's contact interface with time, because we are tracking fluorescent post tips and not the neutrophil membrane itself, we can still approximate the spreading velocity in terms of the

propagation rate of the radial protrusive force (Fig. 5.1 and Fig. 5.5 C). We estimate that our neutrophil contact interface grows as $R \sim t^{0.4}$ which is reasonably close to the short time $R \sim t^{0.5}$ dependency predicted by Cuvelier in the viscous shell surrounding a liquid drop model. It is interesting to note that Waugh and coworkers observed a decrease in spreading velocity when the IL-8 receptor (CXCR1) was blocked coupled with the fact that we observe a slower spreading velocity in neutrophils on the adhesive ligand fibronectin as compared to the chemoattractant IL-8. While intracellular signaling is important in both cases (blocking IL-8 and FN receptors reduces or eliminates spreading, see Fig. 5.9 A, *ii*) it does appear that neutrophil spreading on FN induces a mechanically distinct response as compared to IL-8. The increased spreading velocity on immobilized chemoattractant may have biologically relevant implications as chemoattractant immobilization and expression on endothelial cells is a critical homing cue, inducing neutrophils to execute the leukocyte adhesion cascade and exit the vasculature at the locus of trauma or infection (6).

Our observation of neutrophil spreading on FN post arrays is also consistent with previous Hammer laboratory measurements of neutrophil spreading rates on FN as measured by RICM (88) where the total cell contact radius grew as approximately $R \sim t^{0.45}$. There are significant points of departure from the Cuvelier RICM validation experiments in which the region of intimate cell-substrate contact in mesenchymal cells grew as a radially symmetric disk. In neutrophils this symmetry was not observed, as the regions of intimate cell-substrate contact decorated the periphery with virtually no intimate contact at the core. An additional discrepancy is the observation we make in this thesis that cytochalasin B softening of the cortical shell decreases spreading velocity (Fig.

5.7 *B*) whereas cytochalasin D treatment in HeLA cells was found to increase spreading velocity in the Cuvelier studies.

Beyond single mesenchymal cell spreading, the Cuvelier viscous shell surrounding a liquid droplet model has been successfully applied to recapitulate the hydrodynamics of multi-cellular aggregate spreading with relevance to the fields of tissue morphogenesis and cancer metastasis (89-90). In the context of the data presented in this thesis, previous Hammer lab measurements, and recent Waugh lab observations it appears the Cuvelier model may be a reasonable course-grained representation of macroscopic neutrophil spreading on adhesive fields of FN but not immobilized fields of chemoattractant. The “course-grained” qualifier is with respect to the significant points of departure previously enumerated. In summary, our observations suggest that physical wetting is a useful toy model to help conceptualize the competition of interface energies at play during neutrophil spreading, but it alone is not sufficient to reconcile the cumulative body of empirical observations.

References

1. Janeway, C. 2005. *Immunobiology : the immune system in health and disease*. Garland Science, New York.
2. Borregaard, N. 2010. Neutrophils, from marrow to microbes. *Immunity* 33:657-670.
3. McDonald, B., K. Pittman, G. B. Menezes, S. A. Hirota, I. Slaba, C. C. M. Waterhouse, P. L. Beck, D. A. Muruve, and P. Kubers. 2010. Intravascular Danger Signals Guide Neutrophils to Sites of Sterile Inflammation. *Science* 330:362-366.
4. Nathan, C. 2006. Neutrophils and immunity: challenges and opportunities. *Nat Rev Immunol* 6:173-182.
5. Ley, K., C. Laudanna, M. I. Cybulsky, and S. Nourshargh. 2007. Getting to the site of inflammation: the leukocyte adhesion cascade updated. *Nature Reviews Immunology* 7:678-689.
6. Alon, R., and K. Ley. 2008. Cells on the run: shear-regulated integrin activation in leukocyte rolling and arrest on endothelial cells. *Curr Opin Cell Biol* 20:525-532.
7. Evans, R., I. Patzak, L. Svensson, K. De Filippo, K. Jones, A. McDowall, and N. Hogg. 2009. Integrins in immunity. *J Cell Sci* 122:215-225.
8. Smith, C. W., S. D. Marlin, R. Rothlein, C. Toman, and D. C. Anderson. 1989. Cooperative interactions of LFA-1 and Mac-1 with intercellular adhesion molecule-1 in facilitating adherence and transendothelial migration of human neutrophils in vitro. *J Clin Invest* 83:2008-2017.
9. Kinashi, T. 2005. Intracellular signalling controlling integrin activation in lymphocytes. *Nature Reviews Immunology* 5:546-559.
10. Hind, L. E., J. L. Mackay, D. Cox, and D. A. Hammer. 2014. Two-dimensional motility of a macrophage cell line on microcontact-printed fibronectin. *Cytoskeleton (Hoboken)* 71:542-554.
11. Ricart, B. G., B. John, D. Lee, C. A. Hunter, and D. A. Hammer. 2011. Dendritic cells distinguish individual chemokine signals through CCR7 and CXCR4. *J Immunol* 186:53-61.
12. Ricart, B. G., M. T. Yang, C. A. Hunter, C. S. Chen, and D. A. Hammer. 2011. Measuring traction forces of motile dendritic cells on micropost arrays. *Biophys J* 101:2620-2628.
13. Smith, L. A., H. Aranda-Espinoza, J. B. Haun, M. Dembo, and D. A. Hammer. 2007. Neutrophil traction stresses are concentrated in the uropod during migration. *Biophys J* 92:L58-60.
14. Lee, D., and M. R. King. 2008. Microcontact printing of P-selectin increases the rate of neutrophil recruitment under shear flow. *Biotechnol Prog* 24:1052-1059.
15. Jannat, R. A., G. P. Robbins, B. G. Ricart, M. Dembo, and D. A. Hammer. 2010. Neutrophil adhesion and chemotaxis depend on substrate mechanics. *J Phys-Condens Mat* 22.
16. Charras, G., and E. Sahai. 2014. Physical influences of the extracellular environment on cell migration. *Nat Rev Mol Cell Biol* 15:813-824.

17. Vogel, V., and M. Sheetz. 2006. Local force and geometry sensing regulate cell functions. *Nat Rev Mol Cell Biol* 7:265-275.
18. Oakes, P. W., D. C. Patel, N. A. Morin, D. P. Zitterbart, B. Fabry, J. S. Reichner, and J. X. Tang. 2009. Neutrophil morphology and migration are affected by substrate elasticity. *Blood* 114:1387-1395.
19. Stroka, K. M., and H. Aranda-Espinoza. 2009. Neutrophils display biphasic relationship between migration and substrate stiffness. *Cell Motil Cytoskeleton* 66:328-341.
20. Stroka, K. M., H. N. Hayenga, and H. Aranda-Espinoza. 2013. Human neutrophil cytoskeletal dynamics and contractility actively contribute to trans-endothelial migration. *PLoS One* 8:e61377.
21. Hind, L. E., M. Dembo, and D. A. Hammer. 2015. Macrophage motility is driven by frontal-towing with a force magnitude dependent on substrate stiffness. *Integr Biol (Camb)*.
22. Lammermann, T., B. L. Bader, S. J. Monkley, T. Worbs, R. Wedlich-Soldner, K. Hirsch, M. Keller, R. Forster, D. R. Critchley, R. Fassler, and M. Sixt. 2008. Rapid leukocyte migration by integrin-independent flowing and squeezing. *Nature* 453:51-55.
23. Hung, W.-C., S.-H. Chen, C. D. Paul, K. M. Stroka, Y.-C. Lo, J. T. Yang, and K. Konstantopoulos. 2013. Distinct signaling mechanisms regulate migration in unconfined versus confined spaces. *The Journal of Cell Biology* 202:807-824.
24. Dominguez, G. A., N. R. Anderson, and D. A. Hammer. 2015. The direction of migration of T-lymphocytes under flow depends upon which adhesion receptors are engaged. *Integr Biol (Camb)* 7:345-355.
25. Henry, S. J., J. C. Crocker, and D. A. Hammer. 2014. Ligand density elicits a phenotypic switch in human neutrophils. *Integrative Biology* 6:348-356.
26. Dominguez, G. A., and D. A. Hammer. 2014. Effect of adhesion and chemokine presentation on T-lymphocyte haptokinesis. *Integr Biol (Camb)* 6:862-873.
27. Zigmond, S. H. 1978. Chemotaxis by polymorphonuclear leukocytes. *J Cell Biol* 77:269-287.
28. Malawista, S. E., and A. d. B. Chevance. 1997. Random locomotion and chemotaxis of human blood polymorphonuclear leukocytes (PMN) in the presence of EDTA: PMN in close quarters require neither leukocyte integrins nor external divalent cations. *Proceedings of the National Academy of Sciences* 94:11577-11582.
29. Irimia, D., S.-Y. Liu, W. G. Tharp, A. Samadani, M. Toner, and M. C. Poznansky. 2006. Microfluidic system for measuring neutrophil migratory responses to fast switches of chemical gradients. *Lab Chip* 6:191-198.
30. Butler, L. M., S. Khan, G. Ed Rainger, and G. B. Nash. 2008. Effects of endothelial basement membrane on neutrophil adhesion and migration. *Cell Immunol* 251:56-61.
31. Houk, A. R., A. Jilkine, C. O. Mejean, R. Boltyanskiy, E. R. Dufresne, S. B. Angenent, S. J. Altschuler, L. F. Wu, and O. D. Weiner. 2012. Membrane Tension Maintains Cell Polarity by Confining Signals to the Leading Edge during Neutrophil Migration. *Cell* 148:175-188.

32. Sackmann, E. K., E. Berthier, E. W. K. Young, M. A. Shelef, S. A. Wernimont, A. Huttenlocher, and D. J. Beebe. 2012. Microfluidic kit-on-a-lid: a versatile platform for neutrophil chemotaxis assays. *Blood* 120:e45-e53.
33. Yanai, M., J. P. Butler, T. Suzuki, H. Sasaki, and H. Higuchi. 2004. Regional rheological differences in locomoting neutrophils. *American Journal of Physiology - Cell Physiology* 287:C603-C611.
34. Cassimeris, L., H. McNeill, and S. H. Zigmond. 1990. Chemoattractant-stimulated polymorphonuclear leukocytes contain two populations of actin filaments that differ in their spatial distributions and relative stabilities. *The Journal of Cell Biology* 110:1067-1075.
35. Matzner, Y., I. Vlodaysky, R. I. Michaeli, and A. Eldor. 1990. Selective inhibition of neutrophil activation by the subendothelial extracellular matrix: possible role in protection of the vessel wall during diapedesis. *Exp Cell Res* 189:233-240.
36. Barnhart, E. L., K. C. Lee, K. Keren, A. Mogilner, and J. A. Theriot. 2011. An adhesion-dependent switch between mechanisms that determine motile cell shape. *PLoS Biol* 9:e1001059.
37. Ziebert, F., and I. S. Aranson. 2013. Effects of adhesion dynamics and substrate compliance on the shape and motility of crawling cells. *PLoS One* 8:e64511.
38. Engel, J., E. Odermatt, A. Engel, J. A. Madri, H. Furthmayr, H. Rohde, and R. Timpl. 1981. Shapes, domain organizations and flexibility of laminin and fibronectin, two multifunctional proteins of the extracellular matrix. *J Mol Biol* 150:97-120.
39. van den Berg, J. M., F. P. Mul, E. Schippers, J. J. Weening, D. Roos, and T. W. Kuijpers. 2001. Beta1 integrin activation on human neutrophils promotes beta2 integrin-mediated adhesion to fibronectin. *Eur J Immunol* 31:276-284.
40. Lishko, V. K., V. P. Yakubenko, and T. P. Ugarova. 2003. The interplay between integrins alphaMbeta2 and alpha5beta1 during cell migration to fibronectin. *Exp Cell Res* 283:116-126.
41. Kishimoto, T. K., M. A. Jutila, E. L. Berg, and E. C. Butcher. 1989. Neutrophil Mac-1 and MEL-14 adhesion proteins inversely regulated by chemotactic factors. *Science* 245:1238-1241.
42. Jordan, M. S., and G. A. Koretzky. 2010. Coordination of receptor signaling in multiple hematopoietic cell lineages by the adaptor protein SLP-76. *Cold Spring Harb Perspect Biol* 2:a002501.
43. Ruiz, S. A., and C. S. Chen. 2007. Microcontact printing: A tool to pattern. *Soft Matter* 3:168-177.
44. Kumar, A., and G. M. Whitesides. 1993. Features of gold having micrometer to centimeter dimensions can be formed through a combination of stamping with an elastomeric stamp and an alkanethiol "ink" followed by chemical etching. *Applied Physics Letters* 63:2002-2004.
45. Kane, R. S., S. Takayama, E. Ostuni, D. E. Ingber, and G. M. Whitesides. 1999. Patterning proteins and cells using soft lithography. *Biomaterials* 20:2363-2376.
46. Arnold, M., E. A. Cavalcanti-Adam, R. Glass, J. Blummel, W. Eck, M. Kantlehner, H. Kessler, and J. P. Spatz. 2004. Activation of integrin function by nanopatterned adhesive interfaces. *Chemphyschem* 5:383-388.

47. Desai, R. A., M. K. Khan, S. B. Gopal, and C. S. Chen. 2011. Subcellular spatial segregation of integrin subtypes by patterned multicomponent surfaces. *Integr Biol* 3:560-567.
48. Chen, C. S., M. Mrksich, S. Huang, G. M. Whitesides, and D. E. Ingber. 1997. Geometric control of cell life and death. *Science* 276:1425-1428.
49. Chen, C. S., J. L. Alonso, E. Ostuni, G. M. Whitesides, and D. E. Ingber. 2003. Cell shape provides global control of focal adhesion assembly. *Biochemical and Biophysical Research Communications* 307:355-361.
50. Lehnert, D., B. Wehrle-Haller, C. David, U. Weiland, C. Ballestrem, B. A. Imhof, and M. Bastmeyer. 2004. Cell behaviour on micropatterned substrata: limits of extracellular matrix geometry for spreading and adhesion. *Journal of Cell Science* 117:41-52.
51. Shen, K., V. K. Thomas, M. L. Dustin, and L. C. Kam. 2008. Micropatterning of costimulatory ligands enhances CD4⁺ T cell function. *Proceedings of the National Academy of Sciences* 105:7791-7796.
52. Tong, Z., L. S. Cheung, K. J. Stebe, and K. Konstantopoulos. 2012. Selectin-mediated adhesion in shear flow using micropatterned substrates: multiple-bond interactions govern the critical length for cell binding. *Integr Biol (Camb)* 4:847-856.
53. James, C. D., R. C. Davis, L. Kam, H. G. Craighead, M. Isaacson, J. N. Turner, and W. Shain. 1998. Patterned Protein Layers on Solid Substrates by Thin Stamp Microcontact Printing. *Langmuir* 14:741-744.
54. Ye, H., Z. Gu, and D. H. Gracias. 2006. Kinetics of Ultraviolet and Plasma Surface Modification of Poly(dimethylsiloxane) Probed by Sum Frequency Vibrational Spectroscopy. *Langmuir* 22:1863-1868.
55. Tan, J. L., J. Tien, and C. S. Chen. 2002. Microcontact Printing of Proteins on Mixed Self-Assembled Monolayers. *Langmuir* 18:519-523.
56. Nelson, C. M., S. Raghavan, J. L. Tan, and C. S. Chen. 2003. Degradation of micropatterned surfaces by cell-dependent and -independent processes. *Langmuir* 19:1493-1499.
57. Tan, J. L., W. Liu, C. M. Nelson, S. Raghavan, and C. S. Chen. 2004. Simple approach to micropattern cells on common culture substrates by tuning substrate wettability. *Tissue Eng* 10:865-872.
58. Perry, C. C., T. S. Sabir, W. J. Livingston, J. R. Milligan, Q. Chen, V. Maskiewicz, and D. S. Boskovic. 2011. Fluorescence of commercial Pluronic F127 samples: Temperature-dependent micellization. *J Colloid Interface Sci* 354:662-669.
59. Harris, A. K., P. Wild, and D. Stopak. 1980. Silicone-Rubber Substrata - New Wrinkle in the Study of Cell Locomotion. *Science* 208:177-179.
60. Pelham, R. J., and Y. Wang. 1997. Cell locomotion and focal adhesions are regulated by substrate flexibility. *Proceedings of the National Academy of Sciences of the United States of America* 94:13661-13665.
61. Dembo, M., and Y. L. Wang. 1999. Stresses at the cell-to-substrate interface during locomotion of fibroblasts. *Biophys J* 76:2307-2316.

62. Legant, W. R., J. S. Miller, B. L. Blakely, D. M. Cohen, G. M. Genin, and C. S. Chen. 2010. Measurement of mechanical tractions exerted by cells in three-dimensional matrices. *Nat Meth* 7:969-971.
63. Balaban, N. Q., U. S. Schwarz, D. Riveline, P. Goichberg, G. Tzur, I. Sabanay, D. Mahalu, S. Safran, A. Bershadsky, L. Addadi, and B. Geiger. 2001. Force and focal adhesion assembly: a close relationship studied using elastic micropatterned substrates. *Nat Cell Biol* 3:466-472.
64. Galbraith, C. G., and M. P. Sheetz. 1997. A micromachined device provides a new bend on fibroblast traction forces. *Proc Natl Acad Sci U S A* 94:9114-9118.
65. Tan, J. L., J. Tien, D. M. Pirone, D. S. Gray, K. Bhadriraju, and C. S. Chen. 2003. Cells lying on a bed of microneedles: An approach to isolate mechanical force. *Proceedings of the National Academy of Sciences of the United States of America* 100:1484-1489.
66. du Roure, O., A. Saez, A. Buguin, R. H. Austin, P. Chavrier, P. Silberzan, and B. Ladoux. 2005. Force mapping in epithelial cell migration. *Proceedings of the National Academy of Sciences of the United States of America* 102:2390-2395.
67. Desai, R., M. Yang, N. Sniadecki, W. Legant, and C. S. Chen. 2007. Microfabricated Post-Array-Detectors (mPADs): an Approach to Isolate Mechanical Forces. *Journal of Visualized Experiments* 7.
68. Yang, M. T., J. Fu, Y. K. Wang, R. A. Desai, and C. S. Chen. 2011. Assaying stem cell mechanobiology on microfabricated elastomeric substrates with geometrically modulated rigidity. *Nat Protoc* 6:187-213.
69. Beer, F. P., E. R. Johnston, and J. T. DeWolf. 2006. *Mechanics of Materials*. McGraw-Hill Higher Education, Boston.
70. Schoen, I., W. Hu, E. Klotzsch, and V. Vogel. 2010. Probing cellular traction forces by micropillar arrays: contribution of substrate warping to pillar deflection. *Nano Lett* 10:1823-1830.
71. Engler, A. J., S. Sen, H. L. Sweeney, and D. E. Discher. 2006. Matrix elasticity directs stem cell lineage specification. *Cell* 126:677-689.
72. Ghibaudo, M., A. Saez, L. Trichet, A. Xayaphoummine, J. Browaeys, P. Silberzan, A. Buguin, and B. Ladoux. 2008. Traction forces and rigidity sensing regulate cell functions. *Soft Matter* 4:1836-1843.
73. Lemmon, C. A., N. J. Sniadecki, S. A. Ruiz, J. L. Tan, L. H. Romer, and C. S. Chen. 2005. Shear force at the cell-matrix interface: enhanced analysis for microfabricated post array detectors. *Mech Chem Biosyst* 2:1-16.
74. Carter, S. B. 1967. Haptotaxis and the Mechanism of Cell Motility. *Nature* 213:256-260.
75. Evans, E., and B. Kukan. 1984. Passive material behavior of granulocytes based on large deformation and recovery after deformation tests. *Blood* 64:1028-1035.
76. Evans, E., and A. Yeung. 1989. Apparent viscosity and cortical tension of blood granulocytes determined by micropipet aspiration. *Biophys J* 56:151-160.
77. Needham, D., and R. M. Hochmuth. 1992. A sensitive measure of surface stress in the resting neutrophil. *Biophys J* 61:1664-1670.
78. Tsai, M. A., R. S. Frank, and R. E. Waugh. 1994. Passive mechanical behavior of human neutrophils: effect of cytochalasin B. *Biophys J* 66:2166-2172.

79. Lomakina, E. B., C. M. Spillmann, M. R. King, and R. E. Waugh. 2004. Rheological analysis and measurement of neutrophil indentation. *Biophys J* 87:4246-4258.
80. Tsai, M. A., R. S. Frank, and R. E. Waugh. 1993. Passive mechanical behavior of human neutrophils: power-law fluid. *Biophys J* 65:2078-2088.
81. Sheterline, P., and C. R. Hopkins. 1981. Transmembrane linkage between surface glycoproteins and components of the cytoplasm in neutrophil leukocytes. *The Journal of Cell Biology* 90:743-754.
82. Holzinger, A. 2009. Jasplakinolide: an actin-specific reagent that promotes actin polymerization. *Methods Mol Biol* 586:71-87.
83. Sheikh, S., W. B. Gratzler, J. C. Pinder, and G. B. Nash. 1997. Actin polymerisation regulates integrin-mediated adhesion as well as rigidity of neutrophils. *Biochem Biophys Res Commun* 238:910-915.
84. MacLean-Fletcher, S., and T. D. Pollard. 1980. Mechanism of action of cytochalasin B on actin. *Cell* 20:329-341.
85. Cuvelier, D., M. Thery, Y. S. Chu, S. Dufour, J. P. Thiery, M. Bornens, P. Nassoy, and L. Mahadevan. 2007. The universal dynamics of cell spreading. *Curr Biol* 17:694-699.
86. McGrath, J. L. 2007. Cell spreading: the power to simplify. *Curr Biol* 17:R357-358.
87. Lomakina, E. B., G. Marsh, and R. E. Waugh. 2014. Cell Surface Topography Is a Regulator of Molecular Interactions during Chemokine-Induced Neutrophil Spreading. *Biophys J* 107:1302-1312.
88. Sengupta, K., H. Aranda-Espinoza, L. Smith, P. Janmey, and D. Hammer. 2006. Spreading of neutrophils: from activation to migration. *Biophys J* 91:4638-4648.
89. Douezan, S., K. Guevorkian, R. Naouar, S. Dufour, D. Cuvelier, and F. Brochard-Wyart. 2011. Spreading dynamics and wetting transition of cellular aggregates. *Proc Natl Acad Sci U S A* 108:7315-7320.
90. Beaune, G., T. V. Stirbat, N. Khalifat, O. Cochet-Escartin, S. Garcia, V. V. Gurchenkov, M. P. Murrell, S. Dufour, D. Cuvelier, and F. Brochard-Wyart. 2014. How cells flow in the spreading of cellular aggregates. *Proc Natl Acad Sci U S A* 111:8055-8060.

Chapter 3

Ligand Density Elicits a Phenotypic Switch in Human Neutrophils

Preface

The content of this chapter has been adapted from its published version in the journal *Integrative Biology* (2014, Vol. 6:348-356, DOI: 10.1039/C3IB40225H) by permission of The Royal Society of Chemistry. The published manuscript was coauthored by **Steven J. Henry**, John C. Crocker, and Daniel A. Hammer. The content has been reproduced with knowledge of the coauthors. Specific author contributions were as follows: **SJH** designed and executed experiments, analyzed data, and wrote the manuscript; **JCC** consulted on design of analysis routines, data interpretation, and edited the manuscript. **DAH** supported the work, consulted on data interpretation, and edited the manuscript. Supplementary movies referenced in the prose can be retrieved from the published version online.

Abstract

Neutrophils are mediators of innate immunity and motility is critical to their function. We used microcontact printing to investigate the relationship between density of adhesive ligands and the dynamics of neutrophil motility. We show that neutrophils adopt a well-spread morphology without a uropod on moderate densities of adhesion ligand. As density is increased, the morphology switches to a classic amoeboid shape. In

addition to the morphological differences, the dynamics of motility were quantitatively distinct. Well-spread cells without uropods glide slowly with high persistence while amoeboid cells made frequent directional changes, migrating quickly with low persistence. Using an antibody panel against various integrin chains, we show that adhesion and motility on fibronectin were mediated by MAC-1 ($\alpha_M\beta_2$). The phenotypic switch could be generalized to other surface ligands, such as bovine serum albumin, to which the promiscuous MAC-1 also binds. These results suggest that neutrophils are capable of displaying multiple modes of motility as dictated by their adhesive environment.

Introduction

Leukocytes are important mediators of immunity, and motility is critical to their function. Neutrophils in particular act as first responders to pathogen challenges (1) as well as sterile trauma (2) resulting in inflammation. The role of soluble chemoattractants in stimulating and directing neutrophil motility has long been of interest (1) and has been explored in various engineered *in vitro* systems (3-4). Recently, attention has shifted to environment dimensionality in dictating the mode of leukocyte migration (5-7). As the empirical body of leukocyte observations has grown, it is now appreciated that these cells can employ an assortment of migratory mechanisms.

On a majority of two-dimensional *in vitro* substrates, neutrophils exhibit an amoeboid morphology (3-4, 8-12). The distinguishing features of this phenotype are an elongated cell body with a frontward ruffled-lamellipodium, a midregion that contains the nucleus, and rearward knob-like uropod (8). Detailed images of this morphology have been captured with high resolution scanning electron microscopy (SEM) (13-14).

However, there have also been observations of neutrophils assuming a very different, well-spread phenotype without uropods on two dimensional substrates (15-17). In those instances the alternative phenotype was attributed to the underlying stiffness of the material, as neutrophils on softer substrates were shown to re-assume an amoeboid phenotype. Yet, neutrophils also display the amoeboid phenotype on stiff substrates, such as those in the previously cited SEM studies, suggesting substrate stiffness is not a unique controller of cell morphology (18-19). Hypothesizing that another factor was involved in modulating these two phenotypes, our study focused on the role of ligand density.

In this paper, we investigated neutrophil morphology and motility on increasing densities of the extracellular matrix protein fibronectin (FN). We observed that neutrophils exhibited a well-spread, uropod-absent phenotype on sub-saturating, intermediate densities of FN. On high densities of ligand this phenotype was replaced with the amoeboid phenotype. The modes of motility associated with these two morphologies were quantifiably distinct as shown by comparison of their mean squared displacement with time. Finally, we determined that the FN adhesion and motility were mediated by MAC-1 ($\alpha_M\beta_2$). The phenotypic switch could be generalized to other surface ligands, such as bovine serum albumin, to which the promiscuous MAC-1 also binds.

Materials and Methods

Media

Rinsing buffer was Hanks' Balanced Salt Solution (Invitrogen, Carlsbad, CA) without calcium or magnesium supplemented with 10 mM HEPES (Invitrogen) and pH adjusted to 7.4. Storage buffer was rinsing buffer supplemented with 2 mg/mL glucose. Running buffer was storage buffer supplemented with 1.5 mM Ca^{2+} and 2 mM Mg^{2+} .

Fibronectin (FN) was from human plasma (BD Biosciences, Bedford, MA). Low-endotoxin bovine serum albumin (BSA) (Sigma) was prepared at 2 % and 0.2 % w/v in PBS without calcium and magnesium (PBS(-)). Labeling of proteins *via* Alexa Fluor carboxylic acid, succinimidyl ester (Invitrogen) was performed in accordance with the manufacturer's recommended protocol. Stock N-formylmethionyl-leucyl-phenylalanine (fMLF) (Sigma, Saint Louis, MO) was reconstituted in glacial acetic acid before dilution. The nonionic triblock copolymer Pluronic F-127 (Sigma) was prepared at 0.2 % w/v in PBS(-). All solutions were sterile filtered or prepared sterile. Bicinchoninic acid protein assays (Pierce Biotechnology, Rockford, Il) were performed on stock solutions of proteins to measure concentration.

Substrates

Poly(dimethylsiloxane) (PDMS) (Sylgard 184 Silicone Elastomer, Dow Corning, Midland, MI) coated coverslips were prepared from number one thickness glass coverslips (Fisher Scientific, Hampton, NH) of 25 mm diameter spun with degassed PDMS (10:1 base:cure by weight). Spinning at 4000 rpm for 1 min, leveling at RT, and baking at 65 °C overnight resulted in a 12.5 ± 0.4 μm layer of PDMS. Bare glass coverslips were cleaned *via* piranha wash (2:1 by volume $\text{H}_2\text{SO}_4:\text{H}_2\text{O}_2$) and thoroughly rinsed in diH_2O . Coverslips were dried completely in a 90 °C oven. Coverslips, bare and PDMS-coated, were affixed to the bottom of six-well tissue culture plates which had either been hot-punched or laser-cut to generate a 22 mm diameter opening in the bottom of the wells. Coverslip bonding was performed using a continuous bead of Norland Optical Adhesive 68 (Thorlabs, Newton, NJ), cured for 20 min under a long wavelength ultraviolet lamp.

Protein Deposition and Blocking

Stamps for printing were prepared from PDMS, mixed at 10:1 base:cure by weight, degassed, and poured over a silicon wafer. The polymer was cured by baking for 2 hr or longer at 90 °C. Trimmed stamps were sonicated in 200 proof ethanol for 10 min, rinsed twice in diH₂O and dried in a gentle stream of filtered N_{2(g)}. The face of the PDMS stamp previously cast against the silicon wafer was covered with a sessile drop of protein solution. After incubation, stamps were rinsed twice in a submerging quantity (~ 50 mL) of diH₂O and dried in a gentle stream of filtered N_{2(g)}. For motility studies stamps were 1 cm², inked with 200 μL of protein solution for 2 hrs at RT. For all other experiments, stamps were 0.36 cm², inked with 50 μL of protein solution for 1 hr at RT. After stamp inking and drying, mounted PDMS-coated coverslips were treated for 7 min with ultraviolet ozone (UVO Cleaner Model 342, Jelight, Irvine, CA) to render the surface hydrophilic (20). Stamps were placed in conformal contact with the activated substrate for approximately 30 s.

For physisorption experiments, sterile flexiPERM (Sigma) silicone gaskets were affixed to the substrates to hold an aliquot of protein at a concentration and volume that preserved the number of protein molecules per unit area of exposed surface for comparison with the printed conditions.

Blocking printed or adsorbed surfaces by submersion in 0.2 % w/v solutions of Pluronic F-127 or BSA (0.2 % or 2 %) was performed for 1 hr. Native glass is not amenable to Pluronic blocking until silanized by immersion in 5 % dimethyl dichlorosilane (Sigma) in dicholorobenzene (Sigma) (21). After blocking, each well was rinsed four or five times with 2 mL PBS(-) without dewetting the functionalized surface

to prevent Pluronic sloughing. If substrates were not used the day of fabrication, they were stored overnight at 4 °C under PBS(-). Prior to cell plating, storage PBS(-) was exchanged for running buffer, without dewetting, and equilibrated to 37 °C at 5 % CO₂ in a cabinet incubator.

Neutrophil Isolation

Whole blood was obtained from human donors *via* venipuncture and collection in heparin vials. Samples were collected with University of Pennsylvania Institutional Review Board approval from consenting adult volunteers. Volunteers were required to be in good health and abstain from alcohol and all over-the-counter medication for 24-48 hrs prior to donation. Blood samples were allowed to cool to RT (15-30 min) and layered in a 1:1 ratio of whole blood to Polymorphprep (Axis-Shield, Oslo, Norway). Vials were spun for 45 min at 500 x *g* and 21 °C. After separation, the polymorphonuclear band and underlying separation media layer were aspirated into fresh round-bottom tubes. The isolated solution of cells and separation-media was diluted with rinsing buffer and spun for 5 min at 350 x *g* and 21 °C. Red Blood Cells (RBC) were eliminated from the resulting cell pellet *via* hypotonic lysis. After lysis, vials were centrifuged for 5 min at 350 x *g* and 21 °C and the RBC-free pellets resuspended in storage buffer. Neutrophils were stored at 5×10^5 - 1×10^6 cells/mL on a tube rotisserie at 4 °C until time of plating.

Cell Motility Experiments

For a given experimental condition, 7.5×10^4 neutrophils were seeded on a pre-equilibrated substrate under 1.5 mL of running buffer. Visual confirmation was made that cells had a rounded (i.e. not polarized) morphology at time of plating. Substrates and cells were incubated 10 min at 37 °C and 5 % CO₂ to allow settling and gently rinsed

twice with 1 mL of fresh running buffer to remove non-adherent cells. Prior to rinsing, visual observation of the cells confirmed a transition from rounded to a well-spread morphology. Cell density was minimized to prevent cell-cell collisions but sufficiently dense to acquire reasonable sample sizes for statistical testing. Adherent neutrophils at multiple locations on the same substrate were imaged by time-lapse videomicroscopy for 30 min or longer at 30-90 s intervals in a temperature controlled chamber.

Phase-contrast image stacks corresponding to each imaging location of a particular experimental condition were processed *via* a custom MATLAB (The MathWorks, Natick, MA) script that identified cell boundaries, computed geometric centroids, and connected centroids in consecutive frames to form trajectories. Portions of trajectories were only retained for cells prior to cell-cell collisions and for cells that did not undergo apoptosis. To improve statistical power, multiple locations were imaged per condition. Summing across all field of views (FOVs) acquired we observed a total of 2688 neutrophils, 60 % of which ($n = 1606$) were tracked and their trajectories utilized in MSD construction and curve fitting. Within this group of observed and tracked cells 75 % ($n = 1204$) were tracked for the entire duration of the 30 min observation window and used in model-independent analyses. The remaining cells were tracked for only a portion of that observation window as they subsequently underwent cell-cell collisions. Of those cells that were observed but not tracked ($n = 1082$), 88 % were excluded on the basis of cell-cell contact, residing at the edge of the FOV, or exiting the FOV. The remaining 12 % were excluded on the basis of having an anomalous phenotype (e.g. appearing apoptotic). Cell tracking, mean-squared displacement computation, and error analysis

were based upon the multiple particle tracking method reviewed by Crocker and Hoffman (22). Annotated code used in the analysis of this chapter is reported in Appendix A.

Integrin Blocking

The following panel of function-blocking antibodies against various integrin chains was assembled and used at final concentrations of 50 $\mu\text{g}/\text{mL}$: anti- β_1 clone MAb13 (BD Biosciences), anti- β_2 clone L130 (BD Biosciences), anti- α_M clone ICRF44 (eBioscience), and anti- α_5 clone SAM1 (eBioscience). Isotype controls to IgG1 and IgG2a were purchased from eBioscience. 5×10^5 neutrophils in 200 μL running buffer were incubated for 10 min with antibodies at RT with periodic mixing before exposure to the FN substrate. Substrates and cells were incubated 10 min at 37 °C and 5 % CO_2 and immediately fixed in a solution of 4 % formaldehyde (Fisher) or 10 % neutral buffered formalin (Sigma) for 30 min at RT with periodic mixing. After fixation substrates were rinsed thoroughly with PBS to remove nonadherent cells.

Results and Discussion

Observation of Neutrophil Phenotypes on Two Different Substrates

A common method of preparing two-dimensional surfaces for motility studies is to adsorb an adhesive ligand onto glass or polystyrene and subsequently wash with a solution of bovine serum albumin (BSA). The BSA wash is intended to mask bare regions of the substrate, unoccupied by protein, and impede non-specific cell-substrate interactions. When we plated human neutrophils on such a surface (fibronectin (FN)-adsorbed and BSA-blocked), the cells assumed an amoeboid phenotype having elongated cell bodies, trailing uropods, and narrow lamellipodia (Fig. 3.1 A). The associated motility was undular, with cells undergoing frequent directional changes (Movie S1). The

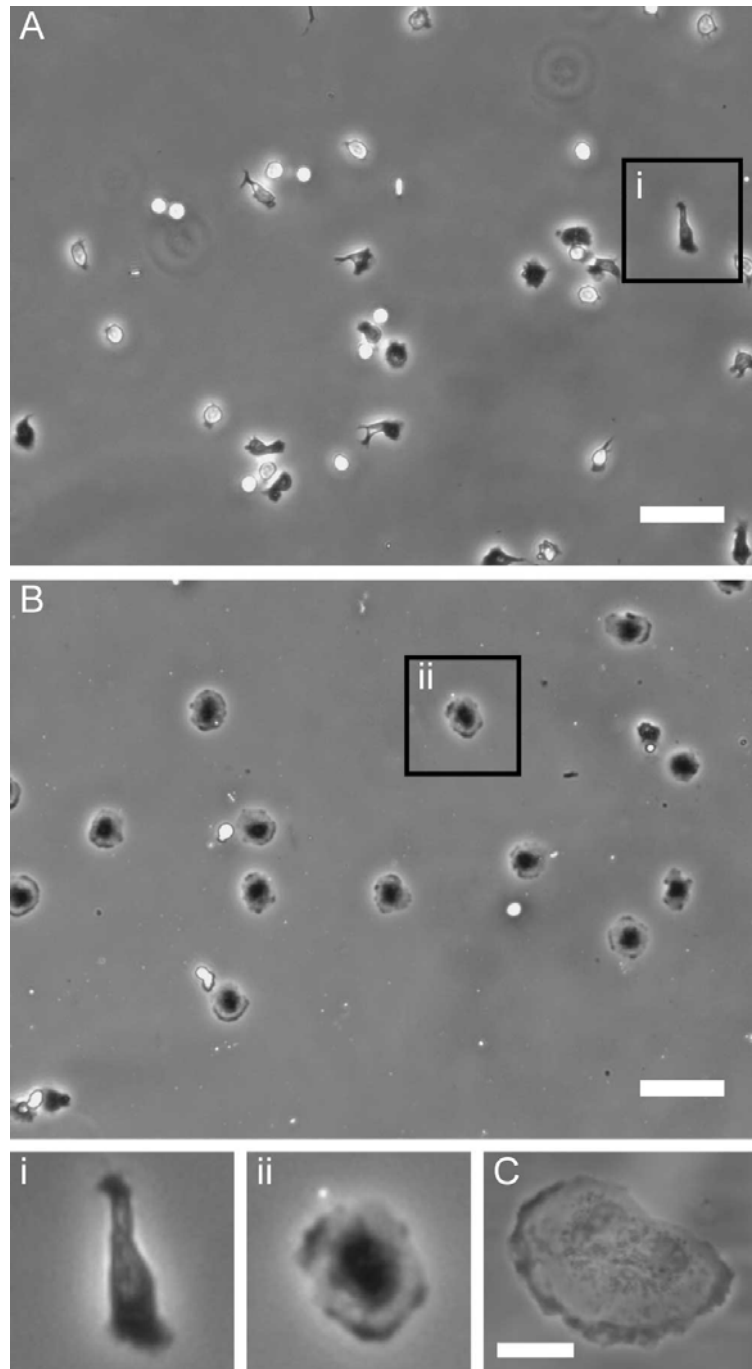


Figure 3.1 Two neutrophil morphologies. (A) FN-adsorbed glass, blocked with BSA. Scalebar is 50 μm . Region (i) is enlarged 3X. (B) FN-printed PDMS, blocked with Pluronic. Scalebar is 50 μm . Region (ii) is enlarged 3X. (C) Same preparation as (B) but higher magnification image. Scalebar is 10 μm .

amoeboid phenotype has been reported elsewhere (3-4, 8-14) for neutrophils on various two-dimensional surfaces. By contrast, when we printed a poly(dimethylsiloxane) (PDMS) surface with FN and blocked with Pluronic, we elicited a very different phenotype. In the latter case the neutrophils were highly spread and no trailing uropods were discernible (Fig. 3.1 *B*). With this phenotype, the cells appeared to glide and were highly persistent in their direction (Movie S2). Our impression is that this latter phenotype was more qualitatively reminiscent of fish-keratocytes (18) than amoeboid cells.

Complementary controls of neutrophils on FN-printed glass and FN-adsorbed PDMS demonstrated that the phenotypic differences depended on the blocking agent, not the method of protein deposition (Fig. 3.2 *A-D*). Quantitative fluorescence measurements of fluorophore-labeled FN confirmed that total FN loading of glass and PDMS surfaces were comparable (Fig. 3.2 *E*). When we silanized glass and then blocked surfaces with Pluronic, we found the well-spread, uropod-absent phenotype could be elicited on FN functionalized glass (Fig. 3.3 *A*). Our interpretation is that when blocking with Pluronic, cell binding was solely due to the underlying FN, and the blocking agent did not contribute to the adhesion (Fig. 3.4 *C-E*).

We hypothesized that the amoeboid phenotype is a result of adhesion to high densities of surface ligand, and that blocking with BSA served to increase the total ligand content. To test this hypothesis, we used microcontact printing to systematically control the density and type of surface ligand (Fig. 3.5 *A*). Microcontact printing is a tool to spatially pattern cellular adhesive ligands (20, 23) and to print the tips of polymeric posts for force measurements (24). While microcontact printing has been extensively used to

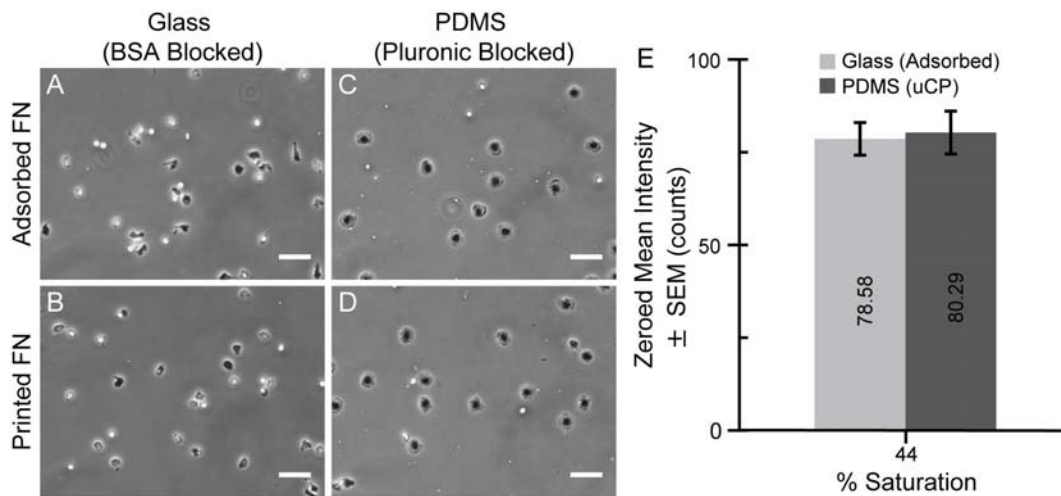


Figure 3.2 Phenotype does not follow method of protein deposition. To determine if method of protein deposition dictated the two cell phenotypes we compared the following surface preparation strategies: (A) FN-adsorbed glass, BSA blocked (reproduced from Fig. 3.1 A), (B) FN-printed glass, BSA blocked, (C) FN-adsorbed PDMS, Pluronic blocked, and (D) FN-printed PDMS, Pluronic blocked (reproduced from Fig. 3.1 B). Scalebars = 50 μ m. Phenotype followed the method of blocking not the method of FN deposition. (E) Mean intensity of FN594 (FN conjugated to Alexa Fluor 594 dye) adsorbed onto glass and printed onto PDMS. Images were acquired under identical settings and the mean pixel intensity computed. For each preparation, the mean pixel intensity of the corresponding negative control was subtracted to produce the “zeroed mean intensity”. Error bars are \pm standard error of the mean (n = 2 independent experiments). Amount of deposited FN on both surfaces is comparable. This figure was presented in the supplementary text of the original manuscript.

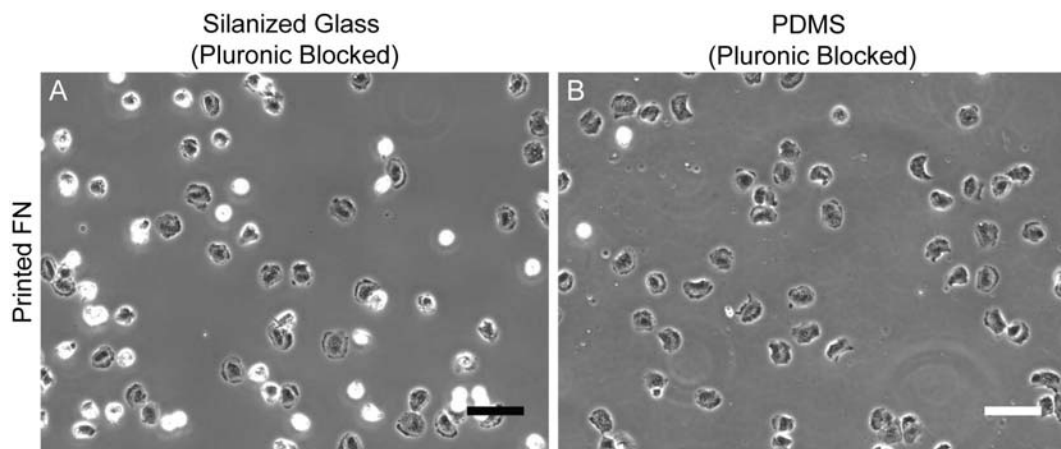


Figure 3.3 Keratocyte-like phenotype recapitulated on Pluornic-blocked glass. To determine if substrate type (i.e. glass vs. PDMS) dictated the two cell phenotypes we performed the following controls: (A) FN-printed silanized glass, Pluronic blocked (B) FN-printed PDMS, Pluronic blocked. Surfaces functionalized at 40 % FN surface saturation. Scalebars = 50 μ m. This figure was presented in the supplementary text of the original manuscript.

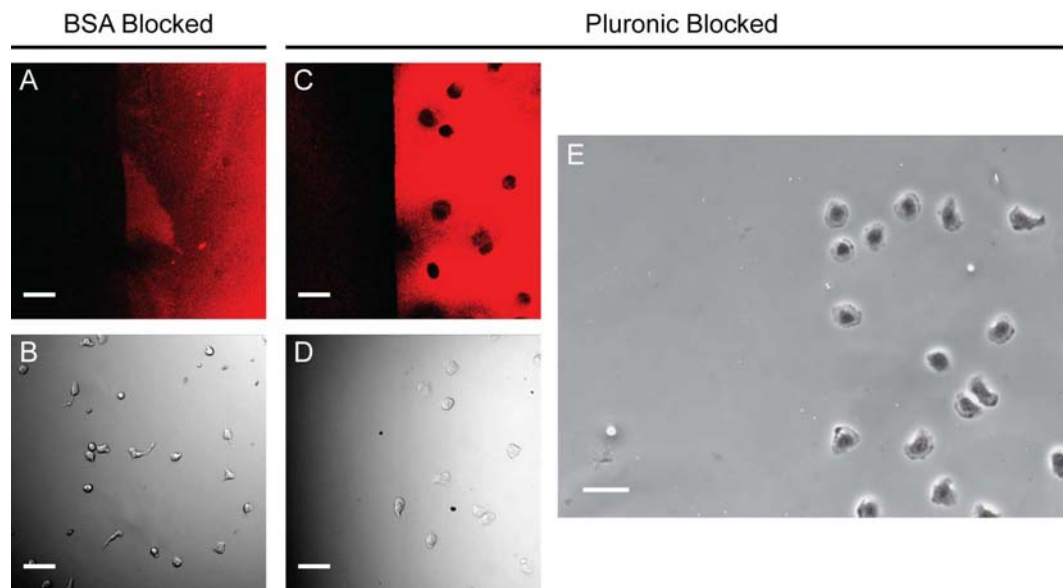


Figure 3.4 Exquisite cell-ligand specificity on Pluronic-blocked substrates. Pluronic F-127 blocking of PDMS substrates allows complete inhibition of non-specific binding in human neutrophils. (A) FN conjugated to Alexa Fluor 647 (FN647) after adsorption to a piranha cleaned coverslip, blocked with 0.2 % BSA in PBS (w/v). The distinct edge shown was achieved by affixing a single-well flexiPERM gasket to the coverslip which was removed prior to blocking and cell plating. (B) DIC image of fixed human neutrophils in same location as (A). Observe that cell adhesion is seen in regions of the substrate not functionalized with FN. (C) FN647 after printing on a PDMS spin-coated coverslip, blocked with 0.2 % Pluronic F-127 in PBS (w/v). (D) DIC image of fixed human neutrophils on microcontact printed substrate in same location as (C). No adhesion outside of the functionalized area is observed. (E) Phase contrast image of fixed cells at a different edge location on same substrate (C-D). All scale bars are 40 μm . Note: non-uniform image acquisition parameters preclude comparison of fluorescent signal intensities between the glass and PDMS conditions (A, C). Surfaces functionalized at 40 % FN surface saturation. This figure was presented in the supplementary text of the original manuscript.

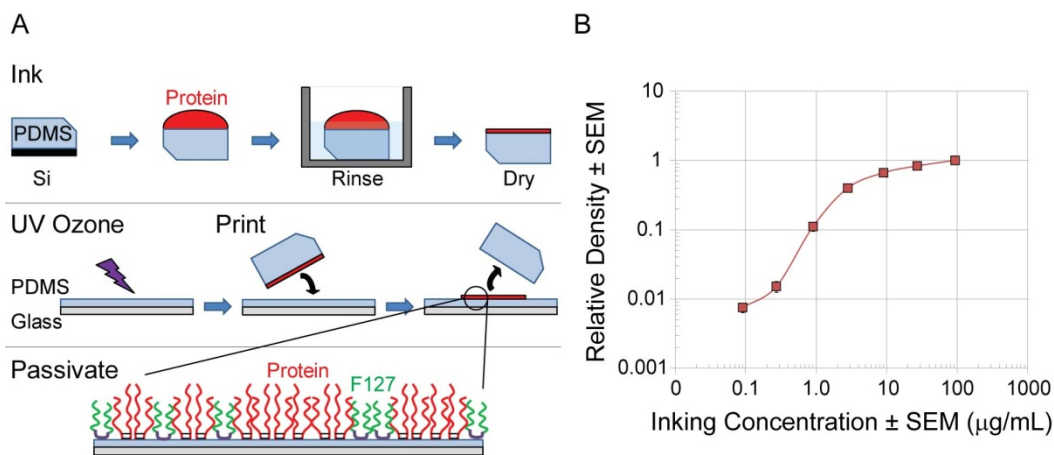


Figure 3.5 Microcontact printing overview and sub-saturating density quantification. (A) PDMS is cast against a silicon wafer to generate a smooth inking face. Stamps are trimmed and a sessile drop of protein solution at known concentration is used to coat the smooth stamping face. Stamps are rinsed and dried gently in a stream of nitrogen. Separately PDMS-spun coverslips are rendered hydrophilic by exposure to UV ozone for 7 min. When the inked stamps are brought into contact with the spun coverslip there is preferential transfer of the protein from the natively hydrophobic stamp to the hydrophilic coverslip. Finally the substrate is passivated by submersion in a nonionic triblock copolymer sold under the tradename Pluronic F-127. Bare regions of the PDMS not occupied by adhesive ligand are rendered stealth to neutrophils by Pluronic coating. (B) Quantitative fluorescence microscopy to determine the relative density of protein on printed substrates by titrating inking concentration. The saturating condition was considered to be 100 $\mu\text{g/mL}$. Error bars are standard error of the mean ($n = 7-9$ independent experiments). This figure was presented in the supplementary text of the original manuscript.

study the behavior of mesenchymal (20, 23-27) cells, it has only recently been applied in studies of hematopoietic-derived cells (28-31). In our study, microcontact printing was used to immobilize different densities of FN on PDMS. By titrating the inking concentration of the protein solution used to prepare the stamps, we could reproducibly achieve sub-saturating densities of deposited FN (Fig. 3.5 B). After fabricating a series of PDMS surfaces with systematically varied densities of FN, all blocked with Pluronic F-127, we scored the resulting neutrophil phenotypes observed (Fig. 3.6).

On surfaces printed with little or no FN and blocked with Pluronic, cells failed to polarize or spread and remained spherical, presenting as bright white circles under phase contrast imaging (Fig. 3.6 *i*). On intermediate densities of printed-FN, blocked with Pluronic, the well-spread, uropod-absent phenotype was observed (Fig. 3.6 *ii*). Frequency of the keratocyte-like phenotype peaked at 40 % surface saturation. As density of FN increased, the well-spread phenotype was observed less frequently. Once surface density reached 83 % saturation the amoeboid phenotype was predominate (Fig. 3.6 *iv*).

Others have observed this well-spread, uropod-absent phenotype in neutrophils on FN-conjugated polyacrylamide gels (15-17). In those instances the morphology was attributed to the underlying stiffness of the material, as neutrophils on softer gels were more amoeboid. Indeed, our relatively thick PDMS layers (~ 12 μm) and the use of a 10:1 formulation (base:cure, w/w) means the substrates were quite stiff, with Young's moduli on the order of megapascals (32-33). However, here we demonstrated that the well-spread phenotype on stiff surfaces is only inducible for sub-saturating densities of ligand. This observation contributes to the growing empirical body of evidence showing neutrophils and other leukocytes can adopt a variety of motile mechanisms to achieve

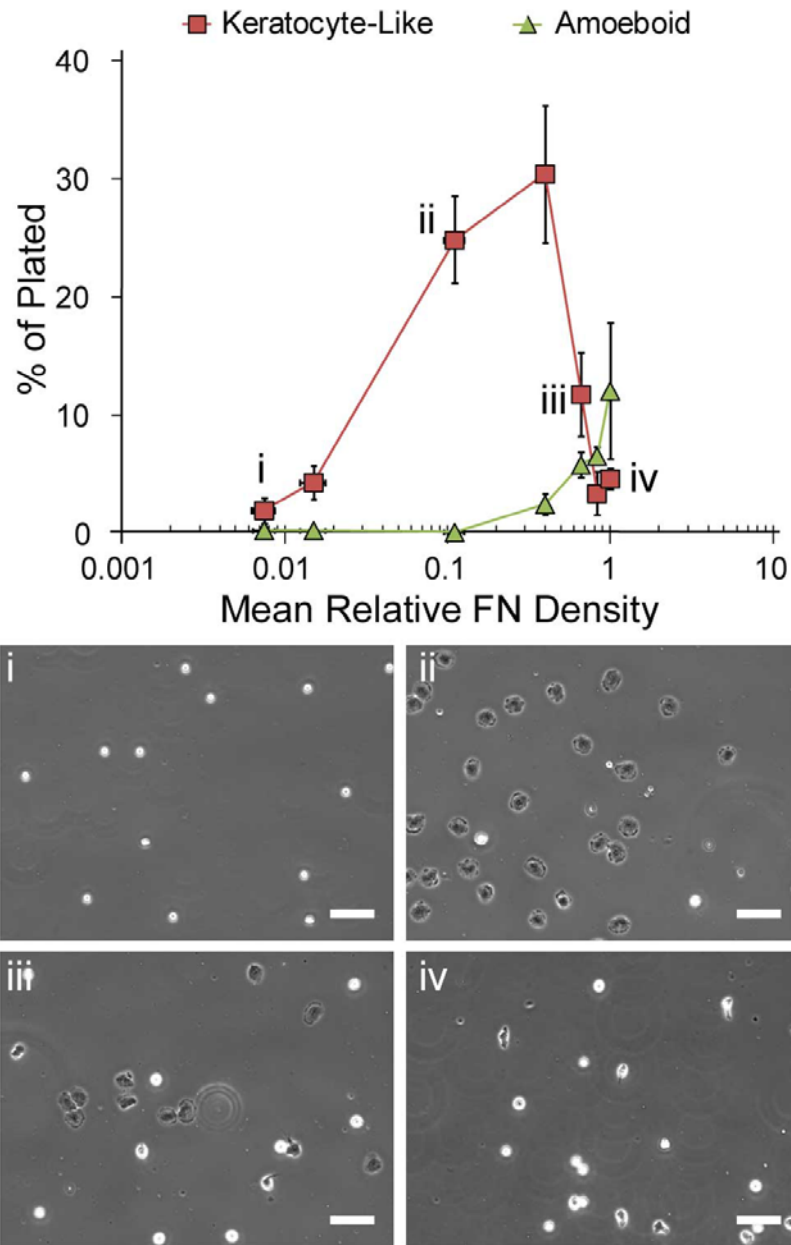


Figure 3.6 Neutrophil phenotype on increasing densities of fibronectin (FN). Adherent neutrophils as percentage of total plated cells per sub-saturating densities of FN. Representative images from a single experiment on (i) 0.7 % (ii) 11 % (iii) 66 % and (iv) 100 % saturated FN substrates. Scalebars are 50 μm . Error bars are \pm standard error of the mean. Substrate density was measured *via* quantitative fluorescence microscopy (Fig. 3.5 B). All substrates were FN-printed and Pluronic-blocked PDMS.

translocation and helps reconcile the occurrence of both phenotypes elsewhere in the literature of neutrophil motility on stiff substrates.

Ziebert and Aranson have constructed a biophysical model of cell motility that demonstrates phenotypic transitions in the mode of migration as a function of underlying substrate stiffness and surface adhesivity (19). On stiff substrates their model predicts a transition from stick-slip to gliding motion as surface ligand density is increased. While we have not observed stick-slip motion at low adhesivity we have found an intermediate ligand density window in which neutrophils display a highly persistent gliding phenotype. It will be interesting to see if the incorporation of intracellular viscoelasticity into their future models can recapitulate our transition from gliding motion to amoeboid motion at saturating conditions of adhesive ligand. A transition from gliding to more erratic motion has also been reported of fish keratocytes on stiff substrates as surface adhesivity increases (18).

Our study of how neutrophil phenotype depends on adhesion draws an interesting qualitative comparison with recent work on the capacity of physical confinement to dictate migratory cell phenotype. Migratory cells in physically confined channels or on narrow one-dimensional tracks of ligand have been shown to lose characteristics of conventional two-dimensional migration (7). Hung and co-workers have also found that the mechanism of propulsion differs as a function of substrate dimensionality (34). In the future, immunocytochemical staining and small molecule inhibitor studies of our amoeboid versus keratocyte-like morphologies may reveal similar discrepancies driven by ligand density.

Quantifying Motility of Amoeboid and Keratocyte-Like Phenotypes

The dynamics of amoeboid and keratocyte-like motility were distinct, as revealed by comparing their mean squared displacements (MSD) as a function of time (Fig. 3.7). On log-log axes, the slopes of MSD vs. time for the two populations were different. Neutrophils undergoing amoeboid migration accumulated squared displacement diffusively (slope ~ 1) while neutrophils undergoing keratocyte-like migration accumulated squared displacement superdiffusively (slope > 1). Fitting the curves for MSD vs. time with the persistent random walk model of cell kinesis (35-36) ($\langle \text{MSD}(\tau) \rangle = 2S^2P[\tau - P(1 - \exp(-\tau/P))]$) allowed us to quantify neutrophil motility in terms of the best-fit parameters speed (S) and persistence (P). Doing so confirmed our qualitative assessment that amoeboid motility was faster and less persistent ($S_{\text{amoeboid}} = 6 \mu\text{m}/\text{min}$, $P_{\text{amoeboid}} = 0.5 \text{ min}$) than keratocyte-like motility ($S_{\text{keratocyte-like}} = 3 \mu\text{m}/\text{min}$, $P_{\text{keratocyte-like}} = 15 \text{ min}$). Comparing the cytoskeletal architecture of these two dramatically different phenotypes remains to be done. It will be interesting to learn how stress fibers are organized in the keratocyte-like cell, compared to the amoeboid cell.

To this point neutrophils were induced to adhere and be motile on FN substrates without prior or concurrent stimulation by soluble chemoattractant. Therefore, the resulting motility was haptokinetic, driven by FN stimulation at the cell-substrate interface. A control study quantifying selectin-expression (37) *via* flow cytometry confirmed neutrophils were not primed for integrin-based adhesion to FN surfaces by virtue of isolation or storage stresses (Fig. 3.8).

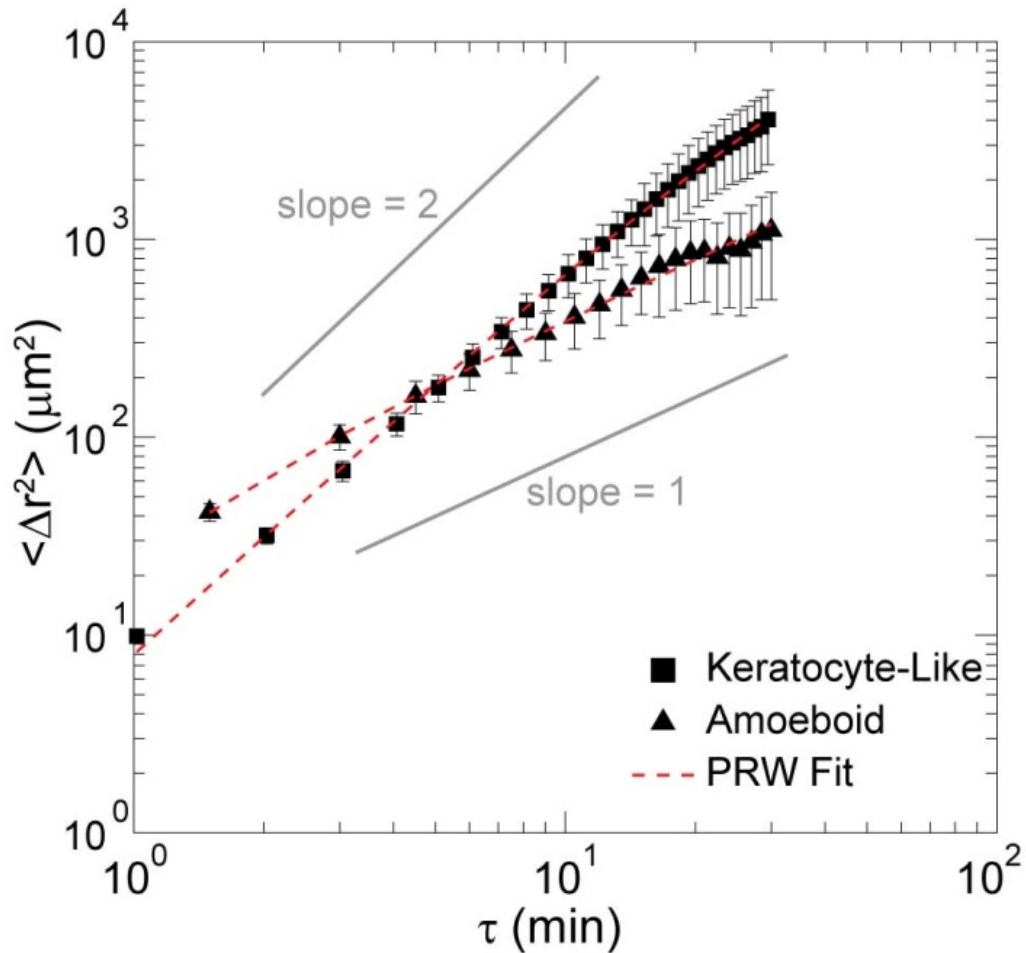


Figure 3.7 Mean squared displacements (MSDs) of two motility modes. Time and ensemble averaged MSDs of neutrophils undergoing amoeboid motility or keratocyte-like motility. Amoeboid cells acquire displacement diffusively, slope ~ 1 . Keratocyte-like cells acquire displacement superdiffusively, slope > 1 . Dotted line is fit of empirical data with persistent random walk (PRW) model of cell motility. Error bars are \pm standard error of the mean.

Flow Cytometry to Assess Activation State

The prose in this section was presented in the supplementary text of the original manuscript. Because neutrophils were robustly haptokinetic on FN alone without the addition of chemoattractant, we verified that cells were not primed for binding to the adhesion ligand as a result of stresses experienced prior to FN exposure. We used L-selectin as the marker of cell activation state. Kishimoto and coworkers demonstrated that L-selectin is a sensitive marker of a neutrophil's transition from quiescence to a phenotype primed for integrin-mediated binding (37), a transition denoted by rapid L-selectin shedding.

Staining of all treatment conditions was for 45 min on ice in the dark immediately followed by fixation in 2 % formaldehyde for 20 min. After fixation, vials were spun to pellet cells (350 x g, 5 min, 4 °C) and resuspended in HBSS without calcium or magnesium. This rinsing sequence was repeated three times. After the final resuspension, cells were stored overnight on ice in the dark until flow cytometry measurements the following day. Antibodies were mouse-anti-human CD62L-PE-Cy5 (eBioscience) and mouse IgG1 κ -PE-Cy5 isotype control (eBioscience).

Immediately after isolation, neutrophils were stained for L-selectin (Fig. 3.8 A). Positive (i.e. activated) controls were generated by exposing isolated neutrophils to the chemoattractants TNF α and fMLF immediately following isolation (Fig. 3.8 B-C). A decrease in L-selectin expression by cells exposed to chemoattractant, relative to the post-isolation control, demonstrated the isolated neutrophils had the capacity to be activated. To mimic the conditions cells would experience prior to plating on a FN-printed PDMS substrate, a separate aliquot of cells was subjected to storage, buffer

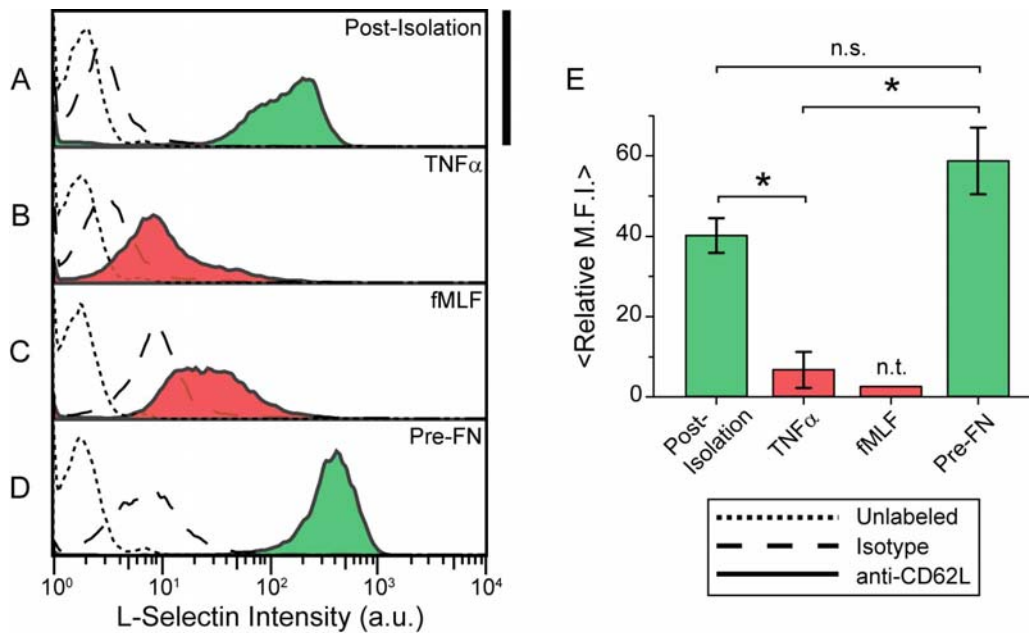


Figure 3.8 Quantification of L-selectin expression levels *via* flow cytometry. Expression levels were assayed under the following conditions: (A) immediately after isolation from whole blood, (B) immediately after isolation including 100 U/mL TNF α or (C) 100 nM fMLF as positive activation controls, and (D) prior to FN exposure mimicking the storage, buffer exchange, and re-warming steps experienced by plated cells. Scalebar is 400 counts. Mean relative median fluorescence intensity (Relative M.F.I) was computed for each experimental condition (E). Errorbars are standard error of the mean (n = 2 donors). Asterisk denotes significant difference and n.s. denotes a difference not statistically significant as computed by post-hoc SNK Multiple Comparisons Method (p < 0.05). fMLF was excluded from significance testing (n.t.). This figure was presented in the supplementary text of the original manuscript.

exchange, and re-warming consistent with the plating protocol used in our motility studies. Flow cytometry on these pre-FN mimics (Fig. 3.8 D) showed a slight increase in L-selectin expression relative to the post-isolation control.

To quantify the extent of these shifts, the relative median fluorescence intensity (Relative M.F.I. = $(M.F.I._{Sample} - M.F.I._{Isotype})/M.F.I._{Isotype}$) of each condition was computed (Fig. 3.8 E). A statistically significant decrease in L-selectin expression as a function of TNF α was observed relative to the post-isolation control and pre-FN mimic. No statistically significant difference was found between the post-isolation control and pre-FN condition. Thus while the isolated neutrophils were capable of activation, they were not primed for integrin-mediated binding by virtue of isolation or storage stresses prior to FN exposure. This finding, coupled with high cell-FN specificity on Pluronic blocked PDMS substrates, leads us to attribute the post-plating adhesion and haptokinesis solely to the deposited FN.

Effect of Chemoattractant on Keratocyte-Like Motility

We explored the capacity of the potent neutrophil chemoattractant formyl-Met-Leu-Phe (fMLF) (38) to modulate the motility of neutrophils undergoing keratocyte-like migration. On 44 % saturated FN surfaces, the addition of 10 nM fMLF to haptokinetic neutrophils had the effect of increasing the total dispersion of the cell system (Fig. 3.9 A-B). To quantify the extent of motility in a model-independent fashion we extracted the maximum displacements for cells tracked over 30 min. Cell trajectories shorter than 30 min were excluded in this analysis to avoid inadvertently biasing the data. The mean of the maximum displacements ($\langle \max(|\Delta r|) \rangle$) was computed for each combination of FN adhesiveness and fMLF concentration (Fig. 3.9 C). Introducing fMLF, after onset of FN-

induced haptokinesis, potentiated motility in a dosage-dependent manner at an intermediate ligand density of 44 % saturation. However, at a higher surface saturation of 73 %, fMLF was no longer capable of increasing the basal motility induced by FN stimulation. The number of independent observations for each condition and a comprehensive description of mean maximum displacement data are reported in Fig. 3.10.

Computation of the MSD provides dynamic information on the dispersion of cells and allows the incorporation of cell trajectories shorter than the total experimental acquisition time. Time and ensemble-averaged MSDs for each independent observation were computed from all available cell trajectories through 30 min. The MSDs corresponding to Fig. 3.9 *A-B* data are reported in Fig. 3.9 *D*. In general, on log-log axes, the slope of the MSD curves are relatively constant and greater than unity. This denotes superdiffusive motility in which cells accumulate squared displacement faster than expected by pure diffusion. Considering the best-fit parameters speed and persistence, systematic variation in the dose of fMLF alters cell speed at intermediate density FN (Fig. 3.9 *E*), but not the persistence time for any of the FN-fMLF conditions tested (Fig. 3.9 *F*). All MSDs-vs.-time contributing to construction of Fig. 3.9 *D* are compiled in Fig. 3.11 along with complete results of multiple comparisons testing on mean speed data.

In both analyses the capacity of chemoattractant to augment haptokinetic motility in the keratocyte-like phenotype was found to be a function of the underlying adhesiveness. This emphasizes the importance of considering the role of substrate adhesiveness in controlling the cell response to the milieu of soluble chemoattractants and cytokines known to orchestrate directional motility during inflammation.

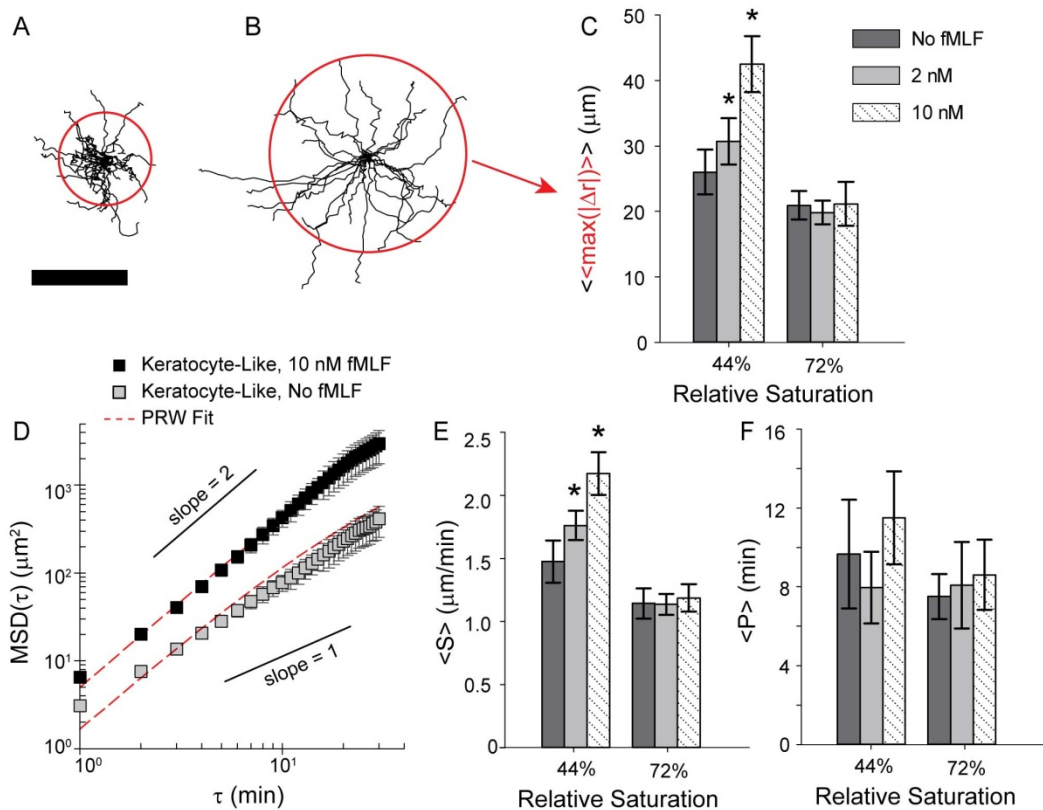


Figure 3.9 Quantification of neutrophil haptokinesis and chemokinesis of keratocyte-like phenotype. Human neutrophil trajectories through 30 min of motility on 44 % saturated FN surface in (A) the absence of fMLF and (B) the presence of 10 nM fMLF. Scalebar is 50 μm . Solid red circle is the mean maximum displacement ($\langle \max(|\Delta r|) \rangle$) of 30 min neutrophil trajectories for (A) $\langle \max(|\Delta r|) \rangle \sim 24 \mu\text{m}$ and (B) $\langle \max(|\Delta r|) \rangle \sim 51 \mu\text{m}$. (C) Mean of the set of mean maximum displacements for all independent observations of a particular FN density and fMLF combination tested ($\langle \langle \max(|\Delta r|) \rangle \rangle$). (D) MSD(τ) corresponding to a single donor's neutrophils migrating on 44 % saturated FN surface in the presence or absence of fMLF. Dotted red line is fit of persistent random walk model (PRW) to empirical data. Model fit parameters (E) speed and (F) persistence. Error bars are \pm standard error of the mean. Asterisk denotes significant difference relative to No fMLF condition as computed by post-hoc SNK Multiple Comparisons Method ($p < 0.05$).

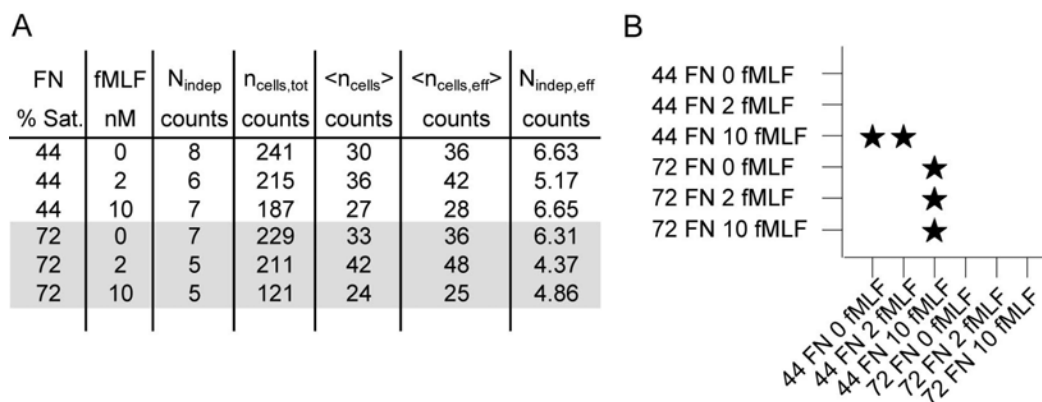


Figure 3.10 Sample sizes per condition and complete results of model-independent significance testing. (A) Table summarizing sample sizes for each experimental condition (FN/fMLF combination). N_{indep} (column 3) is the number of independent observations where an independent observation is a unique donor/donation combination. $N_{\text{cells,tot}}$ (column 4) is the number of total cell trajectories acquired across all independent observations. $\langle n_{\text{cells}} \rangle$ (column 5) is the average number of cells contributed by each independent observation without weighting. Because each independent observation of a condition contributed a different number of cells, weighting is required. Weighting mean values by the number of cells used in the computation of the mean results in an effective number of independent observations on the mean given by $N_{\text{indep,eff}}$ (column 7) and a corresponding effective average number of cells per independent observation $\langle n_{\text{cells,eff}} \rangle$ (column 6). These later two values can be thought of as a hypothetical number of independent observations ($N_{\text{indep,eff}}$) of equal statistical power, each experiment contributing the same number of cells ($\langle n_{\text{cells,eff}} \rangle$). (B) Complete results of significance testing corresponding to the mean maximum displacement metric of Fig. 3.9 C. A star denotes a significance difference as computed by post-hoc SNK Multiple Comparisons Method ($p < 0.05$). This figure was presented in the supplementary text of the original manuscript.

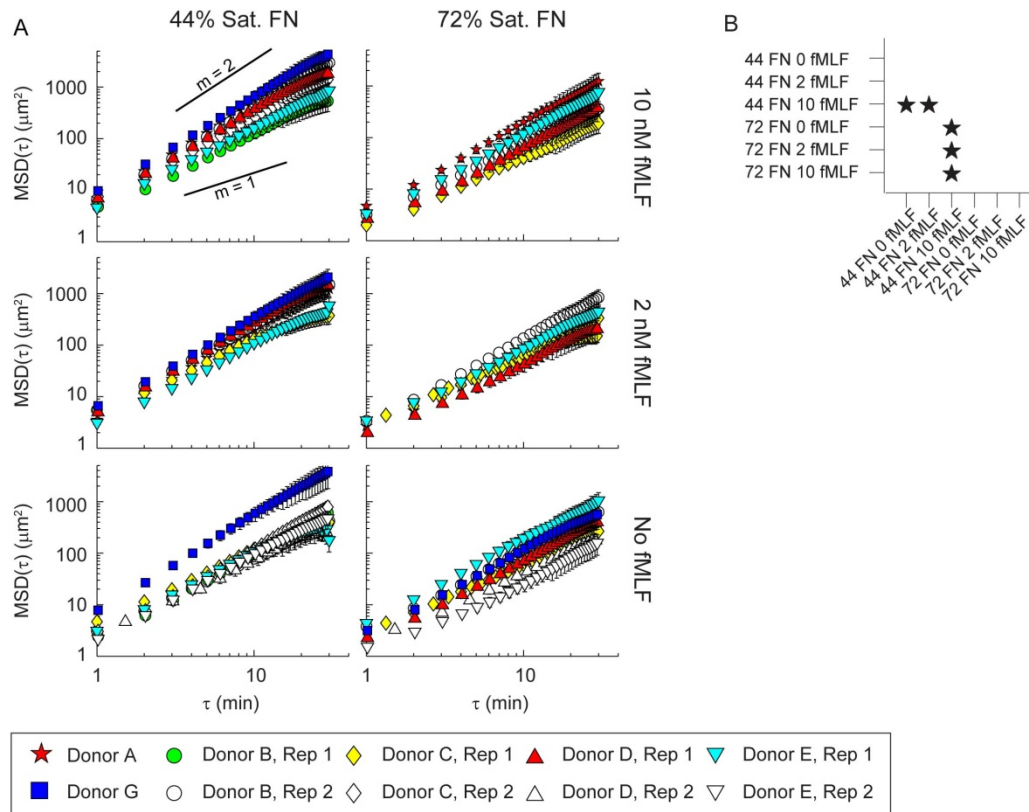


Figure 3.11 MSDs of all independent observations of FN/fMLF experimental conditions tested. (A) For a given elapsed time interval (τ), $\text{MSD}(\tau)$ is the variance of the population of displacements within and across all cells (i.e. time and ensemble averaged). τ_{\min} is the experimental frame rate and τ_{\max} is 30 min. This study utilized six donors (closed symbols), four of which donated on a separate experimental day (open symbols). Variability within a given donor on different experimental days for the same experimental condition led us to treat each donor/donation as an independent observation. Plots are organized by adhesiveness (columns) and concentration of fMLF (rows). All plots are scaled identically. Error bars are \pm standard error of the variance (i.e. of the $\text{MSD}(\tau)$). Eye guides of slope (“ m ”) 1 and 2 are provided for reference. (B) Complete results of significance testing corresponding to the speed parameter “ S ” from the persistent random walk fit to the empirical MSDs. A star denotes a significance difference as computed by post-hoc SNK Multiple Comparisons Method ($p < 0.05$). No statistically significant differences were found among persistence values of Fig. 3.9 E. This figure was presented in the supplementary text of the original manuscript.

Identifying Integrin Chains Responsible for Adhesion

To identify the integrin chains responsible for neutrophil binding to FN, function-blocking antibodies with previously demonstrated efficacy in leukocytes were employed (39-40). Functional blocking of β_2 integrins (Fig. 3.12 D) resulted in a substantial decrease in cell adhesion on FN relative to the positive control without antibody present (Fig. 3.12 A). Targeting the α_M integrin, which coordinates with β_2 integrin to form the MAC-1 heterodimer, was also found to disrupt cell binding on FN significantly (Fig. 3.12 F). In neither case did blocking β_1 (Fig. 3.12 C) nor α_5 (Fig. 3.12 E) integrin chains disrupt binding. These results led us to attribute the observed FN-induced adhesion and subsequent haptokinesis to the β_2 and α_M integrin subunits, or the MAC-1 receptor.

In neutrophils there is known cross talk between β_1 and β_2 integrins when ligating extracellular matrix proteins such as FN (39, 41). Our finding that neutrophils utilize MAC-1 ($\alpha_M\beta_2$) on FN is consistent with other empirical observations. In particular van den Berg and coworkers demonstrated that stimulation of β_1 integrins yields β_2 -mediated adhesion in neutrophils on FN that can be mitigated by function-blocking antibodies against MAC-1 (39). Our blocking study is a probe on the long time-limit (i.e. minute length scale) adhesion of neutrophils to FN. Lishko and co-workers demonstrated that a balance of MAC-1 and VLA-5 ($\alpha_5\beta_1$) is required for neutrophil translocation on FN attributing MAC-1 to adhesion and VLA-5 to migration (41). Our work reveals that the adhesive contribution of MAC-1 is the dominant ligated integrin and may explain the reduced speed of the keratocyte-like phenotype.

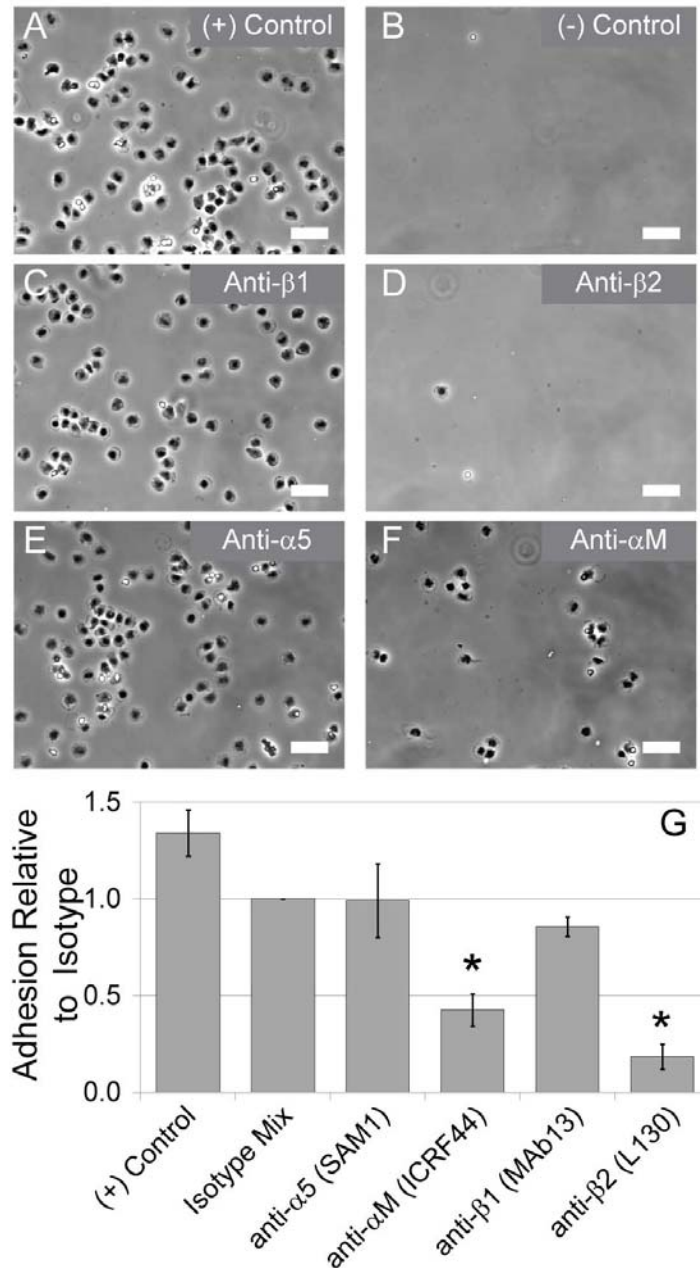


Figure 3.12 Integrin blocking on FN. Integrin blocking of neutrophils pre-incubated with antibodies against various integrin chains before exposure to 44 % saturated FN surface. (A) Positive binding control, no antibodies. (B) Negative binding control, no FN, just Pluronic blocking. (C) anti- β_1 clone MAb13, (D) anti- β_2 clone L130, (E) anti- α_5 clone SAM1, and (F) anti- α_M clone ICRF44. Scalebars are 50 μ m. (G) Mean ratio of adherent cells to isotype control. Error bars are \pm standard error of the mean (n = 3-4). Asterisk denotes significant difference relative to isotype control as computed by post-hoc Dunnet's Method (p < 0.05).

MAC-1 also binds to members of the Ig superfamily (42-43), such as ICAM-1, which illustrates the promiscuity of this integrin. We hypothesized that the emergence of the amoeboid phenotype on BSA-blocked surfaces of intermediate density FN was due to simultaneous binding of MAC-1 to BSA and FN. Indeed, we were able to recapitulate the keratocyte-like phenotype on intermediate densities of BSA alone (Fig. 3.13 A). The percentage of plated neutrophils exhibiting keratocyte-like phenotype on fields of BSA at sub-saturating density was 63 % (n = 3, SE = 22 %). Furthermore, at saturating densities of BSA alone, neutrophils again switched to the amoeboid phenotype (Fig. 3.13 B). The percentage of plated neutrophils exhibiting amoeboid phenotype on fields of BSA at saturating density was 73 % (n = 1, SD = 4 %). When we repeated the function-blocking antibody study on neutrophils exposed to intermediate-density BSA substrates, we again found that MAC-1 was mediating adhesion (Fig. 3.13 C).

The finding that neutrophils were employing the promiscuous integrin MAC-1 to mediate adhesion to our experimental surfaces reinforces the necessity of choosing an appropriate blocking reagent against non-specific cell adhesion. BSA, which is often used to block surfaces, actually functions as an adhesive ligand. Coating surfaces with Pluronic is the only method we have found to reliably eliminate all non-specific background adhesion in our *in vitro* motility assays. This type of exquisite discrimination of the roles of different ligands is only possible with improved surface techniques, such as microcontact printing (20).

Aside from the obvious conclusion that care must be taken to block non-specific binding with appropriately neutral ligands, future work will address how the organization and density of adhesion ligands leads to the morphology of cell response. Now, we can

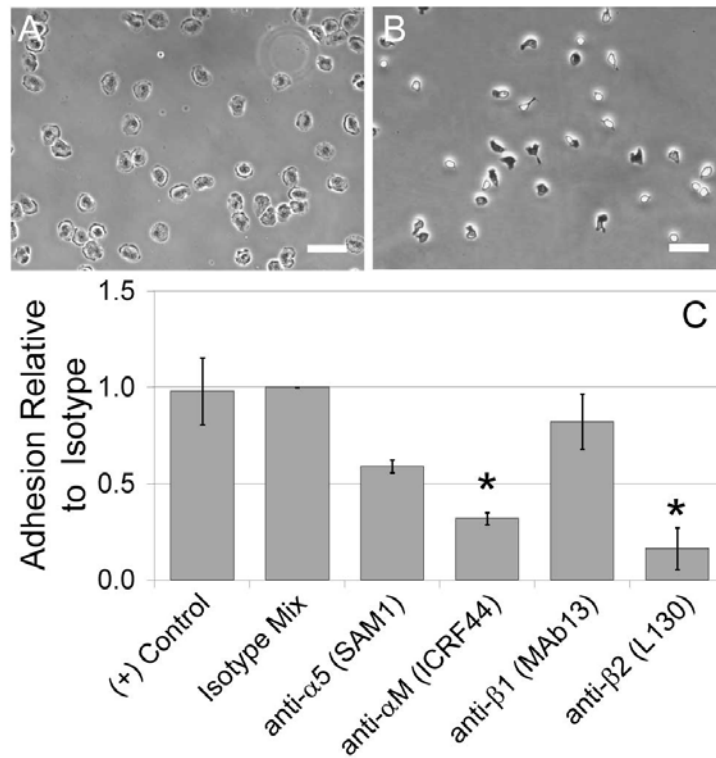


Figure 3.13 Neutrophil adhesion to BSA. (A) Keratocyte-like phenotype of neutrophils on intermediate density of BSA. (B) Amoeboid phenotype returns on saturating density of BSA. (C) Recapitulation of antibody blocking study of neutrophils on intermediate density BSA surfaces. Mean ratio of adherent cells to isotype control. Error bars are \pm standard error of the mean ($n = 2-3$). Asterisk denotes significant difference relative to isotype control as computed by post-hoc Dunnet's Method ($p < 0.05$).

speculate that a high density of adhesion ligands over a large spatial domain promotes uropod formation. If this is the case, distribution of ligands into patches would prevent uropod formation, even if the density in the patches were locally high.

Conclusions

Our work has demonstrated that neutrophils are capable of a phenotypic switch in morphology and associated motility as dictated by adhesion ligand density. The nature of the density sensing remains to be addressed in determining whether neutrophils are sensitive to these changes at the receptor length scale or across their total cell-substrate contact area. We anticipate microcontact printing will be a useful platform in addressing this question. By quantifying the motility associated with the amoeboid and keratocyte-like phenotypes we found the modes of migration to be distinct. The biophysical mechanism that underpins these differences is unclear. We suspect visualizing cytoskeletal architectures will improve our mechanistic insight. Lastly, our finding that the integrin heterodimer MAC-1 was being employed to mediate adhesion to our experimental surfaces reinforces the importance of avoiding BSA as an agent to block non-specific binding in neutrophils.

Acknowledgements

We are grateful to Eric Johnston for technical assistance in the laboratory and Professor Christopher S. Chen, Michael T. Yang, PhD and Ravi A. Desai, PhD for their time and expertise in teaching us microcontact printing. Funding for this work was provided by a National Science Foundation Graduate Research Fellowship to SJH and a grant from the National Institutes of Health (HL18208) to DAH.

References

1. Nathan, C. 2006. Neutrophils and immunity: challenges and opportunities. *Nat Rev Immunol* 6:173-182.
2. McDonald, B., K. Pittman, G. B. Menezes, S. A. Hirota, I. Slaba, C. C. M. Waterhouse, P. L. Beck, D. A. Muruve, and P. Kubers. 2010. Intravascular Danger Signals Guide Neutrophils to Sites of Sterile Inflammation. *Science* 330:362-366.
3. Irimia, D., S.-Y. Liu, W. G. Tharp, A. Samadani, M. Toner, and M. C. Poznansky. 2006. Microfluidic system for measuring neutrophil migratory responses to fast switches of chemical gradients. *Lab Chip* 6:191-198.
4. Sackmann, E. K., E. Berthier, E. W. K. Young, M. A. Shelef, S. A. Wernimont, A. Huttenlocher, and D. J. Beebe. 2012. Microfluidic kit-on-a-lid: a versatile platform for neutrophil chemotaxis assays. *Blood* 120:e45-e53.
5. Lammermann, T., B. L. Bader, S. J. Monkley, T. Worbs, R. Wedlich-Soldner, K. Hirsch, M. Keller, R. Forster, D. R. Critchley, R. Fassler, and M. Sixt. 2008. Rapid leukocyte migration by integrin-independent flowing and squeezing. *Nature* 453:51-55.
6. Hawkins, R. J., M. Piel, G. Faure-Andre, A. M. Lennon-Dumenil, J. F. Joanny, J. Prost, and R. Voituriez. 2009. Pushing off the Walls: A Mechanism of Cell Motility in Confinement. *Physical Review Letters* 102:058103.
7. Konstantopoulos, K., P.-H. Wu, and D. Wirtz. 2013. Dimensional Control of Cancer Cell Migration. *Biophys J* 104:279-280.
8. Zigmond, S. H. 1978. Chemotaxis by polymorphonuclear leukocytes. *J Cell Biol* 77:269-287.
9. Malawista, S. E., and A. d. B. Chevance. 1997. Random locomotion and chemotaxis of human blood polymorphonuclear leukocytes (PMN) in the presence of EDTA: PMN in close quarters require neither leukocyte integrins nor external divalent cations. *Proceedings of the National Academy of Sciences* 94:11577-11582.
10. Butler, L. M., S. Khan, G. Ed Rainger, and G. B. Nash. 2008. Effects of endothelial basement membrane on neutrophil adhesion and migration. *Cell Immunol* 251:56-61.
11. Houk, A. R., A. Jilkine, C. O. Mejean, R. Boltyanskiy, E. R. Dufresne, S. B. Angenent, S. J. Altschuler, L. F. Wu, and O. D. Weiner. 2012. Membrane Tension Maintains Cell Polarity by Confining Signals to the Leading Edge during Neutrophil Migration. *Cell* 148:175-188.
12. Yanai, M., J. P. Butler, T. Suzuki, H. Sasaki, and H. Higuchi. 2004. Regional rheological differences in locomoting neutrophils. *American Journal of Physiology - Cell Physiology* 287:C603-C611.
13. Cassimeris, L., H. McNeill, and S. H. Zigmond. 1990. Chemoattractant-stimulated polymorphonuclear leukocytes contain two populations of actin filaments that differ in their spatial distributions and relative stabilities. *The Journal of Cell Biology* 110:1067-1075.
14. Matzner, Y., I. Vlodaysky, R. I. Michaeli, and A. Eldor. 1990. Selective inhibition of neutrophil activation by the subendothelial extracellular matrix: possible role in protection of the vessel wall during diapedesis. *Exp Cell Res* 189:233-240.
15. Oakes, P. W., D. C. Patel, N. A. Morin, D. P. Zitterbart, B. Fabry, J. S. Reichner, and J. X. Tang. 2009. Neutrophil morphology and migration are affected by substrate elasticity. *Blood* 114:1387-1395.

16. Stroka, K. M., and H. Aranda-Espinoza. 2009. Neutrophils display biphasic relationship between migration and substrate stiffness. *Cell Motil Cytoskeleton* 66:328-341.
17. Jannat, R. A., G. P. Robbins, B. G. Ricart, M. Dembo, and D. A. Hammer. 2010. Neutrophil adhesion and chemotaxis depend on substrate mechanics. *J Phys-Condens Mat* 22.
18. Barnhart, E. L., K. C. Lee, K. Keren, A. Mogilner, and J. A. Theriot. 2011. An adhesion-dependent switch between mechanisms that determine motile cell shape. *PLoS Biol* 9:e1001059.
19. Ziebert, F., and I. S. Aranson. 2013. Effects of adhesion dynamics and substrate compliance on the shape and motility of crawling cells. *PLoS One* 8:e64511.
20. Desai, R. A., M. K. Khan, S. B. Gopal, and C. S. Chen. 2011. Subcellular spatial segregation of integrin subtypes by patterned multicomponent surfaces. *Integr Biol* 3:560-567.
21. Tan, J. L., W. Liu, C. M. Nelson, S. Raghavan, and C. S. Chen. 2004. Simple approach to micropattern cells on common culture substrates by tuning substrate wettability. *Tissue Eng* 10:865-872.
22. Crocker, J. C., and B. D. Hoffman. 2007. Multiple-particle tracking and two-point microrheology in cells. *Method Cell Biol* 83:141-178.
23. Chen, C. S., M. Mrksich, S. Huang, G. M. Whitesides, and D. E. Ingber. 1997. Geometric control of cell life and death. *Science* 276:1425-1428.
24. Tan, J. L., J. Tien, D. M. Pirone, D. S. Gray, K. Bhadriraju, and C. S. Chen. 2003. Cells lying on a bed of microneedles: An approach to isolate mechanical force. *Proceedings of the National Academy of Sciences of the United States of America* 100:1484-1489.
25. Thery, M., V. Racine, A. Pepin, M. Piel, Y. Chen, J.-B. Sibarita, and M. Bornens. 2005. The extracellular matrix guides the orientation of the cell division axis. *Nature Cell Biology* 7:947-953.
26. Tee, S. Y., J. Fu, C. S. Chen, and P. A. Janmey. 2011. Cell shape and substrate rigidity both regulate cell stiffness. *Biophys J* 100:L25-27.
27. Ruiz, S. A., and C. S. Chen. 2007. Microcontact printing: A tool to pattern. *Soft Matter* 3:168-177.
28. Lee, D., and M. R. King. 2008. Microcontact printing of P-selectin increases the rate of neutrophil recruitment under shear flow. *Biotechnol Prog* 24:1052-1059.
29. Ricart, B. G., M. T. Yang, C. A. Hunter, C. S. Chen, and D. A. Hammer. 2011. Measuring traction forces of motile dendritic cells on micropost arrays. *Biophys J* 101:2620-2628.
30. Shen, K., V. K. Thomas, M. L. Dustin, and L. C. Kam. 2008. Micropatterning of costimulatory ligands enhances CD4⁺ T cell function. *Proceedings of the National Academy of Sciences* 105:7791-7796.
31. Tong, Z., L. S. Cheung, K. J. Stebe, and K. Konstantopoulos. 2012. Selectin-mediated adhesion in shear flow using micropatterned substrates: multiple-bond interactions govern the critical length for cell binding. *Integr Biol (Camb)* 4:847-856.
32. Brown, X. Q., K. Ookawa, and J. Y. Wong. 2005. Evaluation of polydimethylsiloxane scaffolds with physiologically-relevant elastic moduli: interplay of substrate mechanics and surface chemistry effects on vascular smooth muscle cell response. *Biomaterials* 26:3123-3129.
33. Fuard, D., T. Tzvetkova-Chevolleau, S. Decossas, P. Tracqui, and P. Schiavone. 2008. Optimization of poly-di-methyl-siloxane (PDMS) substrates for studying cellular adhesion and motility. *Microelectronic Engineering* 85:1289-1293.

34. Hung, W.-C., S.-H. Chen, C. D. Paul, K. M. Stroka, Y.-C. Lo, J. T. Yang, and K. Konstantopoulos. 2013. Distinct signaling mechanisms regulate migration in unconfined versus confined spaces. *The Journal of Cell Biology* 202:807-824.
35. Dunn, G. A. 1983. Characterising a kinesis response: time averaged measures of cell speed and directional persistence. *Agents and Actions Supplements* 12:14-33.
36. Lauffenburger, D. A., and J. J. Linderman. 1993. *Receptors : models for binding, trafficking, and signaling*. Oxford University Press, New York.
37. Kishimoto, T. K., M. A. Jutila, E. L. Berg, and E. C. Butcher. 1989. Neutrophil Mac-1 and MEL-14 adhesion proteins inversely regulated by chemotactic factors. *Science* 245:1238-1241.
38. Schiffmann, E., B. A. Corcoran, and S. M. Wahl. 1975. N-formylmethionyl peptides as chemoattractants for leucocytes. *Proceedings of the National Academy of Sciences* 72:1059-1062.
39. van den Berg, J. M., F. P. Mul, E. Schippers, J. J. Weening, D. Roos, and T. W. Kuijpers. 2001. Beta1 integrin activation on human neutrophils promotes beta2 integrin-mediated adhesion to fibronectin. *Eur J Immunol* 31:276-284.
40. Penberthy, T. W., Y. Jiang, F. W. Luscinikas, and D. T. Graves. 1995. MCP-1-stimulated monocytes preferentially utilize beta 2-integrins to migrate on laminin and fibronectin. *Am J Physiol* 269:C60-68.
41. Lishko, V. K., V. P. Yakubenko, and T. P. Ugarova. 2003. The interplay between integrins alphaMbeta2 and alpha5beta1 during cell migration to fibronectin. *Exp Cell Res* 283:116-126.
42. Henderson, R. B., L. H. Lim, P. A. Tessier, F. N. Gavins, M. Mathies, M. Perretti, and N. Hogg. 2001. The use of lymphocyte function-associated antigen (LFA)-1-deficient mice to determine the role of LFA-1, Mac-1, and alpha4 integrin in the inflammatory response of neutrophils. *J Exp Med* 194:219-226.
43. Phillipson, M., B. Heit, P. Colarusso, L. Liu, C. M. Ballantyne, and P. Kubes. 2006. Intraluminal crawling of neutrophils to emigration sites: a molecularly distinct process from adhesion in the recruitment cascade. *J Exp Med* 203:2569-2575.

Chapter 4

Human Neutrophil Adhesion Density Sensing at the Whole Cell Length Scale

Preface

The content of this chapter has been adapted from the version in preparation for submission to *Annals of Biomedical Engineering*. The manuscript was coauthored by **Steven J. Henry**, John C. Crocker, and Daniel A. Hammer. The content has been reproduced with knowledge of the coauthors. Specific author contributions were as follows: **SJH** designed and executed experiments, analyzed data, and wrote the manuscript; **JCC** consulted on design of analysis routines, data interpretation, and edited the manuscript. **DAH** supported the work, consulted on data interpretation, and edited the manuscript. Supplementary movies referenced in the prose will be retrievable from the published version online.

Abstract

Neutrophils, highly motile immune cells, are capable of a phenotypic switch with respect to their shape and mode of migration as driven by adhesive ligand density. In this study, we engineered planar adhesive environments to elucidate the length scale of neutrophil adhesion density sensing. The engineered surfaces were hybrid in that they presented neutrophils with high and low density cues simultaneously. By controlling island geometry we achieved arrays in which the local (on-island) adhesion density was

high but the global (multi-island) adhesion density over the entire cell-substrate interface was low. These hybrid surfaces were achieved by the stamp-off method of microcontact printing. Neutrophils in contact with these island arrays assumed a well-spread and directionally-persistent motile phenotype in contrast to their classic amoeboid morphology on continuous fields of high adhesion density. By virtue of our rationally designed substrates, we were able to conclude that neutrophils were sensing density at the whole cell length scale, integrating the stimulation received across their entire contact interface and mounting a whole cell response on the timescale of seconds. This work demonstrates the capacity of adhesive microenvironments to direct neutrophil motile phenotype which has broader implications in physiologic processes such as cancer metastasis.

Introduction

Neutrophils are a type of white blood cell (leukocyte) that responds to tissue trauma and infection on the timescale of seconds and minutes. These cells are equipped with a variety of terminal functions including phagocytosis, cytokine secretion, and nuclear-extracellular-trap setting (1). A prerequisite to the execution of any of these terminal functions is the cell's arrival at the locus of trauma (2) or infection (3) *via* vascular rolling, extravasation, and extravascular migration (4). In addition to soluble chemical cues that direct immune cell response and function, cells encounter numerous physical cues (e.g. stiffness, dimensionality, adhesivity, and topology) that are strong determinants of shape, force generation, and gene expression (5-6). Leukocyte response to physical cues such as substrate rigidity (7-9), confinement (10), and adhesion density (11) have been areas of on-going investigation.

Previously, others have demonstrated on planar (2D, unconfined) substrates that neutrophil contact area and force generation were stiffness-dependent (7-9). However we showed that stiffness alone was not a unique controller of adherent neutrophil shape or motility as varying the adhesivity of the surface also dictated cell phenotype on equally stiff substrates (11). In that work, using the method of microcontact printing, we quantified neutrophil shape and motility on sub-saturating densities of the extracellular matrix protein fibronectin (FN). On highly adhesive surfaces neutrophils assumed a classic amoeboid phenotype characterized by an elongated cell body, knob-like trailing uropod, and a narrow, ruffled leading edge lamellipodium (12-13). The observed motility was fast and consisted of frequent directional changes. However on low and moderately adhesive surfaces neutrophils assumed a phenotype reminiscent of fish keratocytes (14-15), characterized by the absence of a trailing uropod and a highly spread fan-like lamellipodium. The observed motility was a slow, but directionally persistent gliding motion. The capacity of adhesion density to alter the phenotypic mode of neutrophil migration drew analogy with adhesion sensitivity in fish keratocytes observed by Barnhart and coworkers (15) and computational predictions of the effect of adhesion on stiff substrates in migratory cells made by Ziebert and Aranson (16).

An open question from our prior work was the length scale over which the neutrophil adhesion density sensing was occurring. Were neutrophils responding to local adhesive cues on the length scale of receptor clusters or integrating adhesive stimulation across their entire contact interface? To address this question we employed the stamp-off variation of microcontact printing (17), to generate a hybrid surface in which high and low density adhesive cues were presented to neutrophils simultaneously. This

engineering approach of spatially organizing a cell's adhesive environment has been widely used to probe integrin clustering (18-19), effect of cell shape on viability (20) and focal adhesion architecture (21), and the role of extracellular matrix distribution on cell spreading (22) in the context of mesenchymal cells. Here we report the effect of adhesive ligand density distribution on neutrophils, a distinct cell of hematopoietic origin.

Materials and Methods

Media and Reagents

Rinsing buffer was Hanks' Balanced Salt Solution (Life Technologies, Carlsbad, CA) without calcium or magnesium supplemented with 10 mM HEPES (Life Technologies) and pH adjusted to 7.4. Storage buffer was rinsing buffer supplemented with 2 mg/mL glucose. Running buffer was storage buffer supplemented with 1.5 mM Ca^{2+} and 2 mM Mg^{2+} . Fibronectin (FN) was from human plasma (BD Biosciences, Bedford, MA). Labeling of FN *via* Alexa Fluor carboxylic acid, succinimidyl ester (Life Technologies) was performed in accordance with the manufacturer's recommended protocol. The nonionic triblock copolymer Pluronic F-127 (Sigma) was prepared at 0.2 % w/v in PBS without calcium and magnesium ("PBS(-)"). All solutions were sterile filtered or prepared sterile. The bicinchoninic acid protein assay (Pierce Biotechnology, Rockford, IL) was performed on stock FN solutions to measure concentration. Poly(dimethylsiloxane) (PDMS) was Sylgard 184 Silicone Elastomer from Dow Corning (Midland, MI) prepared per the specified weight ratio of base:cure agents, mixed vigorously, and degassed until optically clear.

Substrate Production

25:1 base:cure (w/w) PDMS stamps were cast against a silicon wafer to produce an extremely smooth surface. Stamps were trimmed to approximately 25 mm², sonicated in 200 proof ethanol for 10 min, rinsed twice in diH₂O and dried in a gentle stream of filtered N_{2(g)}. The surface of the PDMS stamp, previously cast against the silicon wafer, was incubated with a 50 µL aliquot of fluorescently labeled fibronectin (FN-AlexaFluor594) at a known concentration in PBS(-) for 1 hr at room temperature. After incubation stamps were rinsed twice in a submerging quantity (~ 50 mL) of diH₂O and dried in a gentle stream of filtered N_{2(g)} (Fig. 4.1 A).

For experiments with islands, these inked stamps were subject to stamp-off. An array of holes was generated by casting 10:1 base:cure (w/w) PDMS reliefs of positive silicon microfabricated-Post-Array-Detectors. Silicon masters were manufactured in Professor Christopher S. Chen's laboratory in the manner detailed by Yang et al.(23) Cast PDMS hole arrays were rendered hydrophilic by 7 min treatment in ultraviolet ozone (UVO Cleaner Model 342, Jelight, Irvine, CA) (17). The hydrophilic array was inverted, set atop the inked stamp, and peeled to produce two complimentary surfaces (Fig. 4.1 B).

PDMS coated coverslips were prepared from number one thickness glass coverslips (Fisher Scientific, Hampton, NH) of 25 mm diameter spun with degassed PDMS (10:1 base:cure (w/w)). Bare glass coverslips were cleaned *via* oxygen plasma etching and then spun at 4000 rpm for 1 min under PDMS. Leveling at RT, and baking at 65 °C overnight resulted in an approximately 10 µm thick layer of PDMS. Cured coverslips were affixed to the bottom of six-well tissue culture plates which had either been hot-punched or laser-cut to generate a 22 mm diameter opening in the bottom of the

wells. Coverslips were bonded using Norland Optical Adhesive 68 (Thorlabs, Newton, NJ). Mounted coverslips were rendered hydrophilic by 7 min treatment in ultraviolet ozone and then printed with a continuous field of protein or the stamped-off array of islands (Fig. 4.1 C). Substrates were blocked against non-specific binding by submerging in 0.2 % w/v F-127 in PBS(-) and incubating 30 min at RT (Fig. 4.1 D). After blocking, F-127 was exchanged for PBS(-) by repeated and gentle rinsing with running buffer. Chambers were pre-warmed to 37 °C in a cabinet incubator before cell plating and imaging.

Neutrophil Isolation

Whole blood was obtained from human donors *via* venipuncture. Samples were collected with University of Pennsylvania Institutional Review Board approval from consenting adult volunteers. Volunteers were required to be in good health and abstain from alcohol and all over-the-counter medication for 24 hrs prior to donation. Blood samples were allowed to cool to RT for 15 min and layered in a 1:1 ratio of whole blood to Polymorphprep (Axis-Shield, Oslo, Norway). Vials were spun for 45-60 min at 550-650 x *g* and 21 °C. After separation, the polymorphonuclear band and underlying separation media layer were aspirated into fresh round-bottom tubes. The isolated solution of cells and separation-media was diluted with rinsing buffer and spun for 10 min at 250 x *g* and 21 °C. Red Blood Cells (RBC) were eliminated from the resulting cell pellet *via* hypotonic lysis. After lysis, vials were centrifuged for 10 min at 250 x *g* and 21 °C and the RBC-free pellets resuspended in storage buffer. Neutrophils were stored at 10⁶ cells/mL on a tube rotisserie at 4 °C until time of plating to maintain cells in suspension.

Quantitative Fluorescence Microscopy

A non-flickering mercury bulb within the manufacturer-specified bulb lifetime was used to illuminate samples. Adjustments to bulb alignment and focus were made to achieve a uniform field of illumination. Within a given experimental series all acquisition parameters were held constant and images acquired identically. For each condition (i.e. feature and ligand density combination) multiple fields of view (FOV) were acquired across the entire printed domain as well as appropriate measurements of background fluorescence intensity. To mitigate the effects of photobleaching, focus was set in a region adjacent to the FOV actually imaged. To compare results across independent experiments, mean fluorescent intensities were normalized by the mean intensity of the saturating condition within that series after background subtraction.

Cell Motility Experiments and Data Analysis

Neutrophils were seeded into pre-warmed culture dishes and allowed to gravity sediment onto the printed arrays. Multiple position time-lapse videomicroscopy was performed to track cell shape and position for at least 30 min with images acquired every 15-60 sec. Motility quantification was performed using a custom suite of MATLAB (The MathWorks, Natick, MA) scripts which identified cell boundaries, computed geometric centroids, and connected centroids in consecutive frames to form trajectories. Cell tracking, mean squared displacement computation, and error analysis were based upon the multiple particle tracking method reviewed by Crocker and Hoffman (24).

Results

Engineering Substrates to Present Neutrophils with Two Adhesive Length Scales

By using the stamp-off method of microcontact printing (Fig. 4.1 A-D) we generated hexagonal arrays of submicron diameter islands of the extracellular matrix protein fibronectin (Fig. 4.1 I and J). A spread neutrophil was in contact with many of these islands (~ 100 islands/cell) at once as they were small and tightly spaced relative to the total size of the cell (Fig. 4.5 E). To aid visualization of the islands, contrast was enhanced in fluorescence images of Figure 4.1, however the unenhanced images are provided in Figure 4.2. Printed islands were hexagonally arranged with a measured mean diameter of $0.904 \pm 0.010 \mu\text{m}$ and pitch of $1.932 \pm 0.002 \mu\text{m}$ (Fig. 4.3). Quantities are means \pm standard error of the mean for five independent substrates with an average of 1296 printed islands measured per substrate. Individual islands had a surface area of $0.64 \mu\text{m}^2$ whereas the macroscopic surface area (i.e. a region containing many islands) represented a reduced contact area of 20 % compared to a continuous field.

The principle aim of this study was to generate a hybrid surface in which neutrophils were presented simultaneously with two length scales of adhesive stimulation. This required controlling array geometry and protein loading density such that the final printed surface had locally (i.e. on islands) high protein content but globally (i.e. the area equivalent to a cell body) low average protein content. Inking concentration was a more facile variable to manipulate at the wet bench as compared to island geometry. Therefore, we fixed array geometry and performed a sweep of inking concentrations to identify high and low conditions such that stamp-off of a high content

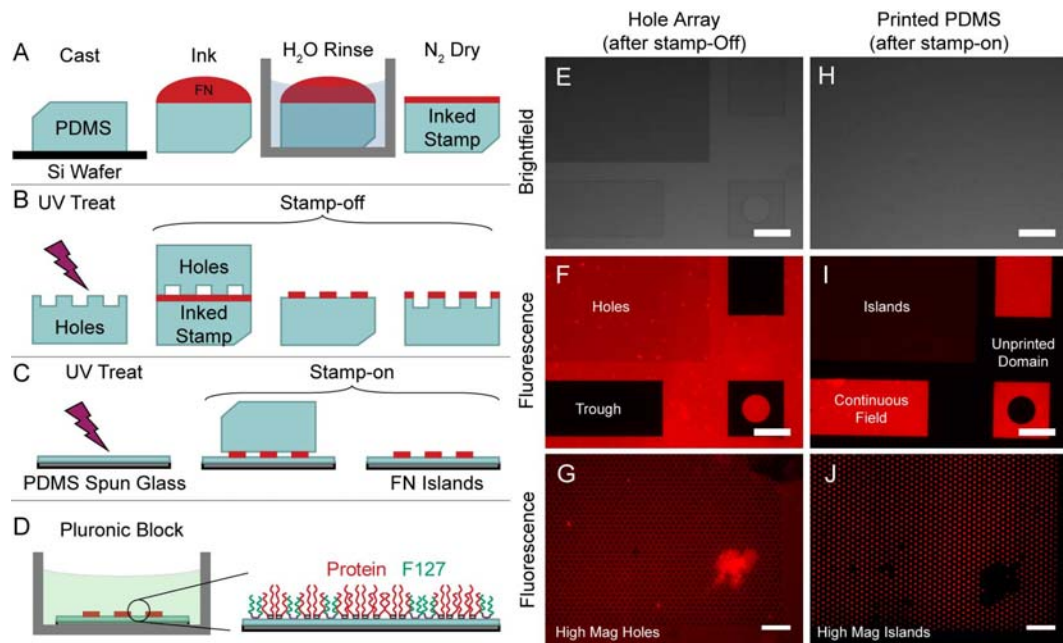


Figure 4.1 Stamp-off method of microcontact printing to generate island arrays. Substrate preparation consisted of: (A) stamp inking, (B) stamp-off, (C) stamp-on, and (D) Pluronic F-127 blocking. (E) Brightfield image of hole array used in stamp-off procedure. (F) Fluorescence image of protein on hole array after stamp-off. (G) Higher magnification image of hole array after stamp-off. (H) Brightfield image of PDMS coverslip after stamp-on. (I) Fluorescence image of protein after stamp-on. (J) Higher magnification image of island array after stamp-on. Fluorescence images were contrast-enhanced to assist island visualization. Unenhanced images are reported in Figure 4.2. Scalebars = 200 μm for E, F, H, and I. Scalebars = 10 μm for G and J.

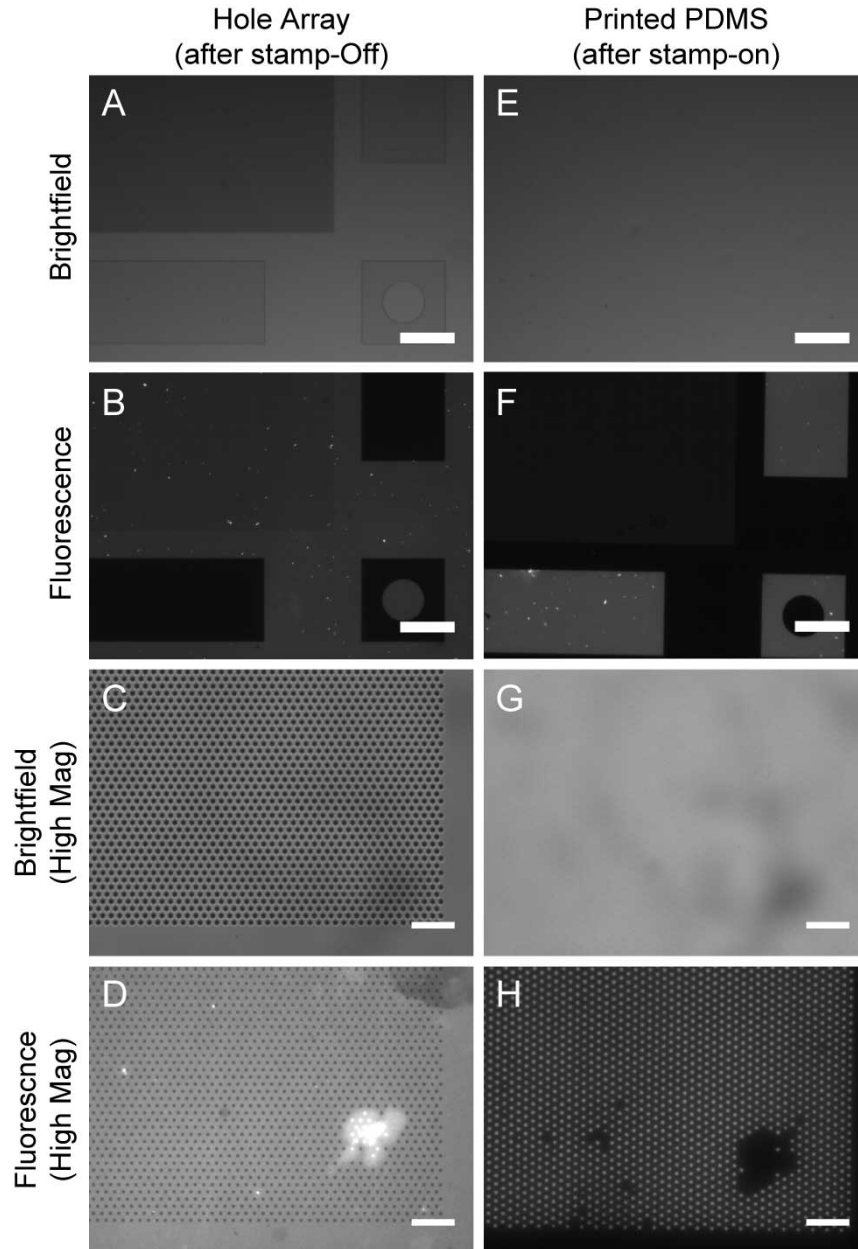


Figure 4.2 Stamp-off method of microcontact printing to generate island arrays: no contrast enhancement. (A) Brightfield image of hole array used in stamp-off procedure. (B) Fluorescence image of protein on hole array after stamp-off. (C) and (D) are higher magnification images of A and B features respectively. (E) Brightfield image of PDMS coverslip after stamp-on procedure. (F) Fluorescence image of protein on flat PDMS coverslip after stamp-on. (G) and (H) are higher magnification images of E and F features respectively. For a given magnification, fluorescence images were captured identically. Scalebars = 200 μm for A, B, E, and F. Scalebars = 10 μm for C, D, G, and H.

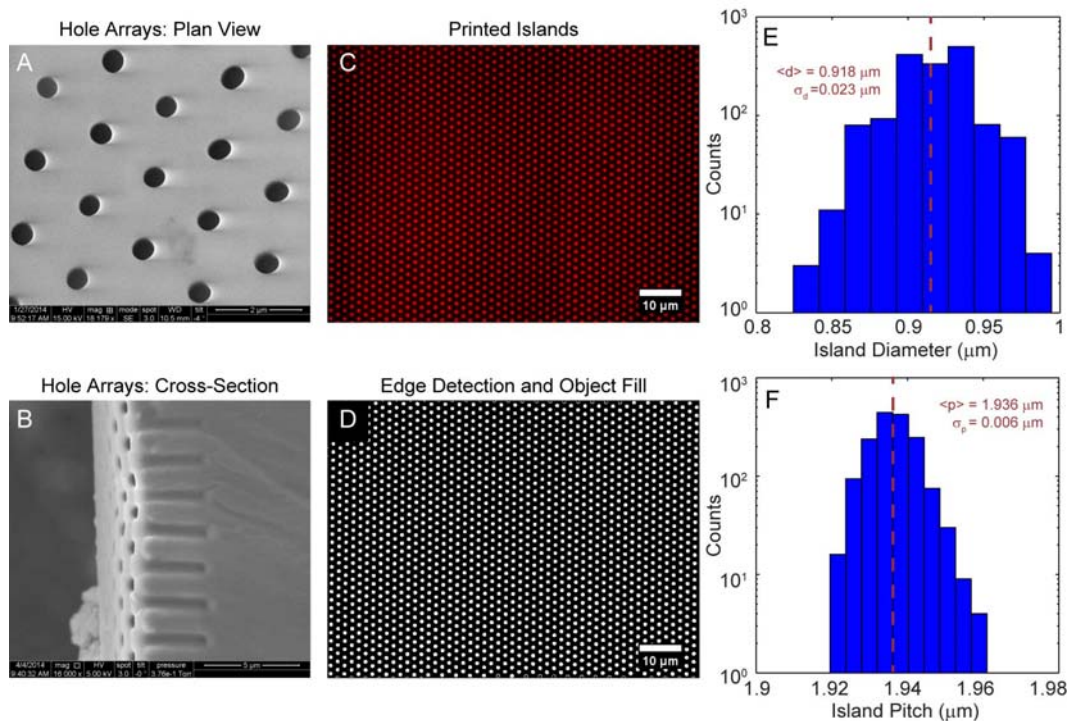


Figure 4.3 Measurement of mean printed island diameter and pitch. Scanning electron micrograph (A) plan view and (B) cross-section view of PDMS hole arrays used in stamp-off procedure. (C) Fluorescence image of printed islands and its (D) corresponding binary bitmap after image processing to measure island positions. (E) histogram of island diameters in D. (F) histogram of nearest neighbor distances (i.e. pitch) in D. The values of diameter and pitch quoted in the main text are the mean and standard error of the mean from five samples, analyzed in the same manner as above.

continuous field produced islands with a global area average equivalent to a low content continuous field.

For our experimental geometry and inking process, we identified that FN-saturated stamps (inking concentration in excess of 30 $\mu\text{g}/\text{mL}$) could be stamped-off to produce islands resulting in a global density equivalent to that of a uniformly inked 2 $\mu\text{g}/\text{mL}$ stamp (Fig. 4.4 A, shaded region). It is important to note that this set of conditions straddled the adhesive threshold (44 % relative to saturation, denoted by the dotted line in Fig. 4.4 A) we previously identified (11) below which the keratocyte-like phenotype occurs, but above which the amoeboid phenotype occurs.

Quantitative fluorescence microscopy was used to measure the on-island (Fig. 4.4 C), area average (Fig. 4.4 D), and between-island (Fig. 4.4 E) densities of printed protein. We found that printed islands had on-island densities (Fig. 4.4 B iii) approaching that of continuous fields of high protein content (Fig. 4.4 B i) while the protein content between islands was nearly zero (Fig. 4.4 B v). The area average protein content of the islands (Fig. 4.4 B iv) was equivalent to that of 2 $\mu\text{g}/\text{mL}$ continuous fields of protein (Fig. 4.4 B ii). For each condition 10-12 fields of view were acquired from each of four independent substrates.

Neutrophils Integrate Adhesive Stimulation Across Entire Contact Interface

We previously published observations of a phenotypic switch in neutrophil shape and motility governed by adhesive density (11). Here, consistent with those findings, we observed the amoeboid phenotype on high density (50 $\mu\text{g}/\text{mL}$) continuous fields of FN (Fig. 4.5 A, D) and the keratocyte-like phenotype on low density (2 $\mu\text{g}/\text{mL}$) continuous fields of FN (Fig. 4.5 B). Both high and low density continuous fields represented

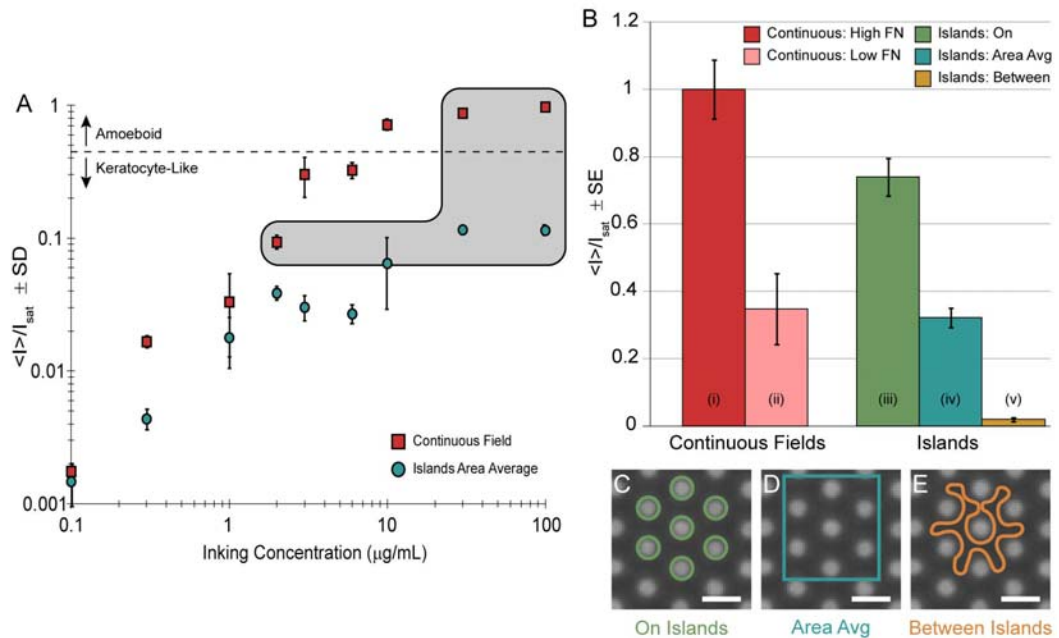


Figure 4.4 Engineering islands with two adhesive length scales. (A) Quantitative fluorescence microscopy of printed continuous fields (red squares) and stamped-off islands (blue circles). Gray shaded region represents domain where stamp-off of high density continuous fields produces islands with an area average equivalent to a low density continuous field. Dotted line denotes adhesive threshold delineating neutrophil phenotypes. Errorbars are \pm standard deviation from 2-4 replicates for each concentration within a single experiment. (B) Quantitative fluorescence microscopy of representative substrates used in adhesion and motility assays: (i) high density FN (50 $\mu\text{g/mL}$) continuous fields, (ii) low density (2 $\mu\text{g/mL}$) continuous fields. (iii) on islands (see ROI of C), (iv) area average of protein density across islands (see ROI of D), and (v) residual protein density between islands (see ROI of E). Scalebars = 2 μm . Errorbars are \pm standard error of the mean from four independent substrate preparations.

surfaces with uniform adhesive stimulation across the cell-substrate interface. By contrast, our hybrid island surfaces presented the cells with two effective adhesive length scales simultaneously. On the scale of single islands the density was high, comparable to that of high protein-content continuous fields. On the scale of multiple islands the density was low, comparable to that of low protein-content continuous fields. We hypothesized that if a neutrophil was sensitive to local density it would assume the amoeboid phenotype whereas if it was sensitive to global density, across its contact interface, it would assume the keratocyte-like phenotype. Consistent with the later hypothesis, neutrophils assumed the keratocyte-like phenotype on engineered islands where the total protein content averaged over the cell contact interface was low (Fig. 4.5 C, E). Both phenotypes were observable in the same field of view when continuous fields were adjacent to discrete islands (Fig. 4.5 G). In Figure 4.6 the phenotype scores for the three experimental conditions across all FOVs acquired are reported. On high density continuous fields the amoeboid phenotype predominates (59 % amoeboid) while on low density continuous fields and islands the keratocyte-like phenotype predominates (70 % and 78% keratocyte-like respectively).

Additionally, we observed that neutrophils could exhibit a rapid change in motile phenotype. Neutrophils migratory in the amoeboid phenotype on high density continuous fields could transform into the keratocyte-like phenotype on the order of seconds when they moved from continuous fields to stamped-off islands. The movie corresponding to Fig. 4.5 G is provided as Electronic Supplementary Material Movie S1. There was a small degree of convective flow in the system that allowed neutrophils to transit across the non-adhesive domain between fields and islands. It is important to note that this non-

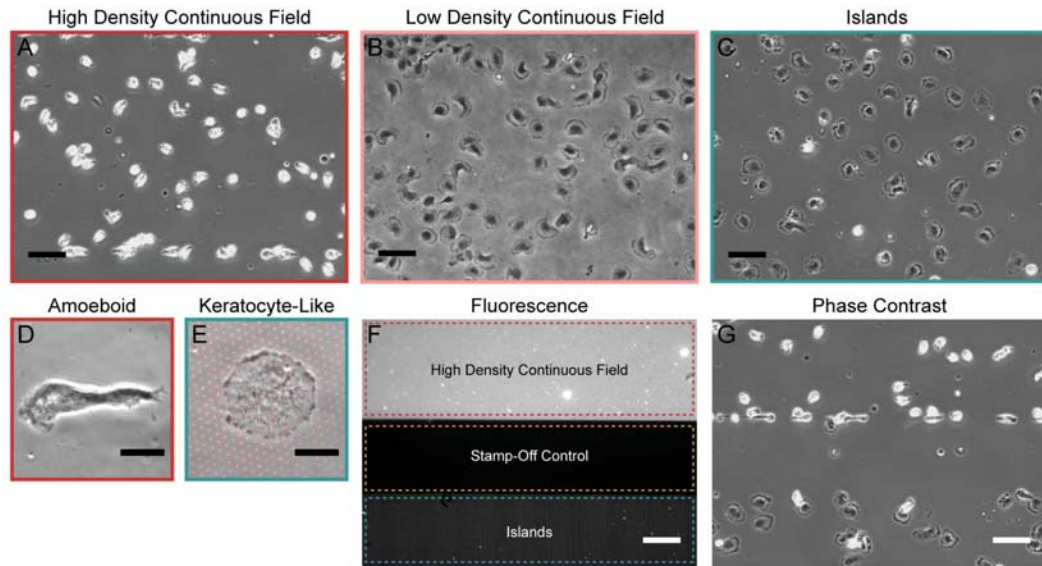


Figure 4.5 Neutrophils sense density at whole cell length scale. On continuous fields of FN neutrophils assume an (A) amoeboid phenotype on high density surfaces and a (B) keratocyte-like phenotype on low density surfaces. (C) However, on discrete islands where local density is high and global density is low, neutrophils assume the keratocyte-like phenotype. Higher magnification images of (D) amoeboid phenotype on continuous field and (D) keratocyte-like phenotype on discrete islands where fluorescent signal has been superimposed. (F) Fluorescence image corresponding to G. (G) Phase contrast image of neutrophils exhibiting amoeboid and keratocyte-like phenotypes in the same FOV with no adhesion in stamp-off control domain. Timelapse movie of neutrophil motility in G is supplied as Electronic Supplementary Movie S1. Scalebars = 50 μm for A, B, C, F, and G. Scalebars = 10 μm for D and E.

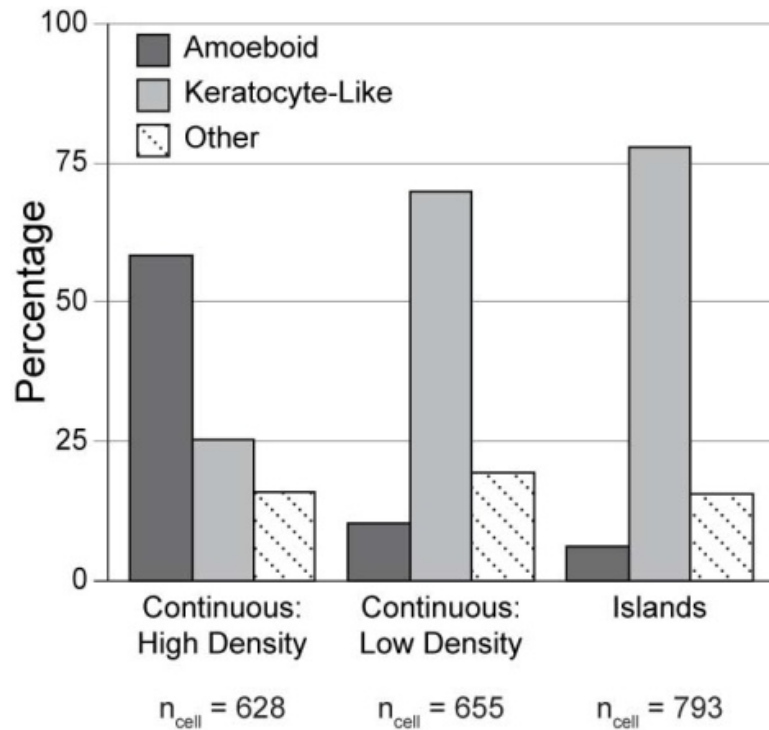


Figure 4.6 Observation of each phenotype per experimental condition. Observed phenotypes for the three experimental conditions across all FOVs acquired were manually scored. On high density continuous fields the largest fraction was amoeboid (59 %). On low density continuous fields and islands the largest fraction was keratocyte-like (70 % and 78 % respectively). Other denotes cells that were adherent but not spread or had ambiguous morphologies.

adhesive domain was a control, establishing that the residual protein content between islands (Fig. 4.4 B v) was not sufficient to support adhesion. This can be concluded because the large non-adhesive band between the continuous field and discrete islands was generated by stamp-off in a manner identical to that used in the interstitial space between islands.

Comparable Neutrophil Motility on Discrete Islands and Continuous Fields

After 30 minutes of motility neutrophils undergoing amoeboid migration on high density continuous fields of FN (Fig. 4.7 A) achieve a greater net dispersal than their keratocyte-like counterparts on low density continuous fields (Fig. 4.7 B) as well as hybrid islands (Fig. 4.7 C). A model-independent metric of dispersal is the mean maximum displacement ($\langle \max(|\Delta r|) \rangle$) of all trajectories followed through 30 min. Cell trajectories followed less than 30 min were excluded in the computation of this analysis to avoid biasing the data. Keratocyte-like motility on low density continuous fields of FN and hybrid islands was statistically indistinguishable (Fig. 4.7 D).

To assess the evolution of the motile cells we also computed mean squared displacements (MSD) as a function of time (Fig. 4.8 A) and fit the curves with the persistent random walk model of cell kinesis ($\langle \Delta r^2(\tau) \rangle = 2S^2P[\tau - P(1 - \exp(-\tau/P))]$) (25–26) in terms of the best-fit parameters speed (S, Fig. 4.8 B), persistence (P, Fig. 4.8 C), and the random motility coefficient ($S^2P/2$, Fig. 4.8 D). This analysis made clear that the origin of the increased dispersion seen in the model-independent analysis (Fig. 4.7 A) was the result of amoeboid neutrophils moving at least twice as fast as keratocyte-like neutrophils (Fig. 4.8 B). Although there was no statistically significant difference in the directional persistence of the two phenotypes we did see an increase in the distribution of

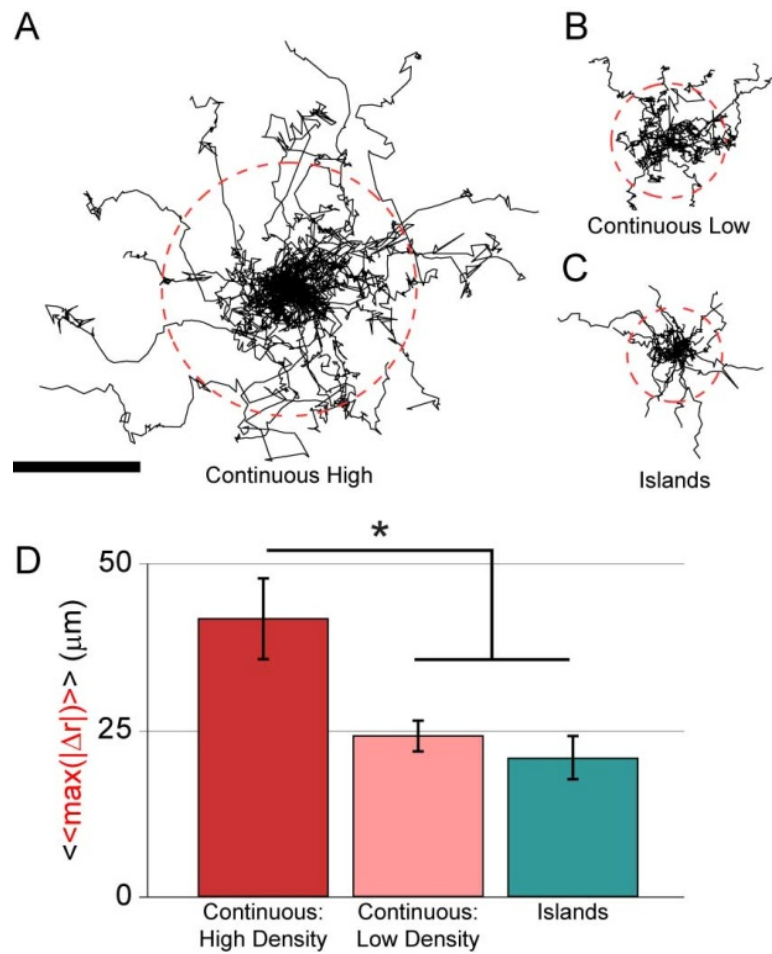


Figure 4.7 Neutrophil motility on islands is comparable to low density continuous fields. Cell trajectories through 30 min of motility from single representative experiments of (A) amoeboid motility on high density continuous fields, (B) keratocyte-like motility on low density continuous fields, and (C) keratocyte-like motility on hybrid islands. Scalebar = 50 μm . Dotted red circle is mean maximum displacement ($\langle\max(|\Delta r|)\rangle$) of set of 30 min trajectories. (D) Mean of the set of mean maximum displacements ($\langle\langle\max(|\Delta r|)\rangle\rangle$) across all independent observations. Error bars are \pm standard error of the mean ($N_{\text{experiments/condition}} = 6-7$, $n_{\text{cell/experiment}} = 17-27$). Asterisk denotes significant difference between populations as computed by post-hoc Dunn-Sidak multiple comparisons method ($p < 0.05$).

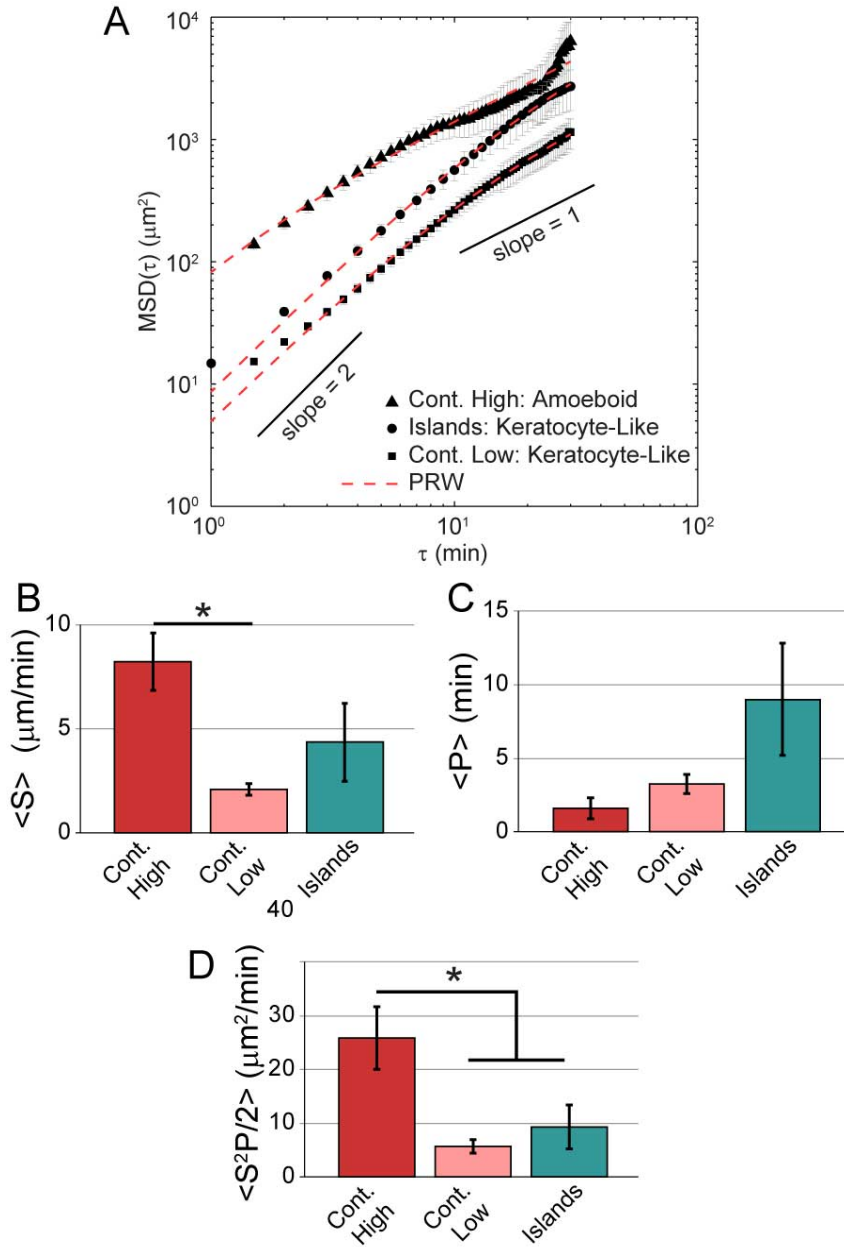


Figure 4.8 MSD analysis of neutrophil motility on islands and fields. (A) Mean squared displacements from single representative experiments for each condition. Dotted red line is fit of persist random walk model (PRW) to empirical data. Mean of the set of model fit parameters (B) speed, (C) persistence, and (D) the random motility coefficient for all independent observations. Error bars are \pm standard error of the mean ($N_{\text{experiments/condition}} = 6-7$, $n_{\text{cell/experiment}} = 36-42$). Asterisk denotes significant difference between populations as computed by post-hoc Dunn-Sidak multiple comparisons method ($p < 0.05$).

persistence values of keratocyte-like neutrophils on islands (Fig. 4.8 C). Finally, we observe a statistically significant increase in amoeboid migration relative to keratocyte-like migration in terms of the random motility coefficient as this metric is dominated by cell speed (Fig. 4.8 D).

Discussion

We previously reported the ability of adhesion ligand density to dictate the shape and mode of neutrophil migration on equally stiff substrates (11). A question that arose from that work was the length scale of the ligand density sensitivity. To address this question we employed the stamp-off method of microcontact printing (17) and engineered adhesive environments to present neutrophils with two adhesive length scales simultaneously. By careful control of protein loading for a given adhesive geometry we were able to achieve a three condition experimental state space that straddled the critical adhesive threshold (44 % saturation) which delineated the keratocyte-like phenotype from the classical amoeboid phenotype. The three conditions explored were as follows: a continuous field of FN at high density (> 44 % saturation) known to elicit the amoeboid phenotype, a continuous field of FN at a low density (< 44 % saturation) known to elicit the keratocyte-like phenotype, and a hybrid island array where the on-island density was high but the area average density was low. On these hybrid adhesive surfaces neutrophils robustly assumed the keratocyte-like phenotype, integrating the adhesive stimuli over their entire contact interface and responding as if the set of discrete islands were a continuous field.

The integration of distributed adhesive contact into a global cell response has been observed in a variety of mesenchymal cells (20, 22). Lehnert and coworkers

explored a large state space of island size and spacing in fibroblasts and melanoma cells and found that these cells spread on discrete islands of $< 1 \mu\text{m}^2$ with pitch $< 5 \mu\text{m}$ as if they were continuous fields of protein. Likewise we observe that neutrophils spread and are motile on discrete islands of $0.64 \mu\text{m}^2$ and $1.9 \mu\text{m}$ pitch as if they were continuous fields of low density protein. However, the dramatic reduction in contact area that occurs when neutrophils assume the highly motile amoeboid phenotype is quite distinct from the behavior of mesenchymal cells in the presence of high density adhesive stimulation. It is interesting to note that the reduced contact area of amoeboid neutrophils ($\sim 100 \mu\text{m}^2$, although admittedly difficult to measure in brightfield) is within the same order of magnitude as the actual adhesive contact of keratocyte-like neutrophils on islands (~ 100 islands/cell $\times 0.64 \mu\text{m}^2/\text{island}$). Thus the phenotypic switch in neutrophils could be driven by the cell's attempt to maintain a constant level of adhesive stimulation across its contact interface.

While the islands we employed in this study are submicron they are large compared to the lateral distance between adhesive ligand binding sites of 58-73 nm necessary to support integrin receptor clustering (18). Clustering of β_2 (CD11b) integrins and the downstream cytoskeletal rearrangement that results is critical to the neutrophil's execution of terminal effector functions like reactive oxygen intermediate generation and proteolytic enzyme secretion (27). We previously showed, using function blocking antibodies, that neutrophils utilize the promiscuous integrin receptor MAC-1 ($\alpha_M\beta_2$) to support haptokinetic migration and density sensitivity (11). Therefore the keratocyte-like phenotype on islands suggests neutrophils do not respond to adhesive ligand density on

the receptor cluster length scale but rather integrate the total adhesive stimulus across all clusters.

The pursuit of constant adhesive stimulation across the contact interface may itself be the consequence of the cell attempting to maintain tensional homeostasis. Our lab has previously demonstrated that neutrophil traction stresses are highest in the rear uropod of motile amoeboid neutrophils (28). This asymmetric rearward contractility is understood to be the mechanism by which the cytoplasm is propelled forward generating a protrusive force despite the contact footprint of the cell being quite small. This is in contrast to the behavior of less polarized mesenchymal cells which show a linear increase in traction generation as contact footprint increases (29). Whereas the keratocyte-like phenotype presumably represents a state of high traction generation doing work against the substrate, the amoeboid phenotype represents a state of high traction generation doing work to deform the cell body itself. In neutrophils these distinct states are archived on equally stiff substrates but elicited by the extent of adhesive stimulation imparted to the cell.

Our findings may have applicability to the study of cancer metastasis and specifically the epithelial to mesenchymal transition model (30-31). It has been established that tumor stiffening drives integrin clustering which supports the malignant cell phenotype (32). Perhaps a concurrent increase in extracellular adhesivity of the stiff tumor microenvironment could subsequently induce a highly motile amoeboid-like transition in previously stationary malignancies.

Acknowledgements

We are grateful to Eric Johnston for laboratory assistance and Christopher S. Chen, PhD and Ravi A. Desai, PhD for their time and expertise in teaching us the stamp-off method of microcontact printing. Funding for this work was provided by a National Science Foundation Graduate Research Fellowship to SJH and a grant from the National Institutes of Health (HL18208) to DAH.

References

1. Borregaard, N. 2010. Neutrophils, from marrow to microbes. *Immunity* 33:657-670.
2. McDonald, B., K. Pittman, G. B. Menezes, S. A. Hirota, I. Slaba, C. C. M. Waterhouse, P. L. Beck, D. A. Muruve, and P. Kubers. 2010. Intravascular Danger Signals Guide Neutrophils to Sites of Sterile Inflammation. *Science* 330:362-366.
3. Nathan, C. 2006. Neutrophils and immunity: challenges and opportunities. *Nat Rev Immunol* 6:173-182.
4. Ley, K., C. Laudanna, M. I. Cybulsky, and S. Nourshargh. 2007. Getting to the site of inflammation: the leukocyte adhesion cascade updated. *Nature Reviews Immunology* 7:678-689.
5. Charras, G., and E. Sahai. 2014. Physical influences of the extracellular environment on cell migration. *Nat Rev Mol Cell Biol* 15:813-824.
6. Vogel, V., and M. Sheetz. 2006. Local force and geometry sensing regulate cell functions. *Nat Rev Mol Cell Biol* 7:265-275.
7. Jannat, R. A., G. P. Robbins, B. G. Ricart, M. Dembo, and D. A. Hammer. 2010. Neutrophil adhesion and chemotaxis depend on substrate mechanics. *J Phys-Condens Mat* 22.
8. Oakes, P. W., D. C. Patel, N. A. Morin, D. P. Zitterbart, B. Fabry, J. S. Reichner, and J. X. Tang. 2009. Neutrophil morphology and migration are affected by substrate elasticity. *Blood* 114:1387-1395.
9. Stroka, K. M., and H. Aranda-Espinoza. 2009. Neutrophils display biphasic relationship between migration and substrate stiffness. *Cell Motil Cytoskeleton* 66:328-341.
10. Lammermann, T., B. L. Bader, S. J. Monkley, T. Worbs, R. Wedlich-Soldner, K. Hirsch, M. Keller, R. Forster, D. R. Critchley, R. Fassler, and M. Sixt. 2008. Rapid leukocyte migration by integrin-independent flowing and squeezing. *Nature* 453:51-55.
11. Henry, S. J., J. C. Crocker, and D. A. Hammer. 2014. Ligand density elicits a phenotypic switch in human neutrophils. *Integrative Biology* 6:348-356.
12. Zigmond, S. H. 1978. Chemotaxis by polymorphonuclear leukocytes. *J Cell Biol* 77:269-287.
13. Cassimeris, L., H. McNeill, and S. H. Zigmond. 1990. Chemoattractant-stimulated polymorphonuclear leukocytes contain two populations of actin filaments that differ in their spatial distributions and relative stabilities. *The Journal of Cell Biology* 110:1067-1075.
14. Lee, J., and K. Jacobson. 1997. The composition and dynamics of cell-substratum adhesions in locomoting fish keratocytes. *J Cell Sci* 110 (Pt 22):2833-2844.
15. Barnhart, E. L., K. C. Lee, K. Keren, A. Mogilner, and J. A. Theriot. 2011. An adhesion-dependent switch between mechanisms that determine motile cell shape. *PLoS Biol* 9:e1001059.
16. Ziebert, F., and I. S. Aranson. 2013. Effects of adhesion dynamics and substrate compliance on the shape and motility of crawling cells. *PLoS One* 8:e64511.

17. Desai, R. A., M. K. Khan, S. B. Gopal, and C. S. Chen. 2011. Subcellular spatial segregation of integrin subtypes by patterned multicomponent surfaces. *Integr Biol* 3:560-567.
18. Arnold, M., E. A. Cavalcanti-Adam, R. Glass, J. Blummel, W. Eck, M. Kanteleiner, H. Kessler, and J. P. Spatz. 2004. Activation of integrin function by nanopatterned adhesive interfaces. *Chemphyschem* 5:383-388.
19. Desai, R., M. Yang, N. Sniadecki, W. Legant, and C. S. Chen. 2007. Microfabricated Post-Array-Detectors (mPADs): an Approach to Isolate Mechanical Forces. *Journal of Visualized Experiments* 7.
20. Chen, C. S., M. Mrksich, S. Huang, G. M. Whitesides, and D. E. Ingber. 1997. Geometric control of cell life and death. *Science* 276:1425-1428.
21. Chen, C. S., J. L. Alonso, E. Ostuni, G. M. Whitesides, and D. E. Ingber. 2003. Cell shape provides global control of focal adhesion assembly. *Biochemical and Biophysical Research Communications* 307:355-361.
22. Lehnert, D., B. Wehrle-Haller, C. David, U. Weiland, C. Ballestrem, B. A. Imhof, and M. Bastmeyer. 2004. Cell behaviour on micropatterned substrata: limits of extracellular matrix geometry for spreading and adhesion. *Journal of Cell Science* 117:41-52.
23. Yang, M. T., J. Fu, Y. K. Wang, R. A. Desai, and C. S. Chen. 2011. Assaying stem cell mechanobiology on microfabricated elastomeric substrates with geometrically modulated rigidity. *Nat Protoc* 6:187-213.
24. Crocker, J. C., and B. D. Hoffman. 2007. Multiple-particle tracking and two-point microrheology in cells. *Method Cell Biol* 83:141-178.
25. Dunn, G. A. 1983. Characterising a kinesis response: time averaged measures of cell speed and directional persistence. *Agents and Actions Supplements* 12:14-33.
26. Lauffenburger, D. A., and J. J. Linderman. 1993. *Receptors : models for binding, trafficking, and signaling*. Oxford University Press, New York.
27. Raptis, S. Z., S. D. Shapiro, P. M. Simmons, A. M. Cheng, and C. T. Pham. 2005. Serine protease cathepsin G regulates adhesion-dependent neutrophil effector functions by modulating integrin clustering. *Immunity* 22:679-691.
28. Smith, L. A., H. Aranda-Espinoza, J. B. Haun, M. Dembo, and D. A. Hammer. 2007. Neutrophil traction stresses are concentrated in the uropod during migration. *Biophys J* 92:L58-60.
29. Wang, N., E. Ostuni, G. M. Whitesides, and D. E. Ingber. 2002. Micropatterning tractional forces in living cells. *Cell Motil Cytoskeleton* 52:97-106.
30. Yao, D., C. Dai, and S. Peng. 2011. Mechanism of the Mesenchymal–Epithelial Transition and Its Relationship with Metastatic Tumor Formation. *Molecular Cancer Research* 9:1608-1620.
31. Thiery, J. P., H. Acloque, R. Y. J. Huang, and M. A. Nieto. 2009. Epithelial-Mesenchymal Transitions in Development and Disease. *Cell* 139:871-890.
32. Paszek, M. J., N. Zahir, K. R. Johnson, J. N. Lakins, G. I. Rozenberg, A. Gefen, C. A. Reinhart-King, S. S. Margulies, M. Dembo, D. Boettiger, D. A. Hammer, and V. M. Weaver. 2005. Tensional homeostasis and the malignant phenotype. *Cancer Cell* 8:241-254.

Chapter 5

Protrusive and Contractile Forces of Spreading Human Neutrophils

Preface

The content of this chapter has been adapted from the version under revision at *Biophysical Journal*. The manuscript was coauthored by **Steven J. Henry**, Christopher S. Chen, John C. Crocker, and Daniel A. Hammer. The content has been reproduced with knowledge of the coauthors. Specific author contributions were as follows: **SJH** designed and executed experiments, analyzed data, and wrote the manuscript; CSC provided mPADs masters and edited the manuscript. JCC consulted on design of analysis routines, data interpretation, and edited the manuscript. DAH supported the work, consulted on data interpretation, and edited the manuscript. Supplementary movies referenced in the prose will be retrievable from the published version online.

Abstract

Human neutrophils are mediators of innate immunity and undergo dramatic shape changes at all stages of their functional life cycle. In this work we quantified the forces associated with a neutrophil's morphological transition from a non-adherent, quiescent sphere to its adherent and spread state. We did this by tracking, with high spatial and temporal resolution, the cell's mechanical behavior during spreading on microfabricated-post-array-detectors (mPADs) printed with the extracellular matrix protein fibronectin.

Two dominant mechanical regimes were observed: transient protrusion and steady state contraction. During spreading, a wave of protrusive force (75 ± 8 pN/post) propagates radially outwards from the cell center (at a speed of 206 ± 28 nm/s). Once completed, the cells enter a sustained contractile state. While post engagement during contraction was continuously varying, posts within the core of the contact zone were less contractile (-20 ± 10 pN/post) than those residing at the geometric perimeter (-106 ± 10 pN/post). The magnitude of the protrusive force was found to be unchanged in response to cytoskeletal inhibitors of lamellipodium formation and myosin II mediated contractility. However, cytochalasin B, known to reduce cortical tension in neutrophils, slowed spreading velocity (61 ± 37 nm/s) without significantly reducing protrusive force. Relaxation of the actin cortical shell was a prerequisite for spreading on post arrays as demonstrated by stiffening in response to jasplakinolide and the abrogation of spreading. ROCK and myosin II inhibition reduced long term-contractility. Function blocking antibody studies revealed haptokinetic spreading was induced by β_2 integrin ligation. Neutrophils were found to moderately invaginate the post arrays to a depth of approximately $1 \mu\text{m}$ as measured from spinning disk confocal microscopy. Our work suggests a competition of adhesion energy, cortical tension, and the relaxation of cortical tension is at play at the onset of neutrophil spreading.

Introduction

Neutrophils are white blood cells of the innate immune system. They act as first responders to tissue trauma (1) and pathogen challenges (2), initiating the body's inflammatory response on the timescale of seconds to minutes. Central to neutrophil

function is spreading in which the cell begins as a quiescent sphere and becomes well-spread and migratory (3). There are numerous observations of the dynamics of neutrophil spreading *in vitro*. Lomakina et al. (4) measured neutrophil spreading as haptokinetically-stimulated by immobilized fields of the chemokine interleukin 8. Sengupta et al. (5) measured neutrophil spreading on continuous fields of fibronectin, induced by soluble formylated chemotactic peptide. Using reflection interference contrast microscopy, it was observed that regions of closest membrane contact to the substrate were present at the periphery of the spreading cell. It was hypothesized that these regions would ultimately correspond to domains of high force generation. In neither study were the tractions associated with neutrophil spreading directly measured.

Our goal in this work was to measure the forces of neutrophil spreading on microfabricated-post-array-detectors (mPADs). While mPADs have long been used to measure forces in mesenchymal cells (6-10) they have only recently been employed to study immune cell function. Ricart et al. (11) used mPADs to measure the traction stresses of dendritic cells undergoing chemotaxis and established that these cells migrate by a frontward pulling mechanism. Bashour et al. (12) explored the mechanics of T-lymphocyte activation and spreading on mPADs functionalized by antibodies to the activation receptors CD28 and CD3. While the mechanodynamics of T-lymphocyte spreading were measured, the role of the cell cytoskeleton was not investigated.

Here, we report the protrusive and contractile behavior of spreading neutrophils with high spatial and temporal resolution on fibronectin printed mPADs. Spreading was a fast, radially symmetric wave sufficiently forceful to generate outward deflections of the underlying posts. After protrusion, cells contracted with posts on the perimeter of the

contact zone exhibiting higher contractility than those in the core. Small molecule inhibitor perturbations of the cellular cytoskeleton revealed that cortical actin relaxation was critical upstream of protrusion but protrusion itself was not myosin II dependent. Conversely, long-time sustained contractility was dependent on ROCK and myosin II. Function blocking antibody studies revealed that haptokinetic spreading on fibronectin was β_2 integrin induced. Confocal z-stacks uncovered moderate post invagination into the cell body which was ultimately fortuitous in reporting the energy associated with the quiescent-to-spread shape change.

Materials and Methods

Media and Reagents

Rinsing buffer was Hanks' Balanced Salt Solution (Life Technologies, Carlsbad, CA) without calcium or magnesium supplemented with 10 mM HEPES (Life Technologies) and pH adjusted to 7.4. Storage buffer was rinsing buffer supplemented with 2 mg/mL glucose. Running buffer was storage buffer supplemented with 1.5 mM Ca^{2+} and 2 mM Mg^{2+} . Fibronectin (FN) was from human plasma (BD Biosciences, Bedford, MA). Labeling of FN *via* Alexa Fluor carboxylic acid, succinimidyl ester (Life Technologies) was performed in accordance with the manufacturer's recommended protocol. The nonionic triblock copolymer Pluronic F-127 (Sigma) was prepared at 0.2% w/v in PBS without calcium and magnesium ("PBS(-)"). Stock delta9-DiI lipophilic membrane dye (Life Technologies) was prepared in 200 proof ethanol at 50 ng/mL. All solutions were sterile filtered or prepared sterile. The bicinchoninic acid protein assay (Pierce Biotechnology, Rockford, Il) was performed on stock FN solutions to measure concentration. Poly(dimethylsiloxane) (PDMS) was Sylgard 184 Silicone Elastomer from

Dow Corning (Midland, MI) prepared per the specified weight ratio of base:cure agents, mixed vigorously, and degassed until optically clear. Silane was Trichloro(1H, 1H, 2H, 2H-perfluorooctyl)silane from Sigma (Saint Louis, MO).

microfabricated-Post-Array-Detectors (mPADs) and Microcontact Printing

mPADs were fabricated and printed as detailed by Yang et al. (13). Scanning electron microscopy of our cast arrays (Fig. 5.1 A) allowed us to characterize the post geometry (diameter = 604 ± 31 nm, length = 5.576 ± 0.286 μ m, $m \pm sd$) and compute an associated spring constant, $k_{\text{spring}} = 0.28 \pm 0.09$ pN/nm. For these posts, with length tenfold longer than width, the Schoen et al. (14) substrate-warping correction to k_{spring} of seven percent was less than the measurement error.

The positive silicon masters were manufactured in and provided directly by the Chen laboratory. From these positive masters, negative PDMS reliefs were cast then silanized by vapor deposition. Silanized molds were coated with a small amount of 10:1 base:cure (w/w) PDMS and degassed. The 10:1 PDMS for positive casting was carefully weighed using a calibrated analytical balance and thoroughly mixed and degassed before coating the molds. PDMS coated molds were pressed against the oxygen plasma cleaned glass coverslips and leveled in a 110 °C for 20 hr. After curing, molds were released in a shallow dish of 200 proof ethanol and sonicated for 2 min. Posts were recovered in a Samdri-PVT-3D critical point drier (Tousimis, Rockville, MD) and stored in a dessicator jar until use.

25:1 base:cure (w/w) PDMS stamps were cast against a silicon wafer to produce an extremely smooth surface. Stamps were trimmed to approximately 25 mm², sonicated in 200 proof ethanol for 10 min, rinsed twice in diH₂O and dried in a gentle stream of

filtered N_{2(g)}. The surface of the PDMS stamp, previously cast against the silicon wafer, was incubated with 50 µL of 100 µg/mL FN-AF488 in PBS(-) for 1 hr at room temperature (RT). After incubation stamps were rinsed twice in a submerging quantity (~ 50 mL) of diH₂O and dried in a gentle stream of filtered N_{2(g)}. Cast mPADs on coverslips were loaded in autoclaved Attofluor chambers (Life Technologies) and rendered hydrophilic by 7 min treatment in ultraviolet ozone (UVO Cleaner Model 342, Jelight, Irvine, CA) (15). Inked stamps were inverted and set atop the UVO treated post tips. After contact the chamber was flooded with 200 proof ethanol and the stamp removed in one quick motion. Posts were observed under fluorescence microscopy to verify printing fidelity and post viability. Subsequently, posts were stained with DiI to facilitate long-duration tracking by incubation for 15 min at RT in the dark. After staining, ethanol was exchanged for PBS(-) *via* repeated and gentle rinsing. Substrates were blocked against non-specific binding by submerging in 0.2% w/v F-127 in PBS(-) and incubating 30 min at RT. After blocking, F-127 was exchanged for PBS(-) by repeated and gentle rinsing with running buffer. Chambers were pre-warmed to 37 °C in a cabinet incubator before cell plating and imaging.

Calculation of Post Spring Constant

For a cantilever beam of uniform cross section carrying a load at its unconstrained terminus, the equation of the beam's elastic deformation curve is given by (Eq. 5.1):

$$F = \left(\frac{3EI}{L^3} \right) \delta \quad (\text{Eq. 5.1})$$

where F, E, I, L and δ are the force exerted at the unconstrained terminus, Young's modulus of the beam material, moment of inertia of the beam cross section, and the

resulting deflection respectively (16). The assumption being made is that the beam is subjected to small deflections which do not cause plastic deformation. The terms within the parenthesis of Eq. 5.1 are collectively referred to as the post spring constant (k_{spring} , Eq. 5.2):

$$k_{spring} = \frac{3EI}{L^3} \quad (\text{Eq. 5.2})$$

Since our fabrication protocol is identical to Yang et al. (13) we employ their measured Young's modulus for PDMS cured 20 hr at 110 °C of $E = 2.5 \pm 0.5$ MPa. The moment of inertia I for a circular cross section is (Eq. 5.3):

$$I = \frac{\pi d^4}{64} \quad (\text{Eq. 5.3})$$

Where d is the diameter of the post. Substituting (Eq. 5.3) into (Eq. 5.2) yields (Eq. 5.4):

$$k_{spring} = \frac{3E\pi d^4}{64L^3} \quad (\text{Eq. 5.4})$$

Using scanning electron microscopy we captured a series of micrographs and measured the post diameter, $d = 604 \pm 31$ nm ($m \pm sd$), and length, $L = 5.576 \pm 0.286$ μ m ($m \pm sd$). Using error propagation (Eq. 5.5) we computed the mean and standard deviation ($\sigma_{k_{spring}}$, Eq. 5.5) of our empirical spring constant as $k_{spring} = 0.28 \pm 0.09$ pN/nm ($m \pm sd$).

$$\sigma_{k_{spring}} = \sqrt{\left(\frac{\partial k_{spring}}{\partial E}\right)^2 \sigma_E^2 + \left(\frac{\partial k_{spring}}{\partial d}\right)^2 \sigma_d^2 + \left(\frac{\partial k_{spring}}{\partial L}\right)^2 \sigma_L^2} \quad (\text{Eq. 5.5})$$

Work by Schoen et al. (14) demonstrated that in low aspect ratio posts (i.e. posts short compared to their width) substrate warping at the base was a substantial contribution to the observed deflection. The authors constructed a table of correction

factors to reduce the apparent spring constant as a function of post aspect ratio. For our posts with $L/d = 9.2$ the interpolated Schoen et al. reducing factor (assuming Poisson ratio of 0.5) is 7.8%. This correction is less than the propagated error in our empirical spring constant calculation and so we anticipate substrate warping is not a major contribution to the observed deflections in this study.

Neutrophil Isolation

Blood was collected with University of Pennsylvania Institutional Review Board approval from consenting adult volunteers. Cells were isolated as previously described (17). Whole blood was obtained from human donors *via* venipuncture and collected in sodium heparin Vacutainers (BD Biosciences). Volunteers were required to be in good health and abstain from alcohol and all over-the-counter medication for 24 hrs prior to donation. Blood samples were allowed to cool to RT for 15 min and layered in a 1:1 ratio of whole blood to Polymorphprep (Axis-Shield, Oslo, Norway). Vials were spun for 45-60 min at $550-650 \times g$ and $21 \text{ }^\circ\text{C}$. After separation, the polymorphonuclear band and underlying separation media layer were aspirated into fresh round-bottom tubes. The isolated solution of cells and separation-media was diluted with rinsing buffer and spun for 10 min at $250 \times g$ and $21 \text{ }^\circ\text{C}$. Red Blood Cells (RBC) were eliminated from the resulting cell pellet *via* hypotonic lysis. After lysis, vials were centrifuged for 10 min at $250 \times g$ and $21 \text{ }^\circ\text{C}$ and the RBC-free pellets resuspended in storage buffer. Neutrophils were stored at 10^6 cells/mL on a tube rotisserie at $4 \text{ }^\circ\text{C}$ until time of plating.

Spreading Experiments

Brightfield and fluorescence microscopy were performed using a spinning disk confocal. Prior to cell plating, the experimental chamber was mounted on a $37 \text{ }^\circ\text{C}$

temperature controlled stage. Images were acquired with a 60X water-immersion lens at a frame rate of 1 frame/sec. Acquisition began prior to cell plating. A small volume of suspended neutrophils were introduced into the experimental chamber and allowed to gravity-sediment onto the FN printed mPADs.

Antibody Blocking and Cytoskeletal Inhibitor Studies

To assess the role of β_2 integrins in neutrophil spreading and adhesion on post arrays, quiescent neutrophils were incubated for 30 min at 4 °C on a tube inverter with anti- β_2 clone L130 (BD Biosciences) at 50 $\mu\text{g}/\text{mL}$. This clone and concentration were previously shown by us (17) and others (18) to be a function blocking antibody of neutrophil adhesion on FN. To assess the roles of various cytoskeletal components during spreading, quiescent neutrophils were incubated for 30 min at 4 °C on a tube inverter with the small molecule inhibitor at the stated final concentration. The corresponding experimental chamber was pretreated at 37 °C for 30 min with the same inhibitor concentration. The small molecule inhibitors explored, having previously been demonstrated to alter hematopoietic cell mechanics, were: 5 μM blebbistatin (Sigma) (19), 1 μM CK666 (Sigma, Lot: 043M4606V) (20), 3 μM cytochalasin B (Sigma) (21), 1 μM jasplakinolide (Life Technologies) (19, 22), and 1 μM Y27632 (EMD Millipore, Billerica, MA) (23).

Cell Profile Imaging

To map the neutrophil vertical profile during spreading, cell membranes were stained with the lipophilic dye delta9-DiI (DiI) at a final concentration of 50 ng/mL for 15 min on a tube inverter at 4 °C. Cells were rinsed twice with fresh storage buffer by gentle centrifugation at 200 x g for 5 min. Z-slices were acquired at 0.25 μm intervals.

For membrane staining experiments, posts were labeled with AlexaFluor-488 conjugated FN (FN-AF488) only, not DiI.

Data Analysis

Fluorescent image stacks focused on the plane of post tips were processed *via* a series of custom MATLAB (The MathWorks, Natick, MA) scripts. These scripts identified fluorescently labeled post centroids, connected centroids in consecutive frames to form trajectories, dedrifted the trajectories, and positioned them relative to their undeflected resting lattice locations. Aspects of our scripts were adapted from the publicly available MATLAB routines (24) of Pelletier et al. (25) which were based upon Crocker and Grier's original particle tracking code (26). Annotated code used in the analysis of this chapter is reported in Appendix B.

Identification of Cell-Engaged Posts

After trajectory dedrifted, constructing a scatter plot of the variances in the tangential and radial directions reveals two populations of trajectories. A compact cloud of data with low variance corresponds to the posts in the field of view outside of the cell-substrate contact zone. The remaining posts correspond to those within the cell-substrate contact zone and are considered "cell-engaged".

Cell Reference Frame Coordinate System

The strong directional bias of peripheral posts (Fig. 5.1 *D*) towards the cell centroid motivated us to translate post trajectories from a laboratory reference frame $(x(t), y(t))$ into a cell reference frame $(r_{\parallel}(t), r_{\perp}(t))$. For each post a vector connecting the geometric centroid and the resting lattice position of that post was constructed. This

vector was used to position the orthogonal ($r_{\parallel}(t)$, $r_{\perp}(t)$) pair such that the radial axis was parallel with the connecting vector.

Computing a Spreading Velocity

For each cell we constructed a scatter plot of the time at which a post's maximum protrusive force was observed (relative to the first protrusive event which denoted the onset of spreading) as a function of the radial distance of the post from the cell centroid. A best fit linear equation was computed, subject to the constraint $t_{F_{\max}}(r) = 0$. The inverse of the slope of the curve was the propagation velocity. The mean velocity quoted in Fig. 5.5 C) is the mean and standard error of the ensemble of 14 spreading velocities acquired in this manner.

Results and Discussion

Neutrophil Spreading on mPADs

Quiescent neutrophils were capable of spreading atop a plane of FN printed post tips. The onset of spreading was concomitant with strong outward deflections observed at a few posts in the center of the final contact zone and propagated in a radially symmetric wave until the cell's final and maximum spread area was reached (Fig. 5.1 B). This transient protrusive signature was replaced by a sustained contractile phase a few minutes after spreading ceased. The complete spreading sequence with superimposed deflection vectors of Fig. 5.1 B) is provided in Movie S1. Post positions were tracked in the fluorescence channel as cell lensing obscured tip detection under brightfield microscopy. Cell-engaged posts experienced significant deflections compared to their non-engaged counterparts (Fig. 5.1 C). This fact was exploited to filter cell-engaged from non-engaged posts in the field of view by considering the variance of the trajectories (Fig. 5.2). The

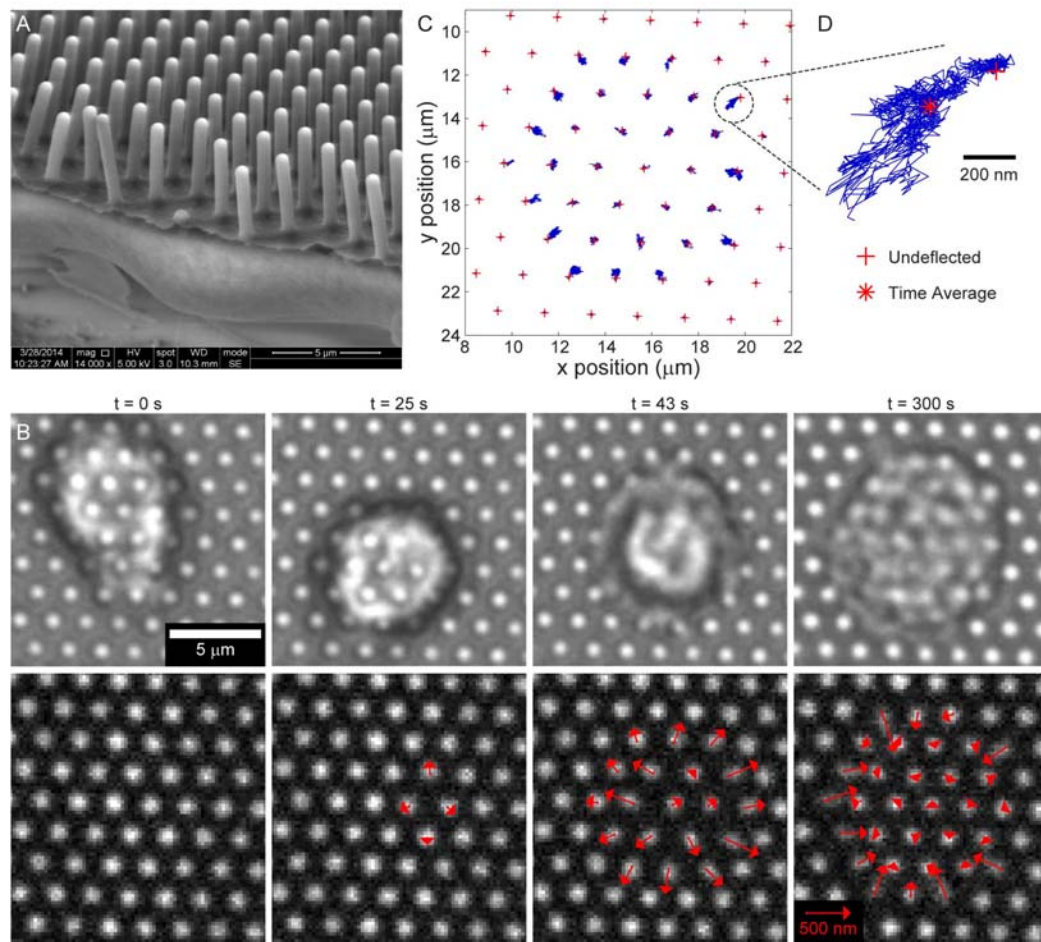


Figure 5.1 Human neutrophil spreading on fibronectin printed mPADs. (A) SEM image of mPADs used in this study. (B, *top row*) Brightfield frames from time-lapse sequence of a single neutrophil rapidly spreading across an array of posts. (B, *bottom row*) Corresponding frames in fluorescence channel of post tips with superimposed deflection vectors (enlarged 5X to aid visualization). Frames were taken from the full time-lapse sequence provided in Movie S1. (C) Trajectories of each post in (B) as recorded for 25 min. Red crosshairs denote the resting lattice position of the undeflected posts. The dotted circle is the enlarged post trajectory of panel (D), where the red asterisk marks the time average position of the trajectory.

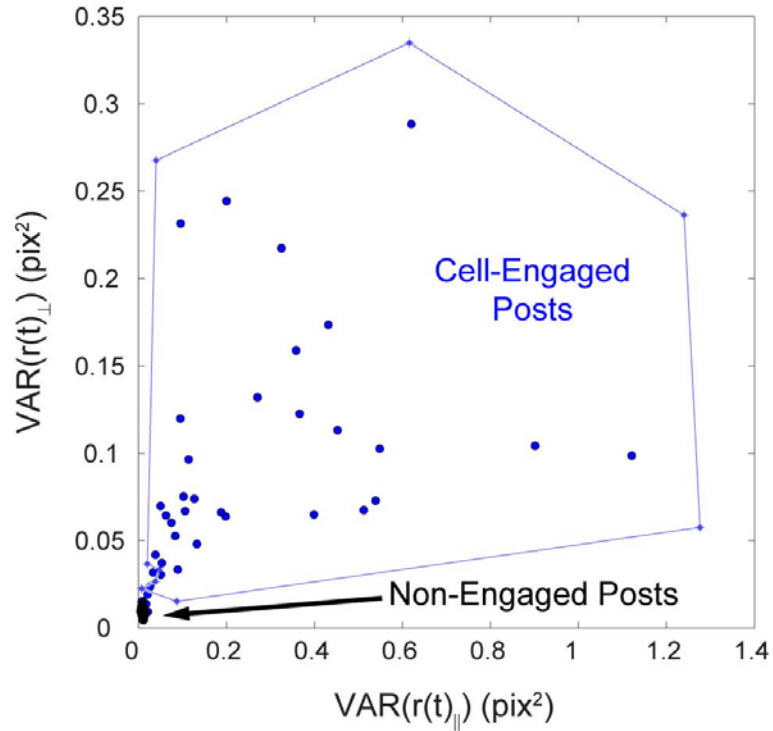


Figure 5.2 Identification of cell-engaged posts by variance analysis. Scatter plot of post trajectory variances in the tangential (\perp) and radial (\parallel) directions for data corresponding to Fig. 5.1 and Fig. 5.4. The compact cloud of posts with low variance corresponds to posts outside the cell-substrate contact zone. The remaining diffuse cloud is declared “cell-engaged” and used in data analysis. 1 pixel = 0.192 μm .

enlargement of a single perimeter post (Fig. 5.1 *D*) reveals a strong radial bias in the post's motion away from and towards the center of the cell's final contact zone.

From post deflections, we quantified force trajectories in the cell reference frame in the radial and tangential directions (Fig. 5.3). For each post, a force trajectory was constructed by multiplying the deflection from resting lattice position with the known spring constant of the posts ($k_{\text{spring}} = 0.28 \pm 0.09$ pN/nm). The force detection floor for our system was 9 ± 2 pN as determined by calculation of the mean displacement of posts not contacted by the cell and subsequent application of the spring constant. At maximum cell-generated protrusion and contraction this detection threshold resulted in signal-to-noise ratios of 8:1 and 12:1 respectively.

When we compared an ensemble plot of the radial force of each post between the periphery and the core as a function of time (gray lines, Fig. 5.4 *B*), a clear stratification of the data occurred. By mapping the deflection trajectories of posts within the top (low contractility) and bottom (high contractility) bands to the spatial position of the posts in the contact zone two groups of posts emerged. Perimeter posts were generally strongly contractile at long times as compared to core posts. However, both sets exhibited a strong transient protrusive spike. The ensemble averages of Fig. 5.4 *B* show two major mechanical regimes: initial transient protrusion and long-time sustained contraction. While tangential deflections were present throughout the experiment, no net asymmetry in the form of cell rotation or twist was observed in either perimeter or core posts (Fig. 5.4 *C*).

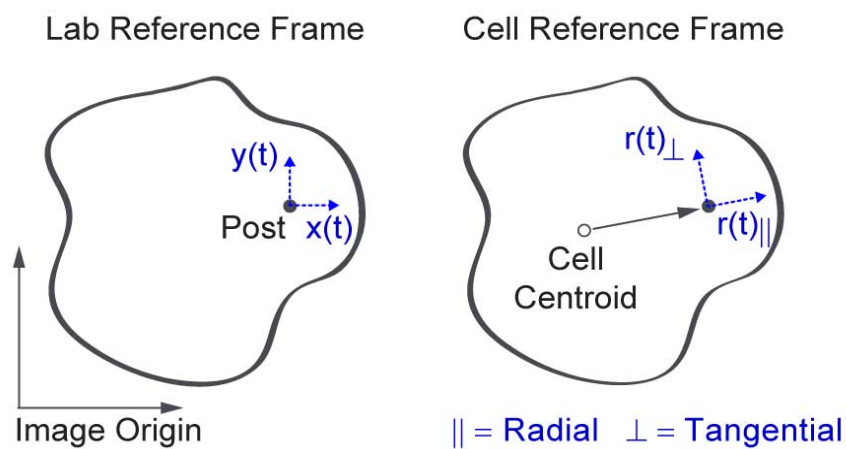


Figure 5.3 Cell reference frame coordinate transformation schematic. In the lab reference frame post trajectories are positioned relative to the field-of-view origin. A cell reference frame is more intuitive and constructed by translating post trajectories in terms of the orthogonal axes ($r_{||}(t)$, $r_{\perp}(t)$) such that the radial axis is parallel to a vector connecting the geometric centroid of the cell and the resting lattice position of the post.

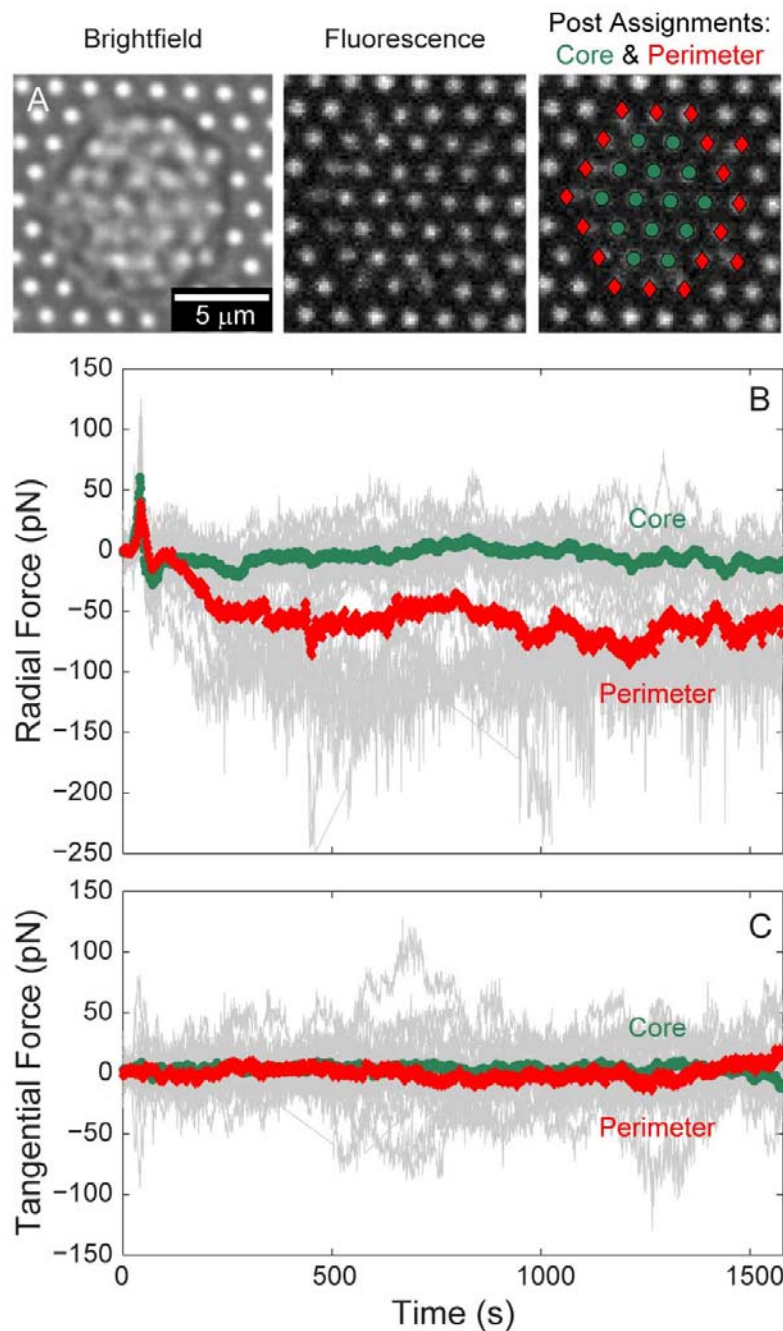


Figure 5.4 Spatial dichotomization of force trajectories. (A) Brightfield and fluorescence channel frames at $t = 300$ s from Fig 5.1 B. Green circles and red diamonds denote the subset of posts residing in the core and at the perimeter of the contact zone respectively. (B) Radial force trajectories over time. (C) Tangential force trajectories over time. In (C) and (D) individual gray lines correspond to individual cell-engaged posts. Ensemble averages of the subset of perimeter (red diamonds) and core (green circles) posts are superimposed.

Metrics of Spreading and Contractility

The behavior of the single spreading neutrophil illustrated in Fig. 5.1, Fig. 5.4, and Movie S1 is representative of our entire set of observations of spreading under control conditions ($n = 14$ cells, 4 different donors, 386 post trajectories) as shown in Fig. 5.5 A. Whereas in Fig. 5.4 B the mean curves were of the ensemble of posts beneath a single cell, in Fig. 5.5 A the mean curves are of the ensemble of all mean trajectories for our entire set of 14 spreading cells. To achieve this mean-of-means, the independent mean radial trajectories were aligned on their respective protrusive maxima and assigned the elapsed event time $\tau = 0$.

The qualitative and quantitative similarity of the protrusive event for core and perimeter posts is evident in the expanded view of Fig. 5.5 B in which the forcefulness and duration of the protrusive events are similar. The protrusive event was immediately followed by a contractile rebound. Outwardly deflected posts did not settle back to their resting lattice position but were summarily deflected inwards. In the core of the cell, the rebound resulted in a transient contractile maximum that relaxed to a less contractile steady state. However, in the perimeter, the posts continuously deflected to a steady state contractile maximum.

To better capture the wave-like propagation of the protrusive front during spreading we plotted the time at which protrusive force was a maximum as a function of the radial distance of the protrusive event from the cell centroid for each cell and fit the data with a linear equation. The inverse of the best-fit slope was the cell's spreading velocity. Fig. 5.5 C shows the ensemble best-fit equation for all spreading events (all per-

cell fits are reported in Fig. 5.6). Using this analysis we computed a mean neutrophil spreading velocity of 206 ± 28 nm/s ($m \pm \text{sem}$).

We considered a variety of metrics to characterize the radial forces during the transient protrusive (Fig. 5.5 *D*) and steady state contractile (Fig. 5.5 *E*) regimes. Consistent with our qualitative observations, the protrusive signatures of core and perimeter posts were not significantly different with respect to the maximum force generated (~ 75 pN) (Fig. 5.5 *D i*), duration of the protrusive deflection (FWHM ~ 17 s) (Fig. 5.5 *D ii*) or the variance in the ensemble of maximum forces (~ 24 pN²) (Fig. 5.5 *D iii*). We did however find a significant decrease in the fraction of perimeter posts (perim: 0.67 ± 0.05) that exhibited a protrusive spike as compared to the fraction of core posts (core: 0.83 ± 0.05) (Fig. 5.5 *D iv*). Thus, during spreading, when a post was protrusively engaged by the cell, the basic dynamic form of the deflection did not depend on whether the post was at the core or the periphery. However, as distance from the cell centroid increased the occurrence of protrusion decreased.

Within the steady state contractile regime we found significant differences in core and perimeter posts with respect to the sustained contractile force (core: -20 ± 10 vs. perim: -106 ± 10 pN/post) and its variance (core: 16 ± 4 vs. perim: 46 ± 4 pN²/post). Perimeter posts were five times more contractile (Fig. 5.5 *E i*) and had three times greater variability (i.e. larger distributions in force) in their sustained contractility (Fig. 5.5 *E ii*) compared to their core counterparts. Our observation that spread neutrophils were most contractile at their periphery compliments the RICM measurements of spreading neutrophils on FN by Sengupta et al. (5). In that prior work, the region of intimate membrane-substrate contact was located at the periphery of the spreading neutrophil. It

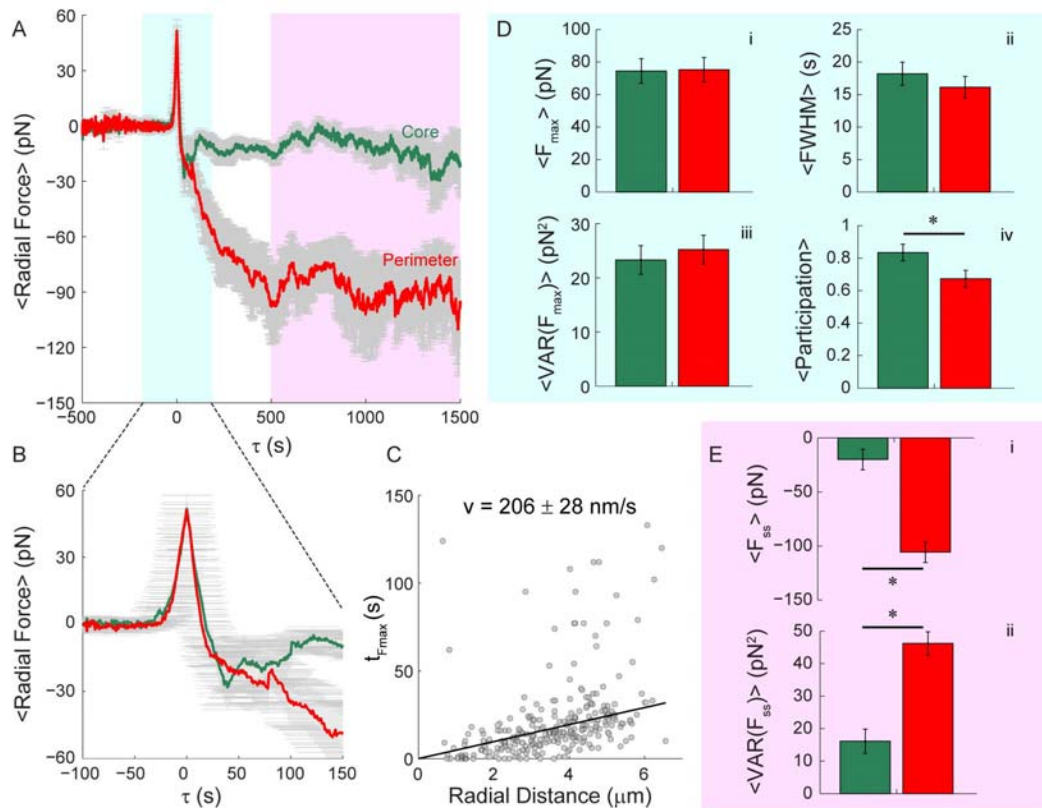


Figure 5.5 Characterizing protrusion and contraction using the ensemble of neutrophil spreading events. (A) Mean radial force trajectories of core (green) and perimeter (red). The transient protrusive and steady state contractile regimes are denoted by the cyan and lavender shaded regions respectively. (B) An expanded temporal resolution of the protrusive regime in A. (C) The time at which protrusive force is maximal as a function of radial distance from the cell centroid. Per cell fits are shown in Fig. 5.6. (D) Mean metrics of transient protrusion: (i) force maximum, (ii) spreading duration *via* full width at half force maximum, (iii) variance in the ensemble of force maxima, and (iv) the fraction of posts in each geometric group that exhibited a protrusive spike (i.e. the participation ratio). (E) Mean metrics of steady state contraction: (i) force, (ii) variance in the ensemble of mean steady state force. All error bars are \pm standard error of the mean ($n = 14$ cells). Asterisk denotes significant difference between populations as computed by post-hoc Tukey least significant difference method ($p < 0.05$).

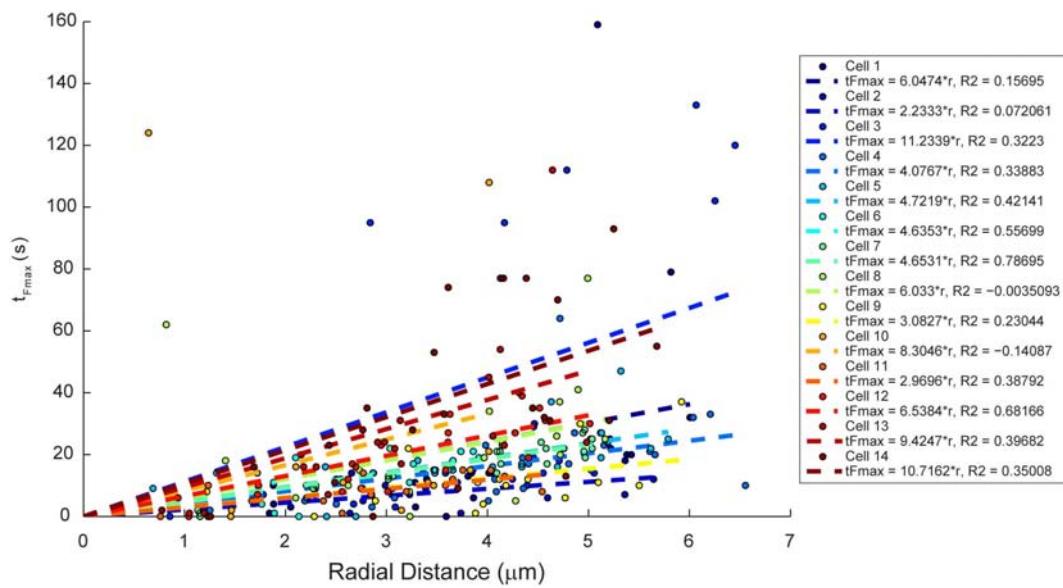


Figure 5.6 Per-cell spreading velocity analysis. Scatter plots for 14 spreading cells of the time at which a post's maximum protrusive force occurred as a function of the post's radial distance from the cell centroid.

was hypothesized there, and experimentally demonstrated here, that those regions of intimate membrane-substrate contact are concurrently regions of greatest force generation.

Contrasting our work with Bashour et al. (12), we see greater protrusive and contractile behavior of spreading neutrophils as compared to T-lymphocytes. Spreading neutrophils were approximately six fold more protrusive and two fold more contractile than activated T-lymphocytes. Bashour and coworkers describe a transient regime between spreading and steady state contraction in their data in which T lymphocyte tractions were highly uncoordinated. In our data, we do not see a latent period of uncoordinated traction. Rather, we observe outward protrusion immediately followed by an inward contractile rebound. At the perimeter, this rebound evolves into a highly contractile steady state. It is important to note that the Bashour et al. work was considering the mechanics associated with T-lymphocyte activation through the CD3 T-cell receptor (TCR) and the CD28 coreceptor. Ligation of these receptors induces cytoskeletal rearrangement but is upstream of integrin activation, representing an inside-out pathway. While the Bashour et al. inside-out activation route shares certain scaffolding proteins (e.g. SLP-76) with the outside-in activation route we are engaging in neutrophils, the pathways are not identical (27).

Biochemical Perturbations of the Cell Cytoskeleton

To study the role of the cytoskeleton during neutrophil spreading on post arrays we pretreated quiescent cells with small molecule inhibitors targeting various cytoskeletal components. Actin in a quiescent neutrophil is confined to a thin cortical shell proximal to the cytoplasmic membrane (28). It has been demonstrated that this actin shell gives

rise to cortical tension (21-22). We began by considering the effect of jasplakinolide on neutrophil spreading. Jasplakinolide is a cyclic depsipeptide capable of polymerizing and stabilizing filamentous actin (29). In neutrophils, pretreatment with jasplakinolide has been shown to increase the rigidity of the cortex as measured by micropipette aspiration (22). When we treated quiescent neutrophils with jasplakinolide, the ability of the cells to spread was completely eliminated (Movie S2). Interestingly, the cells were still sensing the presence of the FN as detected by the formation of small processes uniformly decorating the cell body seen with brightfield imaging. These processes were never observed in untreated control cells. It is unclear whether the effect of jasplakinolide in our cells was to stabilize existing F-actin structure or deplete a pool of free actin by polymerizing excess F-actin.

Unlike jasplakinolide, cytochalasin B has been shown to decrease cortical rigidity in neutrophils as measured by micropipette aspiration (21). Cytochalasin B is known to dramatically reduce the rate of actin polymerization and simultaneously interfere with filament-filament interactions that stabilize the actin network (30). When treated with cytochalasin B, our neutrophils were still able to spread but with a substantially reduced velocity of 61 ± 37 nm/s (Fig. 5.7 B). During spreading, the mean protrusive force exerted per post was not significantly different than observed with untreated cells. However, the duration of the protrusive event was longer as seen by a significant increase in the full width at half max force metric (Fig. 5.7 C, $\langle FWHM \rangle$). Inhibition of actin polymerization and filament-filament interaction by cytochalasin B had long time effects as well, significantly decreasing the achieved steady state contractile force of perimeter posts (Fig. 5.7 C, $\langle F_{ss} \rangle$) and eliminating the contractile rebound of core posts (Fig. 5.7

A, Cytochalasin B). Considered in the context of the results with jasplakinolide, spreading requires relaxation of the actin cortical shell.

We next considered whether spreading was conceptually analogous to lamellipodium formation by inhibiting Arp2/3, the actin binding protein necessary for filament branching (31). CK666 inhibits Arp2/3 mediated branching by stabilizing the inactive conformation of the seven subunit complex (32). CK666 had no effect on the protrusive capacity of the spreading cells. These cells were not significantly different in the forcefulness or duration of protrusion than their untreated counterparts. That CK666 did not abrogate protrusion suggests the shape change associated with spreading was not analogous to lamellipodium formation, in which Arp2/3 is known to play a critical role (31). We did observe a significant increase in the variance of the forces exerted on core posts during steady state contractility (Fig. 5.7 C, $\langle \text{VAR}(F_{ss}) \rangle$). This result suggests that a competent actin network may normally dampen post contractility in the core.

Lastly, we hypothesized that steady state contractility would be ROCK and myosin II mediated (19) and tested this by treating neutrophils with Y27632 and blebbistatin (33), respectively. In both cases these inhibitors significantly reduced steady state contractility (Fig. 5.7 C, $\langle F_{ss} \rangle$) of perimeter posts but did not eliminate the contractile rebound following protrusion (Fig. 5.7 A, *Y27632 and Blebbistatin*). In untreated neutrophils, this contractile rebound was only observed in the ensemble of core posts. Treating with Y27632 and blebbistatin revealed that the transient rebound was also occurring in the perimeter posts but was obscured when ROCK and myosin II mediated contractility commenced. Thus the transient contractile rebound is a feature of both core and perimeter posts but masked by long time engagement of the actomyosin-mediated

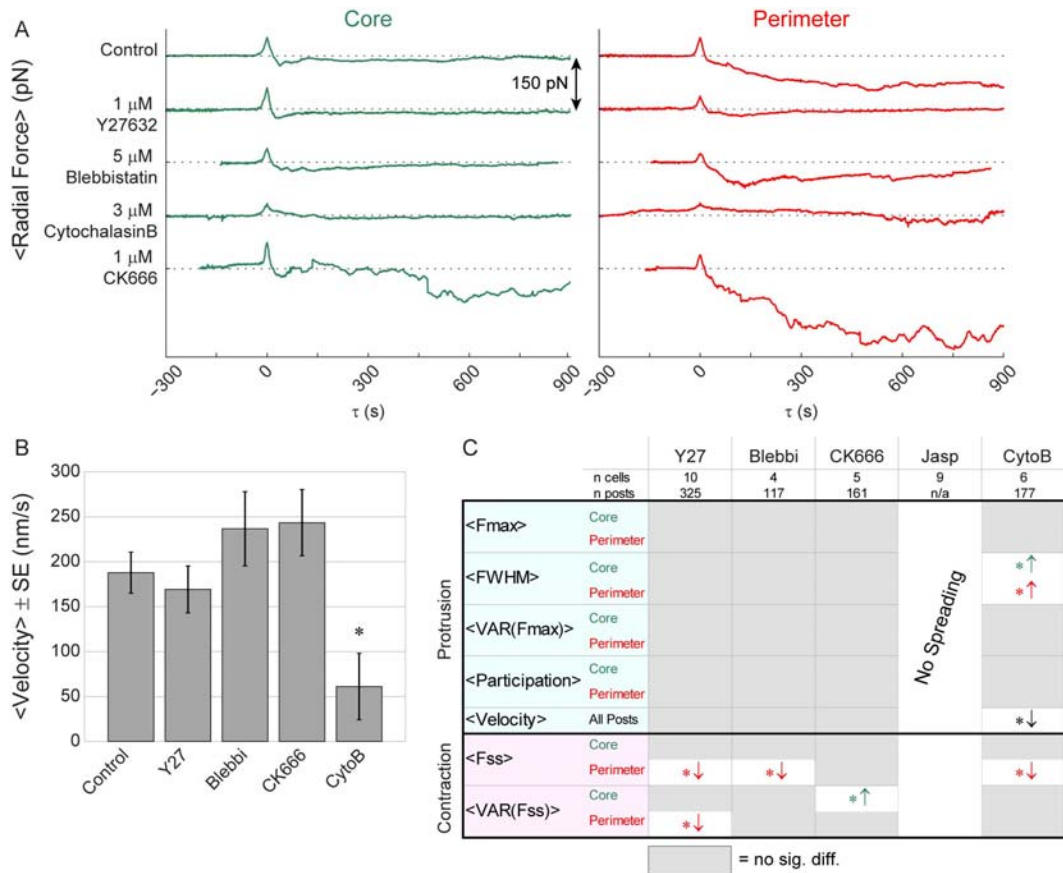


Figure 5.7 Cytoskeletal perturbation via small molecule inhibitors. (A) Mean radial force trajectories of the ensemble of individual cell spreading events observed after 30 min pretreatment with the stated inhibitor. Trajectories were plotted at 150 pN intervals. (B) Effect of inhibitors on spreading velocity. (C) Effect of inhibitors on metrics of protrusion (cyan shading) and contraction (lavender shading). Asterisk denotes significant difference relative to control computed by post-hoc Tukey-Kramer multiple comparisons method ($p < 0.05$). Direction of arrow indicates the direction in which the inhibitor shifted the metric relative to the control, if a significant difference was found.

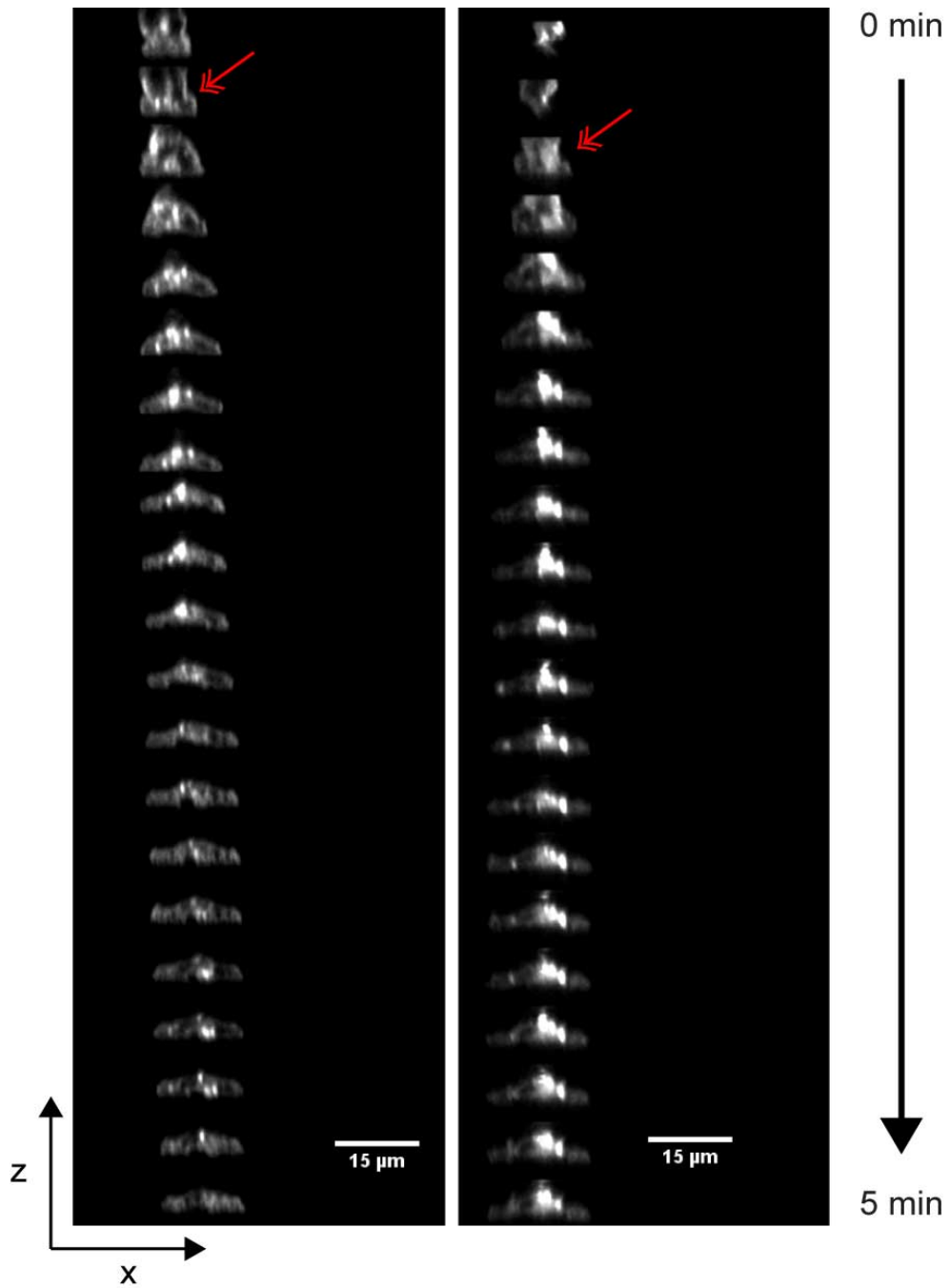


Figure 5.8 XZ kymograph of neutrophil spreading on stiff mPADs. Two representative spreading neutrophils on stiff FN-printed mPADs. Each frame is a vertical (XZ) contour of a spreading neutrophil labelled with the membrane dye DiI. Frames were captured every 15 sec. Double-headed arrows denote observation of necking region.

contractile apparatus at the cell periphery. The implication of this result is that the short time transient rebound is not actomyosin-dependent.

Spreading is Haptokinetically Induced

Neutrophil spreading is induced by haptokinetic interaction with the printed FN. On the soft post arrays ($G \sim 5$ kPa) used in our traction measurements, cells assumed a sessile drop morphology (Fig. 5.9 A *iii*) as captured by spinning disk confocal microscopy z-stacks. The presence of the FN was critical in supporting the transition from a quiescent to spread phenotype. When posts are blocked with Pluronic but not printed (Fig. 5.9 A *i*), the cells remained spherical and there was no non-specific adhesion. Additionally, integrin ligation by FN was required upstream of spreading, since pre-treating quiescent neutrophils with an antibody against β_2 impeded spreading (Fig. 5.9 A *ii*). Haptokinetically-induced neutrophil spreading *via* β_2 integrins is consistent with our published observation that a portion of quiescent neutrophils could be induced to migrate on continuous fields of FN without concurrent or prior stimulation by chemoattractant or selectin-ligation and that this adhesion was mediated by the promiscuous integrin MAC-1 ($\alpha_M\beta_2$) (17).

We hypothesized that the vertical profile of neutrophils on post-arrays had a stiffness dependence and considered the cell shape when spreading on stiff arrays ($G \sim 42$ kPa) and extremely stiff, flat PDMS ($G \sim 833$ kPa). On stiff posts the height (i.e. z-extent) of the cell was reduced (Fig. 5.9 A *iv*) compared to that observed on flat PDMS printed with continuous fields of FN (Fig. 5.9 A *v*). Using Fiji (34), we fit ellipses to the vertical profiles and computed the aspect ratio (i.e. ratio of the major axis length to minor axis length). A clear monotonic trend was observed where aspect ratio of the cell

increased as stiffness increased (Fig. 5.9 B). The dependency of spread area and aspect ratio on discrete post arrays of increasing stiffness is analogous to that observed of neutrophils on continuous polyacrylamide gels of increasing stiffness (35-37). Thus, as established traction methodologies, PDMS post arrays and polyacrylamide gel systems are complementary tools in probing immune cell mechanobiology.

The FN-null and anti- β_2 controls had similar aspect ratios close to unity (unity denotes a perfect circle). Monotonic trends were also revealed in circularity, roundedness, and XY cell-substrate contact area as well (Fig. 5.10). In addition to XZ profile aspect ratio in Fig. 5.9 B, we used Fiji (34) to compute XZ circularity ($4*\pi*area/perimeter^2$, Fig. 5.10 A), roundedness ($1/aspect\ ratio$, Fig. 5.10 B), and XY contact area of the cell-substrate interface (Fig. 5.10 C). As substrate stiffness increases circularity and roundedness monotonically decreased indicating an increasing deviation from a perfect circle. Conversely as substrate stiffness increases the XY contact area monotonically increases. In all metrics FN-null and anti- β_2 conditions were indistinguishable.

These results demonstrate that in our system the FN is required for neutrophils to break quiescence and spread in a β_2 integrin dependent manner and that the extent of spreading increases as a function of underlying stiffness. We explored a larger range of stiffness than Bashour et al. which may explain why XY spread area increases as a function of stiffness in neutrophils but not in T lymphocytes. As a note, for posts we quote approximate shear moduli (G) computed under the assumption that the mode of cell deformation of the post is shear and exerted over its cross sectional area. This assumption is motivated by the empirical work of Lemmon et al. (7) which demonstrated that shear is a larger contribution to post deflection than torque. Alternatively, Ghibaudo

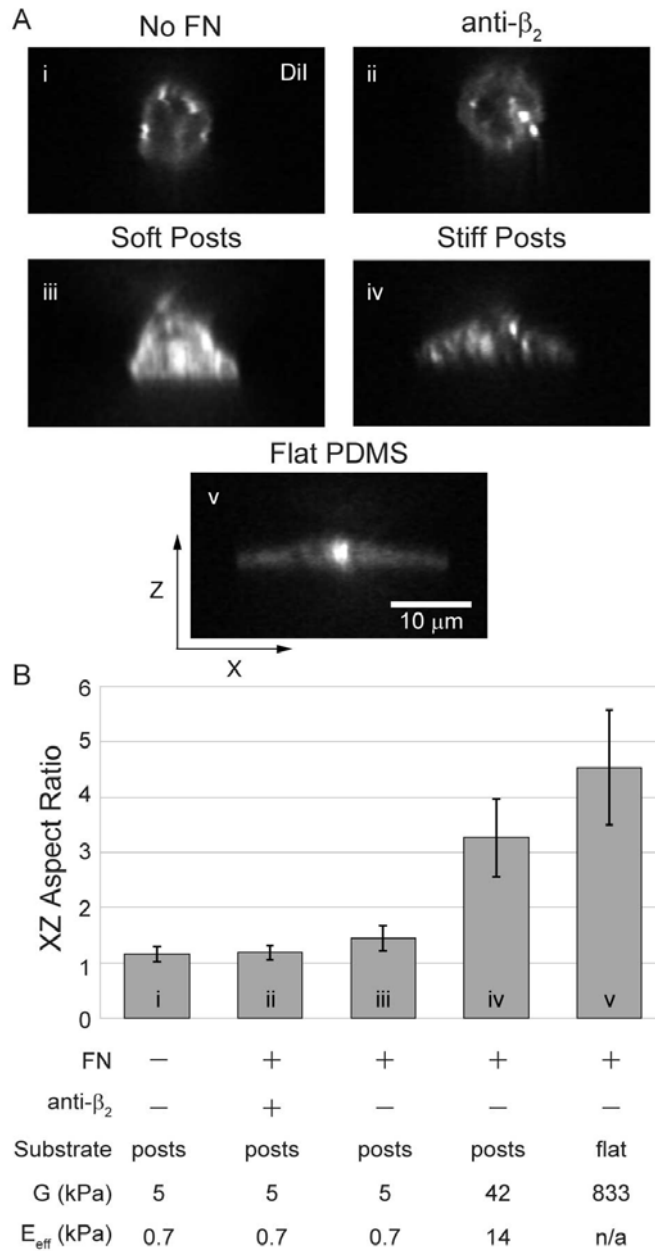


Figure 5.9 Vertical profiles of neutrophils imaged via confocal microscopy. (A) (i) Quiescent neutrophil on an array of posts blocked with Pluronic F-127, but not printed with FN. (ii) A neutrophil on soft FN posts, pre-treated with anti- β_2 integrin antibody. (iii) Spread neutrophil on soft FN posts used in traction mapping. (iv) Spread neutrophil on stiffer FN post arrays. (v) Highly spread neutrophil on extremely stiff, flat FN fields. (B) Aspect ratio of best-fit ellipses to neutrophil profiles. Error bars are \pm standard deviation ($n = 8-15$ cells per condition). Additional metrics showing similar monotonic trends are reported in Fig. 5.10.

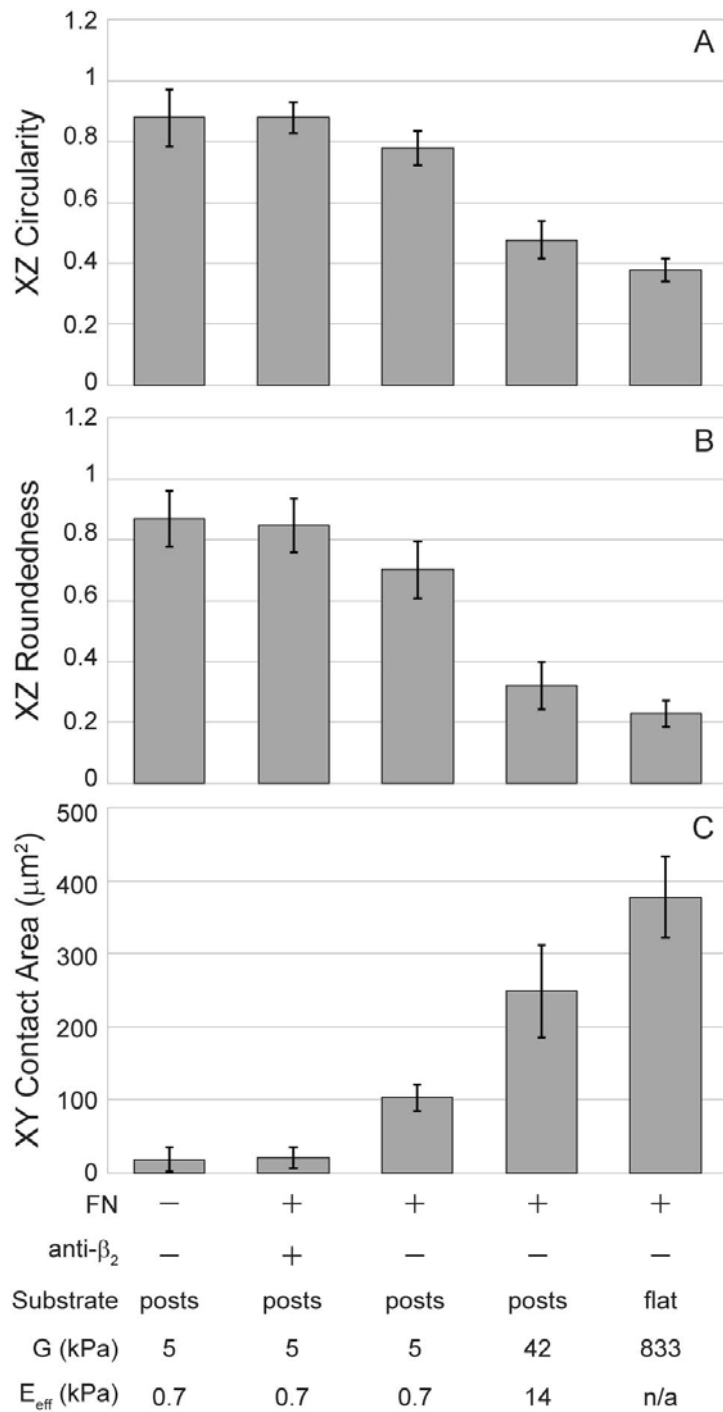


Figure 5.10 Metrics of XZ and XY cell profiles. (A) XZ circularity, (B) XZ roundedness, and (C) XY contact area. Error bars are \pm standard deviation ($n = 8-15$ cells per condition).

and coworkers developed a theoretical description of effective array stiffness by solution of the Green's function for a discretized substrate (38). The Ghibaudo model estimates the Young's moduli of post arrays as being substantially softer than anticipated by a local pure shear model (Fig. 5.9 B "*E_{eff}*").

Estimating Extent of Post Sidewall Printing

From our confocal z stacks we observe that neutrophils invaginate FN printed post arrays to the limit of the post sidewall printing (Fig. 5.13 A). The following series of calculations were used to estimate the extent of this sidewall printing and thus depth of invagination. An apparent image (I) is the convolution of the object's intensity profile (F) with the optical system's airy disc (G) (Eq. 5.6) (39).

$$I = F \otimes G \quad (\text{Eq. 5.6})$$

Assuming the object's intensity profile and the optical airy disc are reasonably approximated as Gaussian distributions, the convolution of two Gaussians produces a variance (σ^2) that is the sum of the variances (Eq. 5.7).

$$\sigma_I^2 = \sigma_F^2 + \sigma_G^2 \quad (\text{Eq. 5.7})$$

Using green fluorescent beads (Molecular Probe FluoSpheres, Catalog: F8813, Lot: 1600255) of known size (diameter = $0.49 \pm 0.015 \mu\text{m}$) we acquired XZ intensity profiles on the spinning disc confocal in the same channel and at the same magnification as our post measurements (Fig. 5.11 B). We normalized each bead intensity profile so the area under the intensity curve equaled unity and the peak of the intensity curve was translated to reside at $x = 0$. Next, each normalized bead intensity curve was fit with a two-parameter Gaussian distribution where mean (μ) and standard deviation (σ) were

free parameters (Fig. 5.11 D). The mean standard deviation of ten beads was 0.94 ± 0.09 μm ($m \pm \text{sd}$).

To estimate the variance of the Gaussian-approximated optical airy disc we must make an assumption about the unconvolved intensity profile of the fluorescent bead. Let the unconvolved intensity profile of the fluorescent bead have a full width at half maximum (FWHM) equal to the known bead diameter. For a Gaussian distribution, FWHM is related to the standard deviation (σ) (Eq. 5.8) *via*:

$$FWHM_{bead} = 2\sqrt{2 \ln(2)}\sigma_{bead} \quad (\text{Eq. 5.8})$$

Rearranging for σ_{bead} and substituting $FWHM_{bead} = 0.49 \mu\text{m}$ results in (Eq. 5.9):

$$\sigma_{bead} = \frac{0.49}{2\sqrt{2 \ln(2)}} \sim 0.21 \mu\text{m} \quad (\text{Eq. 5.9})$$

Solving Eq. 5.7 for the Gaussian-approximation to the confocal's optical airy disc yields (Eq. 5.10):

$$\begin{aligned} \sigma_{optics}^2 &= \sigma_{image}^2 - \sigma_{bead}^2 \\ \sigma_{optics}^2 &= (0.94 \mu\text{m})^2 - (0.21 \mu\text{m})^2 \\ \sigma_{optics}^2 &\sim 0.8403 \mu\text{m}^2 \end{aligned} \quad (\text{Eq. 5.10})$$

Having approximated the contribution of the optical airy disc to the blur in the XZ intensity profile of fluorescent beads of known size, we can now quantify the apparent intensity profile of the printed post arrays and calculate an estimate of the actual extent of sidewall printing. A set of post arrays, printed with FN-AlexaFluor488 in a manner identical to those used in cell spreading experiments, was imaged in an aqueous solution of 90% glycerol. Glycerol was employed to bring the aqueous refractive index closer to that of cured PDMS (Fig. 5.12 A). Next, each normalized post intensity curve was fit with

a two-parameter Gaussian distribution where mean (μ) and standard deviation (σ) were free parameters (Fig. 5.12 C). The mean standard deviation of ten printed posts was $1.00 \pm 0.10 \mu\text{m}$ ($m \pm \text{sd}$). Solving Eq. 5.7 for the variance of the Gaussian-approximation to the true intensity profile of printed posts yields (Eq. 5.11):

$$\begin{aligned}\sigma_{post}^2 &= \sigma_{image}^2 - \sigma_{optics}^2 \\ \sigma_{post}^2 &= (1.00 \mu\text{m})^2 - 0.8403 \mu\text{m}^2 \\ \sigma_{post}^2 &\sim 0.1597 \mu\text{m}^2\end{aligned}\tag{Eq. 5.11}$$

Lastly, we define the extent of sidewall printing as the FWHM of the unconvolved z-intensity profile (Eq. 12):

$$\begin{aligned}FWHM_{post} &= 2\sqrt{2\ln(2)}\sigma_{post} \\ FWHM_{post} &= 2\sqrt{2\ln(2) * 0.1597 \mu\text{m}^2} \\ FWHM_{post} &= 0.9410 \mu\text{m}\end{aligned}\tag{Eq. 5.12}$$

Thus, we conservatively estimate the extent of sidewall printing to be on the order of $1 \mu\text{m}$.

Estimating Energy of Neutrophil-FN Interaction

If cell wetting (i.e. FN ligation of cell surface receptors) alone drives the spherical-to-sessile drop shape change then the energy of this interaction must be sufficient to deform the known cortical tension of quiescent neutrophils. The following is an order of magnitude analysis to estimate the available binding energy of human neutrophils. The total MAC-1 availability of activated human neutrophils is on the order of $\sim 10^5$ receptors (40). Our antibody blocking experiments demonstrated that β_2 integrins were a major mediator of neutrophil-FN binding. From kinetic studies of β_2 integrin ligation, the energy liberated upon binding is known to be on the order of $\sim -10 \text{ k}_B\text{T}$ (41).

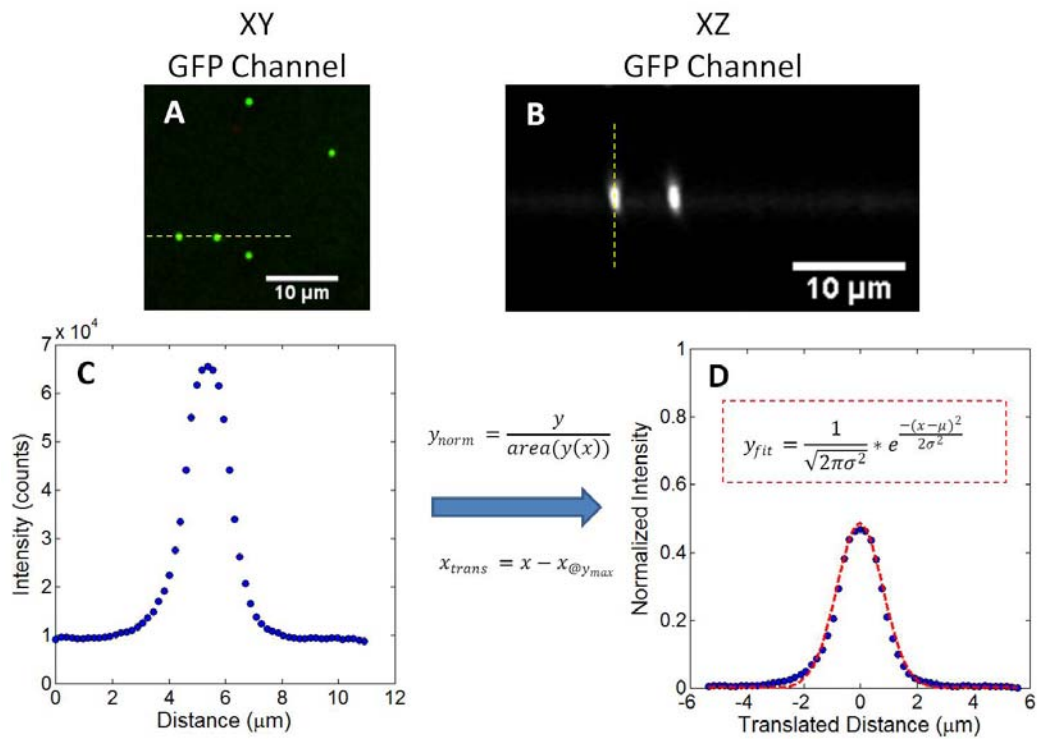


Figure 5.11 Confocal images of 0.49 μm diameter green fluorescent beads using the same magnification and acquisition settings as post arrays. (A) XY plan view of green fluorescent beads. (B) XZ profile view of green fluorescent beads denoted by yellow dotted line in A. (C) Raw bead intensity along yellow dotted line in B. (D) Normalized bead intensity so area beneath intensity curve equals unity and peak intensity occurs at x = 0. Red-dotted line is best-fit two parameter Gaussian curve.

XZ Profile of FN-AF488 printed mPADs in solution of 90% glycerol

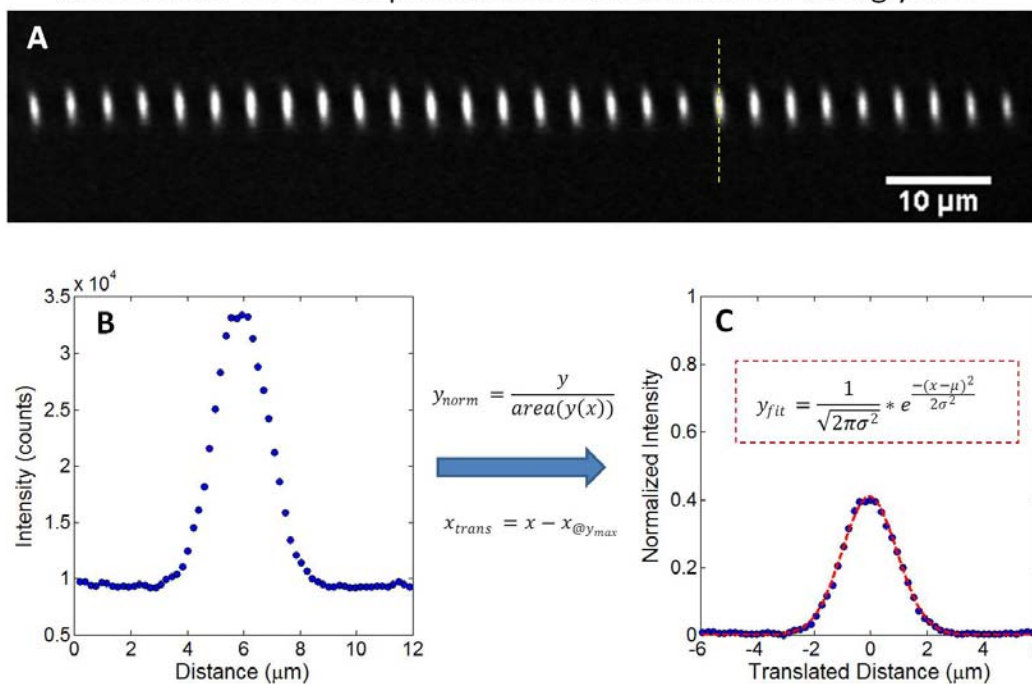


Figure 5.12 Confocal measurements of FN-AlexaFluor488 printed post arrays in a solution of 90% glycerol. (A) XZ profile view of printed posts. (B) Raw post intensity along yellow dotted line in A. (C) Normalized post intensity so area beneath intensity curve equals unity and peak intensity occurs at $x = 0$. Red-dotted line is best-fit two parameter Gaussian curve.

Assuming all MAC-1 is available for binding and FN binding sites are in excess of MAC-1 than an upper estimate on the liberated binding energy ($\gamma_{\text{cell-FN}}$) is on the order of $\sim -10^6$ $k_B T$. Assuming surface energy alone dictates cell shape we can apply Young's equation to relate the observed contact angle of the cell profile to the energy of cell-substrate interaction (Eq. 5.13):

$$0 = \gamma_{\text{cell-FN}} + \gamma_{\text{cell-PBS}} * \cos \theta \quad (\text{Eq. 5.13})$$

Note in Eq. 5.13 we implicitly assumed that the energy of substrate-aqueous (i.e. FN-PBS) interaction is insignificant ($\gamma_{\text{FN-PBS}} \sim 0$). Rearranging Eq. 5.13 and solving for the surface energy of the quiescent neutrophil ($\gamma_{\text{cell-PBS}}$) yields (Eq. 5.14):

$$\gamma_{\text{cell-PBS}} = \frac{-\gamma_{\text{cell-FN}}}{\cos \theta} \quad (\text{Eq. 5.14})$$

From z-stacks of fluorescently labeled neutrophils on FN we can measure the contact angle that the cell forms with the substrate. Contact angles from neutrophils on flat PDMS, microcontact printed with large continuous fields of FN, were used as this case represents the maximum binding energy available to the cell. For flat PDMS, $\theta = 15 \pm 2^\circ$ ($m \pm \text{sd}$, $n = 6$ cells). Substituting Eq. 5.14 for $\gamma_{\text{cell-FN}} \sim -10^6$ $k_B T$ and $\theta = 15^\circ$ yields $\gamma_{\text{cell-PBS}} > 10^6$ $k_B T$. The surface energy of the quiescent neutrophil is the cortical tension (T_{cort}) multiplied by the surface area SA (Eq. 5.15):

$$\gamma_{\text{cell-PBS}} = T_{\text{cort}} * SA \quad (\text{Eq. 5.15})$$

Modeling the spread neutrophil as a hemispherical cap and computing the lateral surface area yields $SA = 446 \pm 56 \mu\text{m}^2$. Substituting $\gamma_{\text{cell-PBS}} \sim 10^3$ $\text{pN}\cdot\mu\text{m}$ (1 $k_B T \sim 0.004114$ $\text{pN}\cdot\mu\text{m}$) and $SA = 446 \mu\text{m}^2$ into Eq. 5.15 yields $T_{\text{cort}} \sim 10$ $\text{pN}/\mu\text{m}$ which is within one order of magnitude of the measured cortical tension of quiescent neutrophils

(41). Our rough analysis suggests that the upper bound of available energy of the cell-FN interaction on flat PDMS is on the order of the surface energy associated with the resting neutrophil's cortical tension. However, the actual binding energy is likely lower on the discretized adhesive environment of the printed post arrays.

Furthermore, if the energy of cell-substrate binding alone were sufficient to explain the deformation we would have expected that reducing cortical tension and decreasing viscosity *via* cytochalasin B treatment would have increased the spreading velocity of neutrophils as was observed of cytochalasin D treated HeLa cells by Cuvelier et al (42). However, in the cytochalasin B case neutrophils spread slower than untreated control cells.

Origin of the Protrusive Signal

Simultaneous acquisition of the cell profile and plane of FN printed post tips revealed that neutrophils moderately invaginate the post arrays to a depth of approximately 1 μm (Fig. 5.13 A). Our prior experience with neutrophils on continuous fields of FN on PDMS blocked with Pluronic F-127 (17) and the absence of spreading in the present FN-null experiments suggests that invagination was a consequence of printing adhesive ligand on the post sidewalls. Sidewall printing may have resulted from using soft stamps to print the post arrays coupled with the fact that the post tips themselves were rounded.

During spreading, posts beneath the propagating cell front reported the forces associated with the cell's shape change from quiescence (spherical) to spread (sessile drop). This was facilitated by the fact that the cell was not spreading exclusively across the top of the plane of post tips but rather through a volume of finite thickness dictated by

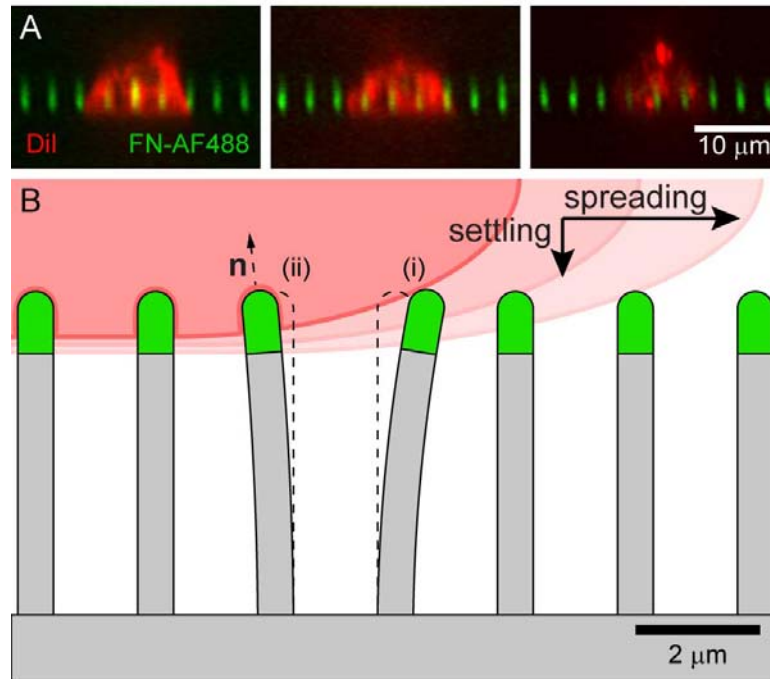


Figure 5.13 Post invagination as origin of protrusive signature. (A) Confocal XZ profiles of neutrophils, cytoplasmic-stained by DiI on FN-AF488 post arrays. Each field of view is a different neutrophil. (B) Schematic of (i) cell spreading through a finite volume of posts as driven by sidewall printing and (ii) a conjecture that the transient contractile rebound is driven by local membrane curvature where \mathbf{n} is a unit normal vector. Schematic is to scale. Extent of sidewall printing was estimated at 1 μm .

the extent of sidewall printing. The posts reported the force of shape change because they physically resided within the cell's spreading path. (Fig. 5.13 *B i*). Our inhibitor studies showed that ROCK and myosin II mediated contractility was not fully matured until approximately 500 s after peak protrusive force was generated. We know that FN was required for spreading as FN-null experiments did not induce shape change. Thus to claim that protrusion was the result of cell spreading across the plane of post tips but not through a finite volume suggests that integrin ligation of FN was responsible for the ~ 75 pN/post protrusive force at short times without mature connection to the actomyosin substructure which requires minutes to develop. If sidewall printing were not present we would have been unable to quantify the force associated with this transformation as connection of the mature actomyosin substructure to the integrin adhesive contacts at the cell-post interface requires minutes to develop.

The energy of the MAC-1/FN interaction was estimated to be within an order of magnitude of the energy necessary to achieve the spherical-to-sessile drop transformation resisted by the cortical tension of quiescent neutrophils. That adhesion energy alone was not in excess of the required deformation energy to achieve spreading suggests an additional mechanism was at play. Our jasplakinolide and cytochalasin B inhibitor studies point to cortical tension release as a possible biophysical mechanism neutrophils employ to permit adhesion driven spreading and invagination. Additionally, the observation of moderate post invagination suggests a possible explanation as to the origin of the transient contractile rebound present in untreated core posts and ROCK/myosin II inhibited perimeter posts. We hypothesize that this rebound results from the invaginated posts assuming a transient orientation normal to the cell membrane to minimize the

energy of the membrane-post interface (Fig. 5.13 *B ii*). Future experiments using time resolved superconfocal microscopy may be able to quantify the post tip orientation relative to the local membrane curvature during spreading. Additionally, future experiments using arrays with a sparse number of non-printed posts could shed light on the mechanical role of integrin ligation during protrusion.

Conclusions

As first responders to tissue trauma and infection, neutrophils are capable of fast and dramatic shape changes (3). In this work we studied the mechanics associated with a neutrophil's transition from a quiescent sphere to a spread and integrin-adherent morphology. *In vivo* spherical neutrophils circulate throughout the vasculature with their shape maintained by an actin cortical shell. Others have demonstrated, using micropipette aspiration, that this shell possesses a characteristic rigidity, tunable by small molecule inhibitors of actin polymerization (22) and depolymerization (21). By observing neutrophil spreading on post arrays in the presence and absence of such inhibitors we quantified protrusive forces associated with spreading and attributed their origin to a biophysical mechanism involving a competition of adhesion energy, cortical tension, and the relaxation of that cortical tension.

Neutrophils were induced to spread on fibronectin (FN) printed post arrays as a result of their haptokinetic interaction with the adhesive ligand alone. This was consistent with our previous demonstration that a fraction of neutrophils in contact with continuous fields of FN could spread and migrate without prior or concurrent stimulation by selectin or chemoattractant (17). This haptokinetic spreading was mediated by the $\alpha_M\beta_2$ (MAC-1) integrin, a promiscuous receptor of multiple adhesive ligands. Our work with

haptokinetically activated neutrophils suggests MAC-1 promiscuity may serve as a biological safeguard, allowing neutrophils to activate at sites of trauma without executing the earliest rolling stages of the leukocyte adhesion cascade.

On flexible post arrays neutrophil spreading was mechanically detected as a circumferential ring of protrusive force (~ 75 pN/post) that propagated radially outwards (~ 200 nm/s) until the cell reached its maximum spread area. The magnitude of the protrusive force was invariant with respect to the post's location beneath the cell. Treatment of neutrophils with CK666, an inhibitor of actin branching, had no effect on protrusion suggesting the protrusive phenomenon was not analogous to lamellipodium formation. However, small molecule inhibitors of actin polymerization and depolymerization did reveal that the quiescent-to-spread shape change required relaxation of the quiescent actin cortical shell. Stiffening cortical actin *via* jaspladkinolide treatment completely eliminated spreading while softening cortical actin *via* cytochalasin B treatment slowed spreading velocity (~ 60 nm/s). Immediately after maximum protrusion, cell-engaged posts underwent a rapid contractile rebound. At the periphery of the contact zone this contractile rebound continuously evolved into a sustained contractile force floor (~ 100 pN/post) that was five-fold greater in magnitude than the transient contractile dip experienced in the core (~ 20 pN/post). While initial protrusion was myosin II independent long-time sustained contractility was ROCK and myosin II dependent as demonstrated by treatment of neutrophils with Y27632 and blebbistatin respectively.

Treating cell spreading as a competition between the energy of the adhesive environment driving the cell to spread and the cell's cohesive forces resisting shape change, has a long history (43). The equilibrium shape of such a droplet in an aqueous

medium is described by Young's equation relating the angle of the droplet-substrate interface to the substrate-medium, droplet-medium, and substrate-droplet interfacial energies. Historically, micropipette aspiration experiments on quiescent neutrophils have motivated their treatment as viscous liquid droplets with apparent surface tension (21, 44-47). Recently, Cuvelier and coworkers developed an alternative model of cell spreading, validated in mesenchymal carcinoma cells and biotinylated red blood cells, which treats the cell as a liquid droplet surrounded by a viscous shell of finite thickness (42). The model predicts two spreading regimes: contact radius evolves as $R \sim t^{0.5}$ at short times and $R \sim t^{0.25}$ at long times when the adhesive patch is comparable to the size of the cell. While we have limited resolution of the evolution of the spreading neutrophil's contact interface with time, as a result of tracking discretized post tips and not the cell membrane itself, we can approximate the spreading velocity in terms of the propagation rate of the radial protrusive force (Fig. 5.5 C and Fig. 5.7 B). We estimate that our neutrophil contact interface grows as $R \sim t^{0.4}$ which is consistent with our previous observations of neutrophil spreading on FN (5) and approaches the short time $R \sim t^{0.5}$ dependency predicted by the Cuvelier model.

However there are significant differences to be noted. In particular the contact interface in the Cuvelier model and RICM validation experiments grows as a radially symmetric disk. In neutrophils this symmetry is absent. In fact, the regions of intimate cell-substrate contact are found to decorate the neutrophil's periphery as a ring with virtually no intimate contact at the core (5). An additional discrepancy is the observation that cytochalasin B softening of the cortical shell decreases spreading velocity in neutrophils whereas cytochalasin D treatment in HeLA cells was found to increase

spreading velocity in the Cuvelier work. This later observation coupled with the additional finding that spreading is abrogated in the absence of competent integrin ligation of FN suggests that cell signaling upstream of spreading is critical and a purely physical treatment of neutrophil spreading is insufficient to reconcile the complete body of experimental work.

Our work extends previous measurements of neutrophil spreading *via* RICM (5) and reveals that regions of close membrane-substrate contact are concurrently regions of high force generation. Our studies also complement recent investigation into the mechanics of T-lymphocyte activation on mPADs (12) by considering the role of the cell cytoskeleton and demonstrating that relaxation of cortical tension is a critical driver of cell shape change. Physiologically, the forces associated with this quiescent-to-spread transition have not been considered as a possible pre-extravasation signal that facilitates transendothelial migration. Work by Rabodzey and coworkers on the forces associated with neutrophil extravasation at endothelial cell junctions demonstrated that nN protrusive forces are exerted by neutrophils when rupturing VE-cadherin junctions (48). These nN forces were attributed directly to neutrophil transmigration and not neutrophil-induced endothelial contraction. That the spherical-to-spread shape change has pN protrusive forces while neutrophil transmigration is a protrusive phenomenon of nN scale suggests a synergistic relationship between transmigrating neutrophils and the underlying endothelial cells.

Future topics to be addressed include the origin of the transient contractile rebound observed in core posts and in the periphery when ROCK/myosin II are inhibited, as well as the organization of the cortical actin shell around posts during invagination.

Additionally, work by Ghassemi and coworkers demonstrated that myosin contractile units form linear chains spanning multiple submicron diameter posts as compared to forming closed rings around single micron diameter posts (10). In our study of adhesion-driven spreading of neutrophils on submicron diameter posts we observe motion or “chatter” in the spatial position of cell engaged posts. This motivates the hypothesis that such motion is biochemically correlated with the organization of these linear contractile units. Furthermore, if these mechanical linkages exist in neutrophils, studies could be performed to search for resulting correlations in neighboring posts.

The role of β_2 clustering in adhesion-driven neutrophil spreading on post arrays also remains an open question. β_2 clustering and downstream cytoskeletal rearrangement are critical to neutrophil processes such as reactive oxygen intermediate generation and enzyme secretion (49). Yu and coworkers demonstrated that β_3 integrin clustering and radially-outward motion of these clusters was upstream of mesenchymal cell spreading on supported lipid bilayers functionalized with RGD and that the basis of the radial motion was actin polymerization (50). In neutrophils, pretreatment with cytochalasin B, an inhibitor of actin polymerization, slowed but did not eliminate spreading. However, a notable difference from the Yu work is that neutrophils on FN printed post arrays spread an order of magnitude faster than mesenchymal cells on supported lipid bilayers functionalized with RGD (~ 200 nm/s vs. ~ 20 nm/s).

Acknowledgements

We are grateful to Michael T. Yang, PhD for his time and expertise in teaching us mPADs production and printing. Funding for this work was provided by the National Institutes of Health grant HL18208 to DAH.

References

1. McDonald, B., K. Pittman, G. B. Menezes, S. A. Hirota, I. Slaba, C. C. M. Waterhouse, P. L. Beck, D. A. Muruve, and P. Kubers. 2010. Intravascular Danger Signals Guide Neutrophils to Sites of Sterile Inflammation. *Science* 330:362-366.
2. Nathan, C. 2006. Neutrophils and immunity: challenges and opportunities. *Nat Rev Immunol* 6:173-182.
3. Ley, K., C. Laudanna, M. I. Cybulsky, and S. Nourshargh. 2007. Getting to the site of inflammation: the leukocyte adhesion cascade updated. *Nature Reviews Immunology* 7:678-689.
4. Lomakina, E. B., G. Marsh, and R. E. Waugh. 2014. Cell Surface Topography Is a Regulator of Molecular Interactions during Chemokine-Induced Neutrophil Spreading. *Biophys J* 107:1302-1312.
5. Sengupta, K., H. Aranda-Espinoza, L. Smith, P. Janmey, and D. Hammer. 2006. Spreading of neutrophils: from activation to migration. *Biophys J* 91:4638-4648.
6. Tan, J. L., J. Tien, D. M. Pirone, D. S. Gray, K. Bhadriraju, and C. S. Chen. 2003. Cells lying on a bed of microneedles: An approach to isolate mechanical force. *Proceedings of the National Academy of Sciences of the United States of America* 100:1484-1489.
7. Lemmon, C. A., N. J. Sniadecki, S. A. Ruiz, J. L. Tan, L. H. Romer, and C. S. Chen. 2005. Shear force at the cell-matrix interface: enhanced analysis for microfabricated post array detectors. *Mech Chem Biosyst* 2:1-16.
8. du Roure, O., A. Saez, A. Buguin, R. H. Austin, P. Chavrier, P. Siberzan, and B. Ladoux. 2005. Force mapping in epithelial cell migration. *Proceedings of the National Academy of Sciences of the United States of America* 102:2390-2395.
9. Sniadecki, N. J., C. M. Lamb, Y. Liu, C. S. Chen, and D. H. Reich. 2008. Magnetic microposts for mechanical stimulation of biological cells: fabrication, characterization, and analysis. *Rev Sci Instrum* 79:044302.
10. Ghassemi, S., G. Meacci, S. Liu, A. A. Gondarenko, A. Mathur, P. Roca-Cusachs, M. P. Sheetz, and J. Hone. 2012. Cells test substrate rigidity by local contractions on submicrometer pillars. *Proc Natl Acad Sci U S A* 109:5328-5333.
11. Ricart, B. G., M. T. Yang, C. A. Hunter, C. S. Chen, and D. A. Hammer. 2011. Measuring traction forces of motile dendritic cells on micropost arrays. *Biophys J* 101:2620-2628.
12. Bashour, K. T., A. Gondarenko, H. Chen, K. Shen, X. Liu, M. Huse, J. C. Hone, and L. C. Kam. 2014. CD28 and CD3 have complementary roles in T-cell traction forces. *Proc Natl Acad Sci U S A* 111:2241-2246.
13. Yang, M. T., J. Fu, Y. K. Wang, R. A. Desai, and C. S. Chen. 2011. Assaying stem cell mechanobiology on microfabricated elastomeric substrates with geometrically modulated rigidity. *Nat Protoc* 6:187-213.
14. Schoen, I., W. Hu, E. Klotzsch, and V. Vogel. 2010. Probing cellular traction forces by micropillar arrays: contribution of substrate warping to pillar deflection. *Nano Lett* 10:1823-1830.
15. Desai, R. A., M. K. Khan, S. B. Gopal, and C. S. Chen. 2011. Subcellular spatial segregation of integrin subtypes by patterned multicomponent surfaces. *Integr Biol* 3:560-567.
16. Beer, F. P., E. R. Johnston, and J. T. DeWolf. 2006. *Mechanics of Materials*. McGraw-Hill Higher Education, Boston.
17. Henry, S. J., J. C. Crocker, and D. A. Hammer. 2014. Ligand density elicits a phenotypic switch in human neutrophils. *Integrative Biology* 6:348-356.

18. Penberthy, T. W., Y. Jiang, F. W. Luscinikas, and D. T. Graves. 1995. MCP-1-stimulated monocytes preferentially utilize beta 2-integrins to migrate on laminin and fibronectin. *Am J Physiol* 269:C60-68.
19. Stroka, K. M., H. N. Hayenga, and H. Aranda-Espinoza. 2013. Human neutrophil cytoskeletal dynamics and contractility actively contribute to trans-endothelial migration. *PLoS One* 8:e61377.
20. Hui, K. L., L. Balagopalan, L. E. Samelson, and A. Upadhyaya. 2015. Cytoskeletal forces during signaling activation in Jurkat T-cells. *Molecular Biology of the Cell* 26:685-695.
21. Tsai, M. A., R. S. Frank, and R. E. Waugh. 1994. Passive mechanical behavior of human neutrophils: effect of cytochalasin B. *Biophys J* 66:2166-2172.
22. Sheikh, S., W. B. Gratzler, J. C. Pinder, and G. B. Nash. 1997. Actin polymerisation regulates integrin-mediated adhesion as well as rigidity of neutrophils. *Biochem Biophys Res Commun* 238:910-915.
23. Jannat, R. A., M. Dembo, and D. A. Hammer. 2011. Traction forces of neutrophils migrating on compliant substrates. *Biophys J* 101:575-584.
24. Kilfoil, M. L. 2014. *Biological Physics Kilfoil Lab*. Web Page. 04 November 2014. <http://people.umass.edu/kilfoil/downloads.html>.
25. Pelletier, V., N. Gal, P. Fournier, and M. L. Kilfoil. 2009. Microrheology of Microtubule Solutions and Actin-Microtubule Composite Networks. *Physical Review Letters* 102:188303.
26. Crocker, J. C., and D. G. Grier. 1996. Methods of Digital Video Microscopy for Colloidal Studies. *Journal of Colloid and Interface Science* 179:298-310.
27. Jordan, M. S., and G. A. Koretzky. 2010. Coordination of receptor signaling in multiple hematopoietic cell lineages by the adaptor protein SLP-76. *Cold Spring Harb Perspect Biol* 2:a002501.
28. Sheterline, P., and C. R. Hopkins. 1981. Transmembrane linkage between surface glycoproteins and components of the cytoplasm in neutrophil leukocytes. *The Journal of Cell Biology* 90:743-754.
29. Holzinger, A. 2009. Jasplakinolide: an actin-specific reagent that promotes actin polymerization. *Methods Mol Biol* 586:71-87.
30. MacLean-Fletcher, S., and T. D. Pollard. 1980. Mechanism of action of cytochalasin B on actin. *Cell* 20:329-341.
31. Svitkina, T. M., and G. G. Borisy. 1999. Arp2/3 complex and actin depolymerizing factor/cofilin in dendritic organization and treadmilling of actin filament array in lamellipodia. *J Cell Biol* 145:1009-1026.
32. Hetrick, B., Min S. Han, Luke A. Helgeson, and Brad J. Nolen. 2013. Small Molecules CK-666 and CK-869 Inhibit Actin-Related Protein 2/3 Complex by Blocking an Activating Conformational Change. *Chemistry & Biology* 20:701-712.
33. Limouze, J., A. Straight, T. Mitchison, and J. Sellers. 2004. Specificity of blebbistatin, an inhibitor of myosin II. *J Muscle Res Cell Motil* 25:337-341.
34. Schindelin, J., I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J. Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, and A. Cardona. 2012. Fiji: an open-source platform for biological-image analysis. *Nat Methods* 9:676-682.
35. Jannat, R. A., G. P. Robbins, B. G. Ricart, M. Dembo, and D. A. Hammer. 2010. Neutrophil adhesion and chemotaxis depend on substrate mechanics. *J Phys-Condens Mat* 22.
36. Oakes, P. W., D. C. Patel, N. A. Morin, D. P. Zitterbart, B. Fabry, J. S. Reichner, and J. X. Tang. 2009. Neutrophil morphology and migration are affected by substrate elasticity. *Blood* 114:1387-1395.

37. Stroka, K. M., and H. Aranda-Espinoza. 2009. Neutrophils display biphasic relationship between migration and substrate stiffness. *Cell Motil Cytoskeleton* 66:328-341.
38. Ghibaudo, M., A. Saez, L. Trichet, A. Xayaphoummine, J. Browaeys, P. Silberzan, A. Buguin, and B. Ladoux. 2008. Traction forces and rigidity sensing regulate cell functions. *Soft Matter* 4:1836-1843.
39. Hecht, E. 1998. *Optics*. Addison-Wesley, Reading, Mass.
40. Diamond, M. S., and T. A. Springer. 1993. A subpopulation of Mac-1 (CD11b/CD18) molecules mediates neutrophil adhesion to ICAM-1 and fibrinogen. *J Cell Biol* 120:545-556.
41. Krasik, E. F., K. E. Caputo, and D. A. Hammer. 2008. Adhesive dynamics simulation of neutrophil arrest with stochastic activation. *Biophys J* 95:1716-1728.
42. Cuvelier, D., M. Thery, Y. S. Chu, S. Dufour, J. P. Thiery, M. Bornens, P. Nassoy, and L. Mahadevan. 2007. The universal dynamics of cell spreading. *Curr Biol* 17:694-699.
43. Carter, S. B. 1967. Haptotaxis and the Mechanism of Cell Motility. *Nature* 213:256-260.
44. Evans, E., and B. Kukan. 1984. Passive material behavior of granulocytes based on large deformation and recovery after deformation tests. *Blood* 64:1028-1035.
45. Evans, E., and A. Yeung. 1989. Apparent viscosity and cortical tension of blood granulocytes determined by micropipet aspiration. *Biophys J* 56:151-160.
46. Needham, D., and R. M. Hochmuth. 1992. A sensitive measure of surface stress in the resting neutrophil. *Biophys J* 61:1664-1670.
47. Lomakina, E. B., C. M. Spillmann, M. R. King, and R. E. Waugh. 2004. Rheological analysis and measurement of neutrophil indentation. *Biophys J* 87:4246-4258.
48. Rabodzey, A., P. Alcaide, F. W. Lusinskas, and B. Ladoux. 2008. Mechanical forces induced by the transendothelial migration of human neutrophils. *Biophys J* 95:1428-1438.
49. Raptis, S. Z., S. D. Shapiro, P. M. Simmons, A. M. Cheng, and C. T. Pham. 2005. Serine protease cathepsin G regulates adhesion-dependent neutrophil effector functions by modulating integrin clustering. *Immunity* 22:679-691.
50. Yu, C. H., J. B. Law, M. Suryana, H. Y. Low, and M. P. Sheetz. 2011. Early integrin binding to Arg-Gly-Asp peptide activates actin polymerization and contractile movement that stimulates outward translocation. *Proc Natl Acad Sci U S A* 108:20585-20590.

Chapter 6

Single Vesicle Patterning of Uniform, Giant Polymersomes

Into Microarrays

Preface

The content of this chapter has been adapted from its published version in the journal *Small* (2013, Vol. 9(13):2272-2276, DOI: 10.1002/sml.201202627) by permission of John Wiley and Sons (License: 3540921353732). The published manuscript was coauthored by Neha P. Kamat, **Steven J. Henry**, Daeyeon Lee, and Daniel A. Hammer. The content has been reproduced with knowledge of the coauthors. Specific author contributions were as follows: NPK produced microfluidic vesicles, performed experiments, and wrote the manuscript; **SJH** produced patterned substrates, performed experiments, and edited the manuscript; DL shared the microfluidic platform, consulted on data interpretation, and edited the manuscript; DAH supported the work, consulted on data interpretation, and edited the manuscript. Supplementary movies referenced in the prose can be retrieved from the published version online.

Abstract

Giant, cell-sized polymersomes are functionalized and patterned at the single vesicle level. Microfluidic methods are employed to generate uniform diameter vesicles with high loading efficiencies and microcontact printing is used to generate patterns of

adhesive ligand. A simple sensory capability is demonstrated with the immobilized array of vesicles.

Introduction

Studies with artificial cells, or protocells, in which synthetic particles are designed to replicate cellular processes are moving beyond single particles to the engineering of coordinated action among multiple particles (1-2). Cells often display multi-cellular communication and coordinate their activities, such as in quorum sensing (3) and paracrine signaling (4). Vesicles are an ideal particle to serve as the structural basis for a protocell. The design and construction of multi-vesicle systems to induce inter-particle communication, however, is challenging. Minimally, such a system requires spatial control of vesicle positioning, the encapsulation of the signaling agents, and functionalization of the responding vesicle for signal detection. Patterning vesicles with spatial precision on a substrate would enable the design and development of structurally well-defined communication systems, and have utility in other applications, such as building biosensor arrays. Microcontact printing is ideally suited towards the fabrication of an ordered array of inter-communicating artificial cells. Here, we demonstrate for the first time the patterning of individual, monodisperse, and functionalized giant polymersomes. Using microfluidics, we prepare functionalized vesicles of controlled size with high encapsulation efficiency and use microcontact printing to immobilize polymersomes in controlled spatial arrangements. Finally, we demonstrate the sensory capability of the resulting array.

Vesicles, comprised of bilayer membranes surrounding an aqueous lumen, are architecturally similar to cells, and provide the spatial compartmentalization that enable

cells to perform a variety of metabolic and sensory functions. Patterning vesicles has facilitated diverse applications ranging from bioenergetic reactions (5-6) to diagnostic assays based on specific recognition (7). Arrays of both lipid (8-10) and polymer vesicles (6, 11) have been built. Vesicles with thick membrane cores are particularly useful because they are able to incorporate hydrophobic solutes in the core of the membrane as well as aqueous solutes in the vesicle lumen. Polymersomes, bilayer vesicles made from di-block copolymers, not only have hyper-thick membrane cores, but possess additional advantages over lipid vesicles, including increased membrane strength and the flexibility to design a wide range of physical and chemical properties into the polymer through chemical synthesis (12).

Beyond technological and medical applications, immobilized bilayer vesicles can also be used to construct systems that reproduce specific functions of cells, like triggered gene expression or chemical reaction cascades (5, 13). Cellular mimicry with synthetic vesicles is quickly advancing to replicate more complex cellular behaviors, such as particle-to-particle (vesicle-to-vesicle) communication (12). For example, theoretical work by Balazs and coworkers (1, 14) has proposed that inanimate, cell sized capsules can be engineered to communicate and induce movement of one another through the exchange of soluble cues that dynamically modulate the underlying adhesive environment. A key technological advance needed to test the principles of these calculations is the assembly of arrays of vesicles with precise spatial organization.

To date, the majority of studies conducted with immobilized vesicles have been limited to small vesicles (with diameters ≤ 400 nm). Large, micron-sized vesicles, however, are closer to the dimensions of biological cells and are therefore appropriately

sized for the study of vesicle-cell communication at a biologically relevant length scale. Yet, patterning large vesicles has proven difficult. When larger, single, micron-scale vesicles have been immobilized, the vesicle size has generally not exceeded several microns (9). Arrays that are assembled with polydisperse vesicles, limit the precision of the intended pattern. Classical vesicle preparation methods, like thin-film hydration, have made it difficult to prepare monodisperse giant vesicles and high encapsulation efficiencies, and have consequently limited our ability to pattern uniform arrays of large vesicles. The advent of vesicle production methods using microfluidic techniques now enables the generation of single, giant, monodisperse polymersomes (15). These vesicles, formed through solvent evaporation from double emulsion templates, have near perfect encapsulation efficiencies and highly uniform diameters (16).

Materials and Methods

Reagents

A polyethylene oxide-polybutadiene diblock copolymer, PEO₃₀-*b*-PBD₄₆, was used for polymersome formation (Polymer Source, Montreal, Canada). Biotinylated polymer was previously functionalized in our laboratory, in which biotin was conjugated onto the terminal polyethylene oxide of PEO₃₀-*b*-PBD₄₆ *via* an intermediate 4-fluoro-3-nitrobenzoic acid linkage that yielded polymer that was approximately 65% biotin-modified. Biocytin, Pluronic F-127, and bovine serum albumin (BSA) were purchased from Sigma. NeutrAvidin-Texas Red conjugate and Biotective Green Reagent were purchased from Life Technologies and were used to pattern substrates and demonstrate vesicle communication, respectively.

Polymersome Preparation

Giant polymersomes were prepared *via* double emulsion templates. Water-in-oil-in-water (W/O/W) double emulsions were produced using glass microcapillary devices, described previously (17). The inner aqueous phase consisted of a sucrose solution (290 mOsm), the middle, organic phase consisted of 1 mg/mL polymer in a mixture of toluene and chloroform (72:28 v/v), and the outer, aqueous phase consisted of phosphate buffered saline (PBS) (290 mOsm) containing either 1 wt % BSA or 0.1 wt % F-127. For functionalization studies, polymersomes were prepared with Pluronic F-127 as the stabilizer to ensure carboxy-linked biocytin modification occurs with the carboxy group on the polymer and not on any residual surfactant (e.g. BSA) that remains in the membrane). For all other studies, BSA was used as the stabilizer. The three fluid streams were co-focused to generate PEO₃₀-*b*-PBD₄₆ double emulsions that were collected in 2 mL of PBS inside 20 mL glass vials. The vials were left loosely capped on a rocker overnight and subsequently tightly capped and rocked until use, generally between 1-2 weeks after formation. The control over vesicle size was demonstrated by changing the outer aqueous phase flow rate between 10-70 mL/hr.

Polymersome Functionalization

To demonstrate functionalization of giant, double emulsion-templated polymersomes, polymer vesicles were formed from either carboxy-terminated diblock copolymer, COOH- PEO₃₀-*b*-PBD₄₆ or biotin-functionalized polymer. In the former case, following polymersome formation, the carboxy-terminated polymer membranes were functionalized *via* an EDC mediated coupling to biocytin. For polymersomes made with either biotin-conjugated polymer or covalently linked to biocytin post-vesicle formation,

NeutrAvidin-Texas Red was incubated with the vesicles to demonstrate the ability to functionalize biotin after attachment to the vesicle surface.

Substrate Fabrication

Micropatterned substrates containing NeutrAvidin-Texas Red islands were fabricated as described by Desai et al (18). Briefly, glass coverslips were spin-coated with poly(dimethylsiloxane) (PDMS, Sylgard 184, Dow-Corning) pre-polymer components at 10:1 (base:curing agent) ratio by weight followed by baking overnight at 60 °C. PDMS stamps were cast against a silicon wafer upon which was etched the negative relief of our desired island array using common photolithography techniques. The stamps were cured overnight at 60 °C, removed from the mask and inked with NeutrAvidin-Texas Red (100 µg/mL) for 1 h. Stamps were rinsed and applied to the substrate, as described previously. Square NeutrAvidin lattices consisted of circular islands either 50 µm in diameter and laterally spaced by 100 µm (50 µm x 100 µm) or of 10 µm diameter islands spaced by 50 µm (10 µm x 50 µm). Texas Red-labeled NeutrAvidin was used to visualize the resulting pattern and uniformity of the printed protein. Vesicle adhesion was restricted to the NeutrAvidin islands by blocking the unprinted surface with 0.2 wt % Pluronic F-127 for at least 10 min and washing thoroughly with PBS without dewetting the printed and blocked surface

Vesicle Patterning

Polymersomes made with biotin-conjugated polymer were incubated on NeutrAvidin patterned substrates at a density of 80 vesicles/mm² and subjected to gentle rotation on a motorized microscope stage. The convection of fluid induced by the stage motion was found to effectively clear unbound vesicles from the patterned and blocked

PDMS substrate provided the island pitch allowed egress. Vesicles made with unmodified PEO₃₀-*b*-PBD₄₆ were used as a control of biotin-avidin specificity as described in the text.

Results and Discussion

Producing Monodisperse Populations of Micron-Scale Microfluidic Vesicles

We prepared polymersomes from microfluidic-generated, water-in-oil-in-water (W/O/W) double emulsions that contain the amphiphilic diblock copolymer PEO₃₀-*b*-PBD₄₆ (MW 1300 and 2500, respectively). We previously verified the unilamellar structure of these vesicles and elimination of organic solvent from their membranes (17). A sucrose solution, toluene and chloroform mixture, and phosphate buffered saline (PBS) made up the inner, middle, and outer phases, respectively, of the double emulsions. Though we have previously shown that polymersomes can be formed without the use of stabilizers, in order to increase yield in this study, the outer phase contained either 1 wt % bovine serum albumin (BSA) or 0.1 wt % Pluronic F-127. By tuning the continuous phase flow rate, we can robustly control the diameter of the resulting vesicles over a range of 20 – 70 microns (Fig. 6.1 A). While small adjustments to the inner and middle flow rates are required to form double emulsions at each continuous flow rate, we find the outer flow rate is the dominant variable in dictating vesicle size (Fig. 6.2 C). Polymersome diameter was found to be a linear function of this continuous phase flow rate and was invariant with respect to the polymer formulations tested at a given continuous phase flow rate (Fig. 6.1 B). Changing the polymer solution from a control PEO₃₀-*b*-PBD₄₆, to biotin-functionalized PEO₃₀-*b*-PBD₄₆, to carboxy-terminated PEO₃₀-

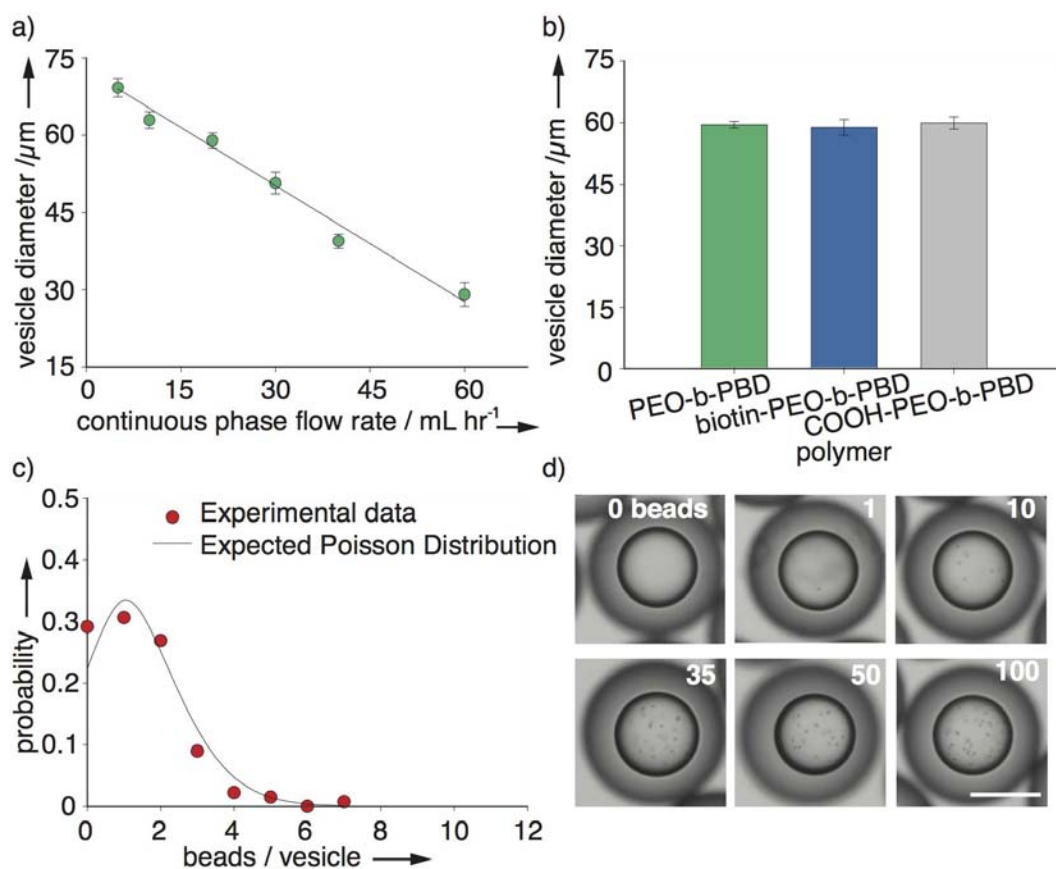


Figure 6.1 Controlling vesicle size and loading. (A) The diameter of polymersomes formed with a microfluidic capillary device is a linear function of the continuous flow rate used to prepare double emulsions ($n > 50$ vesicles for each data point, error bars are standard deviation (s.d.)). (B) Vesicle diameter is invariant with respect to the polymer formulations tested at a given flow rate ($n > 100$). (C) Vesicles made to encapsulate a single bead follows the expected Poisson distribution, where $\langle \text{bead/vesicle} \rangle_{\text{expected}} = 1.5$ ($n = 134$ vesicles, $\langle \text{bead/vesicle} \rangle_{\text{actual}} = 1.3$, C.O.V. = 0.9). (D) Double emulsions are prepared with 1 μm carboxyl modified polystyrene beads in the interior aqueous compartment at loading quantities calculated in white. Scale bar is 50 μm .

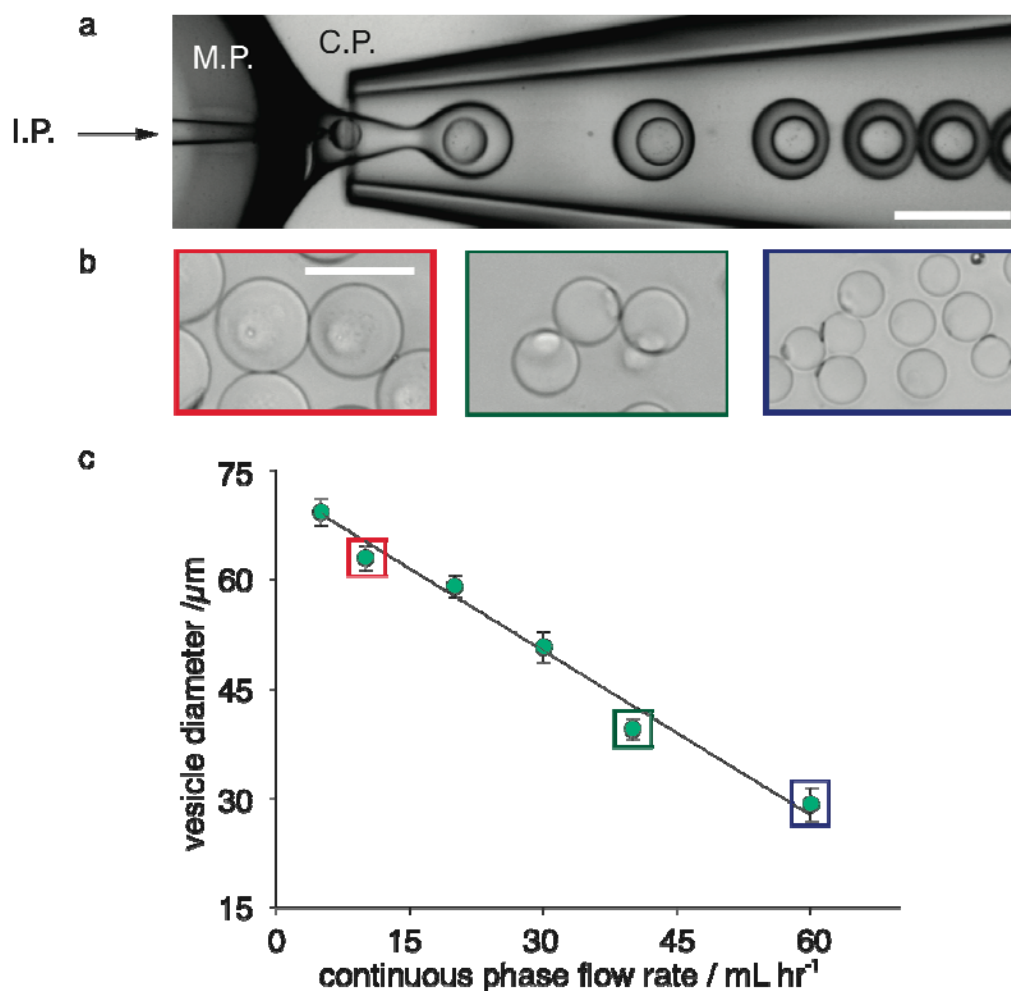


Figure 6.2 Controlling vesicle diameter by adjusting the continuous phase flow rate. (A) A microcapillary device is used to prepare polymersomes by first generating double emulsions. The inner phase (I.P.), middle phase (M.P.) and continuous phase (C.P.) are co-focused to create the double emulsions. (B, C) The diameter of polymersomes can be controlled by changing the flow rates of the different fluid phases. The C.P. flow rate is the dominant variable in dictating vesicle size. Using a device with an inner capillary diameter of 32.6 μm and an outer capillary diameter of 151 μm , polymersomes were prepared with diameters ranging from 20-70 microns. Phase microscopy images of representative polymersomes are depicted for 3 different populations of vesicles appearing in graph C. ($n > 50$ vesicles for each data point, error bars are standard deviation (s.d.)). The I.P., M.P. and C.P. flow rates for each population of vesicles that appears on graph c were: (1) 0.55, 5.0 and 5 mL hr^{-1} (2) 1, 7 and 10 mL hr^{-1} (3) 1, 7 and 20 mL hr^{-1} (4) 1.5, 7 and 30 mL hr^{-1} (5) 1.2, 7.5 and 40 mL hr^{-1} and (6) 0.55, 5 and 60 mL hr^{-1} . Scale bar is 70 μm . This figure was presented in the supplementary text of the original manuscript.

b-PBD₄₆ resulted in the same average diameter of $59.0 \pm 0.5 \mu\text{m}$ demonstrating the consistency of our preparation method regardless of small changes in polymer chemistry.

Controlling Microfluidic Payload Encapsulation

For applications in which arrays of vesicles are to be used as bioreactors, maintaining high encapsulation efficiency and controlling the concentration of encapsulated reactants is critical (19). To illustrate the control microfluidic methods afford in precise payload encapsulation, we prepared double emulsions with different numbers of $1 \mu\text{m}$ carboxylated polystyrene beads. An inner phase solution is prepared to contain the appropriate volume fraction of beads that results in the desired number of particles encapsulated. By determining the actual distribution of beads loaded in a population of vesicles that were prepared to have approximately 1 bead in their interior, we can assess the reproducibility and variation of particle loading in its most variable (i.e. low number) regime. When the volume fraction of particles in a given volume is low and randomly distributed, the distribution of bead loading is expected to follow a Poisson model (20). This distribution was experimentally seen for the volume fraction corresponding to a mean of 1.3 ± 1.2 beads/vesicle (Fig. 6.1 C). Having validated that the encapsulated number of beads can be dictated by the starting volume fraction of the inner phase solution and given that the diameter of the inner aqueous droplet is constant, we produced populations of vesicles with controlled numbers of encapsulated beads (calculated loading values are reported in Fig. 6.1 D).

Surface Functionalizing Vesicles for Controlled Adhesion

In order to adhere vesicles specifically to a patterned surface, the membranes must be functionalized with an appropriate ligand complementary to a ligand

immobilized on a surface. Polymers can be modified prior to vesicle production or after membrane assembly (21). If functional groups are sufficiently hydrophilic, we can advantageously assemble vesicles with pre-functionalized polymer where the number of reactive molecules on a vesicle is known and reproducible between batches. In this study, we demonstrated that microfluidic polymersomes could be functionalized through both aforementioned routes. As shown in Fig. 6.3, PEO₃₀-*b*-PBD₄₆ polymers conjugated to biotin were assembled into vesicles. Alternatively, polymersomes made with carboxy-terminated PEO₃₀-*b*-PBD₄₆ polymer could also be modified after vesicle preparation using an EDC/NHS-mediated coupling reaction to link biocytin to the carboxylic acid groups (22). The latter method results in biotin present only on the outer leaflet of the vesicle, allowing the creation of asymmetric membranes with differing functionalities (Fig. 6.4). Both methods of modification were verified to yield vesicles in which biotin was accessible for binding to Texas Red-labeled NeutrAvidin (NAv). Given the reduced number of steps required to produce biotin-functionalized vesicles from pre-modified polymer, this route was employed to prepare vesicles in subsequent patterning studies.

Spatially Patterning Arrays of Functionalized Vesicles and an Application

We next set out to spatially organize biotin-modified polymersomes *via* immobilization onto NAv-printed surfaces. Microcontact printing is a powerful tool for the precise and complex spatial organization of adhesive ligands on surfaces (23-24). Substrates for polymersome array generation were prepared by microcontact printing NAv onto poly(dimethylsiloxane) (PDMS)-spin coated glass coverslips. NAv islands were 50 μm in diameter with 100 μm spacing (Fig. 6.5 *AI*). Unstamped regions of the substrate were blocked with the triblock copolymer Pluronic F-127 (PEO₁₀₆-*b*-PPO₁₀₆-*b*-

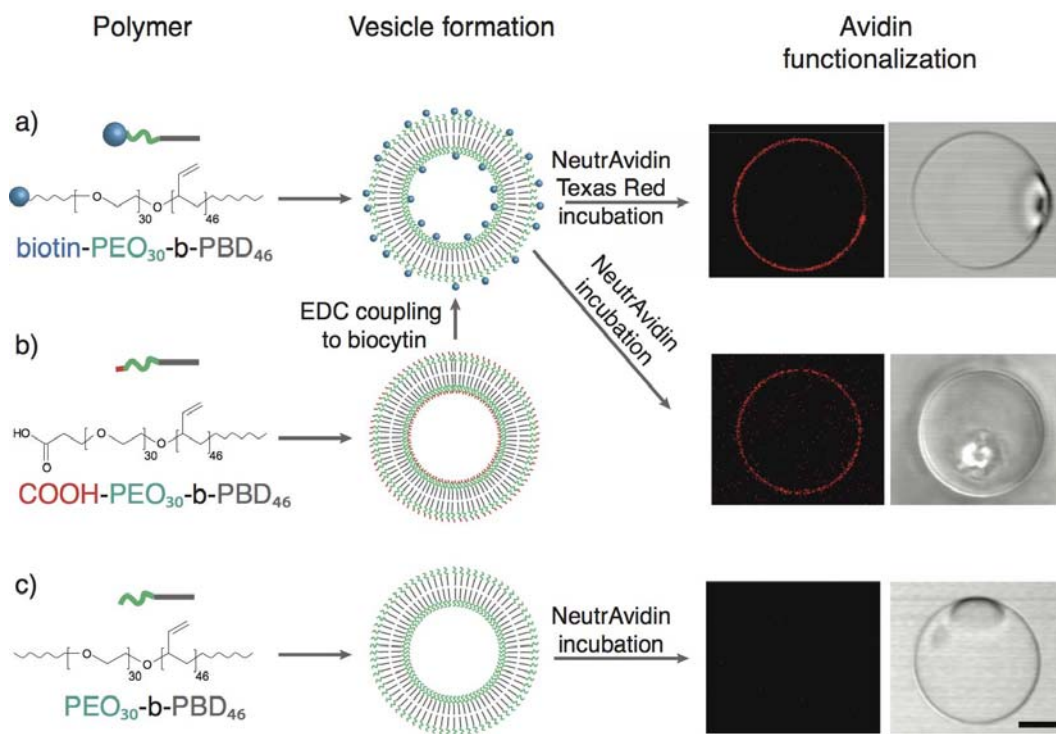


Figure 6.3 Functionalizing polymersomes via biotinylation. (A) The diblock copolymer used to prepare polymersomes is modified prior to vesicle formation to contain a reactive biotin group. (B) Polymer membranes are modified to contain biocytin post-vesicle formation via an EDC-mediated coupling to carboxyl-modified polymer. Polymersomes prepared through either route contain available biotin groups on the membrane surface that bind Texas-Red labeled-NeutrAvidin upon incubation (fluorescent (left) and phase images (right) of a representative vesicle functionalized with NeutrAvidin). (C) Control polymersomes prepared with a polymer that does not contain a reactive group do not bind avidin. Scale bar is 25 μm .

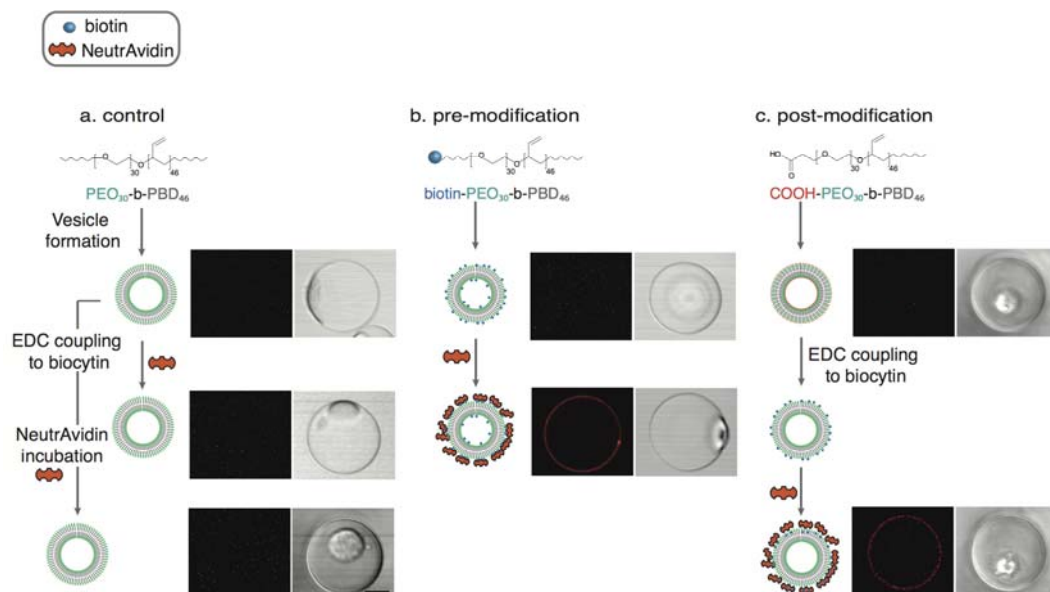


Figure 6.4 NeutrAvidin (NAv) functionalization of polymersomes made from biotin-modified polymer. For functionalization studies, polymersomes were prepared with Pluronic F-127 as the stabilizer to ensure carboxy-linked biocytin modification occurs with the carboxy group on the polymer and not on any residual surfactant that remains in the membrane. (A) Polymersomes that are PEO terminated are not functionalized with NAv. The lack of carboxyl groups on the vesicle surface ensures that EDC mediated reactions to biocytin do not result in biocytin linkage to the polymer membrane. (B) Polymersomes can be made by using polymer that already contains biotin. In this case biotin is available on both the inner and outer leaflets of the membrane and is able to bind NAv upon incubation. (C) Carboxy-terminated polymersomes are also functionalized with biocytin after vesicle formation. In this case, biotin is only added to the outer leaflet of the vesicle. This EDC-mediated coupling could be used to link other amine-containing proteins or molecules to a polymersome surface. Scale bar is 20 μm . This figure was presented in the supplementary text of the original manuscript.

PEO₁₀₆), which results in the presentation of PEO groups on bare PDMS not occupied by adhesive ligand. Vesicles that were ~55 μm in diameter were incubated on the substrate. Gentle movement of the microscope stage created a convective flow of PBS across the substrate face inducing non-adherent vesicles to move. Moving vesicles were either captured by printed NAv islands (Fig. 6.5 *BI* and Fig. 6.6) or glided along the PEO blocked regions between islands. The high mobility of vesicles on F-127 blocked PDMS (Movie S1) is attributed to the steric repulsion between PEO chains at the vesicle-substrate interface.

Selective biological adhesion requires a combination of adhesive and repulsive interactions. In the absence of Pluronic blocking, vesicles failed to specifically pattern, adhering to both bare PDMS and NAv islands (Fig. 6.5 *CI*). To explore the role of biotin-avidin specificity on patterning, non-biotinylated vesicles were incubated with NAv printed surfaces. To our surprise, non-biotinylated polymersomes could still be patterned (Fig. 6.5 *DI* and Fig. 6.7). The repulsive interactions between the blocking F-127 and the PEO chains on the polymersome drove vesicles onto NAv islands, regions of the substrate that minimized the energetically unfavorable repulsive forces between PEO groups. Capture of non-biotinylated polymersomes on NAv islands suggests a level of favorable non-specific interaction (25) between PEO and NAv which is verified by the absence of vesicle motion on continuous fields of the ligand (Fig. 6.8 and Movie S2).

We hypothesized that non-specific interactions between vesicles and NAv patches could be tuned by changing the surface area over which they occur. To test this hypothesis, NAv was printed with a decreased island size (Fig. 6.5 *A2*). On smaller islands biotinylated vesicles were again specifically patterned when substrates were

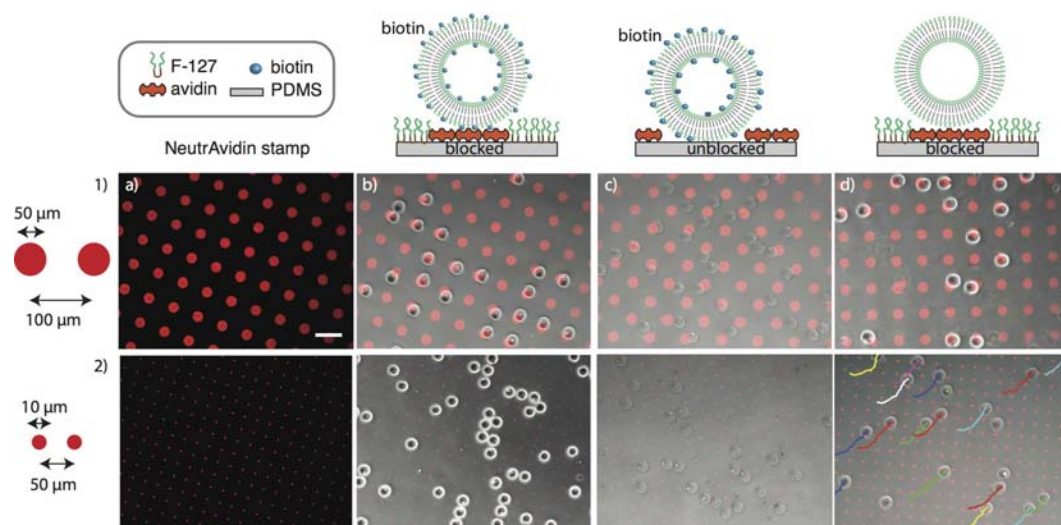


Figure 6.5 Patterning single polymersomes. Giant polymersomes that are functionalized with biotin are patterned in an array by incubation and immobilization onto micropatterned islands of NAV. (A) Fluorescence image of the NAV microcontact-printed array which has islands with a (1) 50 μm diameter and 100 μm spacing or a (2) 10 μm diameter and 50 μm spacing. Scale bar is 100 μm . (B-D) Fluorescence microscopy image of the NAV stamp overlaid with a phase image of polymersomes. (B) Biotinylated polymersomes incubated with a NAV stamped and F-127 blocked surface are specifically patterned. (C) Biotinylated vesicles fail to pattern, binding nonspecifically to bare PDMS. (D) Non-biotinylated control vesicles pattern on a printed and blocked surface provided the island size is sufficiently large. When present, colored tracks indicate the trajectories of mobile vesicles on stamped substrates (B-D).

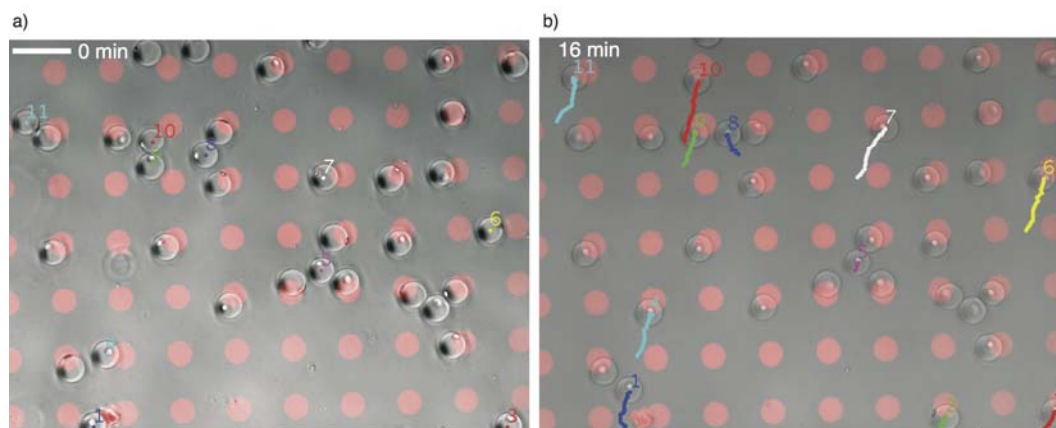


Figure 6.6 Polymersome capture on NAV-printed surfaces. (A) Polymersomes are patterned by incubating vesicles on a NAV-printed surface and placing both the biotinylated polymersomes and the NAV surface on a microscope stage. Low-level motion of the microscope stage that is rotating between different imaging positions creates a convective flow in the polymersome sample. (B) Polymersomes moving along the NAV printed surface are mobile on the blocked regions that contain Pluronic F-127, but are captured by the NAV islands. Overtime, the capture of biotinylated vesicles and movement of non-captured vesicles out of the field of view results in the patterning of polymersomes. Trajectories for mobile vesicles appear in colored tracks that are overlaid onto the merged image of vesicles and the NAV stamp. Vesicles numbered 2, 3, 4, 6, 7, 9, 10, and 11 are captured by NAV islands. Vesicles 5 and 8 are trapped by two patterned vesicles and unable to move to find a NAV island or exit the field of view. Increasing the spacing between NAV islands would allow unbound vesicles to egress more readily. The possibility of utilizing geometric confinement as a patterning force, however, is compelling given this observation of entrapment. Scale bar is 100 μm . The full time course depicting this vesicle capture can be seen in Movie S1. This figure was presented in the supplementary text of the original manuscript.

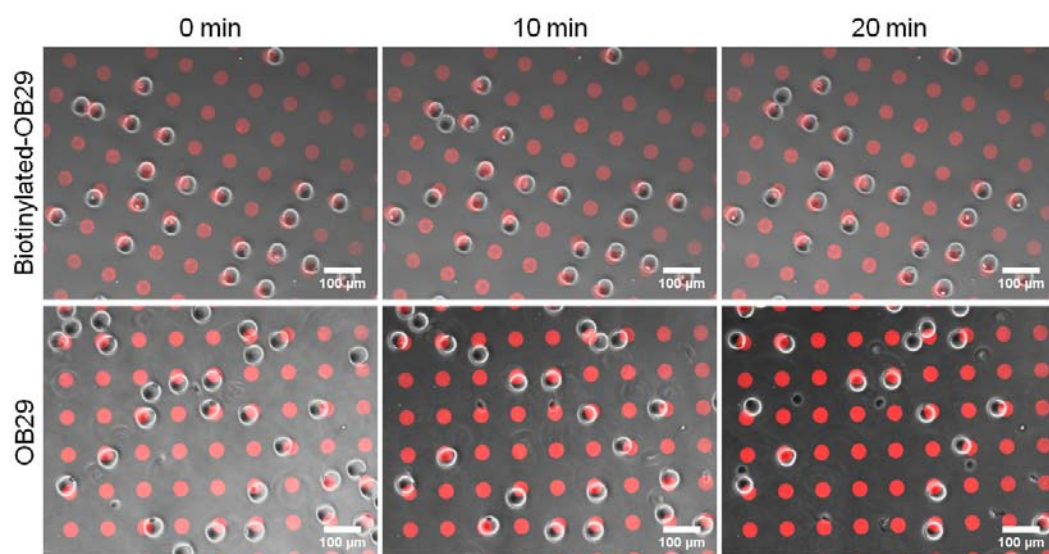


Figure 6.7 Polymersomes patterned on Nav islands of 50 μm diameter and 100 μm pitch. Sustained vesicle patterning in the non-biotinylated case motivated our hypothesis that another driving force such as the repulsive interaction between PEO chains on the vesicles and PEO chains on the blocked substrate was at play. This figure was presented in the supplementary text of the original manuscript.

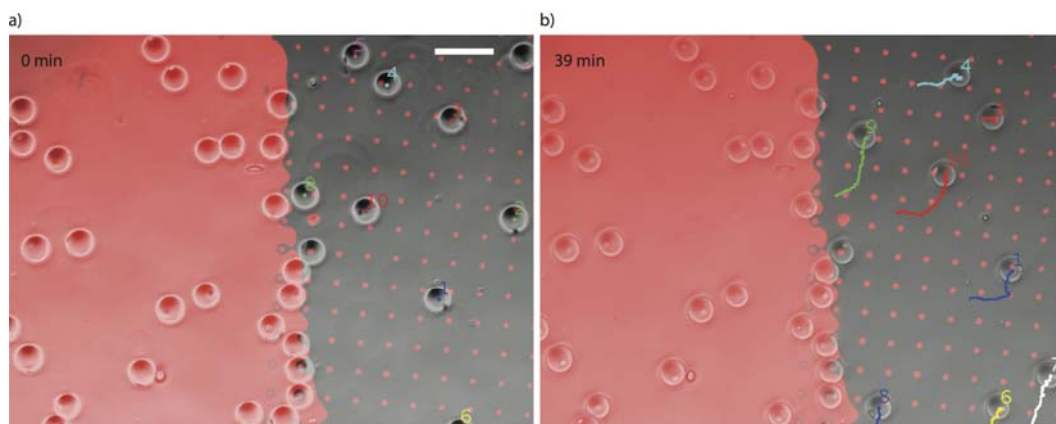


Figure 6.8 Effect of NAV surface area on non-specific binding. To illustrate the effect of available protein surface area on non-specific binding, NAV was printed in a uniform field on the left side of the PDMS substrate and printed in islands on the right side. The fluorescent image of NAV (red) is overlaid with the phase image of polymersomes. The substrate was blocked with Pluronic F-127 and incubated with non-biotinylated polymersomes. Though polymersomes do not contain biotin, they are immobilized through non-specific interactions on the uniform field of NAV. When the area of interaction is decreased to a 10 μm island size, however, the vesicles are mobile and move freely across the substrate. The tracks of motile vesicles are indicated by overlaying the colored vesicle trajectories onto the merged image of the polymersomes and NAV stamp. The full time course of this phenomenon can be seen in Movie S2. This figure was presented in the supplementary text of the original manuscript.

blocked (Fig. 6.5 B2), and bound non-specifically to bare PDMS when left unblocked (Fig. 6.5 C2). When non-biotinylated control vesicles were incubated with the smaller islands of NAv, however, they failed to pattern as previously observed on large 50 μm islands. Instead, these vesicles were found to be continuously motile during observation as indicated by the superimposed trajectories (Fig. 6.5 D2 and Movie S2). By decreasing the island size we effectively eliminated the contribution of nonspecific PEO-NAv interaction allowing us to attribute the high fidelity patterning of biotinylated vesicles to biotin-avidin binding exclusively (Fig. 6.9). Ultimately, we have shown that NAv-printed PDMS, blocked with Pluronic F-127 is ideally suited for the spatial patterning of giant biotinylated polymersomes.

Having successfully patterned giant microfluidic vesicles we sought to demonstrate the array's future applicability to the design of systems capable of inter-vesicle communication. Towards this end, we demonstrate the vesicle array can be used as a biosensing platform. Immobilized polymersomes can report the presence of a soluble molecule added to the vesicle array by capturing the molecule at the vesicle membrane. Biotective Green reagent, an avidin analogue, was used as the bioactive ligand. This molecule is labeled with a fluorescent donor molecule that is quenched through FRET interactions with an acceptor molecule located in the biotin-binding pockets of the reagent. Upon binding biotin, the quencher molecules become displaced and the signaling ligand fluoresces. When this reagent was added to an array of immobilized biotinylated polymersomes, the ligand was captured at the polymersome surface (Fig. 6.10 B). Fluorescent signals from three representative vesicles over the course of 40 minutes are shown in Fig. 6.10 C.

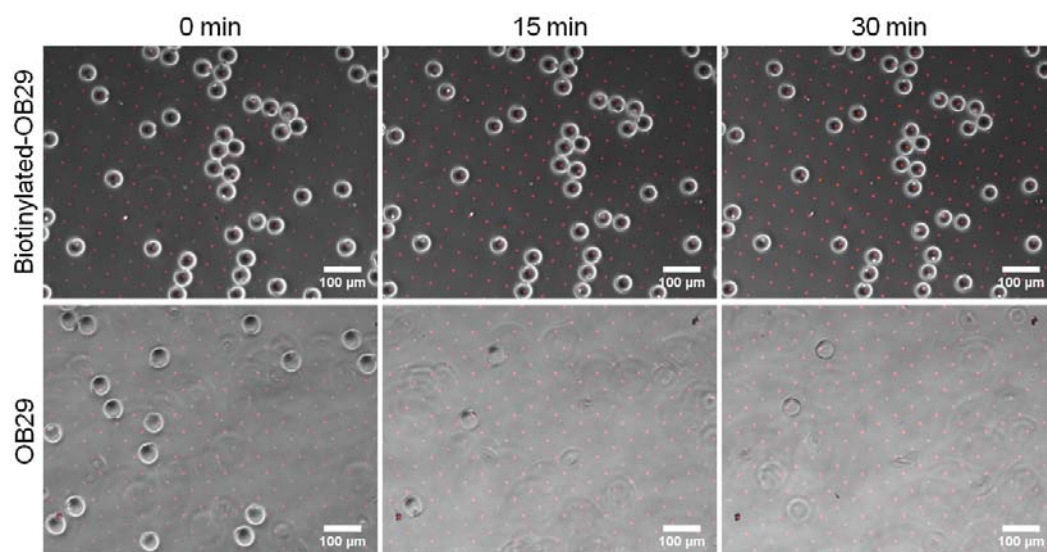


Figure 6.9 Polymersomes patterned on smaller NAV islands of 10 μm diameter and 50 μm pitch. Unlike previously, where non-biotinylated vesicles remained patterned, here, reduction in island size prevents patterning of these same control vesicles. That biotinylated vesicles retain their pattern during stage motion suggests the biotin-avidin interaction is stronger than the non-specific PEO-NAV interaction. This figure was presented in the supplementary text of the original manuscript.

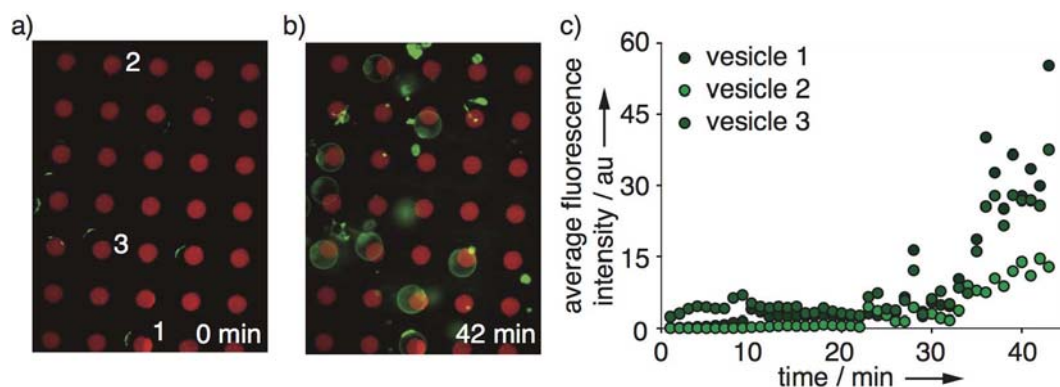


Figure 6.10 Creating sensor arrays. Biotin functionalized vesicles were patterned onto an array of NAV with islands $50\ \mu\text{m}$ in diameter and $100\ \mu\text{m}$ in spacing. (A) At time = 0, when Biotective Green Reagent is added to the system, the vesicles do not fluoresce. (B) At time = 42 min, Biotective Green Reagent bound biotin on the surface of vesicles, the fluorescence signal from the reagent increased and occurred selectively at the vesicle surface. (C) Fluorescence signals from the reagent were tracked at the surface of three vesicles (numbers correspond to panel A) over the course of a 40 min imaging session.

Conclusions

Our system provides a significant advance in the engineering of vesicle-based assemblies. There has been no previous demonstration of the ability to chemically control the spatial organization of single giant polymersomes. We show, by patterning vesicles of precisely controlled diameter and payload encapsulation, that individual polymersomes can be positioned into multi-vesicle arrays that are geometrically governed by the underlying adhesiveness of the surface. In the future, the printing of multiple adhesive ligands (18, 26) or oligonucleotides (27) on a single substrate will enable the patterning of multiple populations of vesicles, each with a distinct biorecognition capability. The precise patterning of giant functionalized polymersomes is an important step towards realizing the full potential of increasingly complex artificial cell systems.

Acknowledgements

This work was supported by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Materials Sciences and Engineering under award DE-FG02-11ER46810. NPK and SJH are supported by NSF Graduate Fellowships. We thank Eric Johnston for assistance in preparing photomasks.

References

1. Kolmakov, G. V., V. V. Yashin, S. P. Levitan, and A. C. Balazs. 2010. Designing communicating colonies of biomimetic microcapsules. *Proc Natl Acad Sci U S A* 107:12417-12422.
2. Kuiper, S. M., M. Nallani, D. M. Vriezema, J. J. L. M. Cornelissen, J. C. M. van Hest, R. J. M. Nolte, and A. E. Rowan. 2008. Enzymes containing porous polymersomes as nano reaction vessels for cascade reactions. *Organic & Biomolecular Chemistry* 6:4315-4318.
3. Miller, M. B., and B. L. Bassler. 2001. Quorum sensing in bacteria. *Annu Rev Microbiol* 55:165-199.
4. Matzuk, M. M., K. H. Burns, M. M. Viveiros, and J. J. Eppig. 2002. Intercellular communication in the mammalian ovary: Oocytes carry the conversation. *Science* 296:2178-2180.
5. Bolinger, P. Y., D. Stamou, and H. Vogel. 2008. An integrated self-assembled nanofluidic system for controlled biological chemistries. *Angew Chem Int Edit* 47:5544-5549.
6. Grzelakowski, M., O. Onaca, P. Rigler, M. Kumar, and W. Meier. 2009. Immobilized Protein-Polymer Nanoreactors. *Small* 5:2545-2548.
7. Choi, J. M., B. Yoon, K. Choi, M. L. Seol, J. M. Kim, and Y. K. Choi. 2012. Micropatterning Polydiacetylene Supramolecular Vesicles on Glass Substrates using a Pre-Patterned Hydrophobic Thin Film. *Macromol Chem Phys* 213:610-616.
8. Bally, M., K. Bailey, K. Sugihara, D. Grieshaber, J. Voros, and B. Stadler. 2010. Liposome and Lipid Bilayer Arrays Towards Biosensing Applications. *Small* 6:2481-2497.
9. Stamou, D., C. Duschl, E. Delamarche, and H. Vogel. 2003. Self-assembled microarrays of attoliter molecular vessels. *Angew Chem Int Edit* 42:5580-5583.
10. Wittenberg, N. J., H. Im, T. W. Johnson, X. H. Xu, A. E. Warrington, M. Rodriguez, and S. H. Oh. 2011. Facile Assembly of Micro- and Nanoarrays for Sensing with Natural Cell Membranes. *Acs Nano* 5:7555-7564.
11. Kim, J. M., E. K. Ji, S. M. Woo, H. W. Lee, and D. J. Ahn. 2003. Immobilized polydiacetylene vesicles on solid substrates for use as chemosensors. *Adv Mater* 15:1118-+.
12. Kamat, N. P., J. S. Katz, and D. A. Hammer. 2011. Engineering Polymersome Protocells. *J Phys Chem Lett* 2:1612-1623.
13. Nourian, Z., W. Roelofsen, and C. Danelon. 2012. Triggered Gene Expression in Fed-Vesicle Microreactors with a Multifunctional Membrane. *Angew Chem Int Edit* 51:3114-3118.
14. Usta, O. B., A. Alexeev, G. Zhu, and A. C. Balazs. 2008. Modeling microcapsules that communicate through nanoparticles to undergo self-propelled motion. *Acs Nano* 2:471-476.
15. Utada, A. S., E. Lenceau, D. R. Link, P. D. Kaplan, H. A. Stone, and D. A. Weitz. 2005. Monodisperse double emulsions generated from a microcapillary device. *Science* 308:537-541.
16. Duncanson, W. J., T. Lin, A. R. Abate, S. Seiffert, R. K. Shah, and D. A. Weitz. 2012. Microfluidic synthesis of advanced microparticles for encapsulation and controlled release. *Lab on a Chip* 12:2135-2145.
17. Kamat, N. P., M. H. Lee, D. Lee, and D. A. Hammer. 2011. Micropipette aspiration of double emulsion-templated polymersomes. *Soft Matter* 7:9863-9866.

18. Desai, R. A., M. K. Khan, S. B. Gopal, and C. S. Chen. 2011. Subcellular spatial segregation of integrin subtypes by patterned multicomponent surfaces. *Integr Biol* 3:560-567.
19. Martino, C., S. H. Kim, L. Horsfall, A. Abbaspourrad, S. J. Rosser, J. Cooper, and D. A. Weitz. 2012. Protein Expression, Aggregation, and Triggered Release from Polymersomes as Artificial Cell-like Structures. *Angew Chem Int Edit* 51:6416-6420.
20. Koster, S., F. E. Angile, H. Duan, J. J. Agresti, A. Wintner, C. Schmitz, A. C. Rowat, C. A. Merten, D. Pisignano, A. D. Griffiths, and D. A. Weitz. 2008. Drop-based microfluidic devices for encapsulation of single cells. *Lab on a Chip* 8:1110-1115.
21. Meeuwissen, S. A., M. F. Debets, and J. C. M. van Hest. 2012. Copper-free click chemistry on polymersomes: pre- vs. post-self-assembly functionalisation. *Polym Chem-Uk* 3:1783-1795.
22. Grabarek, Z., and J. Gergely. 1990. Zero-Length Crosslinking Procedure with the Use of Active Esters. *Anal Biochem* 185:131-135.
23. Tan, J. L., W. Liu, C. M. Nelson, S. Raghavan, and C. S. Chen. 2004. Simple approach to micropattern cells on common culture substrates by tuning substrate wettability. *Tissue Eng* 10:865-872.
24. Ruiz, S. A., and C. S. Chen. 2007. Microcontact printing: A tool to pattern. *Soft Matter* 3:168-177.
25. Hammer, D. A., and M. Tirrell. 1996. Biological adhesion at interfaces. *Annu Rev Mater Sci* 26:651-691.
26. Sekula, S., J. Fuchs, S. Weg-Remers, P. Nagel, S. Schuppler, J. Fragala, N. Theilacker, M. Franueb, C. Wingren, P. Ellmark, C. A. K. Borrebaeck, C. A. Mirkin, H. Fuchs, and S. Lenhart. 2008. Multiplexed Lipid Dip-Pen Nanolithography on Subcellular Scales for the Templating of Functional Proteins and Cell Culture. *Small* 4:1785-1793.
27. Yoshina-Ishii, C., and S. G. Boxer. 2003. Arrays of mobile tethered vesicles on supported lipid bilayers. *Journal of the American Chemical Society* 125:3696-3697.

Chapter 7

Future Directions

Visualizing the Cytoskeleton

The cell analyses contained in the previous chapters largely considered effects of various small molecule perturbations (i.e. chemoattractants, cytoskeletal inhibitors, and antibodies) on biological metrics at the whole cell length scale. We were able to infer the effect of these perturbations by quantifying how the cell's response under treatment differed from the equivalent control cases with respect to metrics of cell shape and motility. However, there is a wealth of information to be gained by visualizing the organization of the cytoskeleton and its rearrangement under the previously explored molecular perturbations. In practice, pilot efforts at fixing neutrophils in the keratocyte-like phenotype have revealed sensitivity in the post-fixation shape to the fixation method. Ubiquitous fixation methodologies such as 10 % neutral buffered formalin and 4 % paraformaldehyde (PFA) (Fig. 7.1 A) were found to abrogate features like the keratocyte-like crescent shape and ruffled lamellipodium visible in live neutrophils (Fig. 7.1 C). With respect to cell area, PFA treatment resulted in a 40 % decrease in total cell area and 60 % increase in nuclear area as compared to live cells in the keartocyte-like phenotype (Fig. 7.1 D).

We previously demonstrated (1) that neutrophils employ the promiscuous integrin receptor MAC-1 ($\alpha_M\beta_2$) to support their haptokinetic motility on fibronectin (FN) fields

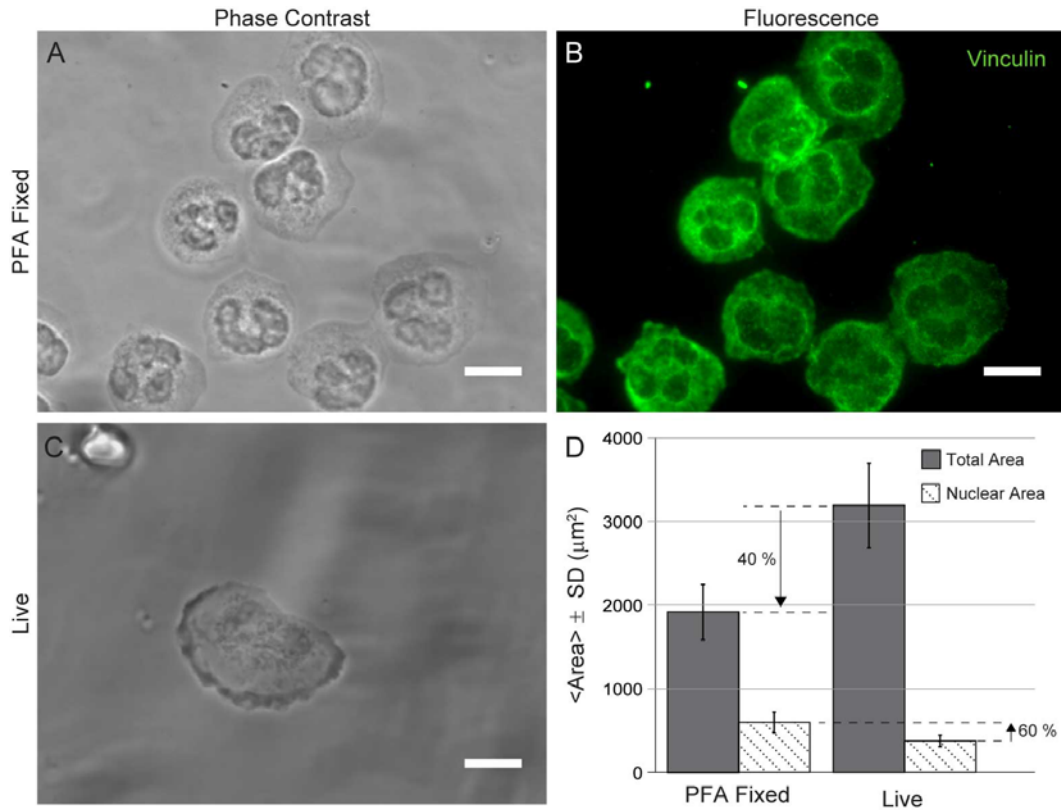


Figure 7.1 Loss of phenotype after paraformaldehyde fixation. (A) Phase contrast image of neutrophils fixed under 4 % paraformaldehyde and permeabilized with 0.5 % TritonX-100. (B) Corresponding fluorescence image of same neutrophils after vinculin staining. (C) Live phase contrast image of keratocyte-like neutrophil. (D) Quantification of paraformaldehyde (PFA) fixation-induced morphological changes. Compared to live cells the total cell area decreased by 40 % whereas the nuclear area increased by 60%. Error bars are \pm standard deviation from $\langle n \rangle = 11$ cells/condition. Scalebars are 10 μm .

(Figs. 3.12 and 3.13). Raptis and coworkers have established that clustering of the β_2 chain (CD11b) is critical in the execution of terminal effector functions like proteolytic enzyme secretion and reactive oxygen intermediate production (2). In mesenchymal cells integrin clustering is a precursor to the formation of three dimensional adhesive plaques called focal adhesions. These adhesion units are composed of numerous protein scaffolds which individually or cumulatively achieve a mechanosensitive linkage between the extracellular integrins and intracellular cytoskeleton (3). One marker of the formation of these focal adhesions is vinculin (4). When neutrophils exhibiting the keratocyte-like phenotype were fixed under 4 % PFA and permeabilized under 0.5 % TritonX-100, vinculin staining resulted in a uniform signal across the cell body (Fig. 7.1 B) with no evidence of discrete adhesive plaques.

However, when an alternative fixation strategy was employed, developed specifically to stabilize microtubules, we found improved phenotype preservation (Fig. 7.2). This alternative fixation strategy using microtubule stabilizing buffer (MTSB) was recommended to us by Ravi A. Desai, PhD from his first hand experience with locomoting NRK-52E cells (a rat kidney cell line available from ATCC) (5). NRK-52E cells bear a striking resemblance to the shape of fish keratocytes (6-7) and our keratocyte-like neutrophils (1) having a very broad leading edge lamellipodium. In contrast to the diffuse vinculin signal of PFA-fixed keratocyte-like neutrophils, both amoeboid and keratocyte-like neutrophils fixed with MTSB and stained for vinculin, revealed distinct plaques at the periphery of the cell (red double-headed arrows of Fig. 7.2). In the amoeboid case, the vinculin structures were predominately located in the rear and sides of

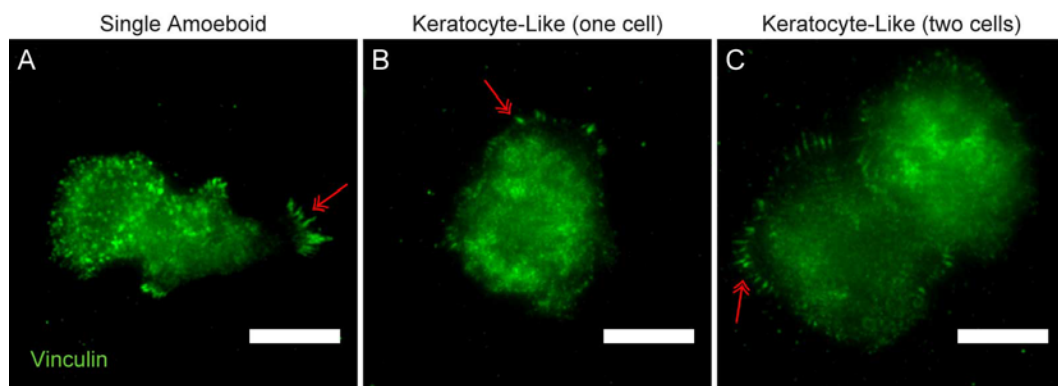


Figure 7.2 Improved phenotype stability and vinculin plaque detection with microtubule stabilizing buffer fixation. Vinculin staining of microtubule stabilizing buffer fixed neutrophils exhibiting: (A) amoeboid phenotype, (B) keratocyte-like phenotype (one cell), and (C) keartocyte-like phenotype (two cells). Red double headed arrows denote vinculin plaques not observed under PFA fixation and TritonX-100 permeabilization of Fig. 7.1 B. Scalebars = 10 μm .

the cell (Fig. 7.2 A) while they uniformly decorated the periphery in the keratocyte-like morphology (Fig. 7.2 B and C).

Previously, Hammer Lab achieved traction measurements of neutrophils chemotaxing in the amoeboid style and observed that the largest force generation was spatially located in the rearward uropod (8). This asymmetric contractility was interpreted to be the basis of the cell's ability to propel its cytoplasm forward in the absence of extensive adhesive contact with the underlying substrate. The asymmetric vinculin plaques seen in the rearward uropod of the amoeboid neutrophil (Fig. 7.2 A) are consistent with the highest traction generation also being observed in the rear of the cell (8). That vinculin plaques uniformly decorate the periphery of keratocyte-like neutrophils is also a possible explanation as to why the cells are slower and more directionally persistent than their amoeboid counterparts (Figs. 3.9, 4.6, and 4.7).

It is interesting to speculate as to whether or not the improved resolution of adhesive plaques in neutrophils *via* MTSB fixation implies that microtubules (MT) are critical to phenotype preservation in living neutrophils. In the zebrafish model of leukocyte migration, the MT organizing center is positioned between the leading edge of the migrating cell and the nucleus (9). Furthermore, perturbation of MT polymerization and depolymerization kinetics was found to impact neutrophil homing and motility. Exploring MT dynamics in the context of the keratocyte-like morphology would be novel.

Also worth considering is the nature of the nonlinear change in cell area as compared to nuclear area under PFA fixation (Fig. 7.1 D). Under PFA conditions, cell area was found to decrease by 40 % while nuclear area was found to increase by 60 %.

The reduction in cell area could be rationalized on the basis of TritonX-100 generating membrane pores and causing cell swelling. A volume increase could have the apparent affect of a reduced projected area. However in this swelling model less nuclear compression would be anticipated and its known that TritonX-100 also punctures the nuclear envelope (10). Thus, why reduced compression and nuclear swelling would result in an increase in projected nuclear area is unclear.

Alternatively, the nonlinearity could be a consequence of induced pores in the cytoplasmic and nuclear envelopes resulting in differing mechanical properties of the two components. TritonX-100 could be targeting a lipid component differentially expressed in the two envelopes. An increase in nuclear spread area could be interpreted as the envelope becoming more mechanically flaccid after pore formation. The mechanics of the cellular nucleus and its molecular basis represent a large and robust field of study (11). Within this field, human neutrophils are a particularly interesting subset of cells to study owing to their characteristic tri-lobed nuclei. It has been observed that lobulated nuclei have reduced lamin A/C content compared to rounded nuclei (12) and that lamin concentration and composition control nuclear stiffness.

Fixation and Vinculin Staining Method Notes

Neutrophils were plated on fields of FN in the usual manner (see Materials and Methods Chapters 3 and 4). For PFA fixation cells were incubated in a final concentration of 4 % PFA in phosphate buffered saline (PBS) from a freshly opened stock bottle of 16 % electron microscopy grade methanol-free PFA (Electron Microscopy Sciences, #15710) for 10 min at room temperature (RT). Cells were rinsed 3X in PBS and permeabilized *via* 10 min incubation under 0.5 % TritonX-100 solution (MP

Biomedicals, #807423). Cells were rinsed 3X in PBS and blocked with 5% bovine serum albumin (BSA) (Sigma, #A70030-100G) solution for 1 hr at RT (13).

For improved phenotypic preservation *via* MTSB fixation, a stock of 10X MTSB was prepared in advance. The final working concentration of MTSB consisted of: 0.1 M PIPES at pH 6.75, 1 mM EGTA, 1 mM MgSO₄, 4 % (w/v) Poly(ethylene glycol) 8000, 1 % TritonX-100, and 2 % Paraformaldehyde. Cells were incubated for 10 min at 37 °C under MTSB and rinsed 3X in PBS. Very gentle rinsing was performed to avoid shearing cell membranes. After rinsing, cells were incubated in 2 % BSA in PBS for 1 hr at RT. Cells were rinsed 3X in PBS and stained for vinculin.

Vinculin staining was a two step immunocytochemical preparation. Fixed and permeabilized cells were incubated at a 1:200 volume dilution of stock mouse mAb to vinculin (hVIN-1, Abcam, # ab11194) for 1 hr at RT. Cells were subsequently rinsed 3X with PBS and incubated in a 1:400 volume dilution of stock AlexaFluor488 goat-anti-mouse IgG₁ (Invitrogen, #A-11001) for 1 hr at RT. Finally, cells were rinsed 3X with PBS and imaged.

Neutrophil Motility on mPADs

An aim not fully realized in this thesis was to elicit neutrophil motility on mPADs and measure the corresponding traction maps. Previously, Brendon Ricart in the Hammer laboratory measured traction maps of dendritic cells (DCs) chemotaxing across large diameter, soft post arrays (14) (post specifications are recorded in Fig. 7.3, “Large, Soft”). The Ricart experiments were an impressive combination of device engineering and biological insight and we initially attempted to simply substitute human neutrophils for DCs in his experimental setup. However, we found that his post geometry failed to

| | Large, Soft | Small, Soft | Small, Stiff | Notes |
|---|-------------------------------|-------------------------------------|--------------------------|---|
| Chen Lab Master Designation | "1.83 μm , #12" | "0.8 μm , #5" | "0.8 μm , #2" | |
| Post Diameter (μm) | 1.83 | 0.604 \pm 0.031 | 0.635 \pm 0.016 | |
| Post Length (μm) | 12.9 | 5.576 \pm 0.286 | 2.116 \pm 0.051 | |
| Post Aspect Ratio (L/D) | 7 | 9 | 3 | |
| Array Pitch (μm) | 5 | 1.932 \pm 0.002 | 1.966 \pm 0.003 | |
| $A_{\text{islands}}/A_{\text{continuous}} \times 100$ (%) | 12 | 18 | 19 | $A_{\text{small islands}} = SA_{\text{hemisphere}} = 2 \cdot \pi \cdot r^2$ |
| k_{spring} (pN/nm) | 1.92 | 0.28 \pm 0.09 | 6.32 \pm 0.031 | $k_{\text{spring}} = 3 \cdot E \cdot \pi \cdot d^4 / (64 \cdot L^3)$ $E_{\text{PDMS}} = 2.5 \text{ MPa}$ Yang et al. 2011. Nat Protocol |
| Warp Correction | 0.90 | 0.93 | 0.79 | Schoen et al. 2010. Nano Letters |
| $k_{\text{spring, Schoen}}$ (pN/nm) | 1.73 | 0.26 | 4.97 | |
| G (pure shear) (kPa) | 9 | 5 | 42 | $G = k_{\text{spring}} \cdot L / (\pi \cdot r^2)$ |
| E_{eff} (kPa) | 1.5 | 0.7 | 14 | $E_{\text{eff}} = 9 \cdot k_{\text{spring}} / (4 \cdot \pi \cdot r)$ Ghibaudo et al. 2008. Soft Matter |
| G_{eff} (kPa) | 0.5 | 0.2 | 5 | $G_{\text{eff}} = E_{\text{eff}} / (2 \cdot (1 + \nu))$ $\nu_{\text{PDMS}} \sim 0.5$ |
| Used In | Ricart et al. 2011 Biophys J. | Chapter 5 (Submitted to Biophys J.) | Chapter 7 | |

Figure 7.3 Summary of mPADs specifications. Error bars are \pm standard deviations from scanning electron micrograph measurements.

elicit neutrophil spreading (Fig. 7.4 A) as was previously observed in DCs. The FN coverage of Ricart post arrays relative to a continuous field was only 12 %. In Chapter 4 we previously demonstrated that neutrophils perceive island geometries with 20 % coverage relative to a continuous field as if the ensemble of islands were a continuous field. In that context we can now infer that there is a critical protein coverage threshold that resides between 12 % and 20 %, below which neutrophils no longer perceive islands as continuous. This is highly consistent with the findings of Lehnert and coworkers in mesenchymal cells where protein coverage less than 20 % dramatically reduced cell spreading (15).

However, when small diameter mPADs with protein coverage of 20% were fabricated, neutrophils were induced to spread (Fig. 7.4 B and C) as was established previously in Chapter 5 (Fig. 5.1 B). While increasing protein coverage recovered spreading, the stiffness of the posts became a critical factor in eliciting motility. On small, soft posts neutrophils spread but did not translocate (Fig. 7.4 B). Neutrophil motility in the region of post collapse adjacent to the posts served as a useful control in this FOV. When the posts were shortened to increase rigidity, a fraction of neutrophils (~ 25 % of total cells plated) were motile (Fig. 7.4 C).

There are a variety of stiffness definitions to describe the discretized environment a cell experiences on mPADs. Figure 7.3 summarizes the specifications of each post array discussed in this chapter as well as a number of stiffness metrics. On the simplest level, each pillar can be modeled as a cantilever subjected to a load at its unconstrained terminus (16) in which case the material spring constant (k_{spring}) is a natural description of pillar stiffness. Small, soft and stiff post spring constants were ~ 0.3 pN/nm and ~ 6

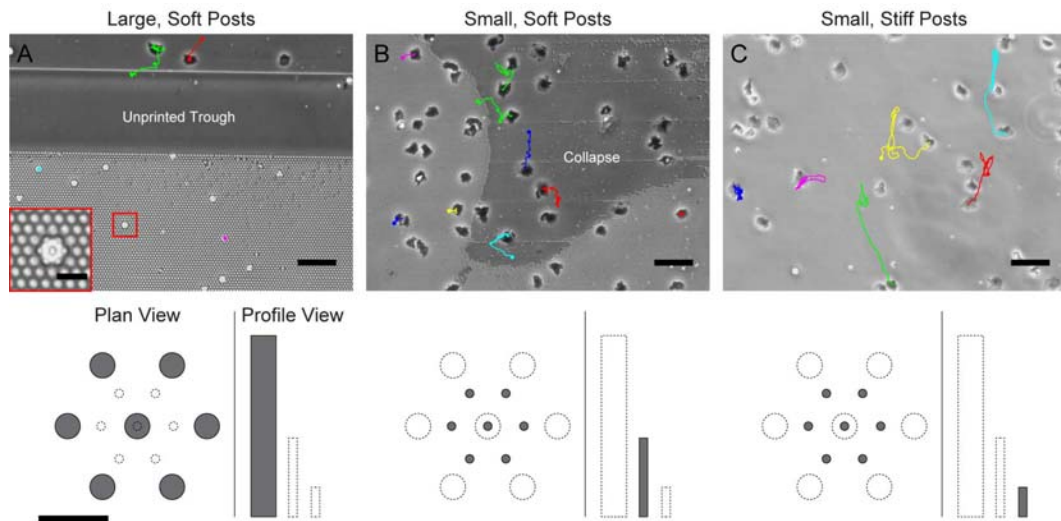


Figure 7.4 Neutrophil motility on post arrays is stiffness dependent. (A) The large, soft posts used to study dendritic cell traction forces during chemotaxis (Ricart et al. 2011. *Biophys J*), fail to elicit spreading in human neutrophils. Neutrophils assume a hexagonal phenotype of one unit cell but do not spread (see inset). Neutrophils on the continuous field of fibronectin in the same FOV are motile. (B) On small, soft posts used to study neutrophil spreading forces (Chapter 5) neutrophils spread but are not motile. A region of collapse after printing supports neutrophil motility in the same FOV as the non-motile cells residing on the posts. (C) On small, stiff posts a fraction of neutrophils are motile. Schematics are to scale and correspond to each experimental condition above. Micrograph scalebars = 50 μm . Inset scalebar = 10 μm . Schematic scalebar = 5 μm .

pN/nm, respectively. Theoretical work done by Schoen and coworkers established a series of corrections to these spring constants as a function of post aspect ratio. The corrections account for the contribution of pillar tilting and base warping to the measured free terminus deflection (17). The magnitude of the correction is minimal for the small, soft posts (7 % reduction) and large for the small, stiff posts (21 % reduction), but the relative ten-fold difference in stiffness between substrates is retained. While the spring constant is a natural description of single pillar stiffness, it is ambiguous with respect to the microscope or ensemble (i.e. multi-post) stiffness perceived by a cell. In a macroscopic context it is more natural to describe the arrays in terms of Young's moduli (E) or shear moduli (G). Applying a simple definition of pure shear we find $G_{\text{small, soft}} \sim 5$ kPa and $G_{\text{small, stiff}} \sim 42$ kPa. Both of these values are within the physiologically relevant domain of stiffnesses (18). The appropriateness of a pure shear model for neutrophil engagement of mPADs is an assumption, but one supported by the empirical work of Lemmon and coworkers which demonstrated that shear is a larger contribution to post deflection than torque (19). Alternatively, Ladoux and coworkers developed a theoretical description of effective array stiffness by solution of the Green's function for a discretized substrate (under certain governing assumptions) (20). The Ladoux model estimates the Young's moduli of our post arrays as $E_{\text{small, soft}} \sim 0.7$ kPa and $E_{\text{small, stiff}} \sim 14$ kPa, substantially softer than anticipated by the local pure shear model. While different definitions of stiffness clearly yield different values, the transcendent point is that the small, soft and small, stiff arrays used in these analyses differ by an order of magnitude.

Increasing post stiffness was necessary to elicit neutrophil motility, but can the tractions (i.e. substrate deformations) still be resolved? On soft but not stiff arrays

neutrophil-induced pillar deflections were readily observed at 100X magnification (Fig. 7.5 B). To determine if neutrophil-induced pillar deflections on stiff arrays were small but nonzero we performed a complete traction analysis of both conditions. The displacement heat maps for all posts beneath and surrounding the three cells in each FOV reveal comparable background noise (i.e. deflections observed in posts surrounding but not under the cells). Quantifying the mean apparent deflection of background posts around the cells we find comparable noise floors on the order of 20-40 nm. Applying the respective material spring constants to these calculated deflection floors resulted in substantially different force floors (Fig. 7.5 D). The mean force per pillar of neutrophil tractions on soft arrays ($\langle F_{\text{cell/post}} \rangle \sim 123$ pN) was 10.3X in excess of the 12 pN force floor, whereas on stiff arrays the mean force per pillar ($\langle F_{\text{cell/post}} \rangle \sim 182$ pN) was only 1.2X in excess of the 150 pN force floor. Hence, the stiff arrays resulted in a substantially reduced signal-to-noise ratio. While our confidence in the measured forces attributed to neutrophils on stiff arrays is low, it was interesting to compute the average per-post strain energy imparted by the cells on each of the arrays (Fig. 7.5 E). Strain energy was not constant but an order of magnitude less on stiff arrays than soft arrays. This suggests a non-linear response in neutrophil traction generation on stiff substrates. If we make the assumption that total cell energy expenditure is constant, the reduction in strain energy on stiff substrates could be interpreted as the cell shunting more energy to other processes such as motility. Indeed, it was only on the stiff post arrays that neutrophil motility was observed, suggesting an inverse relationship between cell speed and contractility.

The pragmatic implication of this section is that the small but stiff arrays used in these traction measurements are slightly stiffer than ideal. Unfortunately the silicon

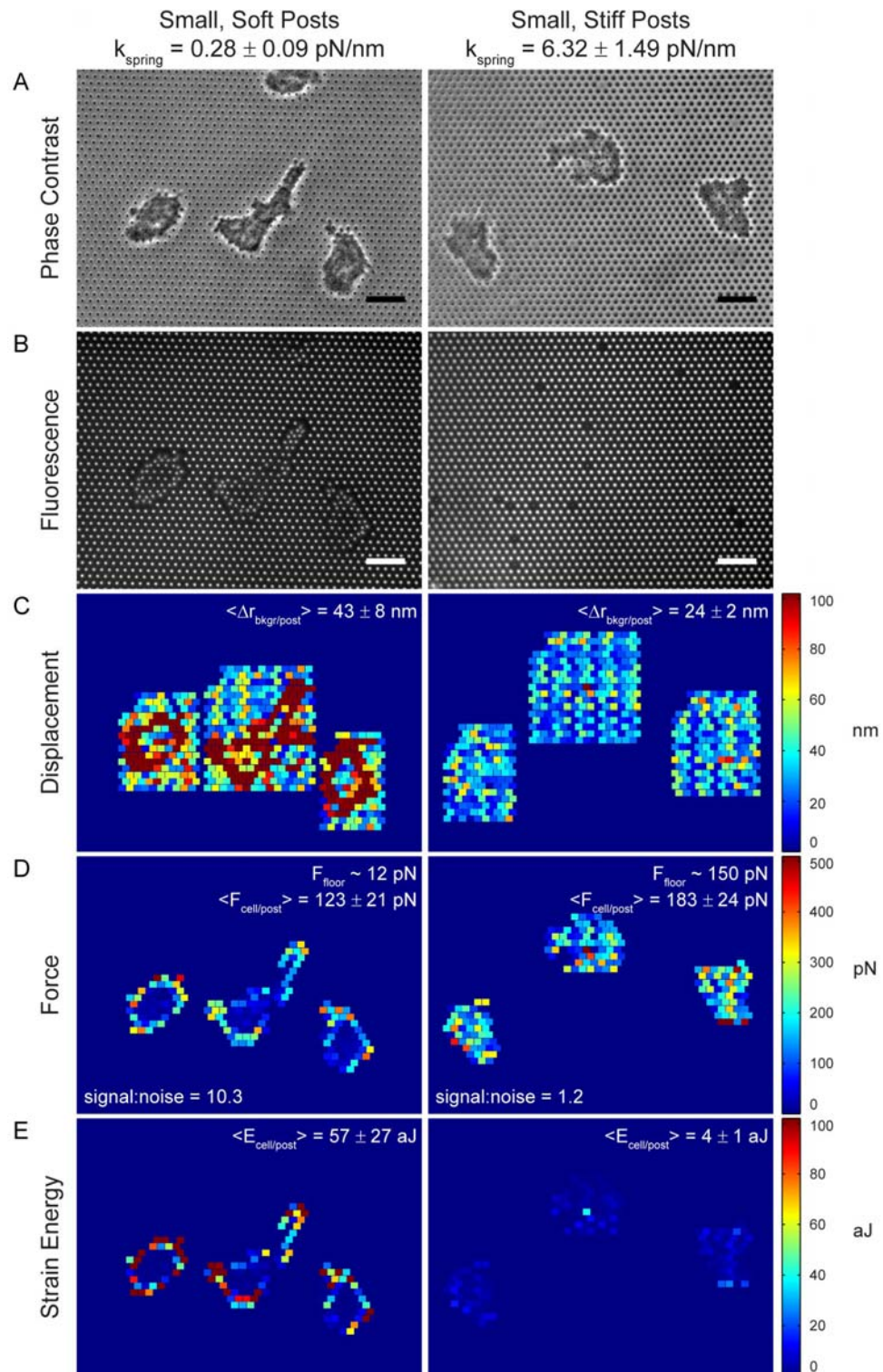


Figure 7.5 Neutrophil traction generation is stiffness dependent. Caption over.

Figure 7.5 Neutrophil traction generation is stiffness dependent (continued). Neutrophils on small, soft pillars (left column data, corresponding to 7.3 *B* condition) or small, stiff pillars (right column data, corresponding to 7.3 *C* condition). (*A*) Phase contrast. (*B*) Fluorescence channel. (*C*) Per-post displacements in rectangular ROI around cell. (*D*) Per-post forces of all posts within cell contact area. Force floor was calculated by applying spring constant to mean background displacement of *B*. (*E*) Per-post strain energy of all posts within cell contact area. Scalebars = 10 μm . Error bars are \pm standard deviation from means of $n = 3$ cells.

masters supplied to Hammer laboratory by Chen laboratory do not presently have an intermediate stiffness between the two arrays tested. Therefore fabrication of a new master is necessary or off-ratio (i.e. not 10:1 base:cure (w/w)) PDMS formulations will be required. While the later avenue is certainly more facile, the reproducibility of off-ratio PDMS formulations with respect to cured stiffness is of concern. However, off-ratio PDMS formulations have been used in mechanically-sensitive applications with success by Huh laboratory (21).

Additional Small Molecule Inhibitor Work

The following comments were informed by discussions with and recommendations by Professor Christopher S. Chen. In Chapter 5 we explored the mechanism by which a neutrophil transitions from a quiescent sphere to a spread and adherent phenotype. Our small molecule inhibitor work identified that spreading was not analogous to lamellipodium formation (no effect with CK666 treatment) but was sensitive to perturbations of actin kinetics which alter actin cortex mechanics (jasplakinolide stiffened the cortex whereas cytocholasin B softened the cortex). These results were previously summarized in Figure 5.7. Our work suggested that neutrophil ligation of FN on the pillar tips induced cell remodeling of the cytoskeleton that resulted in a reduction of cortical stiffness. We previously showed neutrophil haptokinetic engagement of FN was integrin mediated but an outstanding question that remains is the nature of the integrin-actin cortex linkage. Molecular targets thought to be critical in the integrin-actin cortex linkage include focal adhesion kinase (FAK) (22), Src (23), and the TRPV calcium channel family (24). Small molecule inhibitors are commercially

available against these targets and are fast acting, which is important as the timescale of neutrophil spreading is on the order of seconds.

Vesicle Haptokinesis

The original motivation for pursuing the vesicle patterning work of Chapter 6 was to develop an experimental platform for the study of autonomous vesicle motion. Computational modeling done in Professor Anna C. Balazs's group predicted the coordinated motion of a system of semi-permeable particles (or capsules) releasing haptic ligands into their environment (25). The release of haptic ligands generated gradients of local adhesivity which, coupled with hydrodynamic entrainment, resulted in streams of particles moving autonomously in two dimensions.

As an intermediate step en route to empirically realizing this ambitious signaling system, we sought to elicit haptokinetic motion of the Chapter 6 surface-biotinylated microfluidic vesicles by repeatedly and randomly printing small islands of NeutrAvidin (Fig. 7.6 A). The hypothesis was that if the density of islands was sufficiently high the vesicles, by virtue of stochastic formation and dissolution of receptor-ligand bonds, could be induced to haptokinetically move across the surface. Consistent with our previous findings (26) we were able to immobilize surface active micronscale vesicles on 10 μm diameter adhesive islands. Unfortunately, our multi-print method failed to achieve an adequate density of islands to robustly test our autonomous motion hypothesis. At the time these pilot experiments were performed the square lattices of 10 μm diameter, 50 μm pitch were the smallest arrays available to us. However, as demonstrated in Chapter 4, we can now reliably achieve submicron arrays of hexagonally arranged islands with 0.9 μm diameter and 1.9 μm pitch *via* the stamp-off method of microcontact printing

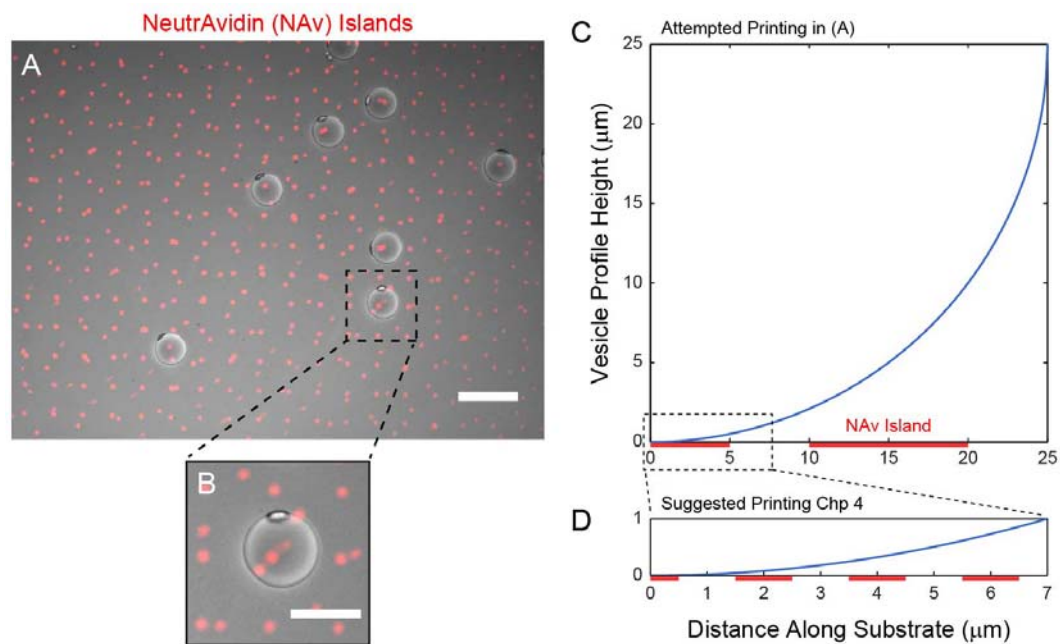


Figure 7.6. Attempted vesicle haptokinesis experiment. (A) Multi-printing of 10 μm islands at 50 μm pitch yielded a few regions where islands were separated by 5 - 10 μm . No multi-island hopping was observed. Scalebar = 100 μm . (B) Expanded view of a single vesicle in A, proximal to 2-3 NeutrAvidin islands. Scalebar = 50 μm . (C) Geometric construction illustrating vesicle surface height above substrate. Islands must be tightly spaced to facilitate multi-island contact of a single vesicle. (D) Island arrays of Chapter 4 will place many more islands in proximity to vesicle surface than the attempt in C yielded.

(Fig. 4.1 *J* and 4.3 *C*). Switching to Chapter 4 island arrays would achieve a substantial increase in island density beneath the biotinylated vesicles (Fig. 7.6 *D*)

An additional empirical parameter available for tuning is the rigidity of the vesicle membranes themselves. The double emulsion templating production method results in a homogeneous population of nearly spherical vesicles. To improve yield during production and solvent evaporation, the membranes are stabilized with surfactants such as BSA or Pluronic F-127. As a simple geometric construction of rigid spherical vesicles makes clear (Fig. 7.6 *C*), the contact interface over which vesicle-substrate, biotin-avidin, interactions can occur is quite limited as the surface of the vesicle rises quadratically over the horizontal plane of islands. Thus, while increasing island density will improve the number of islands available for ligation in the contact zone (Fig. 7.6 *D*), decreasing vesicle rigidity would increase the size of this contact interface as well. Reducing the surfactant content during production and solvent-evaporation is one avenue for consideration. However in our experience reductions in surfactant concentration usually result in poor vesicle survival during solvent evaporation.

An alternative strategy is to return to the thin film method of rehydration for vesicle generation (27) which results in a more flaccid, pancake-like architecture. The cost will be a substantial increase in polydispersity with respect to vesicle size that is a consequence of this production strategy. While not ideal, the Hammer laboratory has previously demonstrated that porphyrin incorporation in vesicle membranes generated by thin film rehydration renders the vesicles amenable to photo-inducible rupture (27). This controlled rupture mechanism could be ideal in generating local adhesive gradients more

akin to the Balazs computational model than static adhesive fields achieved by microcontact printing.

References

1. Henry, S. J., J. C. Crocker, and D. A. Hammer. 2014. Ligand density elicits a phenotypic switch in human neutrophils. *Integrative Biology* 6:348-356.
2. Raptis, S. Z., S. D. Shapiro, P. M. Simmons, A. M. Cheng, and C. T. Pham. 2005. Serine protease cathepsin G regulates adhesion-dependent neutrophil effector functions by modulating integrin clustering. *Immunity* 22:679-691.
3. Kanchanawong, P., G. Shtengel, A. M. Pasapera, E. B. Ramko, M. W. Davidson, H. F. Hess, and C. M. Waterman. 2010. Nanoscale architecture of integrin-based cell adhesions. *Nature* 468:580-584.
4. Chen, C. S., J. L. Alonso, E. Ostuni, G. M. Whitesides, and D. E. Ingber. 2003. Cell shape provides global control of focal adhesion assembly. *Biochemical and Biophysical Research Communications* 307:355-361.
5. Desai, R. A., S. B. Gopal, S. Chen, and C. S. Chen. 2013. Contact inhibition of locomotion probabilities drive solitary versus collective cell migration. *J R Soc Interface* 10:20130717.
6. Barnhart, E. L., K. C. Lee, K. Keren, A. Mogilner, and J. A. Theriot. 2011. An adhesion-dependent switch between mechanisms that determine motile cell shape. *PLoS Biol* 9:e1001059.
7. Lee, J., and K. Jacobson. 1997. The composition and dynamics of cell-substratum adhesions in locomoting fish keratocytes. *J Cell Sci* 110 (Pt 22):2833-2844.
8. Smith, L. A., H. Aranda-Espinoza, J. B. Haun, M. Dembo, and D. A. Hammer. 2007. Neutrophil traction stresses are concentrated in the uropod during migration. *Biophys J* 92:L58-60.
9. Yoo, S. K., P. Y. Lam, M. R. Eichelberg, L. Zasadil, W. M. Bement, and A. Huttenlocher. 2012. The role of microtubules in neutrophil polarity and migration in live zebrafish. *J Cell Sci* 125:5702-5710.
10. Frederiks, W. M., J. James, C. Arnouts, S. Broekhoven, and J. Morreau. 1978. The influence of Triton X-100 on the nuclear envelope of the isolated liver cell nuclei. *Cytobiologie* 18:254-271.
11. Dahl, K. N., A. J. Ribeiro, and J. Lammerding. 2008. Nuclear shape, mechanics, and mechanotransduction. *Circ Res* 102:1307-1318.
12. Yabuki, M., T. Miyake, Y. Doi, T. Fujiwara, K. Hamazaki, T. Yoshioka, A. A. Horton, and K. Utsumi. 1999. Role of nuclear lamins in nuclear segmentation of human neutrophils. *Physiol Chem Phys Med NMR* 31:77-84.
13. 2006. Bovine serum albumin (BSA). *Cold Spring Harbor Protocols* 2006:pdb.rec8452.
14. Ricart, B. G., M. T. Yang, C. A. Hunter, C. S. Chen, and D. A. Hammer. 2011. Measuring traction forces of motile dendritic cells on micropost arrays. *Biophys J* 101:2620-2628.
15. Lehnert, D., B. Wehrle-Haller, C. David, U. Weiland, C. Ballestrem, B. A. Imhof, and M. Bastmeyer. 2004. Cell behaviour on micropatterned substrata: limits of extracellular matrix geometry for spreading and adhesion. *Journal of Cell Science* 117:41-52.

16. Beer, F. P., E. R. Johnston, and J. T. DeWolf. 2006. *Mechanics of Materials*. McGraw-Hill Higher Education, Boston.
17. Schoen, I., W. Hu, E. Klotzsch, and V. Vogel. 2010. Probing cellular traction forces by micropillar arrays: contribution of substrate warping to pillar deflection. *Nano Lett* 10:1823-1830.
18. Engler, A. J., S. Sen, H. L. Sweeney, and D. E. Discher. 2006. Matrix elasticity directs stem cell lineage specification. *Cell* 126:677-689.
19. Lemmon, C. A., N. J. Sniadecki, S. A. Ruiz, J. L. Tan, L. H. Romer, and C. S. Chen. 2005. Shear force at the cell-matrix interface: enhanced analysis for microfabricated post array detectors. *Mech Chem Biosyst* 2:1-16.
20. Ghibaudo, M., A. Saez, L. Trichet, A. Xayaphoummine, J. Browaeys, P. Silberzan, A. Buguin, and B. Ladoux. 2008. Traction forces and rigidity sensing regulate cell functions. *Soft Matter* 4:1836-1843.
21. Huh, D., B. D. Matthews, A. Mammoto, M. Montoya-Zavala, H. Y. Hsin, and D. E. Ingber. 2010. Reconstituting Organ-Level Lung Functions on a Chip. *Science* 328:1662-1668.
22. Mitra, S. K., D. A. Hanson, and D. D. Schlaepfer. 2005. Focal adhesion kinase: in command and control of cell motility. *Nat Rev Mol Cell Biol* 6:56-68.
23. Kuga, T., M. Hoshino, Y. Nakayama, K. Kasahara, K. Ikeda, Y. Obata, A. Takahashi, Y. Higashiyama, Y. Fukumoto, and N. Yamaguchi. 2008. Role of Src-family kinases in formation of the cortical actin cap at the dorsal cell surface. *Exp Cell Res* 314:2040-2054.
24. Sokabe, T., T. Fukumi-Tominaga, S. Yonemura, A. Mizuno, and M. Tominaga. 2010. The TRPV4 Channel Contributes to Intercellular Junction Formation in Keratinocytes. *Journal of Biological Chemistry* 285:18749-18758.
25. Kolmakov, G. V., V. V. Yashin, S. P. Levitan, and A. C. Balazs. 2010. Designing communicating colonies of biomimetic microcapsules. *Proc Natl Acad Sci U S A* 107:12417-12422.
26. Kamat, N. P., S. J. Henry, D. Lee, and D. A. Hammer. 2013. Single-Vesicle Patterning of Uniform, Giant Polymersomes into Microarrays. *Small*:n/a-n/a.
27. Kamat, N. P., G. P. Robbins, J. Rawson, M. J. Therien, I. J. Dmochowski, and D. A. Hammer. 2010. A Generalized System for Photoresponsive Membrane Rupture in Polymersomes. *Advanced Functional Materials* 20:2588-2596.

Appendix A

Custom MATLAB Code for Analysis of Neutrophil Motility

Introduction

The purpose of this appendix is to provide the reader with more detail regarding the data analysis workflow employed to compute neutrophil motility statistics. Broadly speaking the workflow consisted of capturing timelapse images of neutrophil migration (Fig. A.1 A), identifying the cells in each image and computing geometric centroids (Fig. A.1 B), linking centroids into trajectories (Fig. A.1 C) and computing metrics of population dispersion (Fig. A.1 D). The appended code is original, customized to accommodate the specific nuances of our neutrophil experimental data such as the file naming convention used on the microscope and the empirically determined segmentation parameters needed to identify cell bodies. However, the general workflow should be amenable to a variety of motility datasets provided the user tunes some of these empirical parameters.

Methodology

1. Time lapse images should be labeled with the following convention: “pXXttt.tif”. An example would be “p06037.tif” for a phase (“p”) from location “06” corresponding to time “037” seconds. This sequence of images should reside within a folder labeled “Loc_XX”. An example would be “Loc_06” containing all phase images from location “06”. Multiple Loc_XX folders can reside within the same directory and can be processed simultaneously.

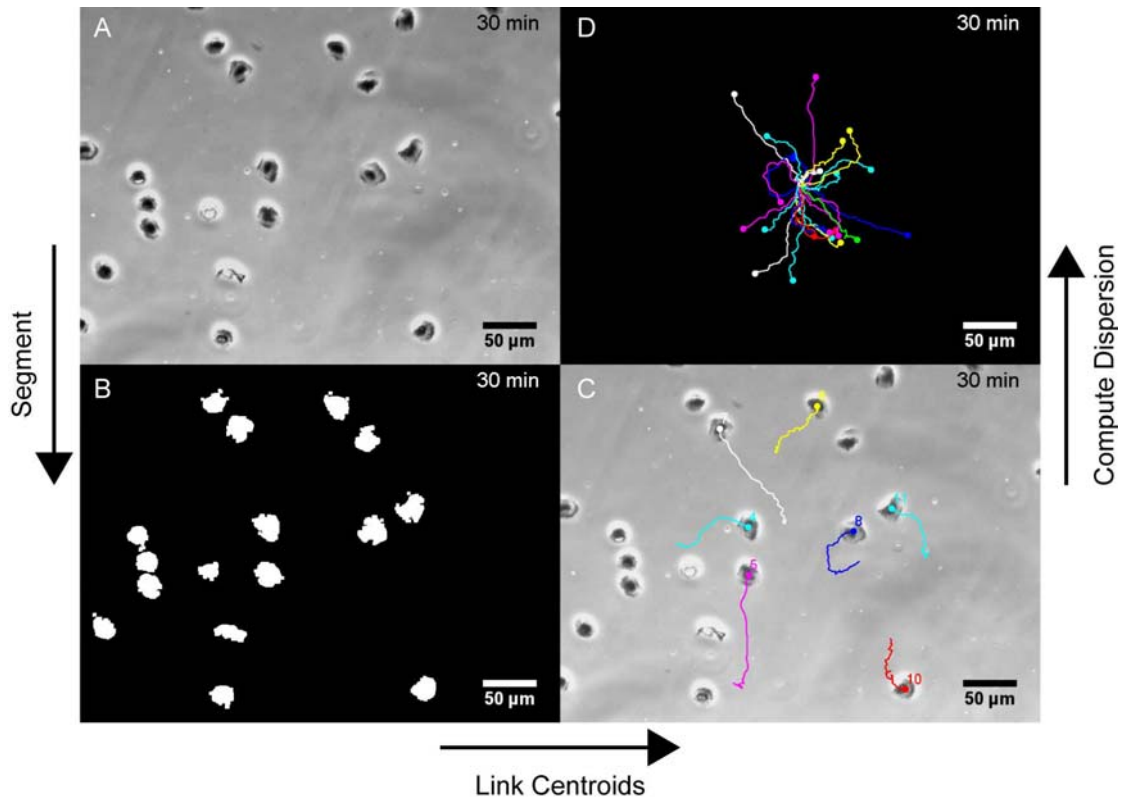


Figure A.1 Neutrophil motility data analysis workflow. (A) Timelapse images of neutrophil migration are captured. (B) Image segmentation is performed to identify cells and geometric centroids are computed. (C) Centroids are linked into trajectories. (D) Population dispersion statistics are computed.

2. In MATLAB, run “Time_Segment_Driver_v7.m”. You must specify the directory containing “Loc_XX” folders at line 87.
 - a. You will be asked if during acquisition you attempted constant time interval imaging. If so specify the attempted imaging period in seconds. This is to handle the real-world acquisition issue of the image not being snapped at an exact integer multiple of the desired imaging period. By specifying the attempted imaging period the program will round the actual acquisition time to the closest integer multiple of the desired acquisition time. This was necessary to improve mean squared displacement (MSD) computation by increasing the statistical power (observation frequency) of a particular τ .
 - b. “Time_Segment_Driver_v7.m” calls the following subroutines:
 - i. “ReadFolderContents_v3.m”
 - ii. “ReadTiffImages_v1.m”
 - iii. “Abs_Time_v1.m”
 - iv. “Sort_Time_v1.m”
 - v. “Bin_Time_v4.m”
 - vi. “Segmentation_v4.m”
3. The output from running “Time_Segment_Driver_v7.m” is a “Time_Segment” folder residing within the specified directory. The numeric prefix to the folder is the ISO 8601 dateform (“yyyymmddTHHMMSS”) for the day and time analysis was performed. Within this “Time_Segment” folder resides “Loc_XX” folders each containing an “Overlay” and “Segmentation” folder. “Overlay” contains the original

- data set superimposed with identified object boundaries and “Segmentation” contains the resulting binary image (cells denoted by ones on a field of zeros).
4. In MATLAB, run “CC_Driver_v5.m”. You must specify the directory containing “yyyymmddTHHMMSS_Time_Segment” folder at line 65.
 - a. “CC_Driver_v5.m” calls the following subroutines:
 - i. “ReadFolderContents_v3.m”
 - ii. “Track_Centroids_v5.m”
 1. “Pos_Selection_v2.m”
 - iii. “IJ_Manual_Track_Prep_v3.m”
 5. The output from running “CC_Driver_v5.m” is a “Loc_XX.mat” and “Loc_XX.txt” file within each “Loc_XX” folder of “yyyymmddTHHMMSS_Time_Segment”. These files contain the cell trajectories in a format compatible with the ImageJ Manual Tracking plugin.
 - a. Open the image sequence in “Overlay” folder of “Loc_XX” as a stack in ImageJ
 - b. Initialize the Manual Tracking plugin
 - c. Select “Load Previous Track File” and navigate to the “Loc_XX.txt” file output from “CC_Driver_v5.m”
 - d. Click on “Show text?” option in Manual Tracking plugin
 - e. Click “Overlay Dots & Lines button in Manual Tracking plugin
 - f. Inspect resulting superposition of tracks and data
 6. If necessary employ “CC_Output_Editor_v2.m” to make manual exclusions of all or portions of trajectories deemed anomalous. At line 56 hardcode path to “yyyymmddTHHMMSS_Time_Segment”. At line 82 hardcode tracks to be entirely

deleted. At line 90 hardcode portions of tracks to be retained (exclude portions outside these bounds). At line 98 hardcode portions of tracks to be eliminated (retain portions outside these bounds).

7. The output from running “CC_Output_Editor_v2.m” are “Loc_XX_edited.mat” and “Loc_XX_edited.txt” files within the “Loc_XX” folder of “yyyymmddTHHMMSS_Time_Segment”.
8. Manually copy all “Loc_XX_edited.mat” files up into the parent “yyyymmddTHHMMSS_Time_Segment” directory. If you did not run “CC_Output_Editor_v2.m” then copy the original “Loc_XX.mat” files. If you ran “CC_Output_Editor_v2.m” twice then copy the “Loc_XX_edited_edited.mat” files or any combination thereof.
9. In MATLAB, Run “Merge_Mats_v4.m”. At line 62 hardcode path to “yyyymmddTHHMMSS_Time_Segment” directory. Navigate to folder containing all “Loc_XX_edited.mat” files. Select “Loc_XX_edited.mat” files to be merged.
10. The output from running “Merge_Mats_v4.m” is a “Merged_Data.mat” file. The numeric prefix to the folder is the ISO 8601 dateform (“yyyymmddTHHMMSS”) for the day and time analysis was performed. This function simply performs a concatenation of the individual location-specific data sets and renames tracks so each trajectory has a unique identification number.
11. Copy all “yyyymmddTHHMMSS_Merged_Data.mat” files to the same location. Manually rename files so they conform to the following naming convention: “DXX_yyyymmdd_yyyymmddTHHMMSS_Merged_Data_XXXpXX_FN_XXXpXX_fMLP.mat”. An example would be:

“D01_20111014_20150216T123011_Merged_Data_005p00_FN_010p00_fMLP.mat” which is the data from donor “D01” acquired on date “20111014”, Merged_Mats_v4.m was performed on “20150216T123011”, and experimental conditions were “005.00” $\mu\text{g/mL}$ FN and “010.00” nM fMLP.

12. In MATLAB, run “Supra_MSD_Driver_v5.m”. This requires you to select if you are analyzing one or multiple experimental conditions. Navigate to the folder containing .mat file(s) and select the file(s) of interest. Enter a pixel to micron conversion in units of microns per single pixel. Elect to analyze full empirical data or a portion of the empirical data. If analyzing a temporal portion of the empirical data specify the upper time limit in minutes beyond which trajectory data will not be used. Elect whether or not to supply an error estimate (ϵ) in the position of cell centroids due to camera noise during acquisition. The correction performed on the empirical MSD curve is a subtraction of $4*\epsilon^2$ from the MSD at all τ values. This ϵ value is experimentally determined via a modeling experiment in which the error in the final MSD as a function of camera noise is established by superimposing additional random noise equal to the camera noise on the centroids of each tracked cell and determining the effect on the final MSD curve. In all of my experiments a correction of $\epsilon = 0.4604$ pix was employed. Lastly, specify if persistent random walk and power-law model fitting should be performed on the full empirical MSD or a portion thereof.
 - a. “Supra_MSD_Driver_v5.m” calls the following subroutines:
 - i. “MSD_Driver_v15.m”
 1. “Parse_Filename_v2.m”

2. "Post_IJ_Manual_Track_v3.m"
3. "Cell_Track_Plotter_v6.m"
4. "Consec_Differentials_v4.m"
5. "Step_Size_Stationarity_v2.m"
6. "Histograms_v3.m"
7. "Path_Length_v6.m"
8. "Mean_Path_Length_v5.m"
9. "Area_v4.m"
10. "Mean_Area_v5.m"
11. "Filter_Exp_Data_v3.m"
12. "Differentials_v5.m"
13. "Neff_v1.m"
14. "Mean_Differentials_v6.m"
15. "MSD_Epsilon_Subtract_v3.m"
16. "Plot_Mean_Differentials_v5.m"
17. "Filter_Mean_Differentials_v4.m"
18. "SandP_v11.m"
19. "Power_Law_v4.m"
20. "Plot_SandP_Fit_v6.m"
21. "Plot_Power_Law_Fit_v4.m"
22. "Van_Hove_Analysis_v3.m"
23. "Tidy_Up_v1.m"

13. The output from running “Supra_MSD_Driver_v5.m” is a “MSD_Driver_v15” folder. The numeric prefix to the folder is the ISO 8601 dateform (“yyyymmddTHHMMSS”) for the day and time MSD analysis was performed. Each folder within the “yyyymmddTHHMMSS_MSD_Driver_v14” directory contains the dispersion analysis (figures, arrays, and log files) corresponding to a “Merged_Data.mat” file. An Excel spreadsheet logs the final dispersion metrics for each condition analyzed.

Code

Note: Missing lines are version history annotation, removed for space considerations.

Time_Segment_Driver_v7.m

```

1 % Steven Henry
2 % 02/16/2015
41 %*****
42 % PURPOSE:
43 % From all image files in a series of Loc_XX folders within a given
44 % experimental condition the aims are to:
45 %
46 % Aim 1: Extract the time stamps from filenames and, if applicable, round
47 % these time stamps to the nearest integer multiple of the user-attempted
48 % constant interval imaging
49 %
50 % Aim 2: Perform image segmentation to identify cell boundaries. This goal
51 % is achieved via adaptation of MATLAB's:
52 % "Detecting a Cell Using Image Segmentation" demo found at:
53 % <http://www.mathworks.com/products/image/demos.html?file=/products/demos/
54 % shipping/images/ipexcell.html> and
55 % "Batch Processing Image Files in Parallel" demo found at:
56 % <http://www.mathworks.com/products/image/demos.html?file=/products/demos/
57 % shipping/images/ipexbatch.html>
58 %
59 % ASSUMPTIONS:
60 % (1) User has reserved "Loc_XX" naming to only those folders within a
61 % particular experimental condition to be analyzed.
62 % (2) User does not have extraneous ".tif" images or stacks present within
63 % a given "Loc_XX" folder.
64 % (3) Only ".tif" images of a particular type (e.g. phase or fluor) reside
65 % within a particular "Loc_XX" folder. The present filtering logic used in
66 % "ReadTiffImages_v1.m" does not differentiate phase images from fluor
67 % images.
68 % (4) Raw image files have time embedded in seconds and take the form
69 % "pXXttt...ttt.tif" or "sXXttt...ttt.tif" where "p/s" denotes "phase" or
70 % "fluor". "XX" denotes the location corresponding to "Loc_XX". "ttt...ttt"
71 % denotes some time stamp in seconds of non-constant length
72 %
73 % FUNCTIONS CALLED:
74 % ReadFolderContents_v3.m
75 % ReadTiffImages_v1.m
76 % Abs_Time_v1.m

```

```

77 % Sort_Time_v1.m
78 % Bin_Time_v4.m
79 % Segmentation_v4.m
80 %*****
81
82 clc
83 clear all
84 close all
85
86 % Have user select the directory:
87 start_path = 'EnterPathToYourDataHere';
88 directory = uigetdir(start_path,'Set Directory');
89
90 % Set directory to user-specified directory:
91 cd(directory);
92
93 % Create a folder that will hold all data analysis from this run: Determine
94 % date and time. Create a string in "dateform" "30" (ISO 8601) which has
95 % the format 'yyyymmddTHHMMSS'.
96 dstr = datestr(now, 30);
97
98 % Concatenate with "_Time_Segment":
99 analysis_folder = [dstr '_Time_Segment'];
100
101 % Create folder in current experimental condition directory:
102 mkdir(analysis_folder);
103
104 % Log path to the "Analysis" folder to store data:
105 analysis_path = [directory '\ analysis_folder'];
106
107 % Start a log file:
108 logfile = [dstr '_Time_Segment_Log.txt'];
109 cd(analysis_path);
110 fid = fopen(logfile,'wt');
111 cd('..');
112
113 % Print directory to log file:
114 fprintf(fid,'%s\n\n',dstr);
115 fprintf(1,'Selected directory is:\n');
116 fprintf(1, '%s \n\n', directory);
117 fprintf(fid,'Selected directory is:\n');
118 fprintf(fid, '%s \n', directory);
119
120 % Send user-specified path to ReadFolderContents.m for generation of a list
121 % of folders that conform to "Loc_XX" naming convention:
122 [num_folders, folderlist] = ReadFolderContents_v3(directory, fid);
123
124 % Send list of "Loc_XX" folders to "ReadTiffImages.m" for generation of a
125 % cell array of ".tif" filenames corresponding to a particular folder
126 % (organized by columns):
127 [frame_array] = ReadTiffImages_v1(num_folders, folderlist, fid);
128
129 % Extract time values from all ".tif" file names in 'frame_array'
130 [t_abs] = Abs_Time_v1(frame_array, fid);
131
132 % Sort t_abs array in ascending order so that first imaging frame occupies
133 % row 1 of t_abs_sorted and any locations with unequal numbers of frames
134 % have padding zeros at end of column.
135 [t_abs_sorted] = Sort_Time_v1(t_abs, fid);
136
137 % At this point we have an array of time sorted values that are "absolute"
138 % time values. We will now generate an array of time values that are
139 % rounded according to the attempted time interval between frames specified
140 % by the user. As an example if the user specified imaging interval was 60
141 % sec and a frame is taken at 64 sec this frame will be rounded to 60 sec.
142 choice = menu('Did you attempt constant time-interval imaging?',...
143     'Yes','No');
144

```

```

145 if choice == 1 % Yes
146
147     % Have user supply what the attempted imaging rate was in seconds:
148     interval = input('\nWhat was the intended imaging time-interval(sec)?:');
149
150     % Record progress:
151     fprintf(1, '\n\nUser attempted constant time interval imaging\n');
152     fprintf(1, 'Attempted imaging rate was = %.0f sec\n\n', interval);
153     fprintf(fid, '\nUser attempted constant time interval imaging\n');
154     fprintf(fid, 'Attempted imaging rate was = %.0f sec\n\n', interval);
155
156     % Go to 'Bin_Time.m' function which will zero the origin of each
157     % location and bin each time stamp to its closesst multiple of
158     % 'interval'.
159     [t_bin] = Bin_Time_v4(t_abs_sorted, interval, fid);
160
161 elseif choice == 2 % No
162
163     % Record progress:
164     fprintf(1, '\n\nUser did not attempt constant time interval imaging\n');
165     fprintf(1, 'As such no binning of time data in "t" was attempted\n\n');
166     fprintf(fid, '\nUser did not attempt constant time interval imaging\n');
167     fprintf(fid, 'As such no binning of time data in "t" was attempted\n\n');
168
169 end
170
171 % Automatically save time arrays:
172 cd(analysis_path);
173 fprintf(1, '\nSaving time arrays...\n\n');
174 fprintf(fid, '\nSaving time arrays...\n\n');
175 save t_abs.mat t_abs
176 save t_abs_sorted.mat t_abs_sorted
177 save t_bin.mat t_bin
178 fprintf(1, '\nItems saved to:\n%s\n\n', pwd);
179 fprintf(fid, '\nItems saved to:\n%s\n\n', pwd);
180 cd('.');
181
182 % Perform image segmentation:
183 Segmentation_v4(frame_array, folderlist, directory, analysis_path, fid)
184
185 fprintf(1, '\nProgram terminated\n\n');
186 fprintf(fid, '\nProgram terminated\n\n');
187
188 fclose(fid);

```

ReadFolderContents_v3.m

```

1 % Steven J. Henry
2 % 04/08/2011
23 %*****
24 % PURPOSE:
25 % This function determines which folders in a user-specified directory
26 % conform to a user-specified name. Currently the function is coded to
27 % identify folders of the form "Loc_XX". The idea is that the user directs
28 % the program to a folder corresponding to a specific experimental
29 % condition in which reside multiple imaging locations. Each of these
30 % locations is to contribute data to the same experimental condition pool
31 % of data and so need to be analyzed in aggregate.
32 %
33 % ASSUMPTIONS:
34 % The following filter logic has two criteria for considering whether or not
35 % a particular element of the directory conforms to the "Loc_XX" naming
36 % criteria. (1) The element must be exactly 6 characters long. (2) The
37 % first four characters must be "Loc_". Thus an assumption is that the last
38 % two characters of "Loc_XX" are integers from 0-9 in the X positions. It
39 % would be possible for "Loc_LL" folders to pass where L represents a
40 % letter A-Z. Why such a folder would exist is not apparent and not
41 % anticipated to occur frequently. A more stringent filter could be coded

```



```

42 % that incorporates a third criteria whereby the last two entries are
43 % confirmed to be integer values. This is not done in the current version
44 % (Version 3).
45 %
46 % INPUT:
47 % directory = user specified path to a folder containing multiple imaging
48 % locations.
49 % fid = file ID to which warnings and progress is printed as a text
50 % file.
51 %
52 % OUPUT:
53 % num_folders = number of folders that conform to "Loc_XX" naming
54 % convention after search is performed and deletions executed.
55 % folderlist = structural array containing list of folders conforming to
56 % "Loc_XX" naming convention as generated via MATLAB's intrinsic 'dir'
57 % command.
58 %*****
59
60 function [num_folders, folderlist] = ReadFolderContents_v3(directory, fid)
61
62 % Set the current directory to that specified by the user:
63 cd(directory);
64
65 % Define a structural array consisting of the information corresponding to
66 % the folders contained within the current directory. The 'pwd' function is
67 % an intrinsic MATLAB function that lists the elements in the current
68 % directory.
69 folderlist = dir(pwd);
70
71 % Initialize the 'del' vector that holds the position of elements that
72 % require deleting from 'folderlist':
73 del = [];
74
75 % Initialize the 'pass' counter that holds the number of elements in
76 % 'folderlist' that passed the "Loc_XX" filter.
77 pass = 0;
78
79 % Determine the number of elements in the 'folderlist' array containing the
80 % contents of the current directory.
81 num_folders = numel(folderlist);
82
83 % This is a loop that identifies elements in 'folderlist' that are not
84 % named in the form "Loc_XX". After these elements are identified the
85 % necessary deletions are made from 'folderlist'. Note nothing in the
86 % actual physical directory is harmed.
87
88 % Iterate over the present number of elements in 'folderlist'
89 for i = 1:num_folders
90
91     % Initialize 'del_flag' to "off". This flag will denote whether or not
92     % a particular entry needs to be deleted. It needs to be reset at the
93     % start of each pass through the loop in the event that a previous
94     % element in 'folderlist' did not pass the filter logic and needed to
95     % be deleted, placing 'del_flag' in the "on" state. Ignore MATLAB's
96     % warning at this line.
97     del_flag = 0;
98
99     % Log the name of element (i) of 'folderlist'
100    name = folderlist(i).name;
101
102    % Determine the number of characters in 'name'
103    num_char = length(name);
104
105    % The folder must contain exactly 6 characters (Loc_XX) to be a
106    % location folder and not an extraneous file, folder, or directory
107    % operator.
108    if num_char ~= 6
109

```

```

110     % If folder name is not exactly 6 characters turn on 'del_flag' for
111     % this element 'i'
112     del_flag = 1;
113
114     % Otherwise the length of the element name is 6 characters and so
115     % we determine if first four characters of the element name are
116     % "Loc_"
117     else
118     % Log the first four characters of the element:
119     lead_actual = name(1:4);
120     lead_desired = 'Loc_';
121
122     % If the actual leading characters are not "Loc_"
123     if strcmp(lead_actual,lead_desired) ~= 1
124
125         % If folder name does not begin with 'Loc_' turn on 'del_flag'
126         % for this element 'i'
127         del_flag = 1;
128
129     else
130
131         % If you made it to this point the given element (i) of
132         % 'folderlist' is **likely** a folder with the form "Loc_XX" so
133         % advance the succesful iteration counter by one. Emphasis is
134         % placed on "likely" because we've only ensured to this point
135         % that the folder name allowed to pass the filter is 6
136         % characters and has "Loc_" as the first four entries. Thus a
137         % folder with some error such as "Loc_LL" where L is a letter
138         % (A-Z) would not be caught by the filter. However such an
139         % error is not anticipated to occur frequently if at all.
140         pass = pass + 1;
141
142         % Ensure that 'del_flag' is still off (this is redundant).
143         del_flag = 0;
144
145     end
146 end
147
148 % If del_flag is "on" than this element 'i' needs to be deleted from
149 % 'folderlist' so record element 'i' position in 'del' vector:
150 if del_flag == 1
151
152     % if 'del' vector is empty then element 'i' is first entry to be
153     % logged:
154     if isempty(del) == 1
155
156         del(1,1) = i;
157
158         % Otherwise 'del' already contains element positions and so we
159         % need to expand the vector by one entry:
160     else
161         del_old = del;
162         num_dels = length(del);
163         clear del;
164         del = zeros(num_dels+1,1);
165         del(1:num_dels,1) = del_old;
166         del(num_dels+1,1) = i;
167     end
168 end
169
170 end
171
172 end
173
174 % Clear 'num_dels' variable for future use:
175 clear num_dels
176
177 % Now we have a vector 'del' mapping to elements in 'folderlist' requiring

```

```

178 % deletion. It is important that we delete these elements in descending
179 % order (from last element position to first element position) to retain
180 % proper mapping. It is not necessary to sort the 'del' vector because the
181 % filter is performed in ascending order (top to bottom), so 'del' must
182 % consist of a vector of 'ascending' entries.
183
184 % If 'del' vector is not empty there are deletions to be made.
185 if isempty(del) == 0
186
187     % Determine number of deletions to be made:
188     num_dels = length(del);
189
190     % Note the following decreasing count works even if 'num_dels' = 1.
191     for j = num_dels:-1:1
192
193         % Delete position 'del(j)' from 'folderlist'
194         folderlist(del(j)) = [];
195
196     end
197
198 end
199
200 % Verify that new size of 'folderlist' is equal to 'pass' the counter that
201 % stores number of entries that passed filter criteria. If not, tell user.
202 clear num_folders
203 num_folders = numel(folderlist);
204 if num_folders ~= pass
205     fprintf(1,'WARNING: Number of entries in "folderlist" is not equal to "pass"\n');
206     fprintf(1,'Error occurred in "ReadFolderContents_v3.m" function\n\n');
207     fprintf(fid,'WARNING: Number of entries in "folderlist" is not equal to "pass"\n');
208     fprintf(fid,'Error occurred in "ReadFolderContents_v3.m" function\n\n');
209 end
210
211 end

```

ReadTiffImages_v1.m

```

1  % Steven J. Henry
2  % 04/08/2011
10 %*****
11 % PURPOSE:
12 % This function identifies files of the form ".tif" in a given
13 % folder. The idea is that a driver cycles through the "Loc_XX" folders
14 % previously identified by "ReadFolderContents_v3.m". This function takes
15 % one of these folders and determines what elements of the folder are
16 % TIFF image files. These frames belong to a sequence of images presumably
17 % of multiple cells from a single location in a single experimental
18 % condition.
19 %
20 % ASSUMPTIONS:
21 % Short Answer:
22 % Folder being explored only contains TIFF files
23 % that comprise the image stack. That is no extraneous TIFF files or stacks
24 % are present.
25 %
26 % Long Answer:
27 % The following filter logic has two criteria for
28 % considering whether or not a particular element of the directory is an
29 % image file. (1) The element must be minimally 5 characters long. (2) The
30 % last four characters must be ".tif". Thus an assumption is that the
31 % remainder of the ".tif" file is of the form "pXXttt...ttt" or
32 % "sXXttt.ttt". p/s denote "phase" or "fluor". XX denotes the location
33 % corresponding to "Loc_XX". ttt...ttt denotes some time stamp in seconds
34 % of non-constant length. It is possible to include more stringent filters
35 % that could separate phase from fluor images but as of the writing of
36 % this function (Version 1, 04/08/2011) this need is not necessary
37 % (anticipated to be required in the future). For now the present design is
38 % sufficient, but requires that the operator only have the desired image

```

```

39 % frames present in the "Loc_XX" folder being analyzed. That is the folder
40 % should not contain both phase and fluor images or image stacks such as
41 % from ImageJ.
42 %
43 % INPUT:
44 % num_folders = number of folders that conform to "Loc_XX" naming
45 % convention via "ReadFolderContents.m"
46 % folderlist = structural array containing list of folders conforming to
47 % "Loc_XX" naming convention via "ReadFolderContents.m"
48 % fid = file ID to which warnings and progress is printed as a text
49 % file.
50 %
51 % OUTPUT:
52 % frame_array = cell array containing names of all ".tif" files associated
53 % with a given location (column)
54 %*****
55
56 function [frame_array] = ReadTiffImages_v1(num_folders, folderlist, fid)
57
58 for i = 1:num_folders
59
60     % Specify the location of folder 'i' in 'folderlist'
61     loc_directory = [pwd '\ folderlist(i).name];
62
63     % Set the current directory to the "Loc_XX" folder containing the
64     % imaging files of interest:
65     cd(loc_directory);
66
67     % Output the path of this new directory (i.e. the path to the current
68     % location folder).
69     fprintf(1, '\nThe current directory is:\n %s \n\n',pwd);
70     fprintf(fid, '\nThe current directory is:\n %s \n\n',pwd);
71
72     % Define a structural array consisting of the information corresponding
73     % to the folders contained within the current directory. The 'pwd'
74     % function is an intrinsic MATLAB function that lists the elements in
75     % the current directory.
76     filelist = dir(pwd);
77
78     % Initialize the 'del' vector that holds the position of elements that
79     % require deleting from 'filelist':
80     del = [];
81
82     % Initialize the 'pass' counter that holds the number of elements in
83     % 'filelist' that passed the "pXX" and ".tif" filters.
84     pass = 0;
85
86     % Determine the number of elements in 'filelist' containing the
87     % contents of the current directory.
88     num_files = numel(filelist);
89
90     % Iterate over the present number of elements in 'filelist'
91     for j = 1:num_files
92
93         % Initialize 'del_flag' to "off". This flag will denote whether or
94         % not a particular entry needs to be deleted. It needs to be reset
95         % at the start of each pass through the loop in the event that a
96         % previous element in 'filelist' did not pass the filter logic
97         % and needed to be deleted, placing 'del_flag' in the "on" state.
98         % Ignore MATLAB's warning at this line.
99         del_flag = 0;
100
101         % Log the name of element 'j' of 'filelist'
102         name = filelist(j).name;
103
104         % Determine the number of characters in 'name'
105         num_char = size(name,2);
106

```

```

107 % Minimally the filename must contain 5 characters (x.xxx) to be a
108 % file and not a directory operator such as '.' or '..' or a folder
109 % with less than 5 characters.
110 if num_char < 5
111
112     % If the element 'j' name is not minimally 5 characters turn
113     % 'del_flag' on
114     del_flag = 1;
115
116     % Otherwise the length of the element name is greater than or
117     % equal to 5 characters and so we determine if the last four
118     % characters of the name are a '.tif' extension
119 else
120     % Flip the element name order:
121     eman = fliplr(name);
122
123     % Log the first four characters which in reverse order
124     % correspond to the extension of the file:
125     ext_actual = eman(1:4);
126     ext_desired = 'fit.';
127
128     % If the reversed extension is not 'fit.'
129     if strcmp(ext_actual,ext_desired) ~= 1
130
131         % Turn 'del_flag' "on"
132         del_flag = 1;
133
134     else
135
136         % If you made it to this point the given element 'j' of
137         % 'filelist' is a file with extension '.tif' so advance the
138         % successful iteration counter by one.
139         pass = pass + 1;
140
141         % Ensure that 'del_flag' is still off (this is redundant)
142         del_flag = 0;
143
144     end
145
146 end
147
148 % If del_flag is "on" than this element 'j' needs to be deleted
149 % from 'filelist' so record element 'j' position in 'del' vector:
150 if del_flag == 1
151
152     % if 'del' vector is empty then element 'j' is first entry to
153     % be logged:
154     if isempty(del) == 1
155
156         del(1,1) = j;
157
158         % Otherwise 'del' already contains element positions and so
159         % we need to expand the vector by one entry:
160     else
161         del_old = del;
162         num_dels = size(del,1);
163         clear del;
164         del = zeros(num_dels+1,1);
165         del(1:num_dels,1) = del_old;
166         del(num_dels+1,1) = j;
167     end
168
169 end
170
171 end
172
173 % Clear 'num_dels' variable for future use:
174 clear num_dels

```

```

175
176 % Now we have a vector 'del' mapping to elements in 'folderlist'
177 % requiring deletion. It is important that we delete these elements in
178 % descending order (from last element position to first element
179 % position) to retain proper mapping. It is not necessary to sort the
180 % 'del' vector because the filter is performed in ascending order (top
181 % to bottom), so 'del' must consist of a vector of 'ascending' entries.
182
183 % If 'del' vector is not empty there are deletions to be made.
184 if isempty(del) == 0
185
186     % Determine number of deletions to be made:
187     num_dels = length(del);
188
189     % Note the following decreasing count works even if 'num_dels' = 1.
190     for k = num_dels:-1:1
191
192         % Delete position 'del(k)' from 'filelist'
193         filelist(del(k)) = [];
194
195     end
196
197 end
198
199 % Verify that new size of 'filelist' is equal to 'pass' the counter
200 % that stores number of entries that passed filter criteria. If not,
201 % tell user.
202 clear num_files
203 num_files = numel(filelist);
204 if num_files ~= pass
205     fprintf(1, '\nWARNING: Number of images in "filelist" is not equal to "pass"\n');
206     fprintf(1, 'Error occurred in "ReadTiffImages.m" function on %s \n\n', foldername(i));
207     fprintf(fid, '\nWARNING: Number of images in "filelist" is not equal to "pass"\n');
208     fprintf(1, 'Error occurred in "ReadTiffImages.m" function on %s \n\n', foldername(i));
209 end
210
211 % If this is the first folder (or only folder being considered)
212 if i == 1
213     % Preallocate a cell array (which is used so we can log text
214     % strings in each cell) memory. We will use the number of frames in
215     % the first folder as an estimate of the size of the array. If the
216     % array needs to grow this can be done during processing.
217     frame_array = cell(num_files, num_folders);
218 end
219
220 % Write the filename for this location to frame_array:
221 for n = 1:num_files
222     frame_array{n,i} = filelist(n).name;
223 end
224
225 % Jump back one level in the directory to the experimental condition
226 % folder
227 cd('../');
228
229 end
230
231 end

```

Abs_Time_v1.m

```

1 % Steven J. Henry
2 % 04/08/2011
6 %*****
7 % PURPOSE:
8 % This function takes in the array 't' containing absolute time
9 % values and performs two functions. First each location (column of data)
10 % is shifted linearly such that the origin fram (first row of each column)
11 % is zero. Second each location is rounded to the nearest multiple of the

```

```

12 % user-specified imaging rate 'interval'. At the end of processing a
13 % matrix 't_bin' results of the same dimensions as 't' but now possessing
14 % either zeros or integer multiples of 'interval.'
15 %
16 % ASSUMPTIONS:
17 % This function assumes that the general form of the filename is
18 % "pXXtt..tst.tif" or "sXXttt...tst.tif". "p"/"s" denotes "phase"/"fluor".
19 % "XX" denotes location number. "tst...tst" is a time-stamp in seconds of
20 % unspecified length. Then it is always true that the time stamp begins at
21 % position '4' and ends at position 'num_char-4' where 'num_char' is the
22 % total length of the name.
23 %
24 % INPUT:
25 % frame_array = cell array containing names of all ".tif" files associated
26 % with a given location (column)
27 % fid = file ID to which warnings and progress is printed as a text
28 % file.
29 %
30 % OUTPUT:
31 % t = array of absolute time values. Rows represent frames. Columns
32 % represent locations. This array is passed sorted from lowest to highest
33 % value. It is not necessarily true that all locations (columns) have the
34 % same number of frames (nonzero rows). The array is designed such that
35 % when two columns do not have an equal number of frames the remainder of
36 % the shorter columns is padded with zeros. For example consider the
37 % following hypothetical three location matrix where the first location
38 % has 5 frames, the second location 3 frames, and the third location 4
39 % frames:
40 %      61  73  87
41 %      123 138 144
42 %      183 195 206
43 %      245 0  266
44 %      306 0  0
45 %*****
46
47 function [t_abs] = Abs_Time_v1(frame_array, fid)
48
49 % Determine size of cell array 'frame_array'
50 [slice_max, loc_max] = size(frame_array);
51
52 % Create a matrix that will contain the embedded times in a given filename.
53 % Rows correspond to slice number and each column is a unique location. It
54 % is NOT necessarily the case that column number represents location
55 % number. For example its possible that a particular imaging set defined
56 % Loc_06 through Loc_09 as a particular experimental condition. However
57 % Loc_06 would be column 1, Loc_07 => column 2, etc...
58 t_abs = zeros(slice_max, loc_max);
59
60 for i = 1:loc_max
61
62     for j = 1:slice_max
63
64         % Load the file name string 'title':
65         title = frame_array{j,i};
66
67         % If 'title' is NOT empty (meaning 'frame_array' entry {j,i} is
68         % not empty:
69         if isempty(title) == 0
70
71             % Make sure the entry is a character array (i.e. string).
72             % If 'title' is not a string tell user:
73             if ischar(title) == 0
74                 % You have a problem because the entry you're dealing with
75                 % is not a string:
76                 fprintf(1, '\nWARNING: The current entry in "frame_array" is not a string.\n');
77                 fprintf(1, 'Occured in "Abs_Time.m" for row = %.0f and col = %.0f\n\n', j, i);
78                 fprintf(fid, '\nWARNING: The current entry in "frame_array" is not a string.\n');
79                 fprintf(fid, 'Occured in "Abs_Time.m" for row = %.0f and col = %.0f\n\n', j, i);

```

```

80
81     % Otherwise you have a string so extract the embedded time
82     % stamp:
83     else
84
85         % Compute number of characters in file name:
86         num_char = length(title);
87
88         % Isolate time portion of file name. This assumes that the
89         % general form of the filename is "pXXtt..tt.tif" or
90         % "sXXttt...tt.tif". "p"/"s" denotes "phase"/"fluo". "XX"
91         % denotes location number. "ttt...tt" is a time-stamp in
92         % seconds of unspecified length. Then it is always true
93         % that the time stamp begins at position '4' and ends at
94         % position 'num_char-4' where 'num_char' is the total
95         % length of the name.
96         t_abs_char = title(4:num_char-4);
97
98         % Convert time portion of file name (a character string) to
99         % a numeric value:
100        t_abs_num = str2double(t_abs_char);
101
102        % Load numeric time corresponding to slice 'j' in array
103        % 't_abs' at row 'j' in column 'i'
104        t_abs(j,i) = t_abs_num;
105
106    end
107
108    % Note if 'title' is empty it just means that this particular
109    % location does not have as many frames for analysis as another
110    % location in the given experimental condition.
111    end
112
113    end
114
115    end
116
117    % Check post-processing dimensionality
118    [t_slices, t_locs] = size(t_abs);
119    if t_slices ~= slice_max
120        fprintf(1, '\nWARNING: # of rows in "t_abs" ~= "frame_array" after time stamp extraction\n\n');
121        fprintf(fid, '\nWARNING: # of rows in "t_abs" ~= "frame_array" after time stamp extraction\n\n');
122    end
123    if t_locs ~= loc_max
124        fprintf(1, '\nWARNING: # of cols in "t_abs" ~= "frame_array" after time stamp extraction\n\n');
125        fprintf(fid, '\nWARNING: # of cols in "t_abs" ~= "frame_array" after time stamp extraction\n\n');
126    end
127
128    end

```

Sort_Time_v1.m

```

1  % Steven J. Henry
2  % 04/08/2011
6  %*****
7  % PURPOSE:
8  % The purpose of this function is to sort an array of number time values in
9  % ascending order. Since not every column will have the same number of
10 % nonzero entries (frame) it is necessary to correct the fact that applying
11 % a global sort to the entire column will result in zeros being the leading
12 % column entries. This function sorts all nonzero elements and then ensures
13 % that the first row of the resulting sorted array has the first imaging
14 % frame of the particular location. Empirically it is never the case that
15 % real imaging frame exists at absolute time t = 0 so we use zero entries
16 % in a column to denote nonexistent frames.
17 %
18 % ASSUMPTIONS:
19 % Zero entries in the time array 't_abs' denote a non-existent frame and

```



```

20 % not a real imaging frame. This assumption is reasonable as it has never
21 % been observed where an image is snapped at the instant the imaging
22 % LABVIEW program is started. Typically the first imaging frame begins at t
23 % = 1-5 sec.
24 %
25 % INPUT:
26 % t_abs = array of UNSORTED absolute time values. Rows represent frames.
27 % Columns represent locations. This array is passed UNSORTED. It is not
28 % necessarily true that all locations (columns) have the same number of
29 % frames (nonzero rows). For example consider the following hypothetical
30 % three location matrix where the first location has 5 frames, the second
31 % location 3 frames, and the third location 4 frames:
32 %      123  195  206
33 %      61  138  144
34 %      183  73   87
35 %      306  0   266
36 %      245  0   0
37 % fid = file ID to which warnings and progress is printed as a text
38 % file.
39 %
40 % OUPUT:
41 % t_abs_sorted = array of SORTED absolute time values. Rows represent
42 % frames. Columns represent locations. This array is passed SORTED from
43 % lowest to highest value. It is not necessarily true that all locations
44 % (columns) have the same number of frames (nonzero rows). The array is
45 % designed such that when two columns do not have an equal number of
46 % frames the remainder of the shorter columns is padded with zeros. For
47 % example consider the following hypothetical three location matrix where
48 % the first location has 5 frames, the second location 3 frames, and the
49 % third location 4 frames:
50 %      61  73   87
51 %      123 138  144
52 %      183 195  206
53 %      245 0   266
54 %      306 0   0
55 %*****
56
57 function [t_abs_sorted] = Sort_Time_v1(t_abs, fid)
58
59 % As Windows may have altered the order in which the files were saved to
60 % the location folder (perhaps as a result of the user applying a sort
61 % within the folder) we now sort the columns of 't_abs' in ascending order:
62 t_abs_sorted = sort(t_abs,1,'ascend');
63
64 % However, in the process of sorting entries in a given column of 't' we
65 % may end up placing zeros in the leading rows if a given location has less
66 % total slices than the location for a given experimental condition with
67 % the most number of slices.
68
69 % Example: If Location 1 has 8 slices but Location 2 has 10 slices after
70 % sorting zeros are now placed in row 1 and row 2 of Location 2's column.
71 % Empirically it is never the case that an image is taken at absolute time
72 % "0" so we can filter zero values to mean non-existent slices. In general
73 % the soonest an image is taken is on the order of 1-5 sec after hitting
74 % "start" on the data collection LabView program.
75
76 % As a check on our manipulations log the total number of nonzero elements
77 % in 't_abs' before anything is done:
78 tot_nz_presort = nnz(t_abs);
79
80 % Determine size of 't_abs_sorted'
81 [max_slices, max_locs] = size(t_abs_sorted);
82
83 % Iterate through each column in 't_abs_sorted'
84 for i = 1:max_locs
85
86     % Determine the number of nonzero elements in each column:
87     nnz_slices = nnz(t_abs_sorted(:,i));

```

```

88
89 % If the number of nonzero entries does not equal the total number of
90 % rows (i.e. value 'max_slices') then there are zeros in this column:
91 if nnz_slices ~= max_slices
92
93     % Save the nonzero entries of column 'i' in array 't_abs_sorted'
94     temp_time = nonzeros(t_abs_sorted(:,i));
95
96     % Write zeros to column 'i' in array 't_abs_sorted'
97     t_abs_sorted(:,i) = 0;
98
99     % Print sorted nonzero values in 'temp_time' vector to column 'i'
100    % in array 't_abs_sorted':
101    t_abs_sorted(1:nnz_slices,i) = temp_time;
102
103 end
104
105 end
106
107 % As a check on our manipulations log the total number of nonzero elements
108 % in 't_abs_sorted' after manipulations are done:
109 tot_nz_postsort = nnz(t_abs_sorted);
110
111 % IF total number of nonzero elements pre/post sorting do not equal you
112 % have a problem!
113 if tot_nz_postsort ~= tot_nz_presort
114     fprintf(1,'\nWARNING: Total number of nonzero elements after sorting does not\n');
115     fprintf(1,'equal total number of nonzero elements before sorting.\n');
116     fprintf(1,'Error occured in "Sort_Time.m"\n');
117     fprintf(fid,'\nWARNING: Total number of nonzero elements after sorting does not\n');
118     fprintf(fid,'equal total number of nonzero elements before sorting.\n');
119     fprintf(fid,'Error occured in "Sort_Time.m"\n');
120 end
121
122 % Determine the number of slices in each location. This corresponds to the
123 % number of nonzero elements in each column. This can be used to verify
124 % that the proper number of entries were achieved for each folder.
125 slice_check = zeros(max_locs,2);
126 for j = 1:max_locs
127     % Column 1 is location number
128     slice_check(j,1) = j;
129     % Column 2 is number of slices in location folder
130     slice_check(j,2) = nnz(t_abs_sorted(:,j));
131 end
132
133 % Output information to user:
134 fprintf(1,'\nTime retrieval completed\n\n');
135 fprintf(fid,'\nTime retrieval completed\n\n');
136 fprintf(1,'Number of locations analyzed = %1.0f\n',max_locs);
137 fprintf(fid,'Number of locations analyzed = %1.0f\n',max_locs);
138 fprintf(1,'The number of slices analyzed for each location:\n');
139 fprintf(fid,'The number of slices analyzed for each location:\n');
140 fprintf(1,'Loc\t# Slices\t\n');
141 fprintf(fid,'Loc\t# Slices\t\n');
142
143 for k = 1:max_locs
144     fprintf(1,'%0ft%0ft\n', slice_check(k,:));
145     fprintf(fid,'%0ft%0ft\n', slice_check(k,:));
146 end

```

Bin_Time_v4.m

```

1 % Steven J. Henry
2 % 02/16/2015
26 %*****
27 % PURPOSE:
28 % This function takes in absolute time values in seconds and rounds (bins)
29 % all values to the closest integer multiple of a user-specified time

```

```

30 % interval. It avoids systematic upwards rounding bias at remainders of
31 % exactly 0.5 by essentially flipping a coin to see whether such remainders
32 % are rounded up or down when such cases arise.
33 %
34 % ASSUMPTIONS:
35 % n/a
36 %
37 % INPUT:
38 % t_abs_sorted = array of SORTED absolute time values. Rows represent
39 % frames. Columns represent locations. This array is passed SORTED from
40 % lowest to highest value. It is not necessarily true that all locations
41 % (columns) have the same number of frames (nonzero rows). The array is
42 % designed such that when two columns do not have an equal number of
43 % frames the remainder of the shorter columns is padded with zeros. For
44 % example consider the following hypothetical three location matrix where
45 % the first location has 5 frames, the second location 3 frames, and the
46 % third location 4 frames:
47 %      61  73  87
48 %      123 138 144
49 %      183 195 206
50 %      245  0  266
51 %      306  0   0
52 % interval = user-specified constant imaging rate attempted (during actual
53 % data collection) in seconds. This is the value the user set the LABVIEW
54 % program on the Nikon microscope.
55 % fid = file ID to which warnings and progress is printed as a text
56 % file.
57 %
58 % OUPUT:
59 %*****
60
61 function [t_bin] = Bin_Time_v4(t_abs_sorted, interval, fid)
62
63 % Get function name:
64 func_name = mfilename;
65
66 % Update log file that function is running:
67 fprintf(1, '\n%s running ... \n', func_name);
68 fprintf(fid, '\n%s running ... \n', func_name);
69
70 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
71 % warning dialog the log file records no errors/warnings generated:
72 warn = 0;
73
74 % Determine dimensionality of 'data':
75 [rows, cols] = size(t_abs_sorted);
76
77 % Reserve space for 't_bin':
78 t_bin = zeros(rows, cols);
79
80 % Before entering the analysis loop set the random number stream generator
81 % to a seed based upon the current time. This will help to increase the
82 % independence of two runs through the program, otherwise the same sequence
83 % of coin flips could result each time this function is called from
84 % start-up:
85
86 % Save a 'defaultStream' that has parameters equivalent to those when
87 % MATLAB first starts up:
88 defaultStream = RandStream('mt19937ar', 'Seed', 0);
89
90 % Generate a time-dependent seed:
91 mySeed = sum(100*clock);
92 % Create a stream based upon this seed:
93 algorithm = 'mt19937ar';
94 myStream = RandStream(algorithm, 'Seed', mySeed);
95 % Set the stream from which 'rand', 'randn', and 'randi' draw numbers to
96 % 'myStream':
97 RandStream.setGlobalStream(myStream);

```

```

98 % Reset internal state or pointer within stream:
99 reset(myStream);
100
101 % Log the seed used:
102 fprintf(1, '\n\tAlgorithm type set to: %s\n', algorithm);
103 fprintf(1, '\n\tSeed used was: %f\n', mySeed);
104 fprintf(fid, '\n\tAlgorithm type set to: %s\n', algorithm);
105 fprintf(fid, '\n\tSeed used was: %f\n', mySeed);
106
107 % Record how many times remainders of 0.5 occur and of these how many were
108 % rounded up vs. down:
109 num_rem = 0;
110 num_down = 0;
111 num_up = 0;
112
113 % Loop over all "Loc_XX" folders
114 for i = 1:cols
115
116     % Loop over all frames within a given folder:
117     for j = 1:rows
118
119         % Since supplied data is sorted we can treat first entry in each
120         % column as the origin frame. We will normalize all subsequent
121         % frames (for the given track 'i') with respect to this origin
122         % frame:
123         if j == 1
124             t_abs_orig = t_abs_sorted(j,i);
125             t_bin(j,i) = 0;
126         else
127
128             % Record the current time-stamp value:
129             t_abs_now = t_abs_sorted(j,i);
130
131             % If it is not zero (in which case it is a padding entry) shift
132             % the value with respect to the track origin and round it to
133             % the nearest multiple of the attempted constant imaging
134             % interval:
135             if t_abs_now ~= 0
136                 % Shift current frame's time-stamp with respect to origin
137                 % frame's time-stamp:
138                 t_abs_shifted = t_abs_now - t_abs_orig;
139
140                 % Q is the quotient after division by the user-specified
141                 % 'interval'. For example if interval = 60 sec and the
142                 % given image has a shifted time stamp of 160 sec than the
143                 % corresponding entry in 'Q' is 2 (b/c 160/60 = 2 + 40/60).
144                 % Use of MATLAB's 'floor' function is to ensure we always
145                 % round down to the nearest whole integer value (the
146                 % quotient). To continue with the above example
147                 % round(160/60) = 3 but floor(160/60) = 2.
148                 Q = floor(t_abs_shifted/interval);
149
150                 % R is the remainder after division normalized by
151                 % 'interval':
152                 R = rem(t_abs_shifted, interval)/interval;
153
154                 % If R is not exactly 0.5:
155                 if R ~= 0.5
156
157                     % Round it according to normal rules:
158                     R = round(R);
159
160                     % If R is exactly 0.5 randomly choose to round up or
161                     % down using a uniform random number generator so that
162                     % over many flips Prob(roundup) ~ Prob(rounddown) ~ 0.5
163                     % (50%).
164                     elseif R == 0.5
165

```

```

166         % Update # times remainders of 0.5 occur:
167         num_rem = num_rem + 1;
168
169         % Flip the coin by drawing a number from (0,1):
170         coin = rand;
171         fprintf(1,'\n\tcoin tossed: rand = %f\n',coin);
172         fprintf(fid,'\n\tcoin tossed: rand = %f\n',coin);
173
174         % If the coin landed on it's edge (0.5) keep flipping
175         % until it lands on a face:
176         while coin == 0.5
177
178             coin = rand;
179
180         end
181
182         % Do a redundant check that coin is not 0.5:
183         if coin == 0.5
184             fprintf(1,'\n\tWARNING: Coin is still on its edge (0.5) but escaped "while" loop.\n');
185             fprintf(1,'\n\tWARNING: Coin is still on its edge (0.5) but escaped "while" loop.\n');
186             warn = 1;
187         end
188
189         % Now invoke the normal rounding rules. Because of the
190         % above logic we need not worry about cases where coin
191         % = 0.5. At this point coin must be less than or
192         % greater but not equal to 0.5.
193         R = round(coin);
194
195         % Log how many roundups and rounddowns occurred:
196         if coin < 0.5
197             num_down = num_down + 1;
198         elseif coin > 0.5
199             num_up = num_up + 1;
200         end
201
202     end
203
204     % Do a redundant check that R is binary. Note MATLAB gives
205     % a warning with the use of "&" instead of short-circuit
206     % operator "&&". Here we need MATLAB to evaluate both
207     % criteria, satisfying a single criterion is not sufficient
208     % in this case so we can't use a short-circuit operator.
209     % Ignore the warning.
210     if R ~= 0 & R ~= 1
211         fprintf(1,'\n\tWARNING: Remainder value R is not binary but should be.\n');
212         fprintf(1,'\tR = %0.5f\n',R);
213         fprintf(fid,'\n\tWARNING: Remainder value R is not binary but should be.\n');
214         fprintf(fid,'\tR = %0.5f\n',R);
215         keyboard
216         warn = 1;
217     end
218
219     % Generate the binned time value:
220     t_bin(j,i) = (Q+R)*interval;
221
222     end
223
224     end
225
226     end
227
228 end
229
230 % Output stats on special rounding cases:
231 fprintf(1,'\n\tNumber of remainders exactly = 0.5 was %f\n',num_rem);
232 fprintf(1,'\t%.0f rounded up and %.0f rounded down\n',num_up,num_down);
233 fprintf(fid,'\n\tNumber of remainders exactly = 0.5 was %f\n',num_rem);

```

```

234 fprintf(fid, 't%.0f rounded up and %.0f rounded down\n', num_up, num_down);
235
236 % Check post-processing dimensionality:
237 [rows2, cols2] = size(t_bin);
238 if rows2 ~= rows
239     fprintf(1, '\n\tWARNING: # of rows in output "t_bin" different than input "t_abs_sorted"\n');
240     fprintf(fid, '\n\tWARNING: # of rows in output "t_bin" different than input "t_abs_sorted"\n');
241     warn = 1;
242 end
243 if cols2 ~= cols
244     fprintf(1, '\n\tWARNING: # of cols in output "t_bin" different than input "t_abs_sorted"\n');
245     fprintf(fid, '\n\tWARNING: # of cols in output "t_bin" different than input "t_abs_sorted"\n');
246     warn = 1;
247 end
248
249 % Return default stream back to conditions at MATLAB startup:
250 RandStream.setGlobalStream(defaultStream);
251 % Reset internal state or pointer within stream:
252 reset(defaultStream);
253
254 % If no warnings generated report so in log file:
255 if warn == 0
256     fprintf(1, '\n\tFunction completed without errors/warnings\n');
257     fprintf(fid, '\n\tFunction completed without errors/warnings\n');
258 end
259
260 % Update log file that function is completed:
261 fprintf(1, '\n%s completed\n', func_name);
262 fprintf(fid, '\n%s completed\n', func_name);
263
264 end

```

Segmentation_v4.m

```

1 % Steven J. Henry
2 % 06/15/2011
74 %*****
75 % PURPOSE:
76 % The purpose of this function is to segment grayscale TIFF images and
77 % identify cell boundaries.
78 %
79 % ASSUMPTIONS:
80 % All images are of similar quality such that the hardcoded segmentation
81 % algorithm is valid for all frames. Raw images were generated such that
82 % the cells have dark-body centers (~black) and light halo outlines
83 % (~white). A histogram of the raw image(s) should reveal a spike at an
84 % intermediate grayscale intensity with no saturation at [0, 255].
85 %
86 % INPUT:
87 % frame_array = cell array containing names of all ".tif" files associated
88 % with a given location (column) via "ReadTiffImages.m"
89 % folderlist = structural array containing list of folders conforming to
90 % "Loc_XX" naming convention via "ReadFolderContents.m"
91 % raw_path = absolute path to folder containing "Loc_XX" folders with TIFF
92 % files to be analyzed
93 % analyzed_path = absolute path to folder containign "Loc_XX" folders with
94 % segmented TIFF files and associated time arrays
95 % fid = file ID to which warnings and progress is printed as a text
96 % file.
97 %
98 % OUPUT: (not returned to driver)
99 % For each image file processed two segmentation files result:
100 % 1) "BW_pXXttt...ttt.tif" is a black/white thresholded image with black
101 % background and white cell bodies.
102 % 2) "pXXttt...ttt.tif" is the original image file
103 % It is necessary to duplicate the
104 %*****
105

```

```

106 function [] = Segmentation_v4(frame_array, folderlist, raw_path, analyzed_path, fid)
107
108 % Get function name:
109 func_name = mfilename;
110
111 % Update log file that function is running:
112 fprintf(1, '\n%s running ...\n', func_name);
113 fprintf(fid, '\n%s running ...\n', func_name);
114
115 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
116 % warning dialog the log file records no errors/warnings generated:
117 warn = 0;
118
119 % Determine dimensionality of 'frame_array'
120 [slices, locs] = size(frame_array);
121
122 % Loop over all "Loc_XX" folders residing in 'folderlist'
123 for i = 1:locs
124
125     % Log current folder name:
126     folder_name = folderlist(i).name;
127
128     % Create a "Loc_XX" folder in the "Analysis" folder to store segmented
129     % images:
130     cd(analyzed_path);
131     mkdir(folder_name)
132     analyzed_loc_path = [analyzed_path '\ folder_name];
133
134     % Specify the location of raw data folder 'i' in 'folderlist':
135     raw_loc_path = [raw_path '\ folder_name];
136
137     % Set the current directory to the "Loc_XX" folder containing the
138     % imaging files of interest:
139     cd(raw_loc_path);
140
141     % Output the path of the folder you are about to perform segmentation
142     % on:
143     fprintf(1, '\n\tSegmenting images in the following directory (%.0f of %.0f):\n\t%s \n\n', i, locs, pwd);
144     fprintf(fid, '\n\tSegmenting images in the following directory (%.0f of %.0f):\n\t%s \n\n', i, locs, pwd);
145
146     % Loop over all ".tif" files in folder 'i':
147     for j = 1:slices
148
149         % If this is the first slice in this location create folders to
150         % contain segmented data:
151         if j == 1
152             % Change directory to the "Loc_XX" folder in "Analysis" folder:
153             cd(analyzed_loc_path);
154             % Make folder names to hold segmentation results:
155             seg_folder = 'Segmentation';
156             overlay_folder = 'Overlay';
157             % Create segmentation results folder:
158             mkdir(seg_folder);
159             mkdir(overlay_folder);
160             % Get paths to these folders:
161             seg_path = [analyzed_loc_path '\ seg_folder];
162             overlay_path = [analyzed_loc_path '\ overlay_folder];
163             % Go back to location of raw data files:
164             cd(raw_loc_path);
165         end
166
167         % Get filename from 'frame_array'
168         filename = frame_array{j,i};
169
170         % If 'filename' is not empty then use it to load the corresponding
171         % image file:
172         if isempty(filename) == 0
173

```

```

174 % Output the slice you're on
175 if mod(j,5)==0
176     fprintf(1,'\tSegmenting image %.0f\n',j);
177 end
178
179 % Load image given by 'filename':
180 I = imread(filename);
181
182 % Perform edge detection using 'roberts' algorithm. This was
183 % empirically determined to give the best signal to noise
184 % ratio.
185 BW = edge(I,'roberts');
186
187 % Create disk structuring element using MATLAB's 'strel'
188 % function.
189 SE_dil = strel('disk', 3);
190
191 % Use this structuring element to dilate the BW image:
192 BWdil = imdilate(BW, SE_dil);
193
194 % Fill dilated image holes:
195 BWfill = imfill(BWdil, 'holes');
196
197 % Clear border elements:
198 BWnobord = imclearborder(BWfill, 8);
199
200 % The ability to perform the complementary erosion operation
201 % after the dilation is commented out below. As both dilation
202 % and erosion are nonlinear operations it is not necessarily
203 % true that eroding a dilated image removes the previous
204 % operation's effect(s). Generally it has been found that the
205 % roberts edge detection underestimates the cell boundary and
206 % so the dilation is not tremendously deleterious.
207
208 % % Create disk structuring element using MATLAB's 'strel'
209 % % function.
210 % SE_ero = strel('disk',1);
211 %
212 % % Erode all edges:
213 % BWerode = imerode(BWnobord, SE_ero);
214
215 % Clear elements less than and empirically determined number of
216 % pixels.
217 % Imaging at 20XLWD, binning 1:
218 % hNeutrophils on uCP hFN are on the order of 4000 pixels in
219 % area. Over the period of an hour area will reduce to
220 % approximately 2000 pixels. Thus set threshold at 1500 pixels.
221 %
222 % Imaging at 20XLWD, binning 2:
223 % Set value at 500 pixels.
224 BWfinal = bwareaopen(BWnobord, 250);
225
226 % Save segmented image:
227 BWname = ['BW_' filename];
228 cd(seg_path);
229 imwrite(BWfinal, BWname, 'tif', 'Compression', 'none');
230 cd(raw_loc_path);
231
232 % Overlay segmentation boundary on original '.tif' file:
233 BWoutline = bwperim(BWfinal);
234 Segout = I;
235 Segout(BWoutline) = 0; % 0 = black, 255 = white
236 BWoutline_name = ['Perim_' filename];
237 cd(overlay_path);
238 imwrite(Segout, BWoutline_name, 'tif', 'Compression', 'none');
239 cd(raw_loc_path);
240
241 end

```



```

242
243     end
244
245     % Jump back to the experimental condition folder containing all raw
246     % Loc_XX folders:
247     cd(raw_path);
248
249 end
250
251 % If no warnings generated report so in log file:
252 if warn == 0
253     fprintf(1,'\n\tFunction completed without errors/warnings\n');
254     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
255 end
256
257 % Update log file that function is completed:
258 fprintf(1,'\n%s completed\n',func_name);
259 fprintf(fid,'\n%s completed\n',func_name);
260
261 end

```

CC_Driver_v5.m

```

1  % Steven Henry
2  % 06/20/2011
34 %*****
35 % PURPOSE:
36 % After "Time_Segment_Driver.m" has been run to generate folders containing
37 % segmented TIFF images this driver is run to track cell centroids.
38 %
39 % ASSUMPTIONS:
40 % (1) In a given experimental condition's "Analysis" folder (generated via
41 % "Time_Segment_Driver.m" user has reserved "Loc_XX" naming for only those
42 % folders containing segmented images.
43 % (2) Segmented images reside in a folder entitled "Segmentation" within
44 % each "Loc_XX" folder.
45 % (3) "Segmentation" folders contain binary (logical) TIFF images
46 % (4) Segmentation image files have names that take the form
47 % "BW_pXXt...t.tif" or "BW_sXXt...t.tif" where p/s denotes "phase"
48 % or "fluor". XX denotes the location corresponding to "Loc_XX".
49 % "t...t" denotes some time stamp in seconds of non-constant length
50 %
51 % DRIVER/FUNCTION MAP:
52 % Level  Name:
53 % 0      CC_Driver_v5.m
54 % 1      ReadFolderContents_v3.m
55 % 1      Track_Centroids_v5.m
56 % 2      Pos_Selection_v3.m
57 % 1      IJ_Manual_Track_Prep_v3.m
58 %*****
59
60 clc
61 clear all
62 close all
63
64 % Have user select the directory:
65 directory = uigetdir(...
66     'EnterPathToYourDataHere','Set Directory');
67
68 % Set directory to user-specified directory:
69 cd(directory);
70
71 % Determine date and time. Create a string in "dateform" "30"
72 % (ISO 8601) which has the format 'yyyymmddTHHMMSS':
73 dstr = datestr(now, 30);
74
75 % Start a log file. Save in user-specified 'directory':
76 % Note: The notation "CC" stands for "connected components"

```

```

77 logfile = [dstr '_CC_Log.txt'];
78 fid = fopen(logfile,'wt');
79
80 % Print directory to log file:
81 fprintf(1,'Selected directory is:\n');
82 fprintf(1, '%s \n\n', directory);
83 fprintf(fid, '%s\n\n', dstr);
84 fprintf(fid, 'Selected directory is:\n');
85 fprintf(fid, '%s \n', directory);
86
87 % Send user-specified path to ReadFolderContents.m for generation of a list
88 % of folders that conform to "Loc_XX" naming convention:
89 [num_folders, folderlist] = ReadFolderContents_v3(directory, fid);
90
91 % Load 't_abs_sorted.mat' from "Sort_Time.m" called in
92 % "Time_Segment_Driver.m". Rows represent frames. Columns represent
93 % locations. This array is passed SORTED from lowest to highest value. It
94 % is not necessarily true that all locations (columns) have the same number
95 % of frames (nonzero rows). The array is designed such that when two
96 % columns do not have an equal number of frames the remainder of the
97 % shorter columns is padded with zeros. For example consider the following
98 % hypothetical three location matrix where the first location has 5 frames,
99 % the second location 3 frames, and the third location 4 frames:
100 %      61  73  87
101 %     123 138 144
102 %     183 195 206
103 %     245  0 266
104 %     306  0  0
105 load t_abs_sorted.mat
106
107 % Load 't_bin.mat' from "Bin_Time.m" called in "Time_Segment_Driver.m"
108 % t_bin = array of binned time-values. This array has taken the absolute
109 % time array in 't_abs_sorted.mat', zeroed each location's origin image
110 % timestamp and binned all remaining times with respect to the closest
111 % multiple of 'interval'. That is all time stamps are now either zero or an
112 % integer multiple of 'interval'.
113 load t_bin.mat
114
115 % Prompt user to supply the maximum number of pixels that an object can
116 % travel between consecutive frames.
117 fprintf(1, '\nSpecify the maximum euclidean distance (pixels) that a cell');
118 fprintf(1, '\ncan travel between two consecutive frames. Typically for');
119 fprintf(1, '\nneutrophils imaged at 20XLWD at 60 sec/frame this value is');
120 fprintf(1, '\n~ 30 pixels\n');
121 fprintf(fid, '\nSpecify the maximum euclidean distance (pixels) that a cell');
122 fprintf(fid, '\ncan travel between two consecutive frames. Typically for');
123 fprintf(fid, '\nneutrophils imaged at 20XLWD at 60 sec/frame this value is');
124 fprintf(fid, '\n~ 30 pixels\n');
125
126 % d_max_pixels = input('\nMaximum distance (pix) = ');
127 % For 20XLWD binning = 1, 60 pix is reasonable
128 % For 20XLWD binning = 2, 30 pix is reasonable
129 d_max_pixels = 30;
130
131 fprintf(1, '\nUser set d_max_pixels = %.0f\n\n', d_max_pixels);
132 fprintf(fid, '\nUser set d_max_pixels = %.0f\n\n', d_max_pixels);
133
134 % Ask the user whether or not these segmented images were from phase
135 % microscopy or fluorescent microscopy. This is required to know whether
136 % the files to be loaded have "BW_pXXt...t.tif" or
137 % "BW_sXXt...t.tif" names:
138 % choice = menu('Are segmented images from phase or fluor imaging?'...
139 % ', 'Phase', 'Fluor');
140 choice = 1;
141 if choice == 1 % Phase
142     prefix = 'p';
143     prefix_print = 'phase "p"';
144 elseif choice == 2 % Fluor

```

```

145     prefix = 's';
146     prefix_print = 'fluor "s"';
147 end
148
149 % Pause to allow menu graphic to clear from view before proceeding:
150 pause(2);
151
152 fprintf(1,'\nUser selected that segmented images are %s\n',prefix_print);
153 fprintf(fid,'\nUser selected that segmented images are %s\n',prefix_print);
154
155 % Have user input values for:
156 % max_frame_skips = user specified # of frames that can be skipped when
157 % linking an object's centroids into a trajectory. Two centroids
158 % separated by a number of frames greater than this value will not be
159 % connected and the object's trajectory will be terminated.
160 % min_frame_track = user specified # of total frames that an object must be
161 % tracked for it to be included in final data set fid = handle to log
162 % file
163 fprintf(1,'\nSpecify max # frames an object can skip and still be tracked. ');
164 fprintf(1,'\nEmpirically 3 frames has been found to be a reasonable value.\n');
165 fprintf(fid,'\nSpecify max # frames an object can skip and still be tracked. ');
166 fprintf(fid,'\nEmpirically 3 frames has been found to be a reasonable value.\n');
167 acceptable = 0;
168 while acceptable == 0;
169
170 %   max_frame_skips = input('\nSet "max_frame_skips" = ');
171     max_frame_skips = 10;
172     fprintf(1,'\n\n\tUser set max_frame_skips = %.0f',max_frame_skips);
173     fprintf(fid,'\n\n\tUser set max_frame_skips = %.0f',max_frame_skips);
174
175     if mod(max_frame_skips,1) ~= 0 || max_frame_skips < 0;
176         fprintf(1,'\n\tWARNING: Entry must be 0 or a positive integer\n');
177         fprintf(fid,'\n\tWARNING: Entry must be 0 or a positive integer\n');
178         acceptable = 0;
179     else
180         fprintf(1,'\n\tEntry acceptable.\n');
181         fprintf(fid,'\n\tEntry acceptable.\n');
182         acceptable = 1;
183     end
184
185 end
186
187 fprintf(1,'\nSpecify min # frames an object must be tracked for inclusion in final data set. ');
188 fprintf(1,'\nMinimally this value must be 2 frames.\n');
189 fprintf(fid,'\nSpecify min # frames an object must be tracked for inclusion in final data set. ');
190 fprintf(fid,'\nMinimally this value must be 2 frames.\n');
191 acceptable = 0;
192 while acceptable == 0;
193
194 %   min_frame_track = input('\nSet "min_frame_track" = ');
195     min_frame_track = 6;
196     fprintf(1,'\n\n\tUser set min_frame_track = %.0f',min_frame_track);
197     fprintf(fid,'\n\n\tUser set min_frame_track = %.0f',min_frame_track);
198
199     if mod(min_frame_track,1) ~= 0 || min_frame_track < 2;
200         fprintf(1,'\n\tWARNING: Entry must be an integer >= 2\n');
201         fprintf(fid,'\n\tWARNING: Entry must be an integer >= 2\n');
202         acceptable = 0;
203     else
204         fprintf(1,'\n\tEntry acceptable.\n');
205         fprintf(fid,'\n\tEntry acceptable.\n');
206         acceptable = 1;
207     end
208
209 end
210
211 % Loop over all "Loc_XX" folders in 'folderlist' which has length
212 % 'num_folders'

```

```

213
214 for i = 1:num_folders
215
216     % Log the name of folder 'i'
217     loc_folder_name = folderlist(i).name;
218
219     % Send column of 't_abs_sorted' and 't_bin' corresponding to folder
220     % 'i':
221     t_abs_vec = t_abs_sorted(:,i);
222     t_bin_vec = t_bin(:,i);
223
224     % Track centroids:
225     [obj] = Track_Centroids_v5(t_abs_vec, t_bin_vec, loc_folder_name, ...
226         directory, d_max_pixels, prefix, max_frame_skips, ...
227         min_frame_track, fid);
228
229     % After all slices have been analyzed for a given "Loc_XX" folder
230     % compile the data into a format acceptable for ImageJ's "Manual
231     % Tracking" plugin which will be used to fine-tune the centroid
232     % analysis.
233     [obj] = IJ_Manual_Track_Prepare_v3(obj, loc_folder_name, directory, fid);
234
235 end
236
237 fprintf(1, '\nProgram terminated\n\n');
238 fprintf(fid, '\nProgram terminated\n\n');
239
240 fclose(fid);

```

Track_Centroids_v5.m

```

1  % Steven J. Henry
2  % 06/20/2011
31 %*****
32 % PURPOSE:
33 % This function tracks cell centroids using binary TIFF images previously
34 % segmented via "Segment.m".
35 %
36 % ASSUMPTIONS:
37 % There is a direct mapping between the entries of the sorted time matrix
38 % 't_abs_sorted' and the files residing within the "Segmentation" folder
39 % of the "_Time_Segment" folder for a particular experimental condition.
40 % Basically, this is a long way of saying, the assumption is the user
41 % hasn't removed segmented files from the "Segmentation" folder prior to
42 % analysis
43 %
44 % INPUT:
45 % t_abs_vec = row vector of SORTED absolute time values. Rows represent
46 % frames. Vector is passed SORTED from lowest to highest value. It is not
47 % necessarily true that all rows have data as a particular Loc_XX may
48 % have had less frames than another Loc_XX in the same experimental
49 % condition. If this is the case the remainder of the column is padded
50 % with zeros. For example consider the following hypothetical three
51 % location matrix where the first location has 5 frames, the second
52 % location 3 frames, and the third location 4 frames:
53 %      61  73  87
54 %      123 138 144
55 %      183 195 206
56 %      245 0  266
57 %      306 0  0
58 % In this case t_abs_vec would be a single column of this array.
59 % t_bin_vec = binned or rounded absolute time values. Same structure and
60 % rules apply as in t_abs_vec, except that the first row should now
61 % contain a zero (i.e. frame 1 occurs at a relative time of zero).
62 % loc_folder_name = string in form of "Loc_XX"
63 % directory = user specified path to an "Analysis" folder containing the
64 % results of segmentation analysis on multiple imaging locations
65 % d_max_pixels = user specified upper bound on euclidean distance (in

```

```

66 % pixels) a cell can be considered to move during a single frame
67 % prefix = string 'p' for phase or 's' for fluor to appropriately load
68 % images either BW_pXXttt...ttt.tif or BW_sXXttt...ttt.tif
69 % max_frame_skips = user specified # of frames that can be skipped when
70 % linking an object's centroids into a trajectory. Two centroids
71 % separated by a number of frames greater than this value will not be
72 % connected and the object's trajectory will be terminated.
73 % min_frame_track = user specified # of total frames that an object must be
74 % tracked for it to be included in final data set fid = handle to log
75 % file
76 %
77 % OUTPUT
78 % obj = cell array of length equal to number of objects identified in first
79 % frame of stack. Each cell contains a matrix with the following
80 % organization:
81 % col 1 = space holder for ImageJ Manual Tracking compatibility
82 % col 2 = object number
83 % col 3 = frame number in which object ('obj' row position) is found
84 % col 4 = x coordinate (pixels) of centroid object ('obj' row position)
85 % col 5 = y coordinate (pixels) of centroid object ('obj' row position)
86 % col 6 = absolute time cooresponding to frame in which object is found
87 % col 7 = binned time corresponding to frame in which object is found
88 % col 8 = area of object
89 %
90 % FUNCTIONS CALLED:
91 % Pos_Selection_v2.m
92 %*****
93
94 function [obj] = Track_Centroids_v5(t_abs_vec, t_bin_vec,...
95     loc_folder_name, directory, d_max_pixels, prefix, max_frame_skips, ...
96     min_frame_track, fid)
97
98 % Get function name:
99 func_name = mfilename;
100
101 % Update log file that function is running:
102 fprintf(1, '\n%s running on %s ...\n', func_name, loc_folder_name);
103 fprintf(fid, '\n%s running on %s ...\n', func_name, loc_folder_name);
104
105 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
106 % warning dialog the log file records no errors/warnings generated:
107 warn = 0;
108
109 % Log the location number of folder 'i' (i.e. the "XX" portion of "Loc_XX")
110 loc_num = sprintf('%02.0f', str2double(loc_folder_name(5:end)));
111
112 % Create a character string to the "Segmentation" folder in the given
113 % "Loc_XX" folder containing segmented iamges (binary TIFF files):
114 seg_folder_path = [directory '\ loc_folder_name "Segmentation"];
115
116 % Change directory to "Segmentation" folder
117 cd(seg_folder_path);
118
119 % Determine number of slices or frames that we must look for in
120 % "Segementation" this is the number of nonzero elements in 't_abs_sorted'
121 % for column 'i' which corresponds to folder 'i' in 'folderlist'
122 num_slices = nnz(t_abs_vec(:));
123
124 % Loop over the number of frames in this folder determined via 't_abs_vec'
125 for j = 1:num_slices
126
127     % Extract time string for slice 'j':
128     t_str = sprintf('%03.0f', t_abs_vec(j));
129
130     % Concatenate desired frame name:
131     base_image_name = [prefix loc_num t_str '.tif'];
132     BW_image_name = ['BW_' base_image_name];
133

```

```

134 % Load frame:
135 BW = imread(BW_image_name);
136
137 % Generate list of connected components:
138 cc = bwconncomp(BW);
139
140 % Create a structural array to hold information on geometric centroid
141 % and area of each object (connected component in 'cc') identified in
142 % this BW image:
143 cc_stats = regionprops(cc, 'Centroid', 'Area');
144
145 % If this is the first frame, generate a cell array to hold object
146 % information (abs_time, bin_time, centroid x pos, centroid y pos, and
147 % object area):
148 if j == 1
149
150     obj = cell(cc.NumObjects,1);
151
152     % Loop over all detected objects in frame 1:
153     for k = 1:cc.NumObjects
154
155         % Create a row vector of 8 positions:
156         obj_info = zeros(1,8);
157         % Position 1 is a spacer that will not hold any meaningful
158         % data. It is necessary for compatibility with ImageJ's Manual
159         % Tracking Plugin.
160         % Position 2 holds object number:
161         obj_info(2) = k;
162         % Position 3 holds frame (slice) number:
163         obj_info(3) = j;
164         % Position 4 holds x component of object 'k' centroid
165         obj_info(4) = cc_stats(k).Centroid(1);
166         % Position 5 holds y component of object 'k' centroid
167         obj_info(5) = cc_stats(k).Centroid(2);
168         % Position 6 holds absolute time value corresponding to frame
169         % j:
170         obj_info(6) = t_abs_vec(j);
171         % Position 7 holds binned time value corresponding to frame j:
172         obj_info(7) = t_bin_vec(j);
173         % Position 8 holds object area (# pixels)
174         obj_info(8) = cc_stats(k).Area;
175         % After all positions have proper values logged, write the
176         % single vector to the appropriate cell in 'obj':
177         obj{k} = obj_info;
178
179     end
180
181     % Retain the total number of initial objects detected. This is the
182     % number of objects tracked for the remainder of processing, it
183     % will not increase. After the first frame the program only seeks
184     % to identify these initialized objects in all following frames.
185     % This prevents tracking of objects that enter the field of view
186     % during data collection. It is not efficient in the sense that the
187     % program will attempt to find objects that may have already left
188     % the field of view. However, from an empirical perspective the
189     % current efficiency is satisfactory.
190     num_obj = cc.NumObjects;
191
192     % Otherwise j ~= 1 so this 'BW' frame is not the first frame and
193     % 'obj' array already exists
194 else
195
196     % Create a column vector of length equal to the number of objects
197     % being tracked from frame 1. Row position corresponds to the object
198     % being tracked from frame 1. The vector will hold the IDs of the
199     % objects in this current non-origin frame (j~=1) that are found to
200     % be continuations of the objects being tracked from frame 1.
201     obj_match = zeros(num_obj,1);

```

```

202
203 % Loop over the number of original objects being tracked:
204 for k = 1:num_obj
205
206     % Load the last observed centroid of object 'k':
207     obj_info = obj{k};
208     x_o = obj_info(end,4);
209     y_o = obj_info(end,5);
210
211     % Reserve a temporary row vector with length = to the number of
212     % newly detected objects in the current frame (~= 1). That is
213     % length(obj) does not necessarily have to equal
214     % length(d_vector):
215     d_vector = zeros(cc.NumObjects,1);
216
217     % Loop over the number of new objects detected in this "new"
218     % frame 'j':
219     for kk = 1:cc.NumObjects
220
221         % Load the centroid of the new object:
222         x_new = cc_stats(kk).Centroid(1);
223         y_new = cc_stats(kk).Centroid(2);
224
225         % Compute Euclidean distance  $d^2 = x^2 + y^2$ :
226         d = sqrt((x_new - x_o)^2 + (y_new - y_o)^2);
227
228         % Log this distance value into 'd_vector'. Note this vector
229         % contains the distances from object 'k' of frame 1 to all
230         % objects 1:'kk' of the present frame corresponding to
231         % image 'BW':
232         d_vector(kk) = d;
233
234     end
235
236     % Identify the minimum euclidean distance between objects in
237     % frame 'j' and objects in frame '1':
238     d_min = min(d_vector);
239
240     % If this minimum value is less than the maximum permissible
241     % travel between consecutive frames:
242     if d_min <= d_max_pixels
243
244         % Determine which object(s) correspond to minimum euclidean
245         % distance. Here 'pos' is a vector which contains the
246         % indices (row #) of entries in 'd_vector' that are set
247         % equal to 'd_min':
248         pos = find(d_vector == d_min);
249
250         if length(pos) > 1
251             % Select which object corresponds to object 'k' via
252             % area consideration:
253             areas = regionprops(cc,'Area');
254             [pos] = Pos_Selection_v2(pos, areas, obj, k);
255             clear areas
256             fprintf(1,'\n\tNote: In frame %.0f object %.0f required area consideration to track centroid.\n',j,k);
257             fprintf(fid,'\n\tNote: In frame %.0f object %.0f required area consideration to track centroid.\n',j,k);
258         end
259
260         % Determine the number of frames that have elapsed
261         % between this frame 'j' and the last frame a centroid
262         % was recorded for object 'k' (column 3):
263         frame_elapse = j - obj_info(end,3);
264
265         % If the number of elapsed frames less than or equal to an
266         % allowed threshold proceed to incorporate the latest 'pos'
267         % info, otherwise skip it:
268         if frame_elapse <= max_frame_skips
269             obj_match(k) = pos;

```

```

270         end
271     end
272 end
273
274 end
275
276 % At this point all object matching is completed and we must check
277 % if multiple objects have declared the same position in frame 'j'
278 % for their next occupancy (implying a collision):
279 clear pos
280 % Identify all nonzero entries in 'obj_match' vector:
281 pos_all = nonzeros(obj_match);
282 % Identify all unique nonzero entries in 'obj_match' vector:
283 pos_unique = unique(pos_all);
284 % If the number of nonzero entries in 'obj_match' is greater than
285 % the number of unique nonzero entries in 'obj_match' a collision
286 % has occurred:
287 if length(pos_all) > length(pos_unique)
288
289     for m = 1:length(pos_all)
290
291         [pos_ind] = find(obj_match(:) == pos_all(m));
292
293         if length(pos_ind) > 1
294
295             for mm = 1:length(pos_ind)
296
297                 obj_match(pos_ind(mm)) = 0;
298
299             end
300
301         end
302
303     end
304
305 end
306
307
308 % Once all collisions have been eliminated update object
309 % information:
310 for k = 1:num_obj
311
312     if obj_match(k) ~= 0
313         % Update object 'k' info by adding a new row to the
314         % existing object's information matrix:
315         obj_info_old = obj{k};
316         [row, col] = size(obj_info_old);
317         obj_info_new = zeros(row+1, col);
318         obj_info_new(1:row, 1:col) = obj_info_old;
319         % Position 2 holds object number:
320         obj_info_new(row+1, 2) = k;
321         % Position 3 holds frame (slice) number:
322         obj_info_new(row+1, 3) = j;
323         % Position 4 holds x component of object 'pos' centroid:
324         obj_info_new(row+1, 4) = cc_stats(obj_match(k)).Centroid(1);
325         % Position 5 holds y component of object 'pos' centroid
326         obj_info_new(row+1, 5) = cc_stats(obj_match(k)).Centroid(2);
327         % Position 6 holds absolute time value corresponding to
328         % frame j:
329         obj_info_new(row+1, 6) = t_abs_vec(j);
330         % Position 7 holds binned time value corresponding to frame
331         % j:
332         obj_info_new(row+1, 7) = t_bin_vec(j);
333         % Position 8 holds object area (# pixels)
334         obj_info_new(row+1, 8) = cc_stats(obj_match(k)).Area;
335         % After all positions have proper values logged, write the
336         % single vector to the appropriate cell in 'obj':
337         obj{k} = [];

```



```

338         obj{k} = obj_info_new;
339     end
340 end
341 end
342 end
343 end
344 end
345 end
346 end
347
348 % Now filter out objects that have been tracked for less than a minimum
349 % number of frames:
350 min_frames = min_frame_track;
351
352 for n = 1:num_obj %Reverse order not required b/c 'obj' is cell array
353
354     obj_frames = length(obj{n}(:,2));
355
356     if obj_frames < min_frames
357
358         obj{n} = [];
359     end
360 end
361 end
362 end
363
364 % If no warnings generated report so in log file:
365 if warn == 0
366     fprintf(1,'\n\tFunction completed without errors/warnings\n');
367     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
368 end
369
370 % Update log file that function is completed:
371 fprintf(1,'\n%s completed\n',func_name);
372 fprintf(fid,'\n%s completed\n',func_name);
373
374 end

```

Pos_Selection_v2.m

```

1 % Steven J. Henry
2 % 04/19/2011
10 %*****
11 % PURPOSE:
12 % This function is written in conjunction with "Track_Centroids.m". In the
13 % event that multiple objects in the present frame reside a distance
14 % 'd_min' from the object 'k' of interest the object with the minimum
15 % change in area is selected as the continuation of object 'k' in this
16 % frame.
17 %
18 % ASSUMPTIONS:
19 % Areas do not vary "dramatically" between frames.
20 %
21 % INPUT:
22 % pos = row vector minimally of length(pos) = 1 containing object IDs of
23 % connected components array cc that reside within the neighborhood of
24 % tracking object 'k'
25 % areas = areas of connected components in present frame
26 % obj = cell array containing all objects being tracked (as determined at
27 % frame 1)
28 % k = object in 'obj' array presently being analyzed in relation to objects
29 % specified by 'pos' mapping to 'cc'
30 %
31 % OUTPUT:
32 % pos = most probable object in 'cc' that is object 'k' in previous frame.
33 %*****
34
35 function [pos] = Pos_Selection_v2(pos, areas, obj, k)

```

```

36
37 % How many connected components in the present frame need to be compared to
38 % object 'k'?
39 num_to_comp = length(pos);
40
41 % Retrieve object 'k' info matrix:
42 obj_info = obj{k};
43 % Isolate area column vector (col 8) from info matrix:
44 obj_area = obj_info(:,8);
45 % Compute mean of area entries (minimally this has a single
46 % entry):
47 obj_m_area = mean(obj_area);
48
49 % Get the areas and euclidean distance values corresponding to objects in
50 % 'pos' store these in 'pos_data' with column 1 = row position reference,
51 % column 2 = area of object 'i' in 'pos', column 3 = euclidean distance of
52 % object 'i' to tracked object 'k'
53 pos_data = zeros(num_to_comp, 2);
54 pos_data(:,1) = pos;
55 for i = 1:num_to_comp
56     pos_data(i,2) = areas(pos(i)).Area;
57 end
58
59 % Compute change in area from object 'k' (previous frame) and neighbor
60 % objects:
61 delta_area = abs(pos_data(:,2) - obj_m_area);
62
63 % Find minimum change in area:
64 delta_area_min = min(delta_area);
65
66 % Find the indices of the entries in 'delta_area' that contain
67 % 'delta_area_min' values:
68 [area_ind] = find(delta_area == delta_area_min);
69
70 % Since 'delta_area', 'pos', and 'pos_data' have the same order, use
71 % 'area_ind' to determine which rows of 'pos_data' and 'pos' to retain
72 pos = pos(area_ind);
73
74 % At this point it is still theoretically possible that 'pos' contains
75 % multiple values. If this is the case just output the first entry as
76 % the correct position:
77 if length(pos) > 1
78     pos = pos(1);
79
80
81 end
82
83 end

```

IJ_Manual_Track_Prep_v3.m

```

1 % Steven J. Henry
2 % 06/20/2011
18 %*****
19 % PURPOSE:
20 % This function takes the 'obj' cell array that contains information of
21 % object centroid tracking from "Track_Centroids.m" and performs
22 % manipulations necessary to make it suitable for import into "ImageJ
23 % Manual Tracking Plugin".
24 %
25 % ASSUMPTIONS:
26 % n/a
27 %
28 % INPUT:
29 % obj_cell_array = cell array of length equal to number of objects
30 % identified in first frame of stack. Each cell contains a matrix with
31 % the following organization:
32 % col 1 = space holder for ImageJ Manual Tracking compatibility

```

```

33 % col 2 = object number
34 % col 3 = frame number in which object ('obj' row position) is found
35 % col 4 = x coordinate (pixels) of centroid object ('obj' row position)
36 % col 5 = y coordinate (pixels) of centroid object ('obj' row position)
37 % col 6 = absolute time cooresponding to frame in which object is found
38 % col 7 = binned time corresponding to frame in which object is found
39 % col 8 = area of object for filter applied in "Area_Consistency.m"
40 % loc_folder_name = string in form of "Loc_XX"
41 % directory = user specified path to an "Analysis" folder containing the
42 % results of segmentation analysis on multiple imaging locations
43 % fid = file ID to which warnings and progress is printed as a text
44 % file.
45 % dstr = date and time string in "dateform" "30" (ISO 8601) which has the
46 % format 'yyyymmddTHHMMSS'.
47 %
48 % OUTPUT:
49 % obj_num_array = numeric array of length equal to number of objects
50 % thought to be cells times the number of total frames those objects were
51 % tracked having the following organization:
52 % col 1 = space holder for ImageJ Manual Tracking compatibility
53 % col 2 = unique track number ID assigned to each object
54 % col 3 = frame number in which object ('obj' row position) is found
55 % col 4 = x coordinate (pixels) of centroid object ('obj' row position)
56 % col 5 = y coordinate (pixels) of centroid object ('obj' row position)
57 % col 6 = absolute time cooresponding to frame in which object is found
58 % col 7 = binned time corresponding to frame in which object is found
59 % col 8 = area of object in pixels
60 % col 9 = track change flag. Entry = 1 if start of new track (yes) or 0
61 % if no (i.e. continuation of an existing track.
62 %*****
63
64 function [obj_num_array] = IJ_Manual_Track_Prepare_v3(obj_cell_array, loc_folder_name, directory, fid)
65
66 % Determine number of objects tracked:
67 num_objs = length(obj_cell_array);
68
69 % Reserve variable name:
70 obj_num_array = [];
71
72 % Loop over these objects
73 for i = 1:num_objs
74
75     % Extract info array for object 'i' from 'obj_cell_array' supplied:
76     obj_info = obj_cell_array{i};
77
78     if isempty(obj_info) == 0
79
80         % If this is the first cell in object with actual data:
81         if isempty(obj_num_array) == 1
82
83             % Set the output array equal to the info array of this object:
84             obj_num_array = obj_info;
85
86             % Otherwise if there are more than one objects:
87             else
88
89                 % Record the previous output information to be expanded
90                 obj_num_array_old = obj_num_array;
91                 % Clear the reserved output variable name
92                 clear obj_num_array
93                 % Concatenate the old output array info with the new object 'i'
94                 % info:
95                 obj_num_array = vertcat(obj_num_array_old, obj_info);
96
97             end
98
99         end
100

```

```

101 end
102
103 % Determine dimensions of output array:
104 [rows, cols] = size(obj_num_array);
105
106 % If the number of columns is not equal to 8 then some error occurred upon
107 % concatenation or "Track_Centroids.m" provided erroneous 'obj' info:
108 if cols ~= 8
109     fprintf(1, '\nWARNING: vertcat resulted in array without 8 columns\n\n');
110     fprintf(fid, '\nWARNING: vertcat resulted in array without 8 columns\n\n');
111 end
112
113 % Add a 9th column to hold "Track Change" flag:
114 obj_num_array_old = obj_num_array;
115 clear obj_num_array
116 obj_num_array = zeros(rows, cols+1);
117 obj_num_array(1:rows, 1:cols) = obj_num_array_old;
118
119 % Turn on "track change flag" when new object ID is detected:
120 % Loop over all entries in 'obj_num_array'
121 for j = 1:rows
122
123     % If this is the first entry set Track Change flag "on" = 1
124     if j == 1;
125
126         obj_num_array(1,9) = 1;
127
128         % Otherwise if this is not the first entry determine if this entry
129         % has the same track ID as the previous entry. If so keep track
130         % change flag "off" (0), if not turn track change flag "on" (1).
131     else
132
133         % Load previous track ID and current track ID:
134         obj_ID_prior = obj_num_array(j-1,2);
135         obj_ID_now = obj_num_array(j, 2);
136
137         % If they are not the same
138         if obj_ID_now ~= obj_ID_prior
139
140             % Turn track chang flag on for this 'j' entry:
141             obj_num_array(j,9) = 1;
142
143         else
144
145             % Otherwise ensure track change flag is off (this is
146             % redundant).
147             obj_num_array(j,9) = 0;
148
149         end
150     end
151 end
152
153 end
154
155 % Now re-assign object ID so objects have consecutive track ID starting at
156 % 1:
157 for jj = 1:rows
158
159     if jj == 1
160
161         obj_num_array(jj,2) = 1;
162
163     else
164
165         obj_ID_prior = obj_num_array(jj-1,2);
166         track_chng_flag = obj_num_array(jj,9);
167
168         if track_chng_flag == 1;

```

```

169
170     obj_ID_now = obj_ID_prior+1;
171
172     else
173
174         obj_ID_now = obj_ID_prior;
175
176     end
177
178     obj_num_array(jj,2) = obj_ID_now;
179
180 end
181
182 end
183
184 % Set directory to "Loc_XX" folder within "Analysis" folder:
185 loc_path = [directory '\ loc_folder_name];
186 cd(loc_path);
187
188 % Write 'obj_num_array' to a text file and save as a .mat file for later
189 % revision:
190 txt_fname = [loc_folder_name '.txt'];
191 fid2 = fopen(txt_fname, 'wt');
192 fprintf(1, '\nSaving %s in:\n', txt_fname);
193 fprintf(1, '%s\n\n', pwd);
194 fprintf(fid, '\nSaving %s in:\n', txt_fname);
195 fprintf(fid, '%s\n\n', pwd);
196 fprintf(fid2, 'n/a\tID\tSlice\tX(pixel)\tY(pixel)\tTime(s)\tBinTime(s)\tArea(pixels)\tTrackChange\n');
197 for k = 1:rows
198     fprintf(fid2, '%.0f\t%.0f\t%.0f\t%f\t%f\t%.0f\t%.0f\t%.0f\t%.0f\n', obj_num_array(k,:));
199 end
200
201 mat_fname = [loc_folder_name '.mat'];
202 save(mat_fname, 'obj_num_array');
203
204 cd('.');
205
206 fclose(fid2);
207
208 end

```

CC_Output_Editor_v2.m

```

1 % Steven J. Henry
2 % 02/16/2015
12 %*****
13 % PURPOSE:
14 % This program is run on individual "Loc_XX.mat" files after "CC_Driver.m"
15 % has generated a data set compatible with ImageJ's Manual Tracking Plugin.
16 % It operates on "Loc_XX.mat" to eliminate entire tracks and portions of
17 % tracks specified by the user (entered manually) to generate an edited
18 % "Loc_XX_edited.mat" and "Loc_XX_edited.txt" file that only contains
19 % cell centroids to be used in MSD computation.
20 %
21 % Note: MATLAB generates warnings related to this program (see orange flags
22 % to the right in the Editor window). These are notifying the user that
23 % care has not been taken with respect to memory conservation. Because the
24 % data sets being processed at a time are relatively small this program is
25 % sloppy and allows arrays to grow and shrink without reserving the
26 % appropriate block of memory.
27 %
28 % ASSUMPTIONS:
29 % n/a
30 %
31 % INPUT:
32 % obj_num_array = numeric array of length equal to number of objects
33 % thought to be cells times the number of total frames those objects were
34 % tracked having the following organization:

```

```

35 % col 1 = space holder for ImageJ Manual Tracking compatibility
36 % col 2 = unique track number ID assigned to each object
37 % col 3 = frame number in which object ('obj' row position) is found
38 % col 4 = x coordinate (pixels) of centroid object ('obj' row position)
39 % col 5 = y coordinate (pixels) of centroid object ('obj' row position)
40 % col 6 = absolute time cooresponding to frame in which object is found
41 % col 7 = binned time corresponding to frame in which object is found
42 % col 8 = area of object in pixels
43 % col 9 = track change flag. Entry = 1 if start of new track (yes) or 0
44 %   if no (i.e. continuation of an existing track.
45 % OUTPUT:
46 % obj_num_array = same structure as input but excluding tracks and portions
47 %   tracks manually deemed unsuitable for inclusion in the final data set
48 %   for MSD computation.
49 %*****
50
51 clc
52 clear all
53 close all
54
55 % Get "Loc_XX.mat" file name and path:
56 [filename, pathname] = uigetfile(...
57   'EnterPathToYourDataHere',...
58   'Select File');
59 % Set directory to user-specified path:
60 cd(pathname);
61
62 num_char = length(filename);
63 filename_no_ext = filename(1:num_char-4);
64
65 % Determine date and time. Create a string in "dateform" "30" (ISO 8601)
66 % which has the format 'yyymmddTHHMMSS'.
67 dstr = datestr(now, 30);
68 log_fname = [dstr '_' filename_no_ext '_CC_Edits_Log.txt'];
69 fid1 = fopen(log_fname, 'wt');
70
71 fprintf(1, 'User selected to edit the following file:\n');
72 fprintf(1, '%s\n', [pathname filename]);
73 fprintf(fid1, '%s\n\n', dstr);
74 fprintf(fid1, 'User selected to edit the following file:\n');
75 fprintf(fid1, '%s\n', [pathname filename]);
76
77 % Load "Loc_XX.mat" file which contains variable 'obj_num_array':
78 load(filename);
79
80 % User supplied vector of track IDs to be completely eliminated. This can
81 % be a row or column vector
82 entire_dels = [];
83 num_entire_dels = length(entire_dels);
84
85 % Eliminate data OUTSIDE (not including) user-specified boundaries (i.e.
86 % frames) for a given track. Input structure should be:
87 % Column 1 = track ID
88 % Column 2 = lower bound, first frame cell should be followed
89 % Column 3 = upper bound, last fraem cell should be followed
90 partial_dels_out = [];
91 num_partial_dels_out = size(partial_dels_out,1);
92
93 % Eliminate data INSIDE (and including) user-specified boundaries (i.e.
94 % frames) for a given track. Input structure should be:
95 % Column 1 = track ID
96 % Column 2 = lower bound, first frame cell data should be eliminated from
97 % Column 3 = upper bound, last frame cell data should be eliminated from
98 partial_dels_in = [];
99 num_partial_dels_in = size(partial_dels_in,1);
100
101 % Perform entire track deletions:
102 if num_entire_dels > 0

```

```

103
104 fprintf(1,'\nTracks tagged for complete deletion:\n');
105 fprintf(fid1,'\nTracks tagged for complete deletion:\n');
106
107 for i = 1:num_entire_dels
108
109     obj_ID = entire_dels(i);
110
111     fprintf(1,'%0f\n',obj_ID);
112     fprintf(fid1,'%0f\n',obj_ID);
113
114     obj_ind = find(obj_num_array(:,2) == obj_ID);
115
116     obj_ind = sort(obj_ind, 'descend');
117
118     for ii = 1:length(obj_ind)
119
120         obj_num_array(obj_ind(ii),:) = [];
121
122     end
123
124 end
125
126 end
127
128 % Perform deletions of data outside of user-specified bounds:
129 if num_partial_dels_out > 0
130
131     fprintf(1,'\nPartial deletions:');
132     fprintf(1,'\nData retained INSIDE (including) the following bounds:');
133     fprintf(1,'\nTrack ID\tStart Frame\tStop Frame\n');
134     fprintf(fid1,'\nPartial deletions:');
135     fprintf(fid1,'\nData retained INSIDE (including) the following bounds:');
136     fprintf(fid1,'\nTrack ID\tStart Frame\tStop Frame\n');
137
138     for j = 1:num_partial_dels_out
139
140         obj_ID = partial_dels_out(j,1);
141
142         obj_first_frame = partial_dels_out(j,2);
143         obj_last_frame = partial_dels_out(j,3);
144
145         fprintf(1,'%0ft%0ft%0f\n',...
146             obj_ID,obj_first_frame,obj_last_frame);
147         fprintf(fid1,'%0ft%0ft%0f\n',...
148             obj_ID,obj_first_frame,obj_last_frame);
149
150         % Find indices in obj_num_array corresponding to track 'j' that
151         % violate lower bound set by user:
152         obj_ind_lb = find((obj_num_array(:,2) == obj_ID) & ...
153             (obj_num_array(:,3) < obj_first_frame));
154
155         % Find indices in obj_num_array corresponding to track 'j' that
156         % violate upper bound set by user:
157         obj_ind_ub = find((obj_num_array(:,2) == obj_ID) & ...
158             (obj_num_array(:,3) > obj_last_frame));
159
160         % Retain the union of these two vectors (concatenation would also
161         % be acceptable as we do not anticipate a given row in
162         % obj_num_array could violate both bounds simultaneously).
163         obj_ind = union(obj_ind_lb, obj_ind_ub);
164         obj_ind = sort(obj_ind, 'descend');
165
166         for jj = 1:length(obj_ind)
167
168             obj_num_array(obj_ind(jj),:) = [];
169
170         end

```

```

171
172     end
173
174 end
175
176 % Perform deletions inside of user-specified bounds:
177 if num_partial_dels_in > 0
178
179     fprintf(1, '\nPartial deletions:');
180     fprintf(1, '\nData retained OUTSIDE (not including) the following bounds:');
181     fprintf(1, '\nTrack ID\tStart Frame\tStop Frame\n');
182     fprintf(fid1, '\nPartial deletions:');
183     fprintf(fid1, '\nData retained OUTSIDE (not including) the following bounds:');
184     fprintf(fid1, '\nTrack ID\tStart Frame\tStop Frame\n');
185
186     for j = 1:num_partial_dels_in
187
188         obj_ID = partial_dels_in(j,1);
189
190         obj_first_frame = partial_dels_in(j,2);
191         obj_last_frame = partial_dels_in(j,3);
192
193         fprintf(1, '%.0ft%.0ft%.0f\n', obj_ID, obj_first_frame, obj_last_frame);
194         fprintf(fid1, '%.0ft%.0ft%.0f\n', obj_ID, obj_first_frame, obj_last_frame);
195
196         % Find indices in obj_num_array corresponding to track 'j' that
197         % violate lower bound set by user:
198         obj_ind_lb = find((obj_num_array(:,2) == obj_ID) & (obj_num_array(:,3) >= obj_first_frame));
199
200         % Find indices in obj_num_array corresponding to track 'j' that
201         % violate upper bound set by user:
202         obj_ind_ub = find((obj_num_array(:,2) == obj_ID) & (obj_num_array(:,3) <= obj_last_frame));
203
204         % Retain the intersection of these two vectors:
205         obj_ind = intersect(obj_ind_lb, obj_ind_ub);
206         obj_ind = sort(obj_ind, 'descend');
207
208         for jj = 1:length(obj_ind)
209
210             obj_num_array(obj_ind(jj), :) = [];
211
212         end
213
214     end
215
216 end
217
218 % Note the following two loops were taken straight out of
219 % 'IJ_Manual_Track_Preparation_v3.m'.
220
221 % Determine dimensions of output array:
222 [rows, cols] = size(obj_num_array);
223
224 % Turn on "track change flag" when new object ID is detected: Loop over all
225 % entries in 'obj_num_array'. We need to repeat this step previously
226 % performed in 'IJ_Manual_Track_Preparation.m' in the event that the user has
227 % deleted an object's position in frame 1 but tracks that object in
228 % subsequent frames, thereby eliminating it's TCF marker.
229 for k = 1:rows
230
231     % If this is the first entry set Track Change flag "on" = 1
232     if k == 1;
233
234         obj_num_array(1,9) = 1;
235
236     % Otherwise if this is not the first entry determine if this entry
237     % has the same track ID as the previous entry. If so keep track
238     % change flag "off" (0), if not turn track change flag "on" (1).

```



```

307     obj_num_array(r,:);
308     fprintf(fid2,...
309         '%.0ft%.0ft%.0ft%.12ft%.12ft%.0ft%.0ft%.0ft%.0ft%',...
310         obj_num_array(r,:));
311 end
312
313 mat_fname = [filename_no_ext '_edited.mat'];
314 save(mat_fname, 'obj_num_array');
315
316 fprintf(1, '\nProgram terminated\n\n');
317 fprintf(fid1, '\nProgram terminated\n\n');
318
319 fclose(fid1);
320 fclose(fid2);

```

Merge_Mats_v4.m

```

1  % Steven Henry
2  % 02/16/2015
27 %*****
28 % PURPOSE:
29 % After "CC_Driver.m" and "CC_Output_Editor.m" have been run and a series
30 % of '.mat' files containing centroid tracking data result, this program is
31 % run to merge all these location specific files into a single data file
32 % representative of all cells tracked for the given experimental condition.
33 %
34 % ASSUMPTIONS:
35 % (1) User only supplies '.mat' files relevant to the given experimental
36 % condition being analyzed
37 %
38 % INPUT:
39 % Individual .mat files containing the array "obj_num_array" that has the
40 % following structure:
41 % col 1 = space holder for ImageJ Manual Tracking compatibility
42 % col 2 = unique track number ID assigned to each object
43 % col 3 = frame number in which object ('obj' row position) is found
44 % col 4 = x coordinate (pixels) of centroid object ('obj' row position)
45 % col 5 = y coordinate (pixels) of centroid object ('obj' row position)
46 % col 6 = absolute time corresponding to frame in which object is found
47 % col 7 = binned time corresponding to frame in which object is found
48 % col 8 = area of object in pixels
49 % col 9 = track change flag. Entry = 1 if start of new track (yes) or 0
50 %   if no (i.e. continuation of an existing track.
51 %
52 % OUTPUT:
53 % Single .mat file and .txt file with same structure as stated in INPUT but
54 % having unique and consecutive track IDs assigned to all cells.
55 %*****
56
57 clc
58 clear all
59 close all
60
61 % Have user select directory where files reside:
62     directory = uigetdir('EnterPathToYourDataHere',...
63         'Select folder containing .mat files to be merged:');
64
65 % Have user select files to merge:
66     mat_file = uigetfile([directory '*.mat'],...
67         'Select .mat files to merge:', 'MultiSelect', 'on');
68
69 % Set directory to user-specified directory:
70     cd(directory);
71
72 % Determine date and time. Create a string in "dateform" "30"
73 % (ISO 8601) which has the format 'yyyymmddTHHMMSS'.
74     dstr = datestr(now, 30);
75

```

```

76 % Start a log file. Save in user-specified directory:
77 logfile = [dstr '_Merge_Mats_Log.txt'];
78 fid = fopen(logfile,'wt');
79
80 % Before merging '.mat' files make sure each '.mat' file has internally
81 % consistent track change flag assignments. This is to make sure that in
82 % the course of using "CC_Output_Editor.m" We did not inadvertently
83 % eliminate a start row and thereby wipeout the track change flag as well.
84 if iscell(mat_file) == 0
85     num_mats = 1;
86 else
87     num_mats = length(mat_file);
88 end
89
90 % If user only selected one filename put this into a single cell array:
91 if num_mats == 1
92     temp{1} = mat_file;
93     clear mat_file
94     mat_file = temp;
95 end
96
97 fprintf(1,'Checking each .mat file for correct track change flag (TCF) assignments:\n\n');
98 fprintf(fid,'Checking each .mat file for correct track change flag (TCF) assignments:\n\n');
99
100 pass = 0;
101
102 for i = 1:num_mats
103
104     fprintf(1,'\nProcessing %s now:\n',mat_file{i});
105     fprintf(fid,'\nProcessing %s now:\n',mat_file{i});
106
107     % Set 'all_clear' flag on. If on after file processing it means file
108     % passes internal consistency check.
109     all_clear = 1;
110
111     % Load file
112     load(mat_file{i});
113
114     % Determine size
115     [rows cols] = size(obj_num_array);
116
117     % % If the file doesn't have exactly 9 columns tell user
118     % if cols ~= 9
119     %     fprintf(1,'WARNING: file has %.0f cols not 9 as required\n',cols);
120     %     fprintf(fid,'WARNING: file has %.0f cols not 9 as required\n',cols);
121     %     all_clear = -1;
122     % end
123
124     % Loop over rows and check that every time a new track ID occurs the
125     % track change flag (TCF) has value = 1:
126     for ii = 1:rows
127
128         if ii == 1
129
130             if obj_num_array(ii,9) ~= 1
131                 fprintf(1,'WARNING: Track %.0f does not have TCF = 1 at row %0.f\n',obj_num_array(ii,2),ii);
132                 fprintf(fid,'WARNING: Track %.0f does not have TCF = 1 at row %0.f\n',obj_num_array(ii,2),ii);
133                 all_clear = -1;
134             end
135
136         else
137
138             ID_prior = obj_num_array(ii-1,2);
139             ID_now = obj_num_array(ii,2);
140             TCF = obj_num_array(ii,9);
141
142             if ID_now ~= ID_prior
143

```

```

144     if TCF ~= 1
145         fprintf(1,'WARNING: Track %.0f does not have TCF = 1 at row %0.f\n',obj_num_array(ii,2),ii);
146         fprintf(fid,'WARNING: Track %.0f does not have TCF = 1 at row %0.f\n',obj_num_array(ii,2),ii);
147         all_clear = -1;
148     end
149
150     else
151
152         if TCF ~= 0
153             fprintf(1,'WARNING: Track %.0f does not have TCF = 0 at row %0.f\n',obj_num_array(ii,2),ii);
154             fprintf(fid,'WARNING: Track %.0f does not have TCF = 0 at row %0.f\n',obj_num_array(ii,2),ii);
155             all_clear = -1;
156         end
157
158     end
159
160     end
161
162 end
163
164 % If you didn't trip the 'all_clear' flag tell user file is OK:
165 if all_clear == 1
166     fprintf(1,'File is internally consistent\n');
167     fprintf(fid,'File is internally consistent\n');
168     pass = pass + 1;
169 end
170
171 clear obj_num_array
172
173 end
174
175 % If all files are internally consistent allow program to proceed:
176 if pass == num_mats
177
178     % Merge mats:
179     for j = 1:num_mats
180
181         load(mat_file{j});
182
183         if j == 1
184             data = obj_num_array;
185         else
186             [new_rows new_cols] = size(obj_num_array);
187             data_old = data;
188             [old_rows old_cols] = size(data_old);
189             clear data
190             data = zeros(old_rows+new_rows, old_cols);
191             data(1:old_rows, 1:old_cols) = data_old;
192             data(old_rows+1:old_rows+new_rows,1:old_cols) = obj_num_array;
193         end
194
195         clear obj_num_array
196
197     end
198
199     % Now re-assign object ID so cells have unique and consecutive track
200     % IDs starting at 1:
201     [data_rows data_cols] = size(data);
202     for jj = 1:data_rows
203
204         if jj == 1
205             data(jj,2) = 1;
206         else
207
208             ID_prior = data(jj-1,2);
209             TCF = data(jj,9);
210
211             if TCF == 1;

```

```

212
213         ID_now = ID_prior+1;
214
215     else
216
217         ID_now = ID_prior;
218
219     end
220
221     data(jj,2) = ID_now;
222
223 end
224
225 end
226
227 % Write 'data' to a text file and save as a .mat file for later analysis
228 txt_fname = [dstr '_Merged_Data.txt'];
229 fid2 = fopen(txt_fname, 'wt');
230 fprintf(1, '\nSaving %s in:\n', txt_fname);
231 fprintf(1, '%s\n\n', pwd);
232 fprintf(fid2, '\nSaving %s in:\n', txt_fname);
233 fprintf(fid2, '%s\n\n', pwd);
234 fprintf(fid2, 'n/a\tID\tSlice\tX(pix)\tY(pix)\tTime(s)\tBinTime(s)\tArea(pix)\tTrackChange\n');
235 for k = 1:data_rows
236     fprintf(fid2, '%.0ft%.0ft%.0ft%.12ft%.12ft%.0ft%.0ft%.0ft%.0fn', data(k,:));
237 end
238
239 mat_fname = [dstr '_Merged_Data.mat'];
240 save(mat_fname, 'data');
241
242 fclose(fid2);
243
244 fprintf(1, '\nProgram terminated.\n');
245 fprintf(fid2, '\nProgram terminated.\n');
246
247 else
248
249     fprintf(1, '\nNot all .mats internally consistent. Program terminated.\n');
250     fprintf(fid2, '\nNot all .mats internally consistent. Program terminated.\n');
251
252 end
253
254 fclose(fid);

```

Supra_MSD_Driver_v5.m

```

1  % Steven J. Henry
2  % 03/03/2015
21 %*****
22 % PURPOSE:
23 % This program calls on 'MSD_Driver_v14.m' without requiring user-input
24 % upon every iteration. It is intended to operate on a folder containing a
25 % series of .mat files, each corresponding to a different condition, within
26 % a given experiment (i.e. donor/day).
27 %
28 % ASSUMPTIONS:
29 % Folder that Supra_MSD_Driver.m' operates on contains .mat centroid files
30 % with necessary structure and all files are to be processed in identical
31 % fashion (for example same objective calibration, same max tau values,
32 % etc...).
33 % filenames conform to the following 53 character naming convention:
34 % DXX_yyyymmdd_yyyymmddThhmmss_Merged_Data_XXXpXX_FN_XXXpXX_fMLP.mat
35 % An example of this format referencing a real filename in the 03/25/2011
36 % data set is:
37 % D05_20110325_20110624T183042_Merged_Data_050p00_FN_000p00_fMLP.mat
38 %
39 % SUPRA DRIVER MAP:
40 % Level   Name:

```

```

41 % -1    Supra_MSD_Driver_v5.m
42 % 0     MSD_Driver_v15.m
43 %*****
44
45 clc
46 clear all
47 close all
48
49 % *****
50 % BEGIN USER INPUT:
51
52 % Have user select file(s) to be analyzed:
53 choice = menu('Analyze multiple conditions or a single condition?', 'Multiple', 'Single');
54 if choice == 1
55     [file_name_list, file_path] = uigetfile('EnterPathToYourDataHere', 'Select *.mat file(s):', 'MultiSelect', 'on');
56 elseif choice == 2
57     [file_name_string, file_path] = uigetfile('EnterPathToYourDataHere', 'Select *.mat file(s):', 'MultiSelect', 'off');
58     file_name_list = cell(1,1);
59     % In case where single file is selected for plotting the function
60     % 'uigetfile' returns a string not a cell array. For compatibility
61     % we need to log the returned string in cell.
62     file_name_list{1} = file_name_string;
63 end
64
65 % Have user supply a calibration value for conversion of pixels to microns
66 % in units of microns/pixel:
67 pixel_calib = input('\nSet pixel to micron conversion factor in units of microns per single pixel:\n');
68
69 % Determine if user wants to perform MSD analysis using entire empirical
70 % data or only up to a user-specified 'exp_t_max' absolute experimental
71 % time:
72 choiceA = menu('Perform MSD analysis on full empirical data or a portion?', 'Full', 'Portion');
73 if choiceA == 1
74     exp_t_max = [];
75 elseif choiceA == 2
76     % exp_t_max = input('\nEnter maximum experimental imaging time (min) to be used in analysis: ');
77     exp_t_max = 30;
78 end
79
80 % Determine if user wishes to supply a random noise value to subtract from
81 % all MSD data points?
82 choiceB = menu('Do you wish to supply a random noise estimate (epsilon) to be subtracted uniformly from all MSD
values?', 'Yes', 'No');
83 if choiceB == 1
84     epsilon = 0.4604; % units are pix not pix^2
85     % epsilon = input('\nEnter epsilon in units of pix NOT pix^2: ');
86 elseif choiceB == 2
87     epsilon = [];
88 end
89
90 % Determine if user wants to:
91 % (1) fit models to MSD from t = 0 min to t = exp_t_max min
92 % (2) fit models to a portion of MSD data between t = 0 min and t =
93 % exp_t_max min. This means the user will set two tau bounds
94 % ('fit_tau_min' and 'fit_tau_max') that will denote the extent of the
95 % MSD data used when fitting. Logically these bounds can at most be
96 % fit_tau_min = 0 and fit_tau_max = exp_t_max.
97 choiceC = menu('Fit complete (i.e. tau = [0 exp_t_max]) or partial MSD data?', 'Complete', 'Partial');
98
99 if choiceC == 1
100     fit_tau_bounds = [];
101 elseif choiceC == 2
102     % Have user supply minimum and maximum tau intervals (min) that should
103     % be considered when fitting the MSD data with motility models
104
105     % Turn exit flag "off" until entered bounds are acceptable:
106     ok_bounds = 0;
107     while ok_bounds == 0;

```

```

108
109 fit_tau_min = input('\nEnter minimum tau interval (minutes) to fit: ');
110 fit_tau_max = input('\nEnter maximum tau interval (minutes) to fit: ');
111
112 % If any of the following conditions are not met have user enter
113 % new values by keeping exit flag "off".
114 if isempty(exp_t_max) == 0
115
116     if (0>fit_tau_min) || (fit_tau_min >= fit_tau_max) || (fit_tau_max > exp_t_max)
117
118         fprintf(1, '\n\tWARNING: entered values do not obey 0 <= fit_tau_min < fi_tau_max <= exp_t_max\n');
119         ok_bounds = 0;
120
121         % Otherwise the entered bounds are logical so turn exit
122         % flag "on"
123     else
124         fprintf(1, '\n\tEntered boundaries are acceptable.\n');
125
126         fit_tau_bounds(1) = fit_tau_min;
127         fit_tau_bounds(2) = fit_tau_max;
128
129         ok_bounds = 1;
130     end
131
132 elseif isempty(exp_t_max) == 1
133
134     if (0>fit_tau_min) || (fit_tau_min >= fit_tau_max)
135
136         fprintf(1, '\n\tWARNING: entered values do not obey 0 <= fit_tau_min < fi_tau_max\n');
137         ok_bounds = 0;
138
139         % Otherwise the entered bounds are logical so turn exit
140         % flag "on"
141     else
142         fprintf(1, '\n\tEntered boundaries are acceptable.\n');
143
144         fit_tau_bounds(1) = fit_tau_min;
145         fit_tau_bounds(2) = fit_tau_max;
146
147         ok_bounds = 1;
148     end
149
150 end
151
152 end
153
154 end
155
156 % END USER INPUT:
157 % *****
158
159 % Set directory to user-specified directory:
160 cd(file_path);
161
162 % Determine date and time. Create a string in "dateform" "30" (ISO 8601)
163 % which has the format 'yyyymmddTHHMMSS'.
164 dstr = datestr(now, 30);
165
166 % Create a folder to hold results of this plotting run:
167 complete_results_folder_name = [dstr '_MSD_Driver_v15'];
168 mkdir(complete_results_folder_name);
169
170 % Prepare an array for holding all the data that will be in the Excel file
171 % for weighted averaging:
172 xls_mimc = zeros(length(file_name_list),16);
173
174 % Loop over all conditions in folder:
175 for i = 1:length(file_name_list)

```

```

176
177 fprintf(1,'\nProcessing file %s of %s\n',num2str(i),num2str(length(file_name_list)));
178
179 % Extract filename
180 file_name = file_name_list{i};
181
182 % Call MSD_Driver.m to process file
183 [Donor, Donation, FN, fMLP, t_max, mi_counts, avg_all_disp, ...
184 std_all_disp, avg_max_disp, std_max_disp, m_counts, ...
185 Sout, Pout, muout, Aout, alphaout, results_folder_name]...
186 = MSD_Driver_v15(file_name, file_path, pixel_calib, exp_t_max, epsilon, fit_tau_bounds);
187
188 % If this is the first file to be processed log header info in Excel
189 % file:
190 if i == 1
191     header_info = {'Donor','Donation','FN','fMLP','t_max',...
192                 'model indep n','<|All Disp|>','STD |All Disp|',...
193                 '<|Max Disp|>','STD |Max Disp|','model n',...
194                 'Sfit','Pfit','mufit','Afit','alpha fit',...
195                 'ID','yyyymmdd','ug/mL','nM','min',...
196                 'counts','um','um',...
197                 'um','um','counts',...
198                 'um/min','min','um^2/min','um^2/min^alpha','unitless'};
199     xls_name = [file_path complete_results_folder_name '.xlsx'];
200     xlswrite(xls_name,header_info,'MSD_Driver_v15','A1');
201 end
202
203 % Otherwise update current excel file with new condition info:
204 data_to_log = {Donor, Donation, FN, fMLP, t_max,...
205 mi_counts, avg_all_disp, std_all_disp,...
206 avg_max_disp, std_max_disp, m_counts,...
207 Sout, Pout, muout, Aout, alphaout};
208 row = i+2;
209 print_start = ['A' num2str(row)];
210 xlswrite(xls_name,data_to_log,'MSD_Driver_v15',print_start);
211
212 % Move condition-specific results folder into experiment folder
213 movefile([file_path results_folder_name],...
214 [file_path complete_results_folder_name]);
215
216 end
217
218 % When all files are processed move Excel file to experiment folder
219 movefile(xls_name,[file_path complete_results_folder_name]);

```

MSD_Driver_v15.m

```

1 % Steven J. Henry
2 % 03/03/2015
123 %*****
124 % PURPOSE:
125 % This driver performs model independent analysis of individual cell
126 % centroid data. This driver also computes time and ensemble mean square
127 % displacements from the individual cell tracks and applies various models
128 % of motility to this data.
129 %
130 % After "Merge_Mats_m" has been run to generate a '.mat' and '.txt' file
131 % containing all cell tracks for a given experimental condition compiled
132 % from multiple imaging locations within the same experimental condition,
133 % this driver is run.
134 %
135 % ASSUMPTIONS:
136 % (1) Merged data file (.mat) contains an array with the variable name
137 % 'data' and has the following structure:
138 % col 1 = space holder for ImageJ Manual Tracking compatibility
139 % col 2 = unique track number ID assigned to each cell
140 % col 3 = frame number in which cell is found
141 % col 4 = x coordinate (pixels) of centroid

```



```

142 % col 5 = y coordinate (pixels) of centroid
143 % col 6 = absolute time cooresponding to frame in which cell is found
144 % col 7 = binned time corresponding to frame in which cell is found
145 % col 8 = area of cell in pixels
146 % col 9 = track change flag. Entry = 1 if start of new track (yes) or 0
147 %     if no (i.e. continuation of an existing track.
148 %
149 % INPUT:
150 % file_name = name of experimental condition .mat file to be loaded
151 % file_path = path to folder containing 'file_name' above
152 % pixel_calib = user-specified objective calibration (um/pix)
153 % exp_t_max = user-specified maximum experimental time to use in data
154 % analysis (min) or empty
155 % epsilon = user-specified correction for camera noise in pix or empty
156 % fit_tau_bounds = user-specified bounds for fitting motility models to a
157 % portion of MSD data between tau = [tau_bounds(1) tau_bounds(2)] min
158 %
159 % OUTPUT:
160 % Donor = number unique to that donor
161 % Donation = date in 'yyyymmdd' format of 'Donor' blood draw
162 % FN = [FN] as auto-read from .mat file (ug/mL)
163 % fMLP = [fMLP] as auto-read from .mat file (nM)
164 % t_max = maximum experimental imaging time (min) used in analysis. This
165 % value is either exp_t_max as set by the user or the maximum time value
166 % observed in the loaded data set depending on whether or not exp_t_max
167 % is defined
168 % mi_counts = number of tracks contributing data to model independent
169 % analysis
170 % avg_all_disp = mean of all absolute displacements observed (um)
171 % std_all_disp = standard deviation of all absolute displacement observed
172 % (um)
173 % avg_max_disp = mean of max absolute displacements observed (um)
174 % std_max_disp = standard deviation of max absolute displacements observed
175 % (um)
176 % m_counts = number of tracks contributing data to model dependent analysis
177 % Sout = best-fit speed parameter from biased random walk model (um/min)
178 % Pout = best-ft persistence parameter from biased random walk model (min)
179 % muout = random motility coefficient using best-fit biased random walk
180 % parameters = 0.5*Sout^2*Pout (um^2/min)
181 % Aout = best-fit coefficient parameter from power law model
182 % (um^2/min^alpha)
183 % alphaout = best-fit power parameter from power law model (unitless)
184 % results_folder_name = name of folder two which condition-specific
185 % analysis is stored
186 %
187 % DRIVER/FUNCTION MAP:
188 % Level Name:
189 % -1 Supra_MSD_Driver_v5.m
190 % 0 MSD_Driver_v15.m
191 % 1 Parse_Filename_v2.m
192 % 1 Post_IJ_Manual_Track_v3.m
193 % 1 Cell_Track_Plotter_v6.m
194 % 1 Consec_Differentials_v4.m
195 % 1 Step_Size_Stationarity_v2.m
196 % 1 Histograms_v3.m
197 % 1 Path_Length_v6.m
198 % 1 Mean_Path_Length_v5.m
199 % 1 Area_v4.m
200 % 1 Mean_Area_v5.m
201 % 1 Filter_Exp_Data_v3.m
202 % 1 Differentials_v5.m
203 % 1 Neff_v1.m
204 % 1 Mean_Differentials_v6.m
205 % 1 MSD_Epsilon_Subtract_v3.m
206 % 1 Plot_Mean_Differentials_v5.m
207 % 1 Filter_Mean_Differentials_v4.m
208 % 1 SandP_v11.m
209 % 1 Power_Law_v4.m

```

```

210 % 1          Plot_SandP_Fit_v6.m
211 % 1          Plot_Power_Law_Fit_v4.m
212 % 1          Van_Hove_Analysis_v2.m
213 % 1          Tidy_Up_v1.m
214 %*****
215
216 function [Donor, Donation, FN, fMLP, t_max, mi_counts, avg_all_disp, std_all_disp, avg_max_disp, std_max_disp,
m_counts, Sout, Pout, muout, Aout, alphaout, results_folder_name] = MSD_Driver_v15(file_name, file_path, pixel_calib,
exp_t_max, epsilon, fit_tau_bounds)
217
218 % Load the file which must contain the variable 'data':
219 load(file_name,'data');
220
221 % Determine date and time. Create a string in "dateform" "30" (ISO 8601)
222 % which has the format 'yyyymmddTHHMMSS'.
223 dstr = datestr(now, 30);
224
225 % Extract condition information from filename (requires filenames obey
226 % standard naming convention):
227 [Donor, Donation, FN, fMLP, run_title] = Parse_Filename_v2(file_name);
228
229 % Create a folder to hold results of analysis on this condition:
230 results_folder_name = [dstr '_' run_title];
231 mkdir(results_folder_name);
232
233 % Set directory to analysis folder:
234 cd([file_path results_folder_name]);
235
236 % Start a log file. Save in new directory:
237 logfile = [results_folder_name '_Log.txt'];
238 fid = fopen(logfile,'wt');
239
240 % Update log file on progress:
241 fprintf(1,'\nMerged data file imported:\n');
242 fprintf(1,'%s\n\n',[file_path file_name]);
243 fprintf(fid,'%s\n',dstr);
244 fprintf(fid,'\nMerged data file imported:\n');
245 fprintf(fid,'%s\n\n',[file_path file_name]);
246
247 fprintf(1,'Title of run:\n');
248 fprintf(1,'%s\n\n', run_title);
249 fprintf(fid,'Title of run:\n');
250 fprintf(fid,'%s\n\n', run_title);
251
252 fprintf(1,'Results of analysis saved at:\n');
253 fprintf(1,'%s\n\n',[file_path results_folder_name]);
254 fprintf(fid,'Results of analysis saved at:\n');
255 fprintf(fid,'%s\n\n',[file_path results_folder_name]);
256
257 % Record pixel calibration value supplied:
258 fprintf(1,'\n\nUser set pixel_calib = %s um/pixel\n\n',num2str(pixel_calib));
259 fprintf(fid,'\n\nUser set pixel_calib = %s um/pixel\n\n',num2str(pixel_calib));
260
261 % Eliminate unnecessary components of 'data' that are artifacts from ImageJ
262 % Manual Tracking Plugin compatibility requirements previously. Also ensure
263 % (again) that start of each unique track ID is consistent with placement
264 % of track change flags:
265 [data] = Post_IJ_Manual_Track_v3(data, fid); %#ok<NODEF>
266
267 % Plot cell trajectories emanating from single origin and compute
268 % associated model-independent statistics:
269 [mi_counts, avg_all_disp, std_all_disp, avg_max_disp, std_max_disp] = Cell_Track_Plotter_v6(exp_t_max,
pixel_calib, data, run_title, fid);
270
271 % Close figures generated here to prevent Java overload:
272 close all
273
274 % Compute absolute differentials in displacement between two consecutive

```

```

275 % frames of all tracked objects. This data will be used for bias analysis
276 % and determination of population stationarity.
277 [CAD] = Consec_Differentials_v4(data, fid);
278
279 % A plot of the absolute value of consecutive absolute differentials in x
280 % and y are plotted as a function of elapsed experimental to determine
281 % extent of population stationarity. Essentially we are plotting the mean
282 % step size as a function of experimental time. We wish to identify the
283 % time period in which this stepsize is essentially constant.
284 % <|delta_x(tau_min)|> vs. experimental time
285 % <|delta_y(tau_min)|> vs. experimental time
286 [SSD, tau_min, t_max] = Step_Size_Stationarity_v2(CAD, run_title, fid);
287
288 % Generate histograms to check for tracking bias:
289 Histograms_v3(data, CAD, tau_min, run_title, fid);
290
291 % Compute cumulative distance traveled (path length) of individual cell
292 % tracks as a function of elapsed experimental imaging time and plot
293 % results:
294 [Path_Length] = Path_Length_v6(data, pixel_calib, run_title, fid);
295
296 % Compute mean (ensemble averaged) cumulative distance traveled (path
297 % length) of individual cell tracks as a function of elapsed experimental
298 % imaging time and plot results:
299 Mean_Path_Length_v5(Path_Length, pixel_calib, run_title, fid);
300
301 % Determine area of individual cell tracks as a function of elapsed time
302 % and plot results:
303 [Area] = Area_v4(data, pixel_calib, run_title, fid);
304
305 % Compute and plot mean (ensemble averaged) area over all cells as a
306 % function of lag time tau in terms of absolute or binned time
307 % (not time intervals):
308 Mean_Area_v5(Area, pixel_calib, run_title, fid);
309
310 % If 'exp_t_max' is empty utilize complete empirical data available to
311 % compute MSD:
312 if isempty(exp_t_max) == 1
313
314     % Update log:
315     fprintf(1, '\n\nUser opted to perform MSD analysis on complete empirical data.\n');
316     fprintf(fid, '\n\nUser opted to perform MSD analysis on complete empirical data.\n');
317
318     fprintf(1, '\n\tTotal experimental imaging duration was %s min\n\n', num2str(t_max));
319     fprintf(fid, '\n\tTotal experimental imaging duration was %s min\n\n', num2str(t_max));
320
321     % If 'exp_t_max' is not empty, work with only a portion of complete
322     % empirical data available:
323 elseif isempty(exp_t_max) == 0
324
325     % Update log:
326     fprintf(1, '\n\nUser opted to perform MSD analysis on PORTION of empirical data.\n');
327     fprintf(fid, '\n\nUser opted to perform MSD analysis on PORTION of empirical data.\n');
328
329     % Record exp_t_max value sent from 'Supre_MSD_Driver.m'
330     fprintf(1, '\n\nUser set max experimental imaging time to consider in MSD analysis as %s
min\n\n', num2str(exp_t_max));
331     fprintf(fid, '\n\nUser set max experimental imaging time to consider in MSD analysis as %s
min\n\n', num2str(exp_t_max));
332
333     % Filter 'data' array so that it contains only rows corresponding to
334     % absolute image time stamps less than or equal to user-specified
335     % 'exp_t_max':
336     [data] = Filter_Exp_Data_v3(data, exp_t_max, fid);
337
338     % Since user has already selected a maximum imaging time we will set
339     % this as the upper bound on any plots that have an abscissa of
340     % time.

```

```

341     t_max = exp_t_max;
342
343 end
344
345 % Report number of tracks that will contribute intervals to model-dependent
346 % analysis:
347 m_counts = length(unique(data(:,1)));
348
349 % Compute squared displacements of all cells using MOVING origin
350 % strategy in preparation for computation of mean (time and ensemble
351 % average) squared displacement as a function of lag time (tau) in terms of
352 % both absolute and binned time intervals:
353 [SD] = Differentials_v5(data, fid);
354
355 % Compute the number of total independent observations possible ("Neff")
356 % corresponding with tau values used in MSD analysis.
357 [Indep_Obs_tabs, Indep_Obs_tbin] = Neff_v1(data, SD, fid);
358
359 % Compute mean (time and ensemble averaged) squared displacements of all
360 % cells as a function of lag time (tau) in terms of both absolute and
361 % binned time intervals:
362 [MSD_tabs, MSD_tbin] = Mean_Differentials_v6(SD, Indep_Obs_tabs, Indep_Obs_tbin, fid);
363
364 % Yes, supply an estimate of random noise (epsilon value):
365 if isempty(epsilon) == 0
366
367     % Update log:
368     fprintf(1, '\n\nUser supplied estimate of random noise for this experimental condition. ');
369     fprintf(1, '\n4*epsilon^2 will be subtracted uniformly from all MSD values. ');
370     fprintf(fid, '\n\nUser supplied estimate of random noise for this experimental condition. ');
371     fprintf(fid, '\n4*epsilon^2 will be subtracted uniformly from all MSD values.\n');
372
373     % Update log:
374     fprintf(1, '\n\nUser set epsilon = %s pix\n\n', num2str(epsilon));
375     fprintf(fid, '\n\nUser set epsilon = %s pix\n\n', num2str(epsilon));
376
377     % Subtract 2*epsilon^2 from all MSD values:
378     [MSD_tabs MSD_tbin] = MSD_Epsilon_Subtract_v3(MSD_tabs, MSD_tbin, epsilon, fid);
379
380     % Create a flag that lets subsequent plotting routines know whether or
381     % not the data is in terms of epsilon corrected values:
382     epsilon_flag = 1;
383
384 elseif isempty(epsilon) == 1
385
386     % Update log:
387     fprintf(1, '\n\nUser did NOT supply estimate of random noise. ');
388     fprintf(1, '\nMSD values are left uncorrected.\n');
389     fprintf(fid, '\n\nUser did NOT supply estimate of random noise. ');
390     fprintf(fid, '\nMSD values are left uncorrected.\n');
391
392     % Create a flag that lets subsequent plotting routines know whether or
393     % not the data is in terms of epsilon corrected values:
394     epsilon_flag = 0;
395
396 end
397
398 % Plot mean (time and ensemble averaged) squared displacements of all cells
399 % as a function of lag time (tau) in terms of both absolute and binned time
400 % intervals:
401 Plot_Mean_Differentials_v5(SD, MSD_tabs, MSD_tbin, pixel_calib, t_max, run_title, epsilon_flag, fid);
402
403 % Fit full MSD data available between [0 exp_t_max] minutes
404 if isempty(fit_tau_bounds) == 1
405
406     % Update log:
407     fprintf(1, '\n\nUser opted to fit model(s) to MSD data between tau = [0 %s] min\n\n', num2str(exp_t_max));
408     fprintf(fid, '\n\nUser opted to fit model(s) to MSD data between tau = [0 %s] min\n\n', num2str(exp_t_max));

```

```

409
410 % Fit filtered MSD array with biased random walk model:
411 [fit_BRW_tabs, fit_BRW_tbin, Sout, Pout, muout] = SandP_v11(MSD_tabs, MSD_tbin, pixel_calib, fid);
412
413 % Fit filtered MSD array with power law model:
414 [fit_PL_tabs, fit_PL_tbin, Aout, alphaout] = Power_Law_v4(MSD_tabs, MSD_tbin, pixel_calib, fid);
415
416 % Fit portion of MSD data between user-defined [fit_tau_bounds(1)
417 % fit_tau_bounds(2)] minutes:
418 elseif isempty(fit_tau_bounds) == 0
419
420 % Update log:
421 fprintf(1, '\n\nUser opted to fit model(s) to MSD data between tau = [%s %s]
min\n\n', num2str(fit_tau_bounds(1)), num2str(fit_tau_bounds(2)));
422 fprintf(fid, '\n\nUser opted to fit model(s) to MSD data between tau = [%s %s]
min\n\n', num2str(fit_tau_bounds(1)), num2str(fit_tau_bounds(2)));
423
424 % Filter MSD arrays so that they contain only those values
425 % corresponding to taus in the range [fit_tau_bounds(1),
426 % fit_tau_bounds(2)]:
427 [MSD_tabs_part, MSD_tbin_part] = Filter_Mean_Differentials_v4(MSD_tabs, MSD_tbin, fit_tau_bounds(1),
fit_tau_bounds(2), epsilon_flag, fid);
428
429 % % Lin spaced data on log axes:
430 % keep_dt = 1.15.^{(0:99)};
431 % [dummy, ind] = unique(keep_dt, 'first');
432 % keep_dt = keep_dt(ind);
433 % ind = find(keep_dt <= round((max(MSD_tbin_part(:,5))/min(MSD_tbin_part(:,5))));
434
435 % Fit filtered MSD array with biased random walk model:
436 [fit_BRW_tabs, fit_BRW_tbin, Sout, Pout, muout] = SandP_v10(MSD_tabs_part, MSD_tbin_part, pixel_calib, fid);
437
438 % Fit filtered MSD array with power law model:
439 [fit_PL_tabs, fit_PL_tbin, Aout, alphaout] = Power_Law_v4(MSD_tabs_part, MSD_tbin_part, pixel_calib, fid);
440
441 end
442
443 % Overlay full empirical MSD data with fit data and plot results:
444 Plot_SandP_Fit_v6(MSD_tabs, MSD_tbin, fit_BRW_tabs, fit_BRW_tbin, pixel_calib, t_max, run_title, epsilon_flag,
fid);
445 Plot_Power_Law_Fit_v4(MSD_tabs, MSD_tbin, fit_PL_tabs, fit_PL_tbin, pixel_calib, t_max, run_title, epsilon_flag,
fid);
446
447 % Close figures so Java doesn't overload
448 close all
449
450 % Perform Van Hove analysis as a check on the MSD analysis completed
451 % previously:
452 Van_Hove_Analysis_v3(pixel_calib, data, MSD_tbin, epsilon_flag, run_title, fid)
453
454 % Update log file that program is terminated:
455 fprintf(1, '\nProgram Terminated\n');
456 fprintf(fid, '\nProgram Terminated\n');
457
458 % Close log file for this condition
459 fclose(fid);
460
461 % Sort all files generated into folders of .fig, .mats, and .txt files
462 % leaving the master log file residing outside the three folders:
463 Tidy_Up_v1(logfile);
464
465 % Close any figures remaining open
466 close all;
467
468 end

```

Parse_Filename_v2.m

```
1 % Steven J. Henry
2 % 11/03/2011
8 %*****
9 % PURPOSE:
10 % This function extracts the concentrations of FN and fMLP from the
11 % filename.
12
13 % REMARKS:
14 % This function is meant for ease of analysis of the hNeutrophils on uCP
15 % hFN in aqueous fMLP study and can be eliminated from 'MSD_Driver.m' to
16 % make that driver and its functionality more general in the future.
17 %
18 %
19 % ASSUMPTIONS:
20 % filenames conform to the following 53 character naming convention:
21 % DXX_yyyymmdd_yyyymmddThmmss_Merged_Data_XXXpXX_FN_XXXpXX_fMLP.mat
22 % An example of this format referencing a real filename in the 03/25/2011
23 % data set is:
24 % D05_20110325_20110624T183042_Merged_Data_050p00_FN_000p00_fMLP.mat
25 %
26 % INPUT:
27 % filename = 566 character string with structure as stated previously in
28 % ASSUMPTIONS
29 %
30 % OUTPUT:
31 % run_title = 'XXXpXX_FN_XXXpXX_fMLP' portion of filename
32 % FN = numeric value of FN concentration in ug/mL
33 % fMLP = numeric value of fMLP concentration in nM
34 %*****
35
36 function [Donor, Donation, FN, fMLP, run_title] = Parse_Filename_v2(file_name)
37
38 % Make sure loaded filename has 66 characters
39 if length(file_name)~= 66
40     fprintf(1,'WARNING: file name loaded does not adhere to naming convention required to auto-read
conditions\n\n');
41     cancel
42 else
43
44     % Retain FN and fMLP portion of current filename for future reference
45     run_title = file_name(1:end-4);
46
47     % Extract numeric FN value:
48     FN_str = file_name(end-24:end-19);
49     FN_str(4)='.';
50     FN = str2num(FN_str);
51
52     % Extract numeric fMLP value:
53     fMLP_str = file_name(end-14:end-9);
54     fMLP_str(4)='.';
55     fMLP = str2num(fMLP_str); %#ok<*ST2NM>
56
57     % Extract Donation Date:
58     Donation_str = file_name(end-61:end-54);
59     Donation = str2num(Donation_str);
60
61     % Extract Donor ID:
62     Donor_str = file_name(2:3);
63     Donor = str2num(Donor_str);
64
65 end
66
67 end
```

Post_IJ_Manual_Track_v3.m

```
1 % Steven J. Henry
2 % 04/30/2011
21 %*****
22 % PURPOSE:
23 % This is a very simple funtion called on by "MSD_Driver_v1.m". It has two
24 % purposes:
25 % (1) Eliminates unnecessary components of supplied data array that are
26 % artifacts as a result of compatibility requirements for interfacing with
27 % ImageJ's Manual Tracking Plugin.
28 % (2) Confirm that start of each unique track ID is consistent with
29 % placement/location of track change flags (values of 1). In theory this is
30 % redundant because "Merge_Mats_v2.m" does this same task prior to saving
31 % the 'data' file imported into this program. However, my concern is that
32 % if the user manually adjusted the 'data' array after "Merge_Mats_v2.m"
33 % but prior to import here the result could be internal inconsistency
34 % especially if the user were to delete a track initiation row (a row in
35 % which the track change flag is set to "on" or value of 1).
36 %
37 % ASSUMPTIONS:
38 % n/a
39 %
40 % INPUT:
41 % data = an array containing all pertinent tracking information for each
42 % cell in the given experimental condition having the following structure:
43 % col 1 = space holder for ImageJ Manual Tracking compatibility
44 % col 2 = unique track number ID assigned to each cell
45 % col 3 = frame number in which cell is found
46 % col 4 = x coordinate (pixels) of centroid
47 % col 5 = y coordinate (pixels) of centroid
48 % col 6 = absolute time cooresponding to frame in which cell is found
49 % (sec)
50 % col 7 = binned time corresponding to frame in which cell is found (sec)
51 % col 8 = area of cell in pixels
52 % col 9 = track change flag. Entry = 1 if start of new track (yes) or 0
53 % if no (i.e. continuation of an existing track.
54 % fid = file ID of log file to which progress is recorded
55 %
56 % OUTPUT:
57 % data = same array but now having the following structure:
58 % col 1 (prev. col 2) = unique track number ID assigned to each cell
59 % col 2 (prev. col 4) = x coordinate (pixels) of centroid
60 % col 3 (prev. col 5) = y coordinate (pixels) of centroid
61 % col 4 (prev. col 6) = absolute time cooresponding to frame in which
62 % cell is found (sec)
63 % col 5 (prev. col 7) = binned time corresponding to frame in which cell
64 % is found (sec)
65 % col 6 (prev. col 8) = area of cell in pixels
66 % col 7 (prev. col 9) = track change flag. Entry = 1 if start of new
67 % track (yes) or 0 if no (i.e. continuation of an
68 % existing track.
69 %*****
70
71 function [data_edited] = Post_IJ_Manual_Track_v3(data,fid)
72
73 % Get function name:
74 func_name = mfilename;
75
76 % Update log file that function is running:
77 fprintf(1,'\n%s running ...\n',func_name);
78 fprintf(fid,'\n%s running ...\n',func_name);
79
80 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
81 % warning dialog the log file records no errors/warnings generated:
82 warn = 0;
83
84 % Determine dimensions of array:
```

```

85 [rows cols] = size(data);
86
87 % Reserve appropriate memory block:
88 data_edited = zeros(rows, 7);
89
90 % Write data to be saved:
91 data_edited(:,1) = data(:,2);
92 data_edited(:,2:7) = data(:,4:9);
93
94 % Check dimensionality:
95 [rows2 cols2] = size(data_edited);
96 if rows2 ~= rows
97     fprintf(1,'\n\tWARNING: # of rows not preserved during editing.\n')
98     fprintf(1,'\tOriginal # rows = %.0f, post editing # rows = %.0f\n',rows,rows2);
99     fprintf(fid,'\n\tWARNING: # of rows not preserved during editing.\n')
100    fprintf(fid,'\tOriginal # rows = %.0f, post editing # rows = %.0f\n',rows,rows2);
101    warn = 1;
102 end
103 if cols2 ~= 7
104     fprintf(1,'\n\tWARNING: # of cols not = 7 after editing.\n');
105     fprintf(1,'\tInstead # cols = %.0f\n',cols2);
106     fprintf(fid,'\n\tWARNING: # of cols not = 7 after editing.\n');
107     fprintf(fid,'\tInstead # cols = %.0f\n',cols2);
108     warn = 1;
109 end
110
111 % Check that track change flags are internally consistent with track IDs:
112 ID_vector = data_edited(:,1);
113 TCF_vector = data_edited(:,7);
114 [junk, ID_ind] = unique(ID_vector, 'first');
115 clear junk
116 [TCF_ind] = find(TCF_vector == 1);
117 check = (ID_ind == TCF_ind);
118
119 if length(ID_ind) ~= length(TCF_ind)
120     fprintf(1,'\n\tWARNING: # Unique Track IDs ~= # Track Change Flags\n');
121     fprintf(fid,'\n\tWARNING: # Unique Track IDs ~= # Track Change Flags\n');
122     warn = 1;
123 else
124     if nnz(check) ~= length(ID_ind)
125         fprintf(1,'\n\tWARNING: Start of each unique track ID not consistent with placement of track change flags.\n');
126         fprintf(fid,'\n\tWARNING: Start of each unique track ID not consistent with placement of track change flags.\n');
127         warn = 1;
128     else
129         fprintf(1,'\n\tStart of each unique track ID confirmed consistent w/placement of track change flags.\n');
130         fprintf(fid,'\n\tStart of each unique track ID confirmed consistent w/placement of track change flags.\n');
131     end
132 end
133
134 % Clear original data array to save memory:
135 clear data
136
137 % If no warnings generated report so in log file:
138 if warn == 0
139     fprintf(1,'\n\tFunction completed without errors/warnings\n');
140     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
141 end
142
143 % Update log file that function is completed:
144 fprintf(1,'\n%s completed\n',func_name);
145 fprintf(fid,'\n%s completed\n',func_name);
146
147 end

```

Cell_Track_Plotter_v6.m

```

1 % Steven J. Henry
2 % 08/11/2012

```



```

73 %*****
74 % PURPOSE:
75 % This program plots all cell trajectories emanating from a single
76 % origin.
77 %
78 % ASSUMPTIONS:
79 % User loads proper file.
80 %
81 % INPUT:
82 % exp_t_max = maximum experimental time to include in analysis (min)
83 % pixel_calib = user specified objective calibration (um/pixel)
84 % data = an array containing all pertinent tracking information for each
85 % cell in the given experimental condition having the following structure
86 % (as a result of 'Post_IJ_Manual_Track.m'):
87 % col 1 = unique track number ID assigned to each cell
88 % col 2 = x coordinate (pixels) of centroid
89 % col 3 = y coordinate (pixels) of centroid
90 % col 4 = absolute time cooresponding to frame in which cell is found
91 % (sec)
92 % col 5 = binned time corresponding to frame in which cell is found (sec)
93 % col 6 = area of cell in pixels
94 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
95 % if no (i.e. continuation of anexisting track.
96 % run_title = user specified string description of experimental condition
97 % fid = file ID of log file to which progress is recorded
98 %
99 % OUTPUT:
100 % mi_counts = 'model independent counts' = number of tracks contributing to
101 % model independent analysis
102 % avg_all_disp = mean of the set of all absolute displacements of each
103 % cell's centroid from its origin.
104 % std_all_disp = standard deviation of the set of all absolute
105 % displacements of each cell's centroid from its origin.
106 % avg_max_disp = mean of the set of all maximum absolute displacements of
107 % each cell's centroid from its origin.
108 % std_max_disp = standard deviation of the set of all maximum absolute
109 % displacements of each cell's centroid from its origin.
110 % Three matlab figures:
111 % Trajectories.fig is a plot of all cell trajectories emanating from a
112 % common origin. Only plotted are those cells that were tracked minimally
113 % 'exp_t_max' in duration. A MATLAB figure (.fig) containing a plot of cell
114 % centroid positions
115 % Trajectories_means.fig is the previous plot but containing two circles.
116 % One of radius 'avg_all_disp' and the other of radius 'avg_max_disp'
117 % Trajectories_hist.fig is a figure containing the histograms of the
118 % absolute displacements computed and the maximum absolute displacements
119 % computed.
120 %*****
121
122 function [mi_counts, avg_all_disp, std_all_disp, avg_max_disp, std_max_disp] = Cell_Track_Plotter_v6(exp_t_max,
pixel_calib, data, run_title, fid)
123
124 % Get function name:
125 func_name = mfilename;
126
127 % Update log file that function is running:
128 fprintf(1, '\n%s running ... \n', func_name);
129 fprintf(fid, '\n%s running ... \n', func_name);
130
131 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
132 % warning dialog the log file records no errors/warnings generated:
133 warn = 0;
134
135 % Retrieve number of tracks in filtered 'data' array. Recall at this point
136 % 'data' array has been processed via 'Post_IJ_Manual_Track_v2.m' and
137 % possibly 'Filter_Exp_Data.m'.
138 uniq_IDs = unique(data(:,1));
139 num_tracks = length(uniq_IDs);

```

```

140
141 % Create a vector that will hold the maximum displacement of each track:
142 max_disp = zeros(num_tracks,1);
143
144 % Reserve variable name 'all_disp' that will hold all displacements of each
145 % track's centroid from its origin position (not just the maximum
146 % displacement as stored in 'max_disp' above):
147 all_disp = [];
148
149 % Create a figure to which we will iteratively plot track trajectories:
150 fig_handle = figure;
151 axes_handle = axes;
152
153 % Create a counter that will keep track of the total number of cells
154 % plotted on the final graph (note it is not necessarily true that the
155 % number of cells plotted will be equal to the number of total cells in
156 % 'data'. To be plotted a track must exist through exp_t_max. In this way we do
157 % not bias the average by incorporating truncated tracks.
158 counts = 0;
159
160 for i = 1:num_tracks
161
162     track = uniq_IDs(i);
163
164     % Does this track 'i' have centroid observations at time 'exp_t_max' or
165     % greater?
166     % The purpose of this check is to ensure we only plot those
167     % tracks that were followed for at least 'exp_t_max' total time.
168     % Otherwise we are biasing the average maximum displacement value
169     % downwards.
170     obs_check = data(:,1)==track & data(:,5)>=exp_t_max*60;
171
172     % If 'obs_check' contains ones then there exist observations of track
173     % 'i' at times equal to or greater than 'exp_t_max' so include this
174     % track in the total plot:
175     if sum(obs_check)>0
176
177         % Determine number of observations of track 'i' which occurred at or
178         % before time 'exp_t_max'. Note: this test is applied to the binned
179         % time interval column of 'data' (col 5).
180         track_ind = data(:,1)==track & data(:,5)<=exp_t_max*60;
181
182         % Provided there exist points to plot, enter the plotting calls:
183         if sum(track_ind)>0
184
185             % Reserve a temporary memory block to hold x,y coordinates
186             % of this track's trajectory to be plotted:
187             coords = data(track_ind,2:3);
188             x_orig = coords(1,1);
189             y_orig = coords(1,2);
190             coords(:,1) = coords(:,1)-x_orig;
191             coords(:,2) = coords(:,2)-y_orig;
192             coords = coords*pixel_calib;
193
194             % Compute displacement of all centroid positions from
195             % initial centroid position (origin):
196             track_disp = sqrt(coords(:,1).^2+coords(:,2).^2);
197
198             % Record all displacements just computed with exception of
199             % initial origin position:
200             if isempty(all_disp)==1
201                 all_disp = track_disp(2:end);
202             else
203                 all_disp(end+1:end+length(track_disp)-1) = track_disp(2:end);
204             end
205
206             % Record only max displacement observed from list of all
207             % displacement for this track:

```

```

208     max_disp(i) = max(track_disp);
209
210     % Plot track trajectory:
211     hold all
212     plot(coords(:,1),coords(:,2),'LineStyle','-','Color','k','LineWidth',1.5,'Marker','none','HandleVisibility','off');
213
214     % Update counter:
215     counts = counts + 1;
216
217     end
218
219     end
220
221 end
222
223 % Send counts out of function:
224 mi_counts = counts;
225
226 % Eliminate zero rows
227 num_del_theory = length(max_disp)-nnz(max_disp);
228 del_ind = max_disp(:)==0;
229 max_disp(del_ind)=[];
230 num_del_practice = sum(del_ind);
231 if num_del_practice ~= num_del_theory
232     fprintf(1,'\n\tWARNING: # deletions from "max_disp" ~= # deletions predicted\n');
233     fprintf(1,'\n\tWARNING: # deletions from "max_disp" ~= # deletions predicted\n');
234     warn = 1;
235     keyboard
236 end
237
238 % Compute mean and std of 'all_disp' and 'max_disp':
239 avg_all_disp = mean(all_disp);
240 std_all_disp = std(all_disp);
241
242 avg_max_disp = mean(max_disp);
243 std_max_disp = std(max_disp);
244
245 % Record number cells plotted on trajectory graph:
246 fprintf(1,'\n\tNumber of trajectories plotted = %s\n',num2str(counts));
247 fprintf(fid,'\n\tNumber of trajectories plotted = %s\n',num2str(counts));
248
249 % Record 'all_disp' mean and std:
250 fprintf(1,'\n\tALL absolute displacement statistics:');
251 fprintf(1,'\n\t\tMean = %0.5f um',avg_all_disp);
252 fprintf(1,'\n\t\tStandard Deviation = %0.5f um',std_all_disp);
253 fprintf(fid,'\n\tALL absolute displacement statistics:');
254 fprintf(fid,'\n\t\tMean = %0.5f um',avg_all_disp);
255 fprintf(fid,'\n\t\tStandard Deviation = %0.5f um',std_all_disp);
256
257 % Record 'max_disp' mean and std:
258 fprintf(1,'\n\tMAX absolute displacement statistics:');
259 fprintf(1,'\n\t\tMean = %0.5f um',avg_max_disp);
260 fprintf(1,'\n\t\tStandard Deviation = %0.5f um',std_max_disp);
261 fprintf(fid,'\n\tMAX absolute displacement statistics:');
262 fprintf(fid,'\n\t\tMean = %0.5f um',avg_max_disp);
263 fprintf(fid,'\n\t\tStandard Deviation = %0.5f um',std_max_disp);
264
265 % Plot a figure that has both a circle of radius 'avg_all_disp' as well
266 % as a circle of radius 'avg_max_disp' on the same axes. This is
267 % the master plot from which we will a second without these circles.
268
269 % Round values for plot cleanliness
270 plot_avg_all_disp = round(avg_all_disp);
271 plot_avg_max_disp = round(avg_max_disp);
272
273 all_circle_x = plot_avg_all_disp*-1.:plot_avg_all_disp;
274 all_circle_y_upper = sqrt(plot_avg_all_disp^2-all_circle_x.^2);
275 all_circle_y_lower = all_circle_y_upper*-1;

```

```

276
277 max_circle_x = plot_avg_max_disp*-1:-1:plot_avg_max_disp;
278 max_circle_y_upper = sqrt(plot_avg_max_disp^2-max_circle_x.^2);
279 max_circle_y_lower = max_circle_y_upper*-1;
280
281 hold all
282 AU = plot(all_circle_x,all_circle_y_upper,'LineStyle','--','Color','b','LineWidth',2,'Marker','none','HandleVisibility','on');
283 hold all
284 AL = plot(all_circle_x,all_circle_y_lower,'LineStyle','--','Color','b','LineWidth',2,'Marker','none','HandleVisibility','off');
285 hold all
286 MU = plot(max_circle_x,max_circle_y_upper,'LineStyle','--','Color','r','LineWidth',2,'Marker','none','HandleVisibility','on');
287 hold all
288 ML = plot(max_circle_x,max_circle_y_lower,'LineStyle','--','Color','r','LineWidth',2,'Marker','none','HandleVisibility','off');
289
290 % Annotation
291 title(run_title,'FontName','Arial','FontSize',18);
292 xlabel('x position (\mum)','FontName','Arial','FontSize',18);
293 ylabel('y position (\mum)','FontName','Arial','FontSize',18);
294 set(axes_handle,'FontName','Arial');
295 set(axes_handle,'FontSize',16);
296 set(axes_handle,'DataAspectRatio',[1 1 1]);
297 h_legend = legend('<\Deltar>','<Max(\Deltar)>','Location','NorthEast');
298 set(h_legend,'FontName','Arial');
299 set(h_legend,'FontSize',14);
300
301 % Get automatically generated axes limits and adjust so square:
302 v = axis;
303 lim = max(abs(v));
304 axis([lim*-1 lim lim*-1 lim]);
305
306 % Generate base filename:
307 fig_title = 'Trajectories';
308
309 % Save figure with mean circles:
310 saveas(fig_handle, [fig_title '_means.fig'], 'fig');
311
312 % Eliminate means and save:
313 delete(AU);
314 delete(AL);
315 delete(MU);
316 delete(ML);
317 delete(h_legend);
318 saveas(fig_handle, [fig_title '.fig'], 'fig');
319
320 % Plot histogram figure
321 num_all_bins = round(sqrt(length(all_disp)));
322 num_max_bins = round(sqrt(length(max_disp)));
323
324 hist_fig = figure;
325
326 h1 = subplot(1,2,1);
327 [freq bin_loc] = hist(all_disp,num_all_bins);
328 bar(bin_loc,freq,1);
329 axis([min(all_disp) max(all_disp) 0 max(freq)]);
330 title('All \Deltar','FontName','Arial','FontSize',18);
331 xlabel('\Deltar (\mum)','FontName','Arial','FontSize',18);
332 ylabel('Count','FontName','Arial','FontSize',18);
333 set(h1,'FontName','Arial','FontSize',16);
334
335 h2 = subplot(1,2,2);
336 [freq bin_loc] = hist(max_disp,num_max_bins);
337 bar(bin_loc,freq,1);
338 axis([min(max_disp) max(max_disp) 0 max(freq)]);
339 title('Max \Deltar','FontName','Arial','FontSize',18);
340 xlabel('\Deltar (\mum)','FontName','Arial','FontSize',18);
341 ylabel('Count','FontName','Arial','FontSize',18);

```

```

342 set(h2,'FontName','Arial','FontSize',16);
343
344 saveas(hist_fig, [fig_title '_hist.fig'], 'fig');
345
346 % If no warnings generated report so in log file:
347 if warn == 0
348     fprintf(1, '\n\tFunction completed without errors/warnings\n');
349     fprintf(fid, '\n\tFunction completed without errors/warnings\n');
350 end
351
352 % Update log file that function is completed:
353 fprintf(1, '\n%s completed\n', func_name);
354 fprintf(fid, '\n%s completed\n', func_name);
355
356 end

```

Consec_Differentials_v4.m

```

1 % Steven J. Henry
2 % 08/11/2012
44 %*****
45 % PURPOSE:
46 % This function computes the absolute differentials in displacement between
47 % two consecutive frames of all tracked objects.
48 %
49 % ASSUMPTIONS:
50 % n/a
51 %
52 % INPUT:
53 % data = array having following structure:
54 % col 1 = unique track number ID assigned to each cell
55 % col 2 = x coordinate (pixels) of centroid
56 % col 3 = y coordinate (pixels) of centroid
57 % col 4 = absolute time cooresponding to frame in which cell is found
58 % (sec)
59 % col 5 = binned time corresponding to frame in which cell is found (sec)
60 % col 6 = area of cell in pixels
61 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
62 % if no (i.e. continuation of an existing track.
63 % fid = file ID of log file to which progress is recorded
64 %
65 % OUTPUT:
66 % CAD = ("consecutive absolute differentials") array containing absolute
67 % displacement differentials between consecutive frames across all tracks
68 % in a given 'data' array. Data from frames separated by an interval of
69 % time greater than the attempted constant imaging rate are excluded.
70 % col 1 = track ID to which this interval belongs
71 % col 2 = dx (absolute displacement along x coordinate axis in pixels)
72 % col 3 = dy (absolute displacement along y coordinate axis in pixels)
73 % col 4 = elapsed experimental imaging time at end of corresponding
74 % displacement (sec)
75 % col 5 = elapsed binned time (sec) between consecutive frames (i.e.
76 % tau_min)
77 %*****
78
79 function [CAD] = Consec_Differentials_v4(data, fid)
80
81 % Get function name:
82 func_name = mfilename;
83
84 % Update log file that function is running:
85 fprintf(1, '\n%s running ... \n', func_name);
86 fprintf(fid, '\n%s running ... \n', func_name);
87
88 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
89 % warning dialog the log file records no errors/warnings generated:
90 warn = 0;
91

```

```

92 % Determine indices of start and stop positions of each unique track in the
93 % overall 'data' array:
94 [junk ind_start] = unique(data(:,1),'first');
95 clear junk
96 [junk ind_stop] = unique(data(:,1),'last');
97 clear junk
98
99 % Make sure lengths of start and stop vectors are same:
100 if length(ind_start) ~= length(ind_stop)
101     fprintf(1,'\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
102     fprintf(fid,'\n\tWARNING: # Unique Track IDs ~= # Track Change Flags\n');
103     warn = 1;
104 end
105
106 % Total number of tracks to be analyzed:
107 num_tracks = length(ind_start);
108
109 % Reserve 'CAD' variable name:
110 CAD = [];
111
112 % Reserve 'num_consec_int_tot' and 'num_dels_tot' for computation of total
113 % number of consecutive intervals computed:
114 num_consec_int_tot = 0;
115 num_del_tot = 0;
116
117 % Begin 08/28/2011 Version 3 edit in determination of constant imaging rate
118 % 'im_rate':
119 % Generate a vector of size length(data) - num_tracks to hold total number
120 % of adjacent frame time differences:
121 num_adj_ints = size(data,1)-num_tracks;
122 bin_time_diff_list = zeros(num_adj_ints,1);
123 print_row = 1;
124
125 % Loop over tracks:
126 for ii = 1:num_tracks
127
128     % Define start and stop position (row #s):
129     r_start = ind_start(ii);
130     r_stop = ind_stop(ii);
131
132     % Loop over all rows between start and second-to-last stop rows and
133     % compute absolute displacements for track
134     for kk = r_start:r_stop-1
135
136         bin_time_now = data(kk,5);
137         bin_time_next = data(kk+1,5);
138         bin_time_diff = bin_time_next-bin_time_now;
139
140         bin_time_diff_list(print_row) = bin_time_diff;
141
142         % Advance print_row:
143         print_row = print_row+1;
144     end
145 end
146
147 end
148
149 im_rate = mode(bin_time_diff_list);
150 % End version 3 edits
151
152 % Loop over tracks:
153 for i = 1:num_tracks
154
155     % Define start and stop position (row #s):
156     r_start = ind_start(i);
157     r_stop = ind_stop(i);
158
159     % Determine the maximum number of consecutive intervals that could

```

```

160 % possibly be observed for this track:
161 num_consec_int_track = r_stop - r_start;
162 track_CAD = zeros(num_consec_int_track,5);
163 num_consec_int_tot = num_consec_int_tot + num_consec_int_track;
164 print_row = 1;
165
166 % Loop over all rows between start and second-to-last stop rows and
167 % compute absolute displacements for track
168 for k = r_start:r_stop-1
169
170     bin_time_now = data(k,5);
171     bin_time_next = data(k+1,5);
172     bin_time_diff = bin_time_next-bin_time_now;
173
174     if bin_time_diff == im_rate
175
176         orig_row = k;
177         adv_row = k+1;
178
179         % Load origin info:
180         track_ID = data(orig_row,1);
181         x_orig = data(orig_row,2);
182         y_orig = data(orig_row,3);
183         t_bin_orig = data(orig_row,5);
184
185         % Load advance row info:
186         x_adv = data(adv_row,2);
187         y_adv = data(adv_row,3);
188         t_bin_adv = data(adv_row,5);
189
190         % Compute differentials:
191         dx = x_adv - x_orig;
192         dy = y_adv - y_orig;
193         dt_bin = t_bin_adv - t_bin_orig;
194
195         % Determine absolute experimental time at end of this
196         % displacement:
197         abs_exp_t = data(adv_row,4);
198
199         % Log values:
200         track_CAD(print_row,1) = track_ID;
201         track_CAD(print_row,2) = dx;
202         track_CAD(print_row,3) = dy;
203         track_CAD(print_row,4) = abs_exp_t;
204         track_CAD(print_row,5) = dt_bin;
205
206         % Advance print_row:
207         print_row = print_row+1;
208     end
209 end
210
211 end
212
213 % Remove rows not utilized in 'track_CAD' because time elapsing between
214 % frames corresponding to that row entry was greater than 'im_rate':
215 del_ind = track_CAD(:,5)==0;
216 num_del_track = sum(del_ind);
217 % Note when 'del_ind' has a zero sum the
218 % following loop is NOT entered (this has been verified).
219 if num_del_track>0
220     track_CAD(del_ind,:) = [];
221     num_del_tot = num_del_tot+num_del_track;
222 end
223
224 % After all consecutive intervals for 'track_ID' are computed make sure
225 % the length of 'track_CAD' is = 'num_consec_int_track - num_del_track':
226 [track_CAD_r track_CAD_c] = size(track_CAD);
227 if track_CAD_r~=num_consec_int_track-num_del_track

```

```

228     fprintf(1, '\n\tWARNING: # consec intervals computed ~= expected #\n');
229     fprintf(1, '\tWarning generated for track = %.0f\n', track_ID);
230     fprintf(fid, '\n\tWARNING: # consec intervals computed ~= expected #\n');
231     fprintf(fid, '\tWarning generated for track = %.0f\n', track_ID);
232     warn = 1;
233 end
234
235 % Concatenate 'track_CAD' with existing 'CAD'
236 if isempty(CAD) == 1
237     CAD = track_CAD;
238     clear track_CAD
239     % In the event only 1 track exists in this 'data' array we want to
240     % have the dimensions for the post-processing dimensionality check.
241     [CAD_new_r CAD_new_c] = size(CAD);
242
243 else
244     CAD_old = CAD;
245     [CAD_old_r CAD_old_c] = size(CAD_old);
246     clear CAD
247     CAD = zeros(CAD_old_r+track_CAD_r, CAD_old_c);
248     CAD(1:CAD_old_r, 1:track_CAD_c) = CAD_old;
249     CAD(CAD_old_r+1:CAD_old_r+track_CAD_r, 1:track_CAD_c) = track_CAD;
250     [CAD_new_r CAD_new_c] = size(CAD);
251
252 end
253
254 end
255
256 % Dimensionality check. There should be five columns after the
257 % concatenations are finished:
258 if CAD_new_c ~= 5
259     fprintf(1, '\n\tWARNING: # cols = %.0f not 5 as expected\n', CAD_new_c);
260     fprintf(1, '\tWarning generated for track = %.0f\n', track_ID);
261     fprintf(fid, '\n\tWARNING: # cols = %.0f not 5 as expected\n', CAD_new_c);
262     fprintf(fid, '\tWarning generated for track = %.0f\n', track_ID);
263     warn = 1;
264 end
265
266 % Dimensionality check. The total number of rows in 'CAD' should be equal
267 % to the sum of each track's previously identified consecutive intervals in
268 % theory minus the number of deletions made in practice:
269 num_consec_int_tot_actual = num_consec_int_tot - num_del_tot;
270 if CAD_new_r ~= num_consec_int_tot_actual
271     fprintf(1, '\n\tWARNING: # rows (consec_intervals) = %.0f not %.0f as
expected\n', CAD_new_r, num_consec_int_tot_actual);
272     fprintf(fid, '\n\tWARNING: # rows (consec_intervals) = %.0f not %.0f as
expected\n', CAD_new_r, num_consec_int_tot_actual);
273     warn = 1;
274 end
275
276 % Save 'CAD' array in .mat and .txt format:
277 save 'CAD.mat' CAD;
278 fid2 = fopen('CAD.txt', 'wt');
279 fprintf(fid2, 'Track\t dx (pix)\t dy (pix)\t Abs. Exp. t (sec)\t tau (sec)\n');
280 for k = 1:CAD_new_r
281     fprintf(fid2, '%.0f\t%.0f\t%.0f\t%.0f\n', CAD(k,:));
282 end
283 fclose(fid2);
284
285 % If no warnings generated report so in log file:
286 if warn == 0
287     fprintf(1, '\n\tFunction completed without errors/warnings\n');
288     fprintf(fid, '\n\tFunction completed without errors/warnings\n');
289 end
290
291 % Update log file that function is completed:
292 fprintf(1, '\n%s completed\n', func_name);
293 fprintf(fid, '\n%s completed\n', func_name);

```


294
295 end

Step_Size_Stationarity_v2.m

```
1 % Steven J. Henry
2 % 08/11/2012
12 %*****
13 % PURPOSE:
14 % This function plots the mean step size (in x and y) of a population of
15 % cells as a function of experimental time:
16 % <delta_x(tau_min)> vs. experimental time
17 % <delta_y(tau_min)> vs. experimental time
18 %
19 % ASSUMPTIONS:
20 % n/a
21 %
22 % INPUT:
23 % % CAD = ("consecutive absolute differentials") array containing absolute
24 % displacement differentials between consecutive frames across all tracks
25 % in a given 'data' array. Data from frames separated by an interval of
26 % time greater than the attempted constant imaging rate are excluded.
27 % col 1 = track ID to which this interval belongs
28 % col 2 = dx (absolute displacement along x coordinate axis in pixels)
29 % col 3 = dy (absolute displacement along y coordinate axis in pixels)
30 % col 4 = elapsed experimental imaging time at end of corresponding
31 % displacement (sec)
32 % col 5 = elapsed binned time (sec) between consecutive frames (i.e.
33 % tau_min)
34 % run_title = user specified string description of experimental condition
35 % fid = file ID of log file to which progress is recorded
36 %
37 % OUTPUT:
38 % SSD = ("stepsize stationarity data") array containing mean stepsize and
39 % standard deviation in x and y as a function of elapsed experimental
40 % imaging time:
41 % col 1 = rounded elapsed experimental imaging time (sec)
42 % Note: These were binned according to multiples of tau
43 % col 2 = <|delta_x(tau)|> (pix)
44 % col 3 = standard deviation of <|delta_x(tau)|> (pix)
45 % col 4 = <|delta_y(tau)|> (pix)
46 % col 5 = standard deviation of <|delta_y(tau)|> (pix)
47 % col 6 = number of observations (replicates) contributing to mean and
48 % s.d.
49 % tau_min = elapsed time interval between most consecutive frames (i.e.
50 % attempted constant imaging rate) (sec)
51 % t_max = maximum elapsed experiemntal imaging time (min)
52 %*****
53
54 function [SSD, tau_min, t_max] = Step_Size_Stationarity_v2(CAD, run_title, fid)
55
56 % Get function name:
57 func_name = mfilename;
58
59 % Update log file that function is running:
60 fprintf(1, '\n%s running ...\n', func_name);
61 fprintf(fid, '\n%s running ...\n', func_name);
62
63 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
64 % warning dialog the log file records no errors/warnings generated:
65 warn = 0;
66
67 % Determine 'tau_min' value:
68 tau_min = unique(CAD(:,5)); % sec
69 % All data residing in CAD should be for the same 'tau_min' value. If multiple
70 % 'tau_min' values are returned from the above operation then there is
71 % more/different data in CAD than you think there is so tell user:
72 if length(tau_min)>1
```

```

73     fprintf(1, '\n\tWARNING: multiple tau_min values in column 5 of CAD array\n');
74     fprintf(fid, '\n\tWARNING: multiple tau_min values in column 5 of CAD array\n');
75     warn = 1;
76     else
77         fprintf(1, '\n\tPlots in terms of tau_min = %.0f sec\n', tau_min);
78         fprintf(fid, '\n\tPlots in terms of tau_min = %.0f sec\n', tau_min);
79     end
80
81     % Place absolute experimental time (column 4 of CAD) in terms of multiples
82     % of 'tau_min'. Note unlike function 'Bin_Time_v2.m' which takes care to account
83     % for upwards rounding bias when quotients of exactly 0.5 are generated,
84     % for the purposes of this program such bias is not of major concern.
85     CAD(:,4) = round(CAD(:,4)/tau_min)*tau_min;
86
87     % Determine unique absolute experimental times observed:
88     abs_exp_t = unique(CAD(:,4));
89     abs_exp_t = sort(abs_exp_t, 'ascend');
90     num_abs_exp_t = length(abs_exp_t);
91
92     % Construct a 6 column array that will hold all data for plotting:
93     % col 1 = absolute experimental time values (abscissa axis)
94     % col 2 = <|delta_x(tau_min)|> (ordinate axis)
95     % col 3 = standard deviation of <|delta_x(tau_min)|>
96     % col 4 = <|delta_y(tau_min)|> (ordinate axis)
97     % col 5 = standard deviation of <|delta_y(tau_min)|>
98     % col 6 = number of observations (replicates) contributing to mean
99     SSD = zeros(num_abs_exp_t, 6);
100
101     % Loop over all unique experimental time entries:
102     for i = 1:num_abs_exp_t
103
104         % Load present experimental time:
105         time = abs_exp_t(i);
106         % Determine indices of those entries in CAD that correspond to
107         % observations of steps over an elapsed period 'tau_min' at the elapsed
108         % experimental imaging time 'time'
109         ind = CAD(:,4)==time;
110         % Extract the corresponding steps in x and y:
111         dx = abs(CAD(ind,2));
112         dy = abs(CAD(ind,3));
113         % Compute the number of observations of cell steps over an elapsed
114         % period 'tau_min' at the elapsed experimental imaging time 'time'
115         reps = length(dx);
116
117         % Consistency check. If dr(i)^2 = dx(i)^2 + dy(i)^2 then there should
118         % always be a pair dx and dy. Check this fact.
119         if length(dx)~=length(dy)
120             fprintf(1, '\n\tWARNING: # dx ~= # dy for experimental time = %.0f\n', time);
121             fprintf(fid, '\n\tWARNING: # dx ~= # dy for experimental time = %.0f\n', time);
122         end
123
124         % Compute means and standard deviations:
125         m_dx = mean(dx);
126         std_dx = std(dx);
127         m_dy = mean(dy);
128         std_dy = std(dy);
129
130         % Log results in appropriate column and row:
131         SSD(i,1) = time;
132         SSD(i,2) = m_dx;
133         SSD(i,3) = std_dx;
134         SSD(i,4) = m_dy;
135         SSD(i,5) = std_dy;
136         SSD(i,6) = reps;
137
138     end
139
140     % Rename columns of SSD for ease of plotting:

```

```

141 T = SSD(:,1)/60; % minutes
142 m_X = SSD(:,2); % pix
143 std_X = SSD(:,3); % pix
144 m_Y = SSD(:,4); % pix
145 std_Y = SSD(:,5); % pix
146
147 % Find the min and max ordinate values across both m_X and m_Y sets of data
148 % so we can configure identical axes on each subplot:
149 yaxis_min = zeros(1,2);
150 yaxis_min(1) = min(m_X-std_X);
151 yaxis_min(2) = min(m_Y-std_Y);
152 yaxis_min = min(yaxis_min);
153 if yaxis_min < 0
154     yaxis_min = 0;
155 end
156
157 yaxis_max = zeros(1,2);
158 yaxis_max(1) = max(m_X+std_X);
159 yaxis_max(2) = max(m_Y+std_Y);
160 yaxis_max = max(yaxis_max);
161
162 % Find max experimental time value:
163 xaxis_min = 0;
164 xaxis_max = max(T);
165 t_max = xaxis_max;
166
167 % First plot will include standard deviations:
168 h1 = figure;
169 sub_h1(1) = subplot(1,2,1);
170
errorbar(T,m_X,std_X,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
171 hold all;
172 sub_h1(2) = subplot(1,2,2);
173
errorbar(T,m_Y,std_Y,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
174 hold all;
175
176 set(sub_h1,'ylim',[yaxis_min yaxis_max]);
177 set(sub_h1,'xlim',[xaxis_min xaxis_max]);
178 set(sub_h1,'FontName','Arial');
179 set(sub_h1,'FontSize',14);
180 h1_x_axis_handles = cell2mat(get(sub_h1,'xlabel'));
181 set(h1_x_axis_handles,'String','Elapsed Experimental Time (min)','FontName','Arial','FontSize',14);
182 h1_y_axis_handles = cell2mat(get(sub_h1,'ylabel'));
183 set(h1_y_axis_handles(1),'String','<math>\Delta x(\tau)> \pm s.d. (pix)','FontName','Arial','FontSize',14);
184 set(h1_y_axis_handles(2),'String','<math>\Delta y(\tau)> \pm s.d. (pix)','FontName','Arial','FontSize',14);
185 h1_title_handles = cell2mat(get(sub_h1,'title'));
186 set(h1_title_handles,'String',{'run_title;[\tau = ' num2str(tau_min) ' sec]'],'FontName','Arial','FontSize',14);
187
188 % Save figure
189 saveas(h1, 'Stationarity_with_std.fig', 'fig');
190
191 % Second plot will not include standard deviations:
192 h2 = figure;
193 sub_h2(1) = subplot(1,2,1);
194
plot(T,m_X,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
195 hold all;
196 sub_h2(2) = subplot(1,2,2);
197
plot(T,m_Y,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
198 hold all;
199
200 set(sub_h2,'ylim',[yaxis_min yaxis_max]);

```

```

201 set(sub_h2,'xlim',[xaxis_min xaxis_max]);
202 set(sub_h2,'FontName','Arial');
203 set(sub_h2,'FontSize',14);
204 h2_x_axis_handles = cell2mat(get(sub_h2,'xlabel'));
205 set(h2_x_axis_handles,'String','Elapsed Experimental Time (min)','FontName','Arial','FontSize',14);
206 h2_y_axis_handles = cell2mat(get(sub_h2,'ylabel'));
207 set(h2_y_axis_handles(1),'String','<math>\Delta x(\tau)>' (pix)','FontName','Arial','FontSize',14);
208 set(h2_y_axis_handles(2),'String','<math>\Delta y(\tau)>' (pix)','FontName','Arial','FontSize',14);
209 h2_title_handles = cell2mat(get(sub_h2,'title'));
210 set(h2_title_handles,'String',{run_title;['\tau = ' num2str(tau_min) ' sec']},'FontName','Arial','FontSize',14);
211
212 % Save figure
213 saveas(h2, 'Stationarity.fig','fig');
214
215 % Compute average number of replicates used in computation of mean and
216 % standard deviation:
217 avg_reps = round(mean(SSD(:,6)));
218 fprintf(1,'\n\tAverage # replicates used to compute mean and s.d. = %.0f\n',avg_reps);
219 fprintf(fid,'\n\tAverage # replicates used to compute mean and s.d. = %.0f\n',avg_reps);
220
221 % Save 'SSD' array in .mat and .txt format:
222 save 'SSD.mat' SSD;
223 fid2 = fopen('SSD.txt','wt');
224 fprintf(fid2,'Exp. t (sec)\t<math>dx>' (pix)\tdx s.d. (pix)\t<math>dy>' (pix)\tdy s.d. (pix)\t# Replicates\n');
225 for k = 1:num_abs_exp_t
226     fprintf(fid2,'%0f\t%f\t%f\t%f\t%f\t%.0f\n',SSD(k,:));
227 end
228 fclose(fid2);
229
230 % If no warnings generated report so in log file:
231 if warn == 0
232     fprintf(1,'\n\tFunction completed without errors/warnings\n');
233     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
234 end
235
236 % Update log file that function is completed:
237 fprintf(1,'\n%s completed\n',func_name);
238 fprintf(fid,'\n%s completed\n',func_name);
239
240 end

```

Histograms_v3.m

```

1 % Steven J. Henry
2 % 06/09/2011
20 %*****
21 % PURPOSE:
22 % This function plots:
23 % (1)Histograms of the fractional part of x-position of the particles and
24 % y-position of the particles (x mod 1, y mod 1)
25 % (2)Histograms of the absolute displacements of cells between consecutive
26 % frames (dx and dy)
27 % These are constructed to determine sample validity consistent with
28 % Crocker and Hoffman's "Multiple-Particle Tracking and Two-Point
29 % Microrheology in Cells" Methods in Cell Biology Vol. 83, 2007.
30 %
31 % ASSUMPTIONS:
32 % n/a
33 %
34 % INPUT:
35 % data = array having following structure:
36 % col 1 = unique track number ID assigned to each cell
37 % col 2 = x coordinate (pixels) of centroid
38 % col 3 = y coordinate (pixels) of centroid
39 % col 4 = absolute time cooresponding to frame in which cell is found
40 % (sec)
41 % col 5 = binned time corresponding to frame in which cell is found (sec)
42 % col 6 = area of cell in pixels

```

```

43 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
44 %     if no (i.e. continuation of an existing track.
45 % CAD = ("consecutive absolute differentials") array containing absolute
46 % displacement differentials between consecutive frames across all tracks
47 % in a given 'data' array. Data from frames separated by an interval of
48 % time greater than the attempted constant imaging rate are excluded.
49 % col 1 = track ID to which this interval belongs
50 % col 2 = dx (absolute displacement along x coordinate axis in pixels)
51 % col 3 = dy (absolute displacement along y coordinate axis in pixels)
52 % col 4 = elapsed experimental imaging time at end of corresponding
53 %     displacement (sec)
54 % col 5 = elapsed binned time (sec) between consecutive frames (i.e. tau)
55 % tau_min = elapsed time interval between most consecutive frames (i.e.
56 % attempted constant imaging rate) (sec)
57 % run_title = user specified string description of experimental condition
58 % fid = file ID of log file to which progress is recorded
59 %
60 % OUTPUT:
61 % n/a
62 %*****
63
64 function [] = Histograms_v3(data, CAD, tau_min, run_title, fid)
65
66 % Get function name:
67 func_name = mfilename;
68
69 % Update log file that function is running:
70 fprintf(1, '\n%s running ...\n', func_name);
71 fprintf(fid, '\n%s running ...\n', func_name);
72
73 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
74 % warning dialog the log file records no errors/warnings generated:
75 warn = 0;
76
77 %*****
78 % Check for systemic error in centroid position:
79
80 % Consider fractional part of x-position of the particles and y-position of
81 % the particles (x mod 1, y mod 1):
82 xmod = mod(data(:,2),1);
83 ymod = mod(data(:,3),1);
84 num_pos = length(xmod);
85
86 % Determine number of bins. A common approach is to use sqrt(num_pos)
87 % number of bins when you suspect the underlying pdf is gaussian. Here the
88 % total number of positions 'num_pos' is equal to the number of rows in
89 % 'data'.
90 num_bins = round(sqrt(num_pos));
91
92 % Record number of samples and bins used to plot data:
93 fprintf(1, '\n\tFor x mod 1 and y mod 1 histograms:\n');
94 fprintf(1, '\tNumber of positions (samples) = %0.f\n', num_pos);
95 fprintf(1, '\tNumber of bins = %0.f\n', num_bins);
96 fprintf(1, '\tDefault computation of bins is sqrt(# samples)\n');
97 fprintf(fid, '\n\tFor x mod 1 and y mod 1 histograms:\n');
98 fprintf(fid, '\tNumber of positions (samples) = %0.f\n', num_pos);
99 fprintf(fid, '\tNumber of bins = %0.f\n', num_bins);
100 fprintf(fid, '\tDefault computation of bins is sqrt(# samples)\n');
101
102 % Start figure:
103 h1 = figure;
104
105 for i = 1:2
106
107     if i == 1
108         % Load x mod 1 values:
109         samples = xmod;
110         % Prepare plot:

```

```

111     subplot(1,2,1);
112     hold all
113     title(run_title,'FontName','Arial','FontSize',14);
114     xlabel('x mod 1 (pix)','FontName','Arial','FontSize',14);
115     ylabel('Counts','FontName','Arial','FontSize',14);
116     set(gca,'FontName','Arial');
117     set(gca,'FontSize',14);
118     elseif i == 2
119         % Load y mod 1 values:
120         samples = ymod;
121         subplot(1,2,2);
122         hold all
123         title(run_title,'FontName','Arial','FontSize',14);
124         xlabel('y mod 1 (pix)','FontName','Arial','FontSize',14);
125         ylabel('Counts','FontName','Arial','FontSize',14);
126         set(gca,'FontName','Arial');
127         set(gca,'FontSize',14);
128     end
129
130     % Use intrinsic MATLAB 'hist' function to determine number of samples
131     % ('counts') that fall into a given bin with centers 'bin_loc' using a
132     % total number of bins equal to 'num_bins':
133     [counts bin_loc] = hist(samples, num_bins);
134
135     % Plot histogram using a bar graph to make bin widths visually equal.
136     % The scalar 1 allows bars to touch.
137     bar(bin_loc, counts, 1);
138
139     % Figure out y_max boundary so both plots can have the same y axis
140     % bounds
141     if i == 1
142         y_max = max(counts);
143     elseif i == 2
144         if max(counts) > y_max
145             y_max = max(counts);
146         end
147     end
148
149     clear counts bin_loc
150
151 end
152
153 % Set y-limits on both subplots:
154 subplot(1,2,1);
155 set(gca,'ylim',[0 y_max]);
156 subplot(1,2,2);
157 set(gca,'ylim',[0 y_max]);
158
159 % Save x mod 1 and y mod 1 figure:
160 saveas(h1, 'xy_mod_1.fig','fig');
161
162 %*****
163
164 %*****
165 % Now plot distribution of displacements associated with consecutive
166 % frames:
167
168 % Load data and determine total number of consecutive intervals (# samples)
169 % computed:
170 dx = CAD(:,2);
171 dy = CAD(:,3);
172 num_samples = length(dx);
173
174 % Determine number of bins. A common approach is to use sqrt(num_samples)
175 % number of bins when you suspect the underlying pdf is gaussian. Here the
176 % total number of samples 'num_samples' is equal to the number of rows in
177 % 'CAD'.
178 num_bins = round(sqrt(num_samples));

```

```

179
180 % Record number of samples and bins used to plot data:
181 fprintf(1,'\n\tFor dx and dy histograms:\n');
182 fprintf(1,'\tNumber differentials from consec. frames (samples) = %0.f\n', num_samples);
183 fprintf(1,'\tNumber of bins = %0.f\n', num_bins);
184 fprintf(1,'\tDefault computation of bins is sqrt(# samples)\n');
185 fprintf(1,'\tPlots in terms of tau_min = %0.f sec\n',tau_min);
186 fprintf(fid,'\n\tFor dx and dy histograms:\n');
187 fprintf(fid,'\tNumber differentials from consec. frames (samples) = %0.f\n', num_samples);
188 fprintf(fid,'\tNumber of bins = %0.f\n', num_bins);
189 fprintf(fid,'\tDefault computation of bins is sqrt(# samples)\n');
190 fprintf(fid,'\tPlots in terms of tau_min = %0.f sec\n',tau_min);
191
192 % Find the max and min delta terms across both dx and dy sets of data so we
193 % can configure identical axes on each subplot:
194 d_min = zeros(1,2);
195 d_min(1) = min(dx);
196 d_min(2) = min(dy);
197 d_max = zeros(1,2);
198 d_max(1) = max(dx);
199 d_max(2) = max(dy);
200
201 d_max = max(d_max);
202 d_min = min(d_min);
203
204 % Generate bin_loc vector which is a vector with length = num_bins which
205 % are linearly spaced bin centers between and including 'd_min' and
206 % 'd_max':
207 bin_loc = linspace(d_min,d_max,num_bins);
208
209 % Generate 'x_counts' and 'y_counts' vectors to hold frequency data:
210 x_counts = hist(dx, bin_loc);
211 y_counts = hist(dy, bin_loc);
212
213 % Start figure:
214 h2=figure;
215
216 % Plot 'dx' results:
217 sub_h(1) = subplot(1,2,1);
218 bar(bin_loc, x_counts, 1);
219 hold all;
220
221 % Plot 'dy' results:
222 sub_h(2) = subplot(1,2,2);
223 bar(bin_loc, y_counts, 1);
224 hold all;
225
226 % Determine maximum number of counts to set y-lim on both subplots:
227 y_max = zeros(2,1);
228 y_max(1) = max(x_counts);
229 y_max(2) = max(y_counts);
230 y_max = max(y_max);
231
232 % Make abscissa bounds symmetrical about zero:
233 d_lim = zeros(1,2);
234 d_lim(1) = abs(d_max);
235 d_lim(2) = abs(d_min);
236 d_lim = max(d_lim);
237
238 set(sub_h,'ylim',[0 y_max]);
239 set(sub_h,'xlim',[d_lim*-1 d_lim]);
240 set(sub_h,'yscale','log');
241 set(sub_h,'YMinorTick','on');
242 set(sub_h,'TickDir','out');
243 set(sub_h,'FontName','Arial');
244 set(sub_h,'FontSize',14);
245 x_axis_handles = cell2mat(get(sub_h,'xlabel'));
246 set(x_axis_handles(1),'String','\Delta x(\tau)','FontName','Arial','FontSize',14);

```

```

247 set(x_axis_handles(2),'String','\Deltay(\tau) (pix)','FontName','Arial','FontSize',14);
248 y_axis_handles = cell2mat(get(sub_h,'ylabel'));
249 set(y_axis_handles(1),'String','Counts','FontName','Arial','FontSize',14);
250 set(y_axis_handles(2),'String','Counts','FontName','Arial','FontSize',14);
251 title_handles = cell2mat(get(sub_h,'title'));
252 set(title_handles,'String',{run_title;["\tau = ' num2str(tau_min) ' sec"]},'FontName','Arial','FontSize',14);
253
254 % Save dx and dy figure:
255 saveas(h2, 'Histogram.fig','fig');
256
257 % If no warnings generated report so in log file:
258 if warn == 0
259     fprintf(1,'\n\tFunction completed without errors/warnings\n');
260     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
261 end
262
263 % Update log file that function is completed:
264 fprintf(1,'\n%s completed\n',func_name);
265 fprintf(fid,'\n%s completed\n',func_name);
266
267 end

```

Path_Length_v6.m

```

1 % Steven J. Henry
2 % 06/09/2011
34 %*****
35 % PURPOSE:
36 % This function plots the cumulative path length of each cell over elapsed
37 % experimental imaging time and saves the results.
38 %
39 % ASSUMPTIONS:
40 % n/a
41 %
42 % INPUT:
43 % data = array having following structure:
44 % col 1 = unique track number ID assigned to each cell
45 % col 2 = x coordinate (pixels) of centroid
46 % col 3 = y coordinate (pixels) of centroid
47 % col 4 = absolute time cooresponding to frame in which cell is found
48 % (sec)
49 % col 5 = binned time corresponding to frame in which cell is found (sec)
50 % col 6 = area of cell in pixels
51 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
52 % if no (i.e. continuation of an existing track.
53 % pixel_calib = user-supplied microns/pixel
54 % run_title = user specified string description of experimental condition
55 % fid = file ID of log file to which progress is recorded
56 %
57 % OUTPUT:
58 % Path_Length = array of cumulative path length data with following
59 % structure:
60 % col 1 = track ID
61 % col 2 = cumulative euclidean distance (pixels)
62 % col 3 = absolute elapsed time (sec)
63 % col 4 = binned elapsed time (sec)
64 % col 5 = track change flag. Entry = 1 if start of new track (yes) or 0
65 % if no (i.e. continuation of an existing track.
66 %*****
67
68 function [Path_Length] = Path_Length_v6(data, pixel_calib, run_title, fid)
69
70 % Get function name:
71 func_name = mfilename;
72
73 % Update log file that function is running:
74 fprintf(1,'\n%s running ...'\n',func_name);
75 fprintf(fid,'\n%s running ...'\n',func_name);

```



```

76
77 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
78 % warning dialog the log file records no errors/warnings generated:
79 warn = 0;
80
81 % Determine number of rows:
82 [rows] = size(data);
83 rows = rows(1);
84 % Reserve memory block for 'Path_Length' array:
85 Path_Length = zeros(rows,5);
86 % Transcribe track IDs and track change flags:
87 Path_Length(:,1) = data(:,1);
88 Path_Length(:,5) = data(:,7);
89
90 % Determine indices of start and stop positions of each unique track in the
91 % overall 'data' array:
92 [junk ind_start] = unique(data(:,1),'first');
93 clear junk
94 [junk ind_stop] = unique(data(:,1),'last');
95 clear junk
96
97 % Make sure lengths of start and stop vectors are same:
98 if length(ind_start) ~= length(ind_stop)
99     fprintf(1,'\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
100     fprintf(fid,'\n\tWARNING: # Unique Track IDs ~= # Track Change Flags\n');
101     warn = 1;
102 end
103
104 % Total number of tracks to be analyzed:
105 num_tracks = length(ind_start);
106
107 % Loop over tracks:
108 for i = 1:num_tracks
109
110     % Define start and stop position:
111     r_start = ind_start(i);
112     r_stop = ind_stop(i);
113
114     % Loop over all rows between start and stop rows:
115     for ii = r_start:r_stop
116
117         % If index is on start row:
118         if ii == r_start
119
120             % All elapsed values (accumulated distance, absolute time, and
121             % binned time) are zero:
122             Path_Length(ii,2:4) = 0;
123
124             % Otherwise if you are on a non-start row:
125             else
126
127                 % Load info of cell (track) in previous frame:
128                 x_prior = data(ii-1,2);
129                 y_prior = data(ii-1,3);
130                 t_abs_prior = data(ii-1,4);
131                 t_bin_prior = data(ii-1,5);
132
133                 % Load info of cell in present frame:
134                 x_now = data(ii,2);
135                 y_now = data(ii,3);
136                 t_abs_now = data(ii,4);
137                 t_bin_now = data(ii,5);
138
139                 % Compute euclidean distance between these two frames:
140                 dx = x_now - x_prior;
141                 dy = y_now - y_prior;
142                 L = sqrt(dx^2+dy^2);
143

```

```

144     % Compute elapsed time between these two frames:
145     dt_abs = t_abs_now - t_abs_prior;
146     dt_bin = t_bin_now - t_bin_prior;
147
148     % Log as the cumulative elapsed path length and time travel as
149     % the elapsed distance and time between these two frames to the
150     % previous cumulative values. Note: in essence we are
151     % computing our sums by adding the differential of each
152     % adjacent frame.
153     Path_Length(ii,2) = Path_Length(ii-1,2)+L;
154     Path_Length(ii,3) = Path_Length(ii-1,3)+dt_abs;
155     Path_Length(ii,4) = Path_Length(ii-1,4)+dt_bin;
156
157     end
158
159 end
160
161 end
162
163 % Plot results:
164 h = figure;
165
166 for j = 1:num_tracks
167
168     % Define start and stop position:
169     r_start = ind_start(j);
170     r_stop = ind_stop(j);
171
172     % Create vectors of track data:
173     PL = Path_Length(r_start:r_stop,2);
174     t_abs = Path_Length(r_start:r_stop,3);
175     t_bin = Path_Length(r_start:r_stop,4);
176
177     % Convert distances (in pixels) to microns. Note: 'pixel_calib' is
178     % supplied in microns/pixel
179     PL = PL*pixel_calib;
180
181     % Convert time vectors to minutes from seconds:
182     t_abs = t_abs/60;
183     t_bin = t_bin/60;
184
185     % Plot accumulated distance 'PL' vs. elapsed time (absolute)
186     % Units are microns and minutes
187     subplot(1,2,1)
188     hold on
189     plot(t_abs,PL,'LineStyle','-','Marker','none','LineWidth',1,'Color','b');
190
191     % Plot accumulated distance 'PL' vs. elapsed time (binned)
192     % Units are microns and minutes
193     subplot(1,2,2)
194     hold on
195     plot(t_bin,PL,'LineStyle','-','Marker','none','LineWidth',1,'Color','b');
196
197 end
198
199 % Set ordinate upperbound as maximum cumulative path length value:
200 L_max = max(Path_Length(:,2))*pixel_calib; % microns
201
202 % Set abscissa upperbound as maximum elapsed experimental imaging time
203 % value:
204 t_max = zeros(2,1);
205 t_max(1) = max(data(:,4)); % sec
206 t_max(2) = max(data(:,5)); % sec
207 t_max = max(t_max)/60; % min
208
209 % Label axes:
210 subplot(1,2,1)
211 title(run_title,'FontName','Arial','FontSize',16);

```

```

212 xlabel('Absolute Elapsed Time (min)', 'FontName', 'Arial', 'FontSize', 16);
213 ylabel('Path Length (\mum)', 'FontName', 'Arial', 'FontSize', 16);
214 set(gca, 'FontName', 'Arial');
215 set(gca, 'FontSize', 14);
216 axis([0 t_max 0 L_max]);
217
218 subplot(1,2,2)
219 title(run_title, 'FontName', 'Arial', 'FontSize', 16);
220 xlabel('Binned Elapsed Time (min)', 'FontName', 'Arial', 'FontSize', 16);
221 ylabel('Path Length (\mum)', 'FontName', 'Arial', 'FontSize', 16);
222 set(gca, 'FontName', 'Arial');
223 set(gca, 'FontSize', 14);
224 axis([0 t_max 0 L_max]);
225
226 % Save figure window generated:
227 saveas(h, 'Path_Length.fig', 'fig');
228
229 % Save 'Path_Length' array in .mat and .txt format:
230 save 'Path_Length.mat' Path_Length
231 fid2 = fopen('Path_Length.txt', 'wt');
232 fprintf(fid2, 'ID\tPL (pix)\tAbs dt (sec)\tBin dt (sec)\tTrackChange\n');
233 for k = 1:rows
234     fprintf(fid2, '%.0f\t%.0f\t%.0f\t%.0f\n', Path_Length(k,:));
235 end
236
237 fclose(fid2);
238
239 % If no warnings generated report so in log file:
240 if warn == 0
241     fprintf(1, '\n\tFunction completed without errors/warnings\n');
242     fprintf(fid, '\n\tFunction completed without errors/warnings\n');
243 end
244
245 % Update log file that function is completed:
246 fprintf(1, '\n%s completed\n', func_name);
247 fprintf(fid, '\n%s completed\n', func_name);
248
249 end

```

Mean_Path_Length_v5.m

```

1 % Steven J. Henry
2 % 08/11/2012
28 %*****
29 % PURPOSE:
30 % This function plots the mean (ensemble averaged) cumulative path length
31 % of each cell over elapsed experimental imaging time and saves the
32 % results.
33 %
34 % ASSUMPTIONS:
35 % n/a
36 %
37 % INPUT:
38 % Path_Length = array of cumulative path length data with following
39 % structure:
40 % col 1 = track ID
41 % col 2 = cumulative euclidean distance (pixels)
42 % col 3 = absolute elapsed time (sec)
43 % col 4 = binned elapsed time (sec)
44 % col 5 = track change flag. Entry = 1 if start of new track (yes) or 0
45 % if no (i.e. continuation of an existing track.
46 % pixel_calib = user-supplied microns/pixel
47 % run_title = user specified string description of experimental condition
48 % fid = file ID of log file to which progress is recorded
49 %
50 % OUTPUT (saved but not passed to main driver):
51 % Mean_Path_Length_tabs = array of mean cell path length with following
52 % structure:

```

```

53 % col 1 = mean cell path length (pixels)
54 % col 2 = standard deviation (pixels)
55 % col 3 = number of observations that contributed to this mean
56 % col 4 = absolute elapsed time (sec)
57 % Mean_Path_Length_tbin = array of mean cell path length with following
58 % structure:
59 % col 1 = mean cell path length (pixels)
60 % col 2 = standard deviation (pixels)
61 % col 3 = number of observations that contributed to this mean
62 % col 4 = binned elapsed time (sec)
63 %*****
64
65 function [] = Mean_Path_Length_v5(Path_Length, pixel_calib, run_title, fid)
66
67 % Get function name:
68 func_name = mfilename;
69
70 % Update log file that function is running:
71 fprintf(1, '\n%s running ... \n', func_name);
72 fprintf(fid, '\n%s running ... \n', func_name);
73
74 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
75 % warning dialog the log file records no errors/warnings generated:
76 warn = 0;
77
78 % Identify all elapsed time intervals that have been observed (using both
79 % absolute and binned time intervals):
80 t_abs_uniq = unique(Path_Length(:,3));
81 t_bin_uniq = unique(Path_Length(:,4));
82
83 for i = 1:2
84
85     if i == 1 % Work with absolute time values
86         t_uniq = t_abs_uniq;
87         t_all = Path_Length(:,3);
88     elseif i == 2 % Work with binned time values
89         t_uniq = t_bin_uniq;
90         t_all = Path_Length(:,4);
91     end
92
93     % Determine number of unique time values:
94     num_uniq = length(t_uniq);
95
96     % Reserve memory for temporary 'Mean_Path_Length'
97     Mean_Path_Length = zeros(num_uniq,4);
98
99     % Loop over all unique elapsed time values:
100    for ii = 1:num_uniq
101
102        % Time interval
103        dt = t_uniq(ii);
104
105        % Find indices in 't_all' that map to positions where entry is
106        % equal to 'dt'
107        ind = t_all == dt;
108
109        % Because 'Path_Length' and 't_all' have the same row
110        % structure/organization use the indices above to extract the
111        % corresponding path length values observed at the given 'dt':
112        PL_dt = Path_Length(ind,2);
113
114        % Compute mean path length for given 'dt'
115        PL_mean_dt = mean(PL_dt);
116
117        % Compute standard deviation for given 'dt'
118        PL_std_dt = std(PL_dt);
119
120        % Log mean path length, standard deviation, the number of

```

```

121     % observations used in the mean calculation, and the elapsed time
122     % interval:
123     Mean_Path_Length(ii,1) = PL_mean_dt;
124     Mean_Path_Length(ii,2) = PL_std_dt;
125     Mean_Path_Length(ii,3) = sum(ind);
126     Mean_Path_Length(ii,4) = dt;
127
128 end
129
130 if i == 1
131     Mean_Path_Length_tabs = Mean_Path_Length;
132     clear Mean_Path_Length
133 elseif i == 2
134     Mean_Path_Length_tbin = Mean_Path_Length;
135     clear Mean_Path_Length
136 end
137
138 end
139
140 % Determine indices of start and stop positions of each unique track in the
141 % 'Path_Length' array:
142 [junk ind_start] = unique(Path_Length(:,1),'first');
143 clear junk
144 [junk ind_stop] = unique(Path_Length(:,1),'last');
145 clear junk
146
147 % Make sure lengths of start and stop vectors are same:
148 if length(ind_start) ~= length(ind_stop)
149     fprintf(1,'\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
150     fprintf(fid,'\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
151     warn = 1;
152 end
153
154 % Total number of tracks to be analyzed:
155 num_tracks = length(ind_start);
156
157 % Plot overlay figure:
158 h = figure;
159
160 for j = 1:num_tracks
161
162     % Define start and stop position:
163     r_start = ind_start(j);
164     r_stop = ind_stop(j);
165
166     % Create vectors of track data:
167     PL = Path_Length(r_start:r_stop,2);
168     t_abs = Path_Length(r_start:r_stop,3);
169     t_bin = Path_Length(r_start:r_stop,4);
170
171     % Convert length (in # pixels) to microns. Note: 'pixel_calib' has
172     % units micron/pixel
173     PL = PL*pixel_calib;
174
175     % Convert time vectors to minutes from seconds:
176     t_abs = t_abs/60;
177     t_bin = t_bin/60;
178
179     % Plot path length vs. elapsed time (absolute)
180     % Units are microns and minutes
181     subplot(1,2,1)
182     hold on
183     plot(t_abs,PL,'LineStyle','none','Marker','.', 'LineWidth',1,'Color','b','HandleVisibility','off');
184
185     % Plot path length vs. elapsed time (binned)
186     % Units are microns and minutes
187     subplot(1,2,2)
188     hold on

```

```

189     plot(t_bin,PL,'LineStyle','none','Marker','.', 'LineWidth',1,'Color','b','HandleVisibility','off');
190
191 end
192
193 % Convert pixels to microns and sec to min:
194 t_abs = Mean_Path_Length_tabs(:,4)/60;
195 m_abs = Mean_Path_Length_tabs(:,1)*pixel_calib;
196 std_abs = Mean_Path_Length_tabs(:,2)*pixel_calib;
197 t_bin = Mean_Path_Length_tbin(:,4)/60;
198 m_bin = Mean_Path_Length_tbin(:,1)*pixel_calib;
199 std_bin = Mean_Path_Length_tbin(:,2)*pixel_calib;
200
201 % Set ordinate upperbound as maximum mean cummulative path length value:
202 PL_max = max(Path_Length(:,2))*pixel_calib; % microns
203
204 % Set abscissa upperbound as maximum elapsed experimental imaging time
205 % value:
206 t_max = zeros(2,1);
207 t_max(1) = max(t_abs); % min
208 t_max(2) = max(t_bin); % min
209 t_max = max(t_max); % min
210
211 % Label axes and overlay mean series:
212 subplot(1,2,1)
213
214 plot(t_abs,m_abs,'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
215 title(run_title,'FontName','Arial','FontSize',16);
216 xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
217 ylabel('Path Length (\mum)','FontName','Arial','FontSize',16);
218 set(gca,'FontName','Arial');
219 set(gca,'FontSize',14);
220 h_legend = legend('Mean Path Length','Location','NorthWest');
221 set(h_legend,'FontName','Arial');
222 set(h_legend,'FontSize',12);
223 axis([0 t_max 0 PL_max]);
224
225 subplot(1,2,2)
226
227 plot(t_bin,m_bin,'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
228 title(run_title,'FontName','Arial','FontSize',16);
229 xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
230 ylabel('Path Length (\mum)','FontName','Arial','FontSize',16);
231 set(gca,'FontName','Arial');
232 set(gca,'FontSize',14);
233 h_legend = legend('Mean Path Length','Location','NorthWest');
234 set(h_legend,'FontName','Arial');
235 set(h_legend,'FontSize',12);
236 axis([0 t_max 0 PL_max]);
237
238 % Save figure window generated:
239 saveas(h, 'Mean_Path_Length_Overlay.fig', 'fig');
240
241 %*****
242 % Plot just mean with error bars:
243
244 % Set ordinate upperbound as maximum accumulated distance value:
245 temp = zeros(1,2);
246 temp(1,1) = max(Mean_Path_Length_tabs(:,1)+Mean_Path_Length_tabs(:,2));
247 temp(1,2) = max(Mean_Path_Length_tbin(:,1)+Mean_Path_Length_tbin(:,2));
248 PL_max = max(temp)*pixel_calib;
249
250 h2 = figure;
251 subplot(1,2,1)
252 errorbar(t_abs,m_abs,std_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);

```

```

251 hold on
252 title(run_title,'FontName','Arial','FontSize',16);
253 xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
254 ylabel('<Path Length> \pm s.d. (\mum)','FontName','Arial','FontSize',16);
255 set(gca,'FontName','Arial');
256 set(gca,'FontSize',14);
257 axis([0 t_max 0 PL_max]);
258
259 subplot(1,2,2)
260
errorbar(t_bin,m_bin,std_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
261 hold on
262 title(run_title,'FontName','Arial','FontSize',16);
263 xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
264 ylabel('<Path Length> \pm s.d. (\mum)','FontName','Arial','FontSize',16);
265 set(gca,'FontName','Arial');
266 set(gca,'FontSize',14);
267 axis([0 t_max 0 PL_max]);
268
269 % Save figure window generated:
270 saveas(h2, 'Mean_Path_Length.fig', 'fig');
271 %*****
272
273 % Save 'Mean_Path_Length' arrays in .mat and .txt format:
274 save 'Mean_Path_Length_tabs.mat' Mean_Path_Length_tabs
275 fid2_1 = fopen('Mean_Path_Length_tabs.txt','wt');
276 fprintf(fid2_1,'Mean PL (# pix)\tStdDev (# pix)\t# Obs\tAbs dt (sec)\n');
277 rows = length(Mean_Path_Length_tabs);
278 for k = 1:rows
279     fprintf(fid2_1,'%f\t%f\t%.0f\t%.0f\n',Mean_Path_Length_tabs(k,:));
280 end
281 fclose(fid2_1);
282
283 save 'Mean_Path_Length_tbin.mat' Mean_Path_Length_tbin
284 fid2_2 = fopen('Mean_Path_Length_tbin.txt','wt');
285 fprintf(fid2_2,'Mean PL (# pix)\tStdDev (# pix)\t# Obs\tBin dt (sec)\n');
286 rows = length(Mean_Path_Length_tbin);
287 for kk = 1:rows
288     fprintf(fid2_2,'%f\t%f\t%.0f\t%.0f\n',Mean_Path_Length_tbin(kk,:));
289 end
290 fclose(fid2_2);
291
292 % If no warnings generated report so in log file:
293 if warn == 0
294     fprintf(1,'\n\tFunction completed without errors/warnings\n');
295     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
296 end
297
298 % Update log file that function is completed:
299 fprintf(1,'\n%s completed\n',func_name);
300 fprintf(fid,'\n%s completed\n',func_name);
301
302 end

```

Area_v4.m

```

1 % Steven J. Henry
2 % 06/09/2011
3 % *****
4 % PURPOSE:
5 % This function plots area of each cell over elapsed experimental imaging
6 % time and saves the results. Since experiments are all 2D migration and
7 % cells are terminally differentiated, cross sectional area (contact area)
8 % can be considered a metric of substrate affinity. If area changes with
9 % time one might correlate this to a change in substrate affinity.
10 %
11 % ASSUMPTIONS:

```

```

32 % n/a
33 %
34 % INPUT:
35 % data = array having following structure:
36 % col 1 = unique track number ID assigned to each cell
37 % col 2 = x coordinate (pixels) of centroid
38 % col 3 = y coordinate (pixels) of centroid
39 % col 4 = absolute time cooresponding to frame in which cell is found
40 % (sec)
41 % col 5 = binned time corresponding to frame in which cell is found (sec)
42 % col 6 = area of cell in pixels
43 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
44 % if no (i.e. continuation of an existing track.
45 % pixel_calib = user-supplied microns/pixel
46 % t_max = user-supplied imaging duration in minutes
47 % run_title = user specified string description of experimental condition
48 % fid = file ID of log file to which progress is recorded
49 %
50 % OUTPUT:
51 % Area = array of track area data with following structure:
52 % col 1 = track ID
53 % col 2 = cell area (pixels)
54 % col 3 = absolute elapsed time (sec)
55 % col 4 = binned elapsed time (sec)
56 % col 5 = track change flag. Entry = 1 if start of new track (yes) or 0
57 % if no (i.e. continuation of an existing track.
58 %*****
59
60 function [Area] = Area_v4(data, pixel_calib, run_title, fid)
61
62 % Get function name:
63 func_name = mfilename;
64
65 % Update log file that function is running:
66 fprintf(1, '\n%s running ...\n', func_name);
67 fprintf(fid, '\n%s running ...\n', func_name);
68
69 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
70 % warning dialog the log file records no errors/warnings generated:
71 warn = 0;
72
73 % Determine number of rows:
74 [rows] = size(data);
75 rows = rows(1);
76 % Reserve memory block for 'Area' array:
77 Area = zeros(rows,5);
78 % Transcribe track IDs, areas, and track change flags:
79 Area(:,1) = data(:,1);
80 Area(:,2) = data(:,6);
81 Area(:,5) = data(:,7);
82
83 % Determine indices of start and stop positions of each unique track in the
84 % overall 'data' array:
85 [junk ind_start] = unique(data(:,1), 'first');
86 clear junk
87 [junk ind_stop] = unique(data(:,1), 'last');
88 clear junk
89
90 % Make sure lengths of start and stop vectors are same:
91 if length(ind_start) ~= length(ind_stop)
92     fprintf(1, '\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
93     fprintf(fid, '\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
94     warn = 1;
95 end
96
97 % Total number of tracks to be analyzd:
98 num_tracks = length(ind_start);
99

```



```

100 % Loop over tracks to compute elapsed times. Note: this was previously done
101 % in 'Path_Length.m' but is duplicated here so that the 'Path_Length'
102 % and 'Area.m' functions are independent.
103 for i = 1:num_tracks
104
105     % Define start and stop position:
106     r_start = ind_start(i);
107     r_stop = ind_stop(i);
108
109     % Loop over all rows between start and stop rows:
110     for ii = r_start:r_stop
111
112         % If index is on start row:
113         if ii == r_start
114
115             % Elapsed time values are zero:
116             Area(ii,3:4) = 0;
117
118             % Otherwise if you are on a non-start row:
119             else
120
121                 % Load info of cell (track) in previous frame:
122                 t_abs_prior = data(ii-1,4);
123                 t_bin_prior = data(ii-1,5);
124
125                 % Load info of cell in present frame:
126                 t_abs_now = data(ii,4);
127                 t_bin_now = data(ii,5);
128
129                 % Compute elapsed time between these two frames:
130                 dt_abs = t_abs_now - t_abs_prior;
131                 dt_bin = t_bin_now - t_bin_prior;
132
133                 % Log as the cumulative elapsed time as the elapsed time
134                 % between these two frames to the previous cumulative values.
135                 % Note: in essence we are computing our sums by adding the
136                 % differential of each adjacent frame.
137                 Area(ii,3) = Area(ii-1,3)+dt_abs;
138                 Area(ii,4) = Area(ii-1,4)+dt_bin;
139
140             end
141
142         end
143
144     end
145
146     % Determine physical area of each pixel (assume pixel is square with edge
147     % length 'pixel_calib'). Units of pixel_area are (um^2/pixel)
148     pixel_area = pixel_calib^2;
149
150     % Plot results:
151     h = figure;
152
153     for j = 1:num_tracks
154
155         % Define start and stop position:
156         r_start = ind_start(j);
157         r_stop = ind_stop(j);
158
159         % Create vectors of track data:
160         A = Area(r_start:r_stop,2);
161         t_abs = Area(r_start:r_stop,3);
162         t_bin = Area(r_start:r_stop,4);
163
164         % Convert areas (in # pixels) to microns^2. Note: 'pixel_area' has
165         % units micron^2/pixel
166         A = A*pixel_area;
167

```

```

168 % Convert time vectors to minutes from seconds:
169 t_abs = t_abs/60;
170 t_bin = t_bin/60;
171
172 % Plot area vs. elapsed time (absolute)
173 % Units are microns^2 and minutes
174 subplot(1,2,1)
175 hold on
176 plot(t_abs,A,'LineStyle','-','Color','b','Marker','none','LineWidth',1);
177
178 % Plot area vs. elapsed time (binned)
179 % Units are microns^2 and minutes
180 subplot(1,2,2)
181 hold on
182 plot(t_bin,A,'LineStyle','-','Color','b','Marker','none','LineWidth',1);
183
184 end
185
186 % Set ordinate upperbound as maximum area value:
187 A_max = max(Area(:,2))*pixel_area;
188
189 % Set abscissa upperbound as maximum elapsed experimental imaging time
190 % value:
191 t_max = zeros(2,1);
192 t_max(1) = max(data(:,4)); % sec
193 t_max(2) = max(data(:,5)); % sec
194 t_max = max(t_max)/60; % min
195
196 % Label axes:
197 subplot(1,2,1)
198 title(run_title,'FontName','Arial','FontSize',16);
199 xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
200 ylabel('Cell Area (\mum^2)','FontName','Arial','FontSize',16);
201 set(gca,'FontName','Arial');
202 set(gca,'FontSize',14);
203 axis([0 t_max 0 A_max]);
204
205 subplot(1,2,2)
206 title(run_title,'FontName','Arial','FontSize',16);
207 xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
208 ylabel('Cell Area (\mum^2)','FontName','Arial','FontSize',16);
209 set(gca,'FontName','Arial');
210 set(gca,'FontSize',14);
211 axis([0 t_max 0 A_max]);
212
213 % Save 'Area' array in .mat and .txt format:
214 save 'Area.mat' Area;
215 fid2 = fopen('Area.txt','wt');
216 fprintf(fid2,'ID\tArea (# pix)\tAbs dt (sec)\tBin dt (sec)\tTrackChange\n');
217 for k = 1:rows
218     fprintf(fid2,'%0ft%ft%.0ft%.0ft%.0ft\n',Area(k,:));
219 end
220 fclose(fid2);
221
222 % Save figure window generated:
223 saveas(h, 'Area.fig', 'fig');
224
225 % If no warnings generated report so in log file:
226 if warn == 0
227     fprintf(1,'\n\tFunction completed without errors/warnings\n');
228     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
229 end
230
231 % Update log file that function is completed:
232 fprintf(1,'\n%s completed\n',func_name);
233 fprintf(fid,'\n%s completed\n',func_name);
234
235 end

```

Mean_Area_v5.m

```
1 % Steven J. Henry
2 % 08/11/2012
24 %*****
25 % PURPOSE:
26 % This function plots mean (ensemble average) cell area as a function of
27 % elapsed time.
28 %
29 % ASSUMPTIONS:
30 % n/a
31 %
32 % INPUT:
33 % Area = array of track area data with following structure:
34 % col 1 = track ID
35 % col 2 = cell area (pixels)
36 % col 3 = absolute elapsed time (sec)
37 % col 4 = binned elapsed time (sec)
38 % col 5 = track change flag. Entry = 1 if start of new track (yes) or 0
39 % if no (i.e. continuation of an existing track.
40 % pixel_calib = user-supplied microns/pixel
41 % run_title = user specified string description of experimental condition
42 % fid = file ID of log file to which progress is recorded
43 %
44 % OUTPUT:
45 % Mean_Area_tabs = array of mean cell area with following structure:
46 % col 1 = mean cell area (pixels)
47 % col 2 = standard deviation (pixels)
48 % col 3 = number of observations that contributed to this mean
49 % col 4 = absolute elapsed time (sec)
50 % Mean_Area_tbin = array of mean cell area with following structure:
51 % col 1 = mean cell area (pixels)
52 % col 2 = standard deviation (pixels)
53 % col 3 = number of observations that contributed to this mean
54 % col 4 = binned elapsed time (sec)
55 %*****
56
57 function [] = Mean_Area_v5(Area, pixel_calib, run_title, fid)
58
59 % Get function name:
60 func_name = mfilename;
61
62 % Update log file that function is running:
63 fprintf(1, '\n%s running ...\n', func_name);
64 fprintf(fid, '\n%s running ...\n', func_name);
65
66 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
67 % warning dialog the log file records no errors/warnings generated:
68 warn = 0;
69
70 % Identify all elapsed time intervals that have been observed (using both
71 % absolute and binned time intervals):
72 t_abs_uniq = unique(Area(:,3));
73 t_bin_uniq = unique(Area(:,4));
74
75 for i = 1:2
76
77     if i == 1 % Work with absolute time values
78         t_uniq = t_abs_uniq;
79         t_all = Area(:,3);
80     elseif i == 2 % Work with binned time values
81         t_uniq = t_bin_uniq;
82         t_all = Area(:,4);
83     end
84
85     % Determine number of unique time values:
86     num_uniq = length(t_uniq);
```

```

87
88 % Reserve memory for temporary 'Mean_Array'
89 Mean_Area = zeros(num_uniq,4);
90
91 % Loop over all unique elapsed time values:
92 for ii = 1:num_uniq
93
94     % Time interval
95     dt = t_uniq(ii);
96
97     % Find indices in 't_all' that map to positions where entry is
98     % equal to 'dt'
99     ind = t_all == dt;
100
101     % Because 'Area' and 't_all' have the same row
102     % structure/organization use the indices above to extract the
103     % corresponding area values observed at the given 'dt':
104     A_dt = Area(ind,2);
105
106     % Compute mean area for given 'dt'
107     A_mean_dt = mean(A_dt);
108
109     % Compute standard deviation for given 'dt'
110     A_std_dt = std(A_dt);
111
112     % Log mean area, standard deviation, the number of observations
113     % used in the mean calculation, and the elapsed time interval:
114     Mean_Area(ii,1) = A_mean_dt;
115     Mean_Area(ii,2) = A_std_dt;
116     Mean_Area(ii,3) = sum(ind);
117     Mean_Area(ii,4) = dt;
118
119 end
120
121 if i == 1
122     Mean_Area_tabs = Mean_Area;
123     clear Mean_Area
124 elseif i == 2
125     Mean_Area_tbin = Mean_Area;
126     clear Mean_Area
127 end
128
129 end
130
131 % Determine indices of start and stop positions of each unique track in the
132 % 'Area' array:
133 [junk ind_start] = unique(Area(:,1),'first');
134 clear junk
135 [junk ind_stop] = unique(Area(:,1),'last');
136 clear junk
137
138 % Make sure lengths of start and stop vectors are same:
139 if length(ind_start) ~= length(ind_stop)
140     fprintf(1, '\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
141     fprintf(fid, '\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
142     warn = 1;
143     keyboard
144 end
145
146 % Total number of tracks to be analyzed:
147 num_tracks = length(ind_start);
148
149 % Determine physical area of each pixel (assume pixel is square with edge
150 % length 'pixel_calib'). Units of pixel_area are (um^2/pixel)
151 pixel_area = pixel_calib^2;
152
153 % Plot overlay figure
154 h = figure;

```

```

155
156 for j = 1:num_tracks
157
158     % Define start and stop position:
159     r_start = ind_start(j);
160     r_stop = ind_stop(j);
161
162     % Create vectors of track data:
163     A = Area(r_start:r_stop,2);
164     t_abs = Area(r_start:r_stop,3);
165     t_bin = Area(r_start:r_stop,4);
166
167     % Convert areas (in # pixels) to microns^2. Note: 'pixel_area' has
168     % units micron^2/pixel
169     A = A*pixel_area;
170
171     % Convert time vectors to minutes from seconds:
172     t_abs = t_abs/60;
173     t_bin = t_bin/60;
174
175     % Plot area vs. elapsed time (absolute)
176     % Units are microns^2 and minutes
177     subplot(1,2,1)
178     hold on
179     plot(t_abs,A,'LineStyle','none','Marker','.', 'LineWidth',1,'Color','b','HandleVisibility','off');
180
181     % Plot area vs. elapsed time (binned)
182     % Units are microns^2 and minutes
183     subplot(1,2,2)
184     hold on
185     plot(t_bin,A,'LineStyle','none','Marker','.', 'LineWidth',1,'Color','b','HandleVisibility','off');
186
187 end
188
189 % Convert pixels to microns and sec to min:
190 t_abs = Mean_Area_tabs(:,4)/60;
191 m_abs = Mean_Area_tabs(:,1)*pixel_area;
192 std_abs = Mean_Area_tabs(:,2)*pixel_area;
193 t_bin = Mean_Area_tbin(:,4)/60;
194 m_bin = Mean_Area_tbin(:,1)*pixel_area;
195 std_bin = Mean_Area_tbin(:,2)*pixel_area;
196
197 % Set ordinate upperbound as maximum area value:
198 A_max = max(Area(:,2))*pixel_area;
199
200 % Set abscissa upperbound as maximum elapsed experimental imaging time
201 % value:
202 t_max = zeros(2,1);
203 t_max(1) = max(t_abs); % min
204 t_max(2) = max(t_bin); % min
205 t_max = max(t_max); % min
206
207 % Label axes and overlay mean series:
208 subplot(1,2,1)
209
plot(t_abs,m_abs,'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','
on');
210 title(run_title,'FontName','Arial','FontSize',16);
211 xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
212 ylabel('Cell Area (\mum^2)','FontName','Arial','FontSize',16);
213 set(gca,'FontName','Arial');
214 set(gca,'FontSize',14);
215 h_legend = legend('Mean Area','Location','NorthEast');
216 set(h_legend,'FontName','Arial');
217 set(h_legend,'FontSize',12);
218 axis([0 t_max 0 A_max]);
219
220 subplot(1,2,2)

```

```

221 plot(t_bin,m_bin,'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','o
n');
222 title(run_title,'FontName','Arial','FontSize',16);
223 xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
224 ylabel('Cell Area (\mum^2)','FontName','Arial','FontSize',16);
225 set(gca,'FontName','Arial');
226 set(gca,'FontSize',14);
227 h_legend = legend('Mean Area','Location','NorthEast');
228 set(h_legend,'FontName','Arial');
229 set(h_legend,'FontSize',12);
230 axis([0 t_max 0 A_max]);
231
232 % Save figure window generated:
233 saveas(h, 'Mean_Area_Overlay.fig', 'fig');
234
235 %*****
236 % Plot just mean with error bars:
237
238 % Set ordinate upperbound as maximum accumulated distance value:
239 temp = zeros(1,2);
240 temp(1,1) = max(Mean_Area_tabs(:,1)+Mean_Area_tabs(:,2));
241 temp(1,1) = max(Mean_Area_tbin(:,1)+Mean_Area_tbin(:,2));
242 A_max = max(temp)*pixel_area;
243
244 h2 = figure;
245 subplot(1,2,1)
246
errorbar(t_abs,m_abs,std_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSi
ze',5);
247 hold on
248 title(run_title,'FontName','Arial','FontSize',16);
249 xlabel('Absolute Elapsed Time (min)','FontName','Arial','FontSize',16);
250 ylabel('<Cell Area> \p m s.d. (\mum^2)','FontName','Arial','FontSize',16);
251 set(gca,'FontName','Arial');
252 set(gca,'FontSize',14);
253 axis([0 t_max 0 A_max]);
254
255 subplot(1,2,2)
256
errorbar(t_bin,m_bin,std_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSiz
e',5);
257 hold on
258 title(run_title,'FontName','Arial','FontSize',16);
259 xlabel('Binned Elapsed Time (min)','FontName','Arial','FontSize',16);
260 ylabel('<Cell Area> \p m s.d. (\mum^2)','FontName','Arial','FontSize',16);
261 set(gca,'FontName','Arial');
262 set(gca,'FontSize',14);
263 axis([0 t_max 0 A_max]);
264
265 % Save figure window generated:
266 saveas(h2, 'Mean_Area.fig', 'fig');
267 %*****
268
269 % Save 'Mean_Area' arrays in .mat and .txt format:
270 save 'Mean_Area_tabs.mat' Mean_Area_tabs;
271 fid2_1 = fopen('Mean_Area_tabs.txt','wt');
272 fprintf(fid2_1,'Mean Area (# pix)\tStd Dev (# pix)\t# Obs\tAbs dt (sec)\n');
273 rows = length(Mean_Area_tabs);
274 for k = 1:rows
275     fprintf(fid2_1,'%ft%ft%.0ft%.0ft\n',Mean_Area_tabs(k,:));
276 end
277 fclose(fid2_1);
278
279 save 'Mean_Area_tbin.mat' Mean_Area_tbin;
280 fid2_2 = fopen('Mean_Area_tbin.txt','wt');
281 fprintf(fid2_2,'Mean Area (# pix)\tStd Dev (# pix)\t# Obs\tBin dt (sec)\n');
282 rows = length(Mean_Area_tbin);

```

```

283 for kk = 1:rows
284     fprintf(fid2_2, '%f\t%f\t%.0f\t%.0f\n', Mean_Area_tbin(kk,:));
285 end
286 fclose(fid2_2);
287
288 % If no warnings generated report so in log file:
289 if warn == 0
290     fprintf(1, '\n\tFunction completed without errors/warnings\n');
291     fprintf(fid, '\n\tFunction completed without errors/warnings\n');
292 end
293
294 % Update log file that function is completed:
295 fprintf(1, '\n%s completed\n', func_name);
296 fprintf(fid, '\n%s completed\n', func_name);
297
298 end

```

Filter_Exp_Data_v3.m

```

1  % Steven J. Henry
2  % 08/11/2012
21 %*****
22 % PURPOSE:
23 % This function filters a supplied array of centroid/area measurements and
24 % retains only those measurements obtained at or before some user-defined
25 % maximum experimental imaging time. It produces an array with the same
26 % structure originally input but only containing data corresponding to
27 % imaging time-stamps less than or equal to the user-specified 'exp_t_max'
28 %
29 % REMARKS:
30 % n/a
31 %
32 % ASSUMPTIONS:
33 % n/a
34 %
35 % INPUT:
36 % data = an array containing all pertinent tracking information for each
37 % cell in the given experimental condition. It contains measurements over
38 % entire course of actual experimental imaging. The array has the following
39 % structure:
40 % col 1 = unique track number ID assigned to each cell
41 % col 2 = x coordinate (pixels) of centroid
42 % col 3 = y coordinate (pixels) of centroid
43 % col 4 = absolute time corresponding to frame in which cell is found
44 % (sec)
45 % col 5 = binned time corresponding to frame in which cell is found (sec)
46 % col 6 = area of cell in pixels
47 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
48 % if no (i.e. continuation of anexisting track.
49 % exp_t_max = user-supplied maximum experimental imaging time (min)
50 % fid = file ID of log file to which progress is recorded
51 %
52 % OUTPUT:
53 % data_filt = same array and structure as input but now only containing
54 % data from images with time-stamps equal to or less than the
55 % user-specified maximum imaging time-stamp to be processed.
56 %*****
57
58 function [data_filt] = Filter_Exp_Data_v3(data, exp_t_max, fid)
59
60 % Get function name:
61 func_name = mfilename;
62
63 % Update log file that function is running:
64 fprintf(1, '\n%s running ... \n', func_name);
65 fprintf(fid, '\n%s running ... \n', func_name);
66
67 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a

```

```

68 % warning dialog the log file records no errors/warnings generated:
69 warn = 0;
70
71 % Convert 'exp_t_max' to sec from min:
72 exp_t_max = exp_t_max*60;
73
74 % Identify binned time values that exceed exp_t_max value in seconds:
75 del_ind = data(:,5) > exp_t_max;
76
77 % Eliminate these rows:
78 data_filt = data;
79 data_filt(del_ind,:) = [];
80
81 % Dimension check:
82 num_del_theory = sum(del_ind);
83 num_del_practice = size(data,1)-size(data_filt,1);
84 if num_del_practice ~= num_del_theory
85     fprintf(1,'\n\tWARNING: # deletions from data ~= # deletions predicted\n');
86     fprintf(1,'\n\tWARNING: # deletions from data ~= # deletions predicted\n');
87     warn = 1;
88     keyboard
89 end
90
91 % If no warnings generated report so in log file:
92 if warn == 0
93     fprintf(1,'\n\tFunction completed without errors/warnings\n');
94     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
95 end
96
97 % Update log file that function is completed:
98 fprintf(1,'\n%s completed\n',func_name);
99 fprintf(fid,'\n%s completed\n',func_name);
100
101 end

```

Differentials_v5.m

```

1 % Steven J. Henry
2 % 08/11/2012
45 %*****
46 % PURPOSE:
47 % This function computes squared differentials in displacement and time for
48 % all intervals **USING A MOVING ORIGIN** along a given cell's track.
49 %
50 % ASSUMPTIONS:
51 % n/a
52 %
53 % INPUT:
54 % data = array having following structure:
55 % col 1 = unique track number ID assigned to each cell
56 % col 2 = x coordinate (pixels) of centroid
57 % col 3 = y coordinate (pixels) of centroid
58 % col 4 = absolute time cooresponding to frame in which cell is found
59 % (sec)
60 % col 5 = binned time corresponding to frame in which cell is found (sec)
61 % col 6 = area of cell in pixels
62 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
63 % if no (i.e. continuation of an existing track.
64 % fid = file ID of log file to which progress is recorded
65 %
66 % OUTPUT:
67 % SD = array containing squared displacements and corresponding times
68 % col 1 = track ID to which this interval belongs
69 % col 2 = d^2 (squared displacement (euclidean distance) in pixels^2)
70 % Note: d^2 values are not rounded to the nearest whole pixel
71 % col 3 = interval absolute time (sec)
72 % col 4 = interval binned time (sec)
73 % col 5 = dx (pix)

```



```

74 % col 6 = dy (pix)
75 %*****
76
77 function [SD] = Differentials_v5(data, fid)
78
79 % Get function name:
80 func_name = mfilename;
81
82 % Update log file that function is running:
83 fprintf(1, '\n%s running ...\n', func_name);
84 fprintf(fid, '\n%s running ...\n', func_name);
85
86 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
87 % warning dialog the log file records no errors/warnings generated:
88 warn = 0;
89
90 % Determine indices of start and stop positions of each unique track in the
91 % overall 'data' array:
92 [junk ind_start] = unique(data(:,1), 'first');
93 clear junk
94 [junk ind_stop] = unique(data(:,1), 'last');
95 clear junk
96
97 % Make sure lengths of start and stop vectors are same:
98 if length(ind_start) ~= length(ind_stop)
99     fprintf(1, '\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
100     fprintf(fid, '\n\tWARNING: # Unique Track IDs ~= # Track Change Flags\n');
101     warn = 1;
102 end
103
104 % Total number of tracks to be analyzed:
105 num_tracks = length(ind_start);
106
107 % Reserve array name:
108 SD = [];
109
110 % Reserve 'num_intervals_tot' to hold total number of intervals computed:
111 num_intervals_tot = 0;
112
113 % Loop over tracks:
114 for i = 1:num_tracks
115
116     % Define start and stop position:
117     r_start = ind_start(i);
118     r_stop = ind_stop(i);
119
120     % Determine the number of intervals that will be observed when using a
121     % moving origin from the first row of this track to the second-to-last
122     % row of the track.
123     N = r_stop - r_start + 1;
124     num_intervals = 0;
125     for j = 1:N
126         num_intervals = (j-1) + num_intervals;
127     end
128     track_SD = zeros(num_intervals, 6);
129     num_intervals_tot = num_intervals_tot + num_intervals;
130     print_row = 1;
131
132     % Loop over all rows between start and second-to-last stop rows:
133     for orig_row = r_start:r_stop-1
134
135         % Load origin info:
136         track_ID = data(orig_row, 1);
137         x_orig = data(orig_row, 2);
138         y_orig = data(orig_row, 3);
139         t_abs_orig = data(orig_row, 4);
140         t_bin_orig = data(orig_row, 5);
141

```

```

142 % Loop over all advance rows ahead of origin between 'orig_row' + 1
143 % and last row ('r_stop'):
144 for adv_row = orig_row + 1:r_stop
145
146     % Load advance row info:
147     x_adv = data(adv_row,2);
148     y_adv = data(adv_row,3);
149     t_abs_adv = data(adv_row,4);
150     t_bin_adv = data(adv_row,5);
151
152     % Compute differentials:
153     dx = (x_adv - x_orig);
154     dx2 = dx^2;
155     dy = (y_adv - y_orig);
156     dy2 = dy^2;
157     dt_abs = t_abs_adv - t_abs_orig;
158     dt_bin = t_bin_adv - t_bin_orig;
159
160     % Compute squared displacement:
161     d2 = dx2 + dy2;
162
163     % Log values:
164     track_SD(print_row,1) = track_ID;
165     track_SD(print_row,2) = d2;
166     track_SD(print_row,3) = dt_abs;
167     track_SD(print_row,4) = dt_bin;
168     track_SD(print_row,5) = dx;
169     track_SD(print_row,6) = dy;
170
171     % Advance print_row:
172     print_row = print_row+1;
173
174 end
175
176 end
177
178 % After all intervals for 'track_ID' are computed make sure the length
179 % of 'track_SD' is = 'num_intervals':
180 [track_SD_r track_SD_c] = size(track_SD);
181 if track_SD_r ~= num_intervals
182     fprintf(1, '\n\tWARNING: # intervals computed ~= expected #\n');
183     fprintf(1, '\tWarning generated for track = %.0f\n', track_ID);
184     fprintf(fid, '\n\tWARNING: # intervals computed ~= expected #\n');
185     fprintf(fid, '\tWarning generated for track = %.0f\n', track_ID);
186     warn = 1;
187     keyboard
188 end
189
190 % Concatenate 'track_SD' with existing 'SD'
191 if isempty(SD) == 1
192     SD = track_SD;
193     clear track_SD
194     % In the event only 1 track exists in this 'data' array we want to
195     % have the dimensions for the post-processing dimensionality check.
196     [SD_new_r SD_new_c] = size(SD);
197
198 else
199     SD_old = SD;
200     [SD_old_r SD_old_c] = size(SD_old);
201     clear SD
202     SD = zeros(SD_old_r+track_SD_r, SD_old_c);
203     SD(1:SD_old_r, 1:track_SD_c) = SD_old;
204     SD(SD_old_r+1:SD_old_r+track_SD_r, 1:track_SD_c) = track_SD;
205     [SD_new_r SD_new_c] = size(SD);
206
207 end
208
209 end

```

```

210
211 % Dimensionality check. There should be six columns after the
212 % concatenations are finished:
213 if SD_new_c ~= 6
214     fprintf(1,'\n\tWARNING: # cols = %.0f not 6 as expected\n',SD_new_c);
215     fprintf(1,'\tWarning generated for track = %.0f\n',track_ID);
216     fprintf(fid,'\n\tWARNING: # cols = %.0f not 6 as expected\n',SD_new_c);
217     fprintf(fid,'\tWarning generated for track = %.0f\n',track_ID);
218     warn = 1;
219     keyboard
220 end
221 % Dimensionality check. The total number of rows in 'SD' should be equal to
222 % the sum of each track's theoretical intervals:
223 if SD_new_r ~= num_intervals_tot;
224     fprintf(1,'\n\tWARNING: # rows (intervals) = %.0f not %.0f as expected\n',SD_new_r,num_intervals_tot);
225     fprintf(fid,'\n\tWARNING: # rows (intervals) = %.0f not %.0f as expected\n',SD_new_r,num_intervals_tot);
226     warn = 1;
227     keyboard
228 end
229
230 % Check for instances in which a squared displacement was measured over an
231 % elapsed time of zero:
232 dt_zero_inds = SD(:,3)==0 | SD(:,4)==0;
233 num_dt_zero_inds = sum(dt_zero_inds);
234
235 if num_dt_zero_inds > 0
236     dt_nz_inds = SD(:,3)~=0 & SD(:,4)~=0;
237     SD = SD(dt_nz_inds,:);
238
239     % Dimensionality check.
240     SD_dt_nz_r = size(SD,1);
241     if SD_dt_nz_r ~= SD_new_r - num_dt_zero_inds
242         fprintf(1,'\n\tWARNING: # zero tau deletions made (%.0f) ~= # zero tau deletions expected (%.0f)\n',SD_new_r-
SD_dt_nz_r,num_dt_zero_inds);
243         fprintf(fid,'\n\tWARNING: # zero tau deletions made (%.0f) ~= # zero tau deletions expected
(%.0f)\n',SD_new_r-SD_dt_nz_r,num_dt_zero_inds);
244         warn = 1;
245         keyboard
246     end
247 end
248
249 % Save 'SD' array in .mat and .txt format:
250 save 'SD.mat' SD;
251 fid2 = fopen('SD.txt','wt');
252 fprintf(fid2,'Track ID\t^2 (pix^2)\tAbs dt (sec)\tBin dt (sec)\tdx (pix)\tdy (pix)\n');
253 final_num_rows = size(SD,1);
254 for k = 1:final_num_rows
255     fprintf(fid2,'%.0f\t%f\t%.0f\t%.0f\t%.0f\t%.0f\n',SD(k,:));
256 end
257 fclose(fid2);
258
259 % If no warnings generated report so in log file:
260 if warn == 0
261     fprintf(1,'\n\tFunction completed without errors/warnings\n');
262     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
263 end
264
265 % Update log file that function is completed:
266 fprintf(1,'\n%s completed\n',func_name);
267 fprintf(fid,'\n%s completed\n',func_name);
268
269 end

```

Neff_v1.m

```

1 % Steven J. Henry
2 % 06/10/2011
7 %*****

```

```

8 % PURPOSE:
9 % This function computes Neff value for computation of standard error of
10 % the variance associated with MSD values. Neff is the number of
11 % **INDEPENDENT** squared displacement observations associated with a given
12 % tau interval.
13 %
14 % The quantity is defined by Crocker and Hoffman in "Multiple-Particle
15 % Tracking and Two-Point Microrheology in Cells", Methods in Cell Biology,
16 % 2007, Vol 83, P141.
17 %
18 % The essential points to consider are:
19 %
20 % (1) To increase the number of squared displacement observations
21 % associated with a given tau we previously invoked a "moving origin"
22 % strategy in "Differentials.m". Doing so means we introduced
23 % correlations in the data which we must account for in the computation of
24 % the MSD error bars. As a result of this moving origin strategy we cannot
25 % simply set Neff equal to the total number of squared displacement
26 % observations for a given tau which is a number larger than the number of
27 % actual independent measurements. Thus we would artificially and
28 % erroneously decrease the magnitude of our error bars.
29 % (2) The Crocker & Hoffman text defines  $N_{eff} \sim N_{cell} * T / \tau$ . However this
30 % equation is for deal data in which one it is possible to track all Ncells
31 % for the duration of the imaging experiment. In our data Ncells is a
32 % number that generally decreases with time and furthermore the duration of
33 % each track of the set of tracks comprising Ncells are not of equal
34 % length.
35 %
36 % In this code Neff is computed to account for both points (1) and (2)
37 % above.
38 %
39 % ASSUMPTIONS:
40 % n/a
41 %
42 % INPUT:
43 % data = array having following structure:
44 % col 1 = unique track number ID assigned to each cell
45 % col 2 = x coordinate (pixels) of centroid
46 % col 3 = y coordinate (pixels) of centroid
47 % col 4 = absolute time cooresponding to frame in which cell is found
48 % (sec)
49 % col 5 = binned time corresponding to frame in which cell is found (sec)
50 % col 6 = area of cell in pixels
51 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
52 % if no (i.e. continuation of an existing track.
53 % SD = array containing squared displacement differentials and linear time
54 % differentials:
55 % col 1 = track ID to which this interval belongs
56 % col 2 =  $d^2$  (squared displacement (euclidean distance) in pixels2)
57 % Note:  $d^2$  values are not rounded to the nearest whole pixel
58 % col 3 = interval absolute time (sec)
59 % col 4 = interval binned time (sec)
60 % col 5 = dx (pix)
61 % col 6 = dy (pix)
62 % fid = file ID of log file to which progress is recorded
63 %
64 % OUTPUT:
65 % Indep_Obs_tabs = ("Independent Observations") array containing Neff
66 % values corresponding to unbinned tau intervals:
67 % col 1 = tau values using absolute time intervals (sec)
68 % col 2 = Neff values associated with absolute taus
69 % Indep_Obs_tbin = ("Independent Observations") array containing Neff
70 % values corresponding to binned tau intervals:
71 % col 1 = tau values using binned time intervals (sec)
72 % col 2 = Neff values associated with binned taus
73 %*****
74
75 function [Indep_Obs_tabs, Indep_Obs_tbin] = Neff_v1(data, SD, fid)

```

```

76
77 % Get function name:
78 func_name = mfilename;
79
80 % Update log file that function is running:
81 fprintf(1,'\n%s running ...\n',func_name);
82 fprintf(fid,'\n%s running ...\n',func_name);
83
84 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
85 % warning dialog the log file records no errors/warnings generated:
86 warn = 0;
87
88 % Determine indices of start and stop positions of each unique track in the
89 % overall 'data' array:
90 [junk ind_start] = unique(data(:,1),'first');
91 clear junk
92 [junk ind_stop] = unique(data(:,1),'last');
93 clear junk
94
95 % Make sure lengths of start and stop vectors are same:
96 if length(ind_start) ~= length(ind_stop)
97     fprintf(1,'\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
98     fprintf(fid,'\n\tWARNING: # Unique Track IDs ~= # Track Change Flags\n');
99     warn = 1;
100 end
101
102 % Total number of tracks to be analyzed:
103 num_tracks = length(ind_start);
104
105 % Identify all elapsed time intervals that have been observed (using both
106 % absolute and binned time intervals):
107 tau_abs = unique(SD(:,3));
108 tau_abs = sort(tau_abs,'ascend');
109 num_tau_abs = length(tau_abs);
110 tau_bin = unique(SD(:,4));
111 tau_bin = sort(tau_bin,'ascend');
112 num_tau_bin = length(tau_bin);
113
114 % Generate vectors that will hold the Neff counts corresponding to each tau
115 % value. Initially all entries are zero but as each track is processed
116 % sequentially the Neff value is updated (running sum).
117 Indep_Obs_tabs = zeros(num_tau_abs,2);
118 Indep_Obs_tabs(:,1) = tau_abs;
119 Indep_Obs_tbin = zeros(num_tau_bin,2);
120 Indep_Obs_tbin(:,1) = tau_bin;
121
122 % Loop over tracks:
123 for i = 1:num_tracks
124
125     % Define start and stop row positions:
126     r_start = ind_start(i);
127     r_stop = ind_stop(i);
128
129     % Get total time 'T' track 'i' has been imaged for:
130     T_abs = data(r_stop,4) - data(r_start,4); % sec
131     T_bin = data(r_stop,5) - data(r_start,5); % sec
132
133     % Compute number of independent (i.e. non-overlapping) observations
134     % possible for this track with respect to each of the possible tau
135     % values. The result of this operation is a vector. Note we round down
136     % to make sure we only count whole intervals:
137     Neff_abs = floor(T_abs./tau_abs);
138     Neff_bin = floor(T_bin./tau_bin);
139
140     % Update output arrays:
141     Indep_Obs_tabs(:,2) = Indep_Obs_tabs(:,2) + Neff_abs;
142     Indep_Obs_tbin(:,2) = Indep_Obs_tbin(:,2) + Neff_bin;
143 end

```

```

144
145 % Save 'Indep_Obs' arrays in .mat and .txt format:
146 save 'Indep_Obs_tabs.mat' Indep_Obs_tabs;
147 fid2 = fopen('Indep_Obs_tabs.txt','wt');
148 fprintf(fid2,'tau (abs dt) (sec)\tNeff (#)\n');
149 for k = 1:num_tau_abs
150     fprintf(fid2,'%0f\t%0f\n',Indep_Obs_tabs(k,:));
151 end
152 fclose(fid2);
153
154 save 'Indep_Obs_tbin.mat' Indep_Obs_tbin;
155 fid3 = fopen('Indep_Obs_tbin.txt','wt');
156 fprintf(fid3,'tau (bin dt) (sec)\tNeff (#)\n');
157 for k = 1:num_tau_bin
158     fprintf(fid2,'%0f\t%0f\n',Indep_Obs_tbin(k,:));
159 end
160 fclose(fid3);
161
162 % If no warnings generated report so in log file:
163 if warn == 0
164     fprintf(1,'\n\tFunction completed without errors/warnings\n');
165     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
166 end
167
168 % Update log file that function is completed:
169 fprintf(1,'\n%s completed\n',func_name);
170 fprintf(fid,'\n%s completed\n',func_name);
171
172 end

```

Mean_Differentials_v6.m

```

1 % Steven J. Henry
2 % 08/10/2011
81 %*****
82 % PURPOSE:
83 % This function computes mean (time and ensemble averaged) squared
84 % displacements of all cells as a function of lag time (tau) in terms of
85 % both absolute and binned time intervals. Because the squared
86 % displacements were previously computed using a moving origin strategy
87 % (see "Differentials.m" code) the elapsed time values are lag times or
88 % "taus". These time intervals were either in terms of absolute time
89 % differences or binned time differences. The binning strategy is to help
90 % improve the number of samples per mean computation.
91 %
92 % REMARKS:
93 % Assuming the underlying errors in measurement are Gaussian then the SEV
94 % is:
95 %  $SEV = 2 * MSD(\tau) / \sqrt{Neff}$ 
96 % where Neff = number of **INDEPENDENT** observations of displacements
97 % associated with a given imaging time interval tau.
98 %
99 % The SEV form was obtained from discussion with John Crocker on 01/31/2011
100 % and reference to Crocker and Hoffman's "Multiple-Particle Tracking and
101 % Two-Point Microrheology in Cells" Methods in Cell Biology Vol. 83, 2007.
102 %
103 % An extremely practical reference regarding error bar construction and the
104 % source of "inferential" vs. "descriptive" definitions above can be found
105 % in Cumming's "Error bars in experimental biology." The Journal of Cell
106 % Biology Vol. 177, 2007.
107 %
108 %
109 % ASSUMPTIONS:
110 % n/a
111 %
112 % INPUT:
113 % SD = array containing squared displacement differentials and linear time
114 % differentials ("taus"):

```

```

115 % col 1 = track ID to which this interval belongs
116 % col 2 = d^2 (squared displacement (euclidean distance) in pixels^2)
117 %     Note: d^2 values are not rounded to the nearest whole pixel
118 % col 3 = tau absolute time (sec)
119 % col 4 = tau binned time (sec)
120 % col 5 = dx (pix)
121 % col 6 = dy (pix)
122 % Indep_Obs_tabs = ("Independent Observations") array containing Neff
123 % values corresponding to unbinned tau intervals:
124 % col 1 = tau values using absolute time intervals (sec)
125 % col 2 = Neff values associated with absolute taus
126 % Indep_Obs_tbin = ("Independent Observations") array containing Neff
127 % values corresponding to binned tau intervals:
128 % col 1 = tau values using binned time intervals (sec)
129 % col 2 = Neff values associated with binned taus
130 % fid = file ID of log file to which progress is recorded
131 %
132 % OUTPUT:
133 % MSD_tabs = array containing mean squared displacements and corresponding
134 % lag time taus in terms of absolute differences with the following
135 % structure:
136 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
137 % col 2 = obsolete 'NaN'
138 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
139 % col 4 = Neff (number of independent observations)
140 % col 5 = tau absolute elapsed time (sec)
141 % col 6 = VAR(dx) (pix^2)
142 % col 7 = VAR(dy) (pix^2)
143 % MSD_tbin = array containing mean squared displacements and corresponding
144 % lag time taus in terms of binned differences with the following
145 % structure:
146 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
147 % col 2 = obsolete 'NaN'
148 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
149 % col 4 = Neff (number of independent observations)
150 % col 5 = tau binned elapsed time (sec)
151 % col 6 = VAR(dx) (pix^2)
152 % col 7 = VAR(dy) (pix^2)
153 %*****
154
155 function [MSD_tabs, MSD_tbin] = Mean_Differentials_v6(SD, Indep_Obs_tabs, Indep_Obs_tbin, fid)
156
157 % Get function name:
158 func_name = mfilename;
159
160 % Update log file that function is running:
161 fprintf(1, '\n%s running ...\n', func_name);
162 fprintf(fid, '\n%s running ...\n', func_name);
163
164 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
165 % warning dialog the log file records no errors/warnings generated:
166 warn = 0;
167
168 % Identify all elapsed time intervals that have been observed (using both
169 % absolute and binned time intervals):
170 t_abs_uniq = unique(SD(:,3));
171 t_abs_uniq = sort(t_abs_uniq, 'ascend');
172 t_bin_uniq = unique(SD(:,4));
173 t_bin_uniq = sort(t_bin_uniq, 'ascend');
174
175 % Check for consistency with tau values previously computed in "Neff.m"
176 % function:
177 if t_abs_uniq ~= Indep_Obs_tabs(:,1)
178     fprintf(1, '\n\tWARNING: Absolute tau values in %s not same as those passed by "Neff.m"\n', func_name);
179     fprintf(fid, '\n\tWARNING: Absolute tau values in %s not same as those passed by "Neff.m"\n', func_name);
180 end
181 if t_bin_uniq ~= Indep_Obs_tbin(:,1)
182     fprintf(1, '\n\tWARNING: Binned tau values in %s not same as those passed by "Neff.m"\n', func_name);

```

```

183     fprintf(fid, '\n\tWARNING: Binned tau values in %s not same as those passed by "Neff.m"\n', func_name);
184 end
185
186 for i = 1:2
187
188     if i == 1 % Work with lag time taus in absolute differences (sec)
189         t_uniq = t_abs_uniq;
190         t_all = SD(:,3);
191         indep = Indep_Obs_tabs(:,2);
192     elseif i == 2 % Work with lag time taus in binned differences (sec)
193         t_uniq = t_bin_uniq;
194         t_all = SD(:,4);
195         indep = Indep_Obs_tbin(:,2);
196     end
197
198     % Determine number of unique time values:
199     num_uniq = length(t_uniq);
200
201     % Reserve memory for temporary 'MSD'
202     MSD = zeros(num_uniq,7);
203
204     % Loop over all unique elapsed time values:
205     for ii = 1:num_uniq
206
207         % Time interval
208         tau = t_uniq(ii);
209
210         % Find indices in 't_all' that map to positions where entry is
211         % equal to 'tau'
212         ind = t_all==tau;
213
214         % Because 'SD' and 't_all' have the same row structure/organization
215         % use the indices above to extract the corresponding square
216         % displacement values observed at the given 'tau' both within and
217         % across all tracks:
218         SD_tau = SD(ind,2);
219
220         % Compute mean squared displacement for given 'tau'
221         varx = var(SD(ind,5));
222         vary = var(SD(ind,6));
223         covxy = cov(SD(ind,5),SD(ind,6));
224         covxy = covxy(1,2);
225         mean_tau = varx+vary+2*covxy;
226
227         % Compute standard error of the variance for given 'tau'. Recall:
228         % SEV = 2*MSD(tau)/sqrt(Neff)
229         % where Neff is called from Indep_Obs array constructed previously
230         % in "Neff.m" function.
231         Neff = indep(ii);
232         sev_tau = 2*mean_tau/sqrt(Neff);
233
234         % Log MSD(tau), SD, SEV, the number of observations (Ncell) used in
235         % the mean calculation, and tau:
236         MSD(ii,1) = mean_tau; % pixels^2
237         MSD(ii,2) = NaN;
238         MSD(ii,3) = sev_tau; % pixels^2
239         MSD(ii,4) = Neff; % integer value
240         MSD(ii,5) = tau; % sec
241         MSD(ii,6) = varx; % pix^2
242         MSD(ii,7) = vary; % pix^2
243
244     end
245
246     if i == 1
247         MSD_tabs = MSD;
248         clear MSD
249     elseif i == 2
250         MSD_tbin = MSD;

```



```

46 % VAR(X-c) = VAR(X)
47 %
48 % Since std = sqrt(VAR)
49 % std(X-c) = std(X)
50 %
51 % ASSUMPTIONS:
52 % n/a
53 %
54 % INPUT:
55 % MSD_tabs = array containing mean squared displacements and corresponding
56 % lag time taus in terms of absolute differences with the following
57 % structure:
58 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
59 % col 2 = obsolete 'NaN'
60 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
61 % col 4 = Neff (number of independent observations)
62 % col 5 = tau absolute elapsed time (sec)
63 % col 6 = VAR(dx) (pix^2)
64 % col 7 = VAR(dy) (pix^2)
65 % MSD_tbin = array containing mean squared displacements and corresponding
66 % lag time taus in terms of binned differences with the following
67 % structure:
68 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
69 % col 2 = obsolete 'NaN'
70 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
71 % col 4 = Neff (number of independent observations)
72 % col 5 = tau binned elapsed time (sec)
73 % col 6 = VAR(dx) (pix^2)
74 % col 7 = VAR(dy) (pix^2)
75 % epsilon = user specified noise constant (pix)
76 % fid = file ID of log file to which progress is recorded
77 %
78 % OUTPUT:
79 % MSD_tabs_epsilon = same structure as input array except with epsilon
80 % subtracted from all MSD values
81 % MSD_tbin_epsilon = same structure as input array except with epsilon
82 % subtracted from all MSD values
83 %*****
84
85 function [MSD_tabs_epsilon MSD_tbin_epsilon] = MSD_Epsilon_Subtract_v3(MSD_tabs, MSD_tbin, epsilon, fid)
86
87 % Get function name:
88 func_name = mfilename;
89
90 % Update log file that function is running:
91 fprintf(1, '\n%s running ...\n', func_name);
92 fprintf(fid, '\n%s running ...\n', func_name);
93
94 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
95 % warning dialog the log file records no errors/warnings generated:
96 warn = 0;
97
98 % Subtract 4*epsilon^2 from all MSD values in column 1:
99 MSD_tabs_epsilon = MSD_tabs;
100 MSD_tabs_epsilon(:,1) = MSD_tabs_epsilon(:,1)-4*epsilon^2;
101 MSD_tbin_epsilon = MSD_tbin;
102 MSD_tbin_epsilon(:,1) = MSD_tbin_epsilon(:,1)-4*epsilon^2;
103
104 % Save epsilon corrected 'MSD' arrays in .mat and .txt format:
105 save 'MSD_tabs_epsilon.mat' MSD_tabs_epsilon
106 fid2_1 = fopen('MSD_tabs_epsilon.txt', 'wt');
107 fprintf(fid2_1, 'MSD-4*epsilon^2 (pix^2)\tNaN\t.e.v. (pix^2)\tNeff (#)\tAbs tau (sec)\tVAR(dx) (pix^2)\tVAR(dy)
(pix^2)\n');
108 rows = length(MSD_tabs_epsilon);
109 for k = 1:rows
110     fprintf(fid2_1, '%f\t%f\t%f\t%.0f\t%.0f\t%.0f\t%.0f\n', MSD_tabs_epsilon(k,:));
111 end
112 fclose(fid2_1);

```

```

113
114 save 'MSD_tbin_epsilon.mat' MSD_tbin_epsilon
115 fid2_2 = fopen('MSD_tbin_epsilon.txt','wt');
116 fprintf(fid2_2,'MSD-4*epsilon^2 (pix^2)\tNaN\ts.e.v. (pix^2)\tNeff (#)\tBin tau (sec)\tVAR(dx) (pix^2)\tVAR(dy)
(pix^2)\n');
117 rows = length(MSD_tbin_epsilon);
118 for k = 1:rows
119     fprintf(fid2_2,'%ft%f\t%.0ft%.0ft%.0ft%.0f\n',MSD_tbin_epsilon(k,:));
120 end
121 fclose(fid2_2);
122
123 % If no warnings generated report so in log file:
124 if warn == 0
125     fprintf(1,'\n\tFunction completed without errors/warnings\n');
126     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
127 end
128
129 % Update log file that function is completed:
130 fprintf(1,'\n%s completed\n',func_name);
131 fprintf(fid,'\n%s completed\n',func_name);
132
133 end

```

Plot_Mean_Differentials_v5.m

```

1 % Steven J. Henry
2 % 08/11/2012
27 %*****
28 % PURPOSE:
29 % This function plots mean squared displacements of all cells as a function
30 % of lag time (tau) in terms of both absolute and binned time intervals.
31 % The elapsed time values are lag times or "taus". These time intervals
32 % were either in terms of absolute time differences or binned time
33 % differences. The binning strategy is to help improve the number of
34 % samples per mean computation.
35 %
36 % REMARKS:
37 % Placing all plotting associated with the MSD arrays into a separate
38 % function allows easier plotting manipulation without having to
39 % reconstruct the MSD arrays each time.
40 %
41 % ASSUMPTIONS:
42 % n/a
43 %
44 % INPUT:
45 % SD = array containing squared displacement differentials and linear time
46 % differentials ("taus"):
47 % col 1 = track ID to which this interval belongs
48 % col 2 = d^2 (squared displacement (euclidean distance) in pixels^2)
49 % Note: d^2 values are not rounded to the nearest whole pixel
50 % col 3 = tau absolute time (sec)
51 % col 4 = tau binned time (sec)
52 % col 5 = dx (pix)
53 % col 6 = dy (pix)
54 % MSD_tabs = array containing mean squared displacements and corresponding
55 % lag time taus in terms of absolute differences with the following
56 % structure:
57 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
58 % col 2 = obsolete 'NaN'
59 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
60 % col 4 = Neff (number of independent observations)
61 % col 5 = tau absolute elapsed time (sec)
62 % col 6 = VAR(dx) (pix^2)
63 % col 7 = VAR(dy) (pix^2)
64 % MSD_tbin = array containing mean squared displacements and corresponding
65 % lag time taus in terms of binned differences with the following

```

```

66 % structure:
67 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
68 % col 2 = obsolete 'NaN'
69 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
70 % col 4 = Neff (number of independent observations)
71 % col 5 = tau binned elapsed time (sec)
72 % col 6 = VAR(dx) (pix^2)
73 % col 7 = VAR(dy) (pix^2)
74 % pixel_calib = user-supplied microns/pixel
75 % t_max = maximum experimental imaging time for which empirical data is
76 % being used to compute MSD values.
77 % run_title = user specified string description of experimental condition
78 % epsilon_flag = 0 if MSD data being filtered is not corrected for
79 % random noise or 1 if MSD data being filtered is corrected for random
80 % noise
81 % fid = file ID of log file to which progress is recorded
82
83 % OUTPUT:
84 % n/a
85 %*****
86
87 function [] = Plot_Mean_Differentials_v5(SD, MSD_tabs, MSD_tbin, pixel_calib, t_max, run_title, epsilon_flag, fid)
88
89 % Get function name:
90 func_name = mfilename;
91
92 % Update log file that function is running:
93 fprintf(1, '\n%s running ...\n', func_name);
94 fprintf(fid, '\n%s running ...\n', func_name);
95
96 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
97 % warning dialog the log file records no errors/warnings generated:
98 warn = 0;
99
100 %*****
101 % Begin plot that overlays MSD values on SD of each track:
102
103 % Determine indices of start and stop positions of each unique track in the
104 % 'SD' array:
105 [junk ind_start] = unique(SD(:,1), 'first');
106 clear junk
107 [junk ind_stop] = unique(SD(:,1), 'last');
108 clear junk
109
110 % Make sure lengths of start and stop vectors are same:
111 if length(ind_start) ~= length(ind_stop)
112     fprintf(1, '\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
113     fprintf(fid, '\n\tWARNING: # Unique Track IDs start positions ~= # stop positions\n');
114     warn = 1;
115 end
116
117 % Total number of tracks to be analyzed:
118 num_tracks = length(ind_start);
119
120 % Plot MSD overlay on squared displacement figure:
121 h1 = figure;
122
123 for j = 1:num_tracks
124
125     % Define start and stop position:
126     r_start = ind_start(j);
127     r_stop = ind_stop(j);
128
129     % Create vectors of track data:
130     d2 = SD(r_start:r_stop, 2);
131     t_abs = SD(r_start:r_stop, 3);
132     t_bin = SD(r_start:r_stop, 4);
133

```

```

134 % Convert squared displacement (in pixels^2) to microns^2. Note:
135 % 'pixel_calib' has units micron/pixel:
136 d2 = d2*pixel_calib^2;
137
138 % Convert time vectors to minutes from seconds:
139 t_abs = t_abs/60; % min
140 t_bin = t_bin/60; % min
141
142 % Plot squared displacement vs. elapsed time (absolute)
143 % Units are microns^2 and minutes
144 h1_sub(1) = subplot(1,2,1);
145 hold on
146 plot(t_abs,d2,'LineStyle','none','Marker','.', 'LineWidth',1, 'Color','b', 'HandleVisibility','off');
147
148
149 % Plot squared displacement vs. elapsed time (binned)
150 % Units are microns^2 and minutes
151 h1_sub(2) = subplot(1,2,2);
152 hold on
153 plot(t_bin,d2,'LineStyle','none','Marker','.', 'LineWidth',1, 'Color','b', 'HandleVisibility','off');
154
155 end
156
157 % Set ordinate upperbound as maximum squared displacement value:
158 d2_max_h1 = max(SD(:,2))*pixel_calib^2; % microns^2
159
160 % Prepare the mean data for plotting:
161 t_abs = MSD_tabs(:,5)/60; % min
162 m_abs = MSD_tabs(:,1)*pixel_calib^2; % microns^2
163 sev_abs = MSD_tabs(:,3)*pixel_calib^2; % microns^2
164
165 t_bin = MSD_tbin(:,5)/60; % min
166 m_bin = MSD_tbin(:,1)*pixel_calib^2; % microns^2
167 sev_bin = MSD_tbin(:,3)*pixel_calib^2; % microns^2
168
169 % Overlay mean series:
170 subplot(h1_sub(1));
171
172 plot(t_abs,m_abs,'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
173 h1_legend_handle = legend('<math>\tau^2</math>','Location','NorthWest');
174 set(h1_legend_handle,'FontName','Arial','FontSize',12);
175
176 subplot(h1_sub(2));
177
178 plot(t_bin,m_bin,'LineStyle','none','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5,'HandleVisibility','on');
179 h1_legend_handle = legend('<math>\tau^2</math>','Location','NorthWest');
180 set(h1_legend_handle,'FontName','Arial','FontSize',12);
181
182 % Set axes properties:
183 set(h1_sub,'ylim',[0 d2_max_h1]);
184 set(h1_sub,'xlim',[0 t_max]);
185 set(h1_sub,'FontName','Arial');
186 set(h1_sub,'FontSize',14);
187
188 h1_x_axis_handles = cell2mat(get(h1_sub,'xlabel'));
189 set(h1_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
190 set(h1_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);
191
192 h1_y_axis_handles = cell2mat(get(h1_sub,'ylabel'));
193 if epsilon_flag == 0
194     ord_label = '<math>\tau^2</math> (\mu\text{m}^2)';
195 elseif epsilon_flag == 1
196     ord_label = '<math>\tau^2 - 4\epsilon^2</math> (\mu\text{m}^2)';
197 end
198 set(h1_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);
199

```

```

198 h1_title_handles = cell2mat(get(h1_sub,'title'));
199 set(h1_title_handles,'String',run_title,'FontName','Arial','FontSize',16);
200
201 % Save figure window generated:
202 saveas(h1, 'MSD_Overlay.fig', 'fig');
203 %*****
204
205
206 %*****
207 % Plot mean +/- s.e.v. (linear-linear axes)
208
209 % Set ordinate upperbound as maximum mean squared displacement value:
210 temp = zeros(1,2);
211 temp(1,1) = max(MSD_tabs(:,1)+MSD_tabs(:,3));
212 temp(1,2) = max(MSD_tbin(:,1)+MSD_tbin(:,3));
213 d2_max_h4_h5 = max(temp)*pixel_calib^2; % microns^2
214
215 % Plot data:
216 h4 = figure;
217 h4_sub(1) = subplot(1,2,1);
218
errorbar(t_abs,m_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
219 hold on
220
221 h4_sub(2) = subplot(1,2,2);
222
errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
223 hold on
224
225 % Set axes properties:
226 set(h4_sub,'ylim',[0 d2_max_h4_h5]);
227 set(h4_sub,'xlim',[0 t_max]);
228 set(h4_sub,'FontName','Arial');
229 set(h4_sub,'FontSize',14);
230
231 h4_x_axis_handles = cell2mat(get(h4_sub,'xlabel'));
232 set(h4_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
233 set(h4_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);
234
235 h4_y_axis_handles = cell2mat(get(h4_sub,'ylabel'));
236 if epsilon_flag == 0
237     ord_label = '<math>\mu\text{m s.e.v.} (\mu\text{m}^2)</math>';
238 elseif epsilon_flag == 1
239     ord_label = '<math>\epsilon^2 - 4\epsilon^2 \mu\text{m s.e.v.} (\mu\text{m}^2)</math>';
240 end
241 set(h4_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);
242
243 h4_title_handles = cell2mat(get(h4_sub,'title'));
244 set(h4_title_handles,'String',run_title,'FontName','Arial','FontSize',16);
245
246 % Save figure window generated:
247 saveas(h4, 'MSD_sev.fig', 'fig');
248 %*****
249
250 %*****
251 % Plot mean +/- s.e.v. (log-log axes)
252
253 % On log-log axes negative values result in output warnings to user. To
254 % avoid this filter for msd - s.d. (lower bounds) that result in negative
255 % values.
256 L_sev_abs = sev_abs;
257 for i = 1:length(L_sev_abs)
258     if m_abs(i)-L_sev_abs(i) < 0
259         L_sev_abs(i) = 0;
260     end
261 end

```

```

262 L_sev_bin = sev_bin;
263 for i = 1:length(L_sev_bin)
264     if m_bin(i)-L_sev_bin(i) < 0
265         L_sev_bin(i) = 0;
266     end
267 end
268
269 % Plot data:
270 h5 = figure;
271 h5_sub(1) = subplot(1,2,1);
272
errorbar(t_abs,m_abs,L_sev_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor',
'k','MarkerSize',5);
273 hold on
274
275 h5_sub(2) = subplot(1,2,2);
276
errorbar(t_bin,m_bin,L_sev_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k',
'MarkerSize',5);
277 hold on
278
279 % Set axes properties:
280 set(h5_sub,'ylim',[1 d2_max_h4_h5]);
281 set(h5_sub,'xlim',[1 t_max]);
282 set(h5_sub,'yscale','log');
283 set(h5_sub,'xscale','log');
284 set(h5_sub,'YMinorTick','on');
285 set(h5_sub,'XMinorTick','on');
286 set(h5_sub,'FontName','Arial');
287 set(h5_sub,'FontSize',14);
288
289 h5_x_axis_handles = cell2mat(get(h5_sub,'xlabel'));
290 set(h5_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
291 set(h5_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);
292
293 h5_y_axis_handles = cell2mat(get(h5_sub,'ylabel'));
294 if epsilon_flag == 0
295     ord_label = '<math>\rho</math> s.e.v. (

```

```

324
325 set(h6_plot,'ylim',[1 axis_lim]);
326 set(h6_plot,'xlim',[1 axis_lim]);
327 set(h6_plot,'yscale','log');
328 set(h6_plot,'xscale','log');
329 set(h6_plot,'DataAspectRatio',[1 1 1]);
330 set(h6_plot,'YMinorTick','on');
331 set(h6_plot,'XMinorTick','on');
332 set(h6_plot,'FontName','Arial');
333 set(h6_plot,'FontSize',14);
334 set(h6_plot,'box','on');
335
336 h6_plot_x_axis_handle = xlabel('\tau (min)');
337 set(h6_plot_x_axis_handle,'FontName','Arial','FontSize',16);
338
339 if epsilon_flag == 0
340     ord_label = '<math>\rho \text{ s.e.v. } (\mu\text{m}^2)</math>';
341 elseif epsilon_flag == 1
342     ord_label = '<math>\rho - 4 \epsilon^2 \text{ s.e.v. } (\mu\text{m}^2)</math>';
343 end
344 h6_plot_y_axis_handle = ylabel(ord_label);
345 set(h6_plot_y_axis_handle,'FontName','Arial','FontSize',16);
346
347 h6_plot_title_handle = title(run_title);
348 set(h6_plot_title_handle,'FontName','Arial','FontSize',16);
349
350 % Save figure window generated:
351 saveas(h6, 'MSD_sev_loglog_final.fig', 'fig');
352 %*****
353
354 % If no warnings generated report so in log file:
355 if warn == 0
356     fprintf(1, '\n\tFunction completed without errors/warnings\n');
357     fprintf(fid, '\n\tFunction completed without errors/warnings\n');
358 end
359
360 % Update log file that function is completed:
361 fprintf(1, '\n%s completed\n', func_name);
362 fprintf(fid, '\n%s completed\n', func_name);
363
364 end

```

Filter_Mean_Differentials_v4.m

```

1 % Steven J. Henry
2 % 08/11/2012
24 %*****
25 % PURPOSE:
26 % This function filters a supplied array of mean squared displacement and
27 % elapsed time values (taus) and produces an array with the same structure
28 % but only containing data corresponding to tau values between and
29 % including user specified bounds.
30 %
31 % REMARKS:
32 % n/a
33 %
34 % ASSUMPTIONS:
35 % n/a
36 %
37 % INPUT:
38 % MSD_tabs_full = array containing mean squared displacements and corresponding
39 % lag time taus in terms of absolute differences with the following
40 % structure:
41 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
42 % col 2 = obsolete 'NaN'
43 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
44 % col 4 = Neff (number of independent observations)
45 % col 5 = tau absolute elapsed time (sec)

```



```

46 % col 6 = VAR(dx) (pix^2)
47 % col 7 = VAR(dy) (pix^2)
48 % MSD_tbin_full = array containing mean squared displacements and corresponding
49 % lag time taus in terms of binned differences with the following
50 % structure:
51 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
52 % col 2 = obsolete 'NaN'
53 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
54 % col 4 = Neff (number of independent observations)
55 % col 5 = tau binned elapsed time (sec)
56 % col 6 = VAR(dx) (pix^2)
57 % col 7 = VAR(dy) (pix^2)
58 % fit_tau_min = user supplied minimum tau interval in (min)
59 % fit_tau_max = user-supplied maximum tau interval in (min)
60 % epsilon_flag = 0 if MSD data being filtered is not corrected for
61 % random noise or 1 if MSD data being filtered is corrected for random
62 % noise
63 % fid = file ID of log file to which progress is recorded
64 %
65 % OUTPUT:
66 % MSD_tabs_part = array containing mean squared displacements and
67 % corresponding lag time taus for a given experimental condition for those
68 % taus that are less than or equal to the user-specified 'tau_max'. Taus
69 % are in terms of absolute differences. The array has the same structure as
70 % before.
71 % MSD_tbin_part = array containing mean squared displacements and
72 % corresponding lag time taus for a given experimental condition for those
73 % taus that are less than or equal to the user-specified 'tau_max'. Taus
74 % are in terms of binned differences. The array has the same structure as
75 % before.
76 %*****
77
78 function [MSD_tabs_part MSD_tbin_part] = Filter_Mean_Differentials_v4(MSD_tabs_full, MSD_tbin_full, fit_tau_min,
fit_tau_max, epsilon_flag, fid)
79
80 % Get function name:
81 func_name = mfilename;
82
83 % Update log file that function is running:
84 fprintf(1, '\n%s running ...\n', func_name);
85 fprintf(fid, '\n%s running ...\n', func_name);
86
87 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
88 % warning dialog the log file records no errors/warnings generated:
89 warn = 0;
90
91 % Convert tau boundaries to sec from min:
92 fit_tau_min = fit_tau_min*60; % sec
93 fit_tau_max = fit_tau_max*60; % sec
94
95 % Identify row position of all elapsed time intervals that pass the filter
96 % 'tau_max' (i.e. that are less than or equal to 'tau_max'):
97 t_abs_pass_ind = fit_tau_min <= MSD_tabs_full(:,5) & MSD_tabs_full(:,5) <= fit_tau_max;
98 t_bin_pass_ind = fit_tau_min <= MSD_tbin_full(:,5) & MSD_tbin_full(:,5) <= fit_tau_max;
99
100 MSD_tabs_part = MSD_tabs_full(t_abs_pass_ind,:);
101 MSD_tbin_part = MSD_tbin_full(t_bin_pass_ind,:);
102
103 % Double check manipulations:
104 if any(MSD_tabs_part(:,5) < fit_tau_min) == 1
105     fprintf(1, '\n\tWARNING: tau entry in "MSD_tabs_part" is less than "fit_tau_min" after filtering.\n')
106     fprintf(fid, '\n\tWARNING: tau entry in "MSD_tabs_part" is less than "fit_tau_min" after filtering.\n')
107     warn = 1;
108     keyboard
109 end
110 if any(MSD_tabs_part(:,5) > fit_tau_max) == 1
111     fprintf(1, '\n\tWARNING: tau entry in "MSD_tabs_part" is greater than "fit_tau_max" after filtering.\n')
112     fprintf(fid, '\n\tWARNING: tau entry in "MSD_tabs_part" is greater than "fit_tau_max" after filtering.\n')

```

```

113     warn = 1;
114     keyboard
115 end
116 if any(MSD_tbin_part(:,5) < fit_tau_min) == 1
117     fprintf(1,'\n\tWARNING: tau entry in "MSD_tbin_part" is less than "fit_tau_min" after filtering.\n')
118     fprintf(fid,'\n\tWARNING: tau entry in "MSD_tbin_part" is less than "fit_tau_min" after filtering.\n')
119     warn = 1;
120     keyboard
121 end
122 if any(MSD_tbin_part(:,5) > fit_tau_max) == 1
123     fprintf(1,'\n\tWARNING: tau entry in "MSD_tbin_part" is greater than "fit_tau_max" after filter.\n')
124     fprintf(fid,'\n\tWARNING: tau entry in "MSD_tbin_part" is greater than "fit_tau_max" after filter.\n')
125     warn = 1;
126     keyboard
127 end
128 if length(t_abs_pass_ind)~=length(MSD_tabs_part(:,1))
129     fprintf(1,'\n\tWARNING: length(t_abs_pass_ind) ~= length(MSD_tabs_part)\n')
130     fprintf(fid,'\n\tWARNING: length(t_abs_pass_ind) ~= length(MSD_tabs_part)\n')
131     warn = 1;
132     keyboard
133 end
134 if length(t_bin_pass_ind)~=length(MSD_tbin_part(:,1))
135     fprintf(1,'\n\tWARNING: length(t_bin_pass_ind) ~= length(MSD_tbin_part)\n')
136     fprintf(fid,'\n\tWARNING: length(t_bin_pass_ind) ~= length(MSD_tbin_part)\n')
137     warn = 1;
138     keyboard
139 end
140
141 % Save filtered 'MSD' arrays in .mat and .txt format:
142 if epsilon_flag == 0
143     MSD_tabs_header = 'MSD(tau) (pix^2)\ts.d. (pix^2)\ts.e.v. (pix^2)\tNeff (#)\tAbs tau (sec)\tVAR(dx)
(pix^2)\tVAR(dy) (pix^2)\n';
144     MSD_tbin_header = 'MSD(tau) (pix^2)\ts.d. (pix^2)\ts.e.v. (pix^2)\tNeff (#)\tBin tau (sec)\tVAR(dx)
(pix^2)\tVAR(dy) (pix^2)\n';
145
146 elseif epsilon_flag == 1
147     MSD_tabs_header = 'MSD(tau)-4*epsilon^2 (pix^2)\ts.d. (pix^2)\ts.e.v. (pix^2)\tNeff (#)\tAbs tau (sec)\tVAR(dx)
(pix^2)\tVAR(dy) (pix^2)\n';
148     MSD_tbin_header = 'MSD(tau)-4*epsilon^2 (pix^2)\ts.d. (pix^2)\ts.e.v. (pix^2)\tNeff (#)\tBin tau (sec)\tVAR(dx)
(pix^2)\tVAR(dy) (pix^2)\n';
149 end
150
151 save 'MSD_tabs_filtered.mat' MSD_tabs_part
152 fid2_1 = fopen('MSD_tabs_filtered.txt','wt');
153 fprintf(fid2_1,MSD_tabs_header);
154 rows = length(MSD_tabs_part);
155 for k = 1:rows
156     fprintf(fid2_1,'%ft%ft%ft%.0ft%.0ft%.0ft%.0ft\n',MSD_tabs_part(k,:));
157 end
158 fclose(fid2_1);
159
160 save 'MSD_tbin_filtered.mat' MSD_tbin_part
161 fid2_2 = fopen('MSD_tbin_filtered.txt','wt');
162 fprintf(fid2_2,MSD_tbin_header);
163 rows = length(MSD_tbin_part);
164 for k = 1:rows
165     fprintf(fid2_2,'%ft%ft%ft%.0ft%.0ft%.0ft%.0ft\n',MSD_tbin_part(k,:));
166 end
167 fclose(fid2_2);
168
169 % If no warnings generated report so in log file:
170 if warn == 0
171     fprintf(1,'\n\tFunction completed without errors/warnings\n');
172     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
173 end
174
175 % Update log file that function is completed:
176 fprintf(1,'\n%s completed\n',func_name);

```

```

177 fprintf(fid, '\n%s completed\n', func_name);
178
179 end

```

SandP_v11.m

```

1 % Steven J. Henry
2 % 03/03/2015
56 %*****
57 % PURPOSE:
58 % The following function calculates root-mean-square speed 'S' and
59 % persistence time 'P' from a nonlinear curve fitting algorithm performed
60 % on <d^2> vs. time interval data using the Lauffenburger persistent random
61 % walk model (6-35a) on p.312 of "Receptors: Models for binding,
62 % trafficking, and signaling" 1993 Oxford University Press.
63 %
64 % REMARKS:
65 % n/a
66 %
67 % ASSUMPTIONS:
68 % Model is appropriate for mode of migration being analyzed.
69 %
70 % INPUT:
71 % MSD_tabs = array containing mean squared displacements and corresponding
72 % lag time taus in terms of absolute differences with the following
73 % structure:
74 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
75 % col 2 = obsolete 'NaN'
76 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
77 % col 4 = Neff (number of independent observations)
78 % col 5 = tau absolute elapsed time (sec)
79 % col 6 = VAR(dx) (pix^2)
80 % col 7 = VAR(dy) (pix^2)
81 % MSD_tbin = array containing mean squared displacements and corresponding
82 % lag time taus in terms of binned differences with the following
83 % structure:
84 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
85 % col 2 = obsolete 'NaN'
86 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
87 % col 4 = Neff (number of independent observations)
88 % col 5 = tau binned elapsed time (sec)
89 % col 6 = VAR(dx) (pix^2)
90 % col 7 = VAR(dy) (pix^2)
91 % pixel_calib = user-supplied microns/pixel
92 % fid = file ID of log file to which progress is recorded
93 %
94 % OUTPUT:
95 % fit_BRW_tabs = (BRW = biased random walk) array with length =
96 % length(MSD_tabs) containing fit data generated using best-fit S and P
97 % parameters having structure:
98 % col 1 = fit mean squared displacement (pixels^2)
99 % col 2 = tau absolute elapsed time (sec)
100 % fit_BRW_tbin = (BRW = biased random walk) array with length =
101 % length(MSD_tabs) containing fit data generated using best-fit S and P
102 % parameters having structure:
103 % col 1 = fit mean squared displacement (pixels^2)
104 % col 2 = tau binned elapsed time (sec)
105 % Sout = best fit speed value (um/min)
106 % Pout = best fit persistence value (min)
107 % muout = random motility coefficient from best-fit values =
108 % 0.5*Sout^2*Pout (um^2/min)
109 %*****
110
111 function [fit_BRW_tabs, fit_BRW_tbin, Sout, Pout, muout] = SandP_v11(MSD_tabs, MSD_tbin, pixel_calib, fid)
112
113 % Get function name:
114 func_name = mfilename;
115

```

```

116 % Update log file that function is running:
117 fprintf(1,'\n%s running ...\n',func_name);
118 fprintf(fid,'\n%s running ...\n',func_name);
119
120 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
121 % warning dialog the log file records no errors/warnings generated:
122 warn = 0;
123
124 for i = 1:2
125
126     if i == 1
127         MSD = MSD_tabs;
128     elseif i == 2
129         MSD = MSD_tbin;
130     end
131
132     % Generate an initial guess for the root mean squared speed in
133     % pixels/sec. Use as the guess the instantaneous speed corresponding to
134     % the <d^2> value that has the most number of samples ("Neff")
135     % contributing to its calculation (i.e. select the MSD value which you
136     % have the most confidence in):
137     [max_Neff, ind_max_Neff] = max(MSD(:,4));
138     d2 = MSD(ind_max_Neff,1); % pix^2
139     tau = MSD(ind_max_Neff,5); % sec
140     d = sqrt(d2); % pix
141     S0 = d/tau; % pixel/sec
142
143     % Generate an initial guess for persistence time in sec from the
144     % long-time approximation that:
145     % <d^2> = 2*S^2*P*tau
146     % Rearranging:
147     % P = <d^2>/(2*S^2*tau) using S = S0
148     [max_tau, ind_max_tau] = max(MSD(:,5));
149     long_d2 = MSD(ind_max_tau,1); % pix^2
150     long_tau = MSD(ind_max_tau,5); % sec
151     if long_tau ~= max_tau
152         fprintf(1,'\n\tWARNING: index reported for "max_tau" does not correspond to "max_tau" in MSD array\n');
153         fprintf(fid,'\n\tWARNING: index reported for "max_tau" does not correspond to "max_tau" in MSD array\n');
154     end
155     P0 = long_d2/(2*S0^2*long_tau); % sec
156
157     if P0 < 0
158         P0 = tau; % sec
159         fprintf(1,'\n\t5 WARNING: P0 < 0 so set P0 = %s (sec)\n',num2str(P0));
160         fprintf(fid,'\n\t5 WARNING: P0 < 0 so set P0 = %s (sec)\n',num2str(P0));
161     end
162
163     if P0 > long_tau
164         P0 = long_tau; % sec
165         fprintf(1,'\n\t5 WARNING: P0 > max tau so set P0 = %s (sec)\n',num2str(P0));
166         fprintf(fid,'\n\t5 WARNING: P0 > max tau so set P0 = %s (sec)\n',num2str(P0));
167     end
168
169     %*****
170     % Use 'lsqcurvefit' function to determine S and P. Create a row vector
171     % to hold the initial guesses at the fit parameters S and P:
172     para0 = [S0,P0];
173
174     % Define empty matrices of lower and upper bounds. Utilization of
175     % Levenberg-Marquardt algorithm does not accommodate boundary constraints
176     % on solver (see discussion below). Thus we pass empty boundaries to
177     % the solver algorithm.
178     lb = [];
179     ub = [];
180
181     % Define 'options'. Note: the Levenberg-Marquardt algorithm was
182     % selected for two reasons. First it was the default algorithm being
183     % used by a grandfathered code from Brendon Ricart in 2010. Second

```

```

184 % discussion of the algorithm in the Help section of the Optimization
185 % Toolbox titled "Least Squares (Model Fitting)" turned up that this
186 % algorithm is not the most efficient but is robust especially when the
187 % solution has a nonzero residual which I anticipate to be the case for
188 % my empirical data:
189 %
190 % "The poorer efficiency is partly because the Gauss-Newton method is
191 % generally more effective when the residual is zero at the solution.
192 % However, such information is not always available beforehand, and the
193 % increased robustness of the Levenberg-Marquardt method compensates
194 % for its occasional poorer efficiency."
195 %
196 options = optimset('MaxFunEvals', 2000, 'Algorithm','levenberg-marquardt');
197
198 % Generate an "anonymous" function that contains the Biased Random Walk
199 % model. Use as the weights vector the Neff values corresponding to
200 % each data point.
201 weights = MSD(:,4);
202 % weights = ones(size(MSD(:,4))); % use this for an unweighted fit
203 BRW_fun = @(para,t) weights.*(2*para(1)^2*(para(2)*t-para(2)^2*(1-exp(-t./para(2)))));
204 [para, resnorm, residual, exitflag] = lsqcurvefit(BRW_fun, para0, MSD(:,5), MSD(:,1).*weights, lb, ub, options);
205
206 % Fit root mean squared speed (pixels/sec) and persistence time (sec):
207 S = para(1);
208 P = para(2);
209
210 % Send S and P out of function with physical units:
211 Sout = S*pixel_calib*60;
212 Pout = P/60;
213
214 % Compute random motility coefficient:
215 mu = 0.5*(S)^2*P;
216 muout = 0.5*(Sout)^2*Pout;
217
218 % Reserve memory for an array that will hold the theoretical MSD values
219 % using returned S and P values and corresponding tau:
220 rows = length(MSD(:,1));
221 F = zeros(rows,2);
222
223 for j = 1:rows
224     tau = MSD(j,5);
225     F(j,1) = 2*S^2*(P*tau-P^2*(1-exp(-tau/P)));
226     F(j,2) = tau;
227 end
228
229 if i == 1
230     fit_tabs = F;
231     tau_type = 'ABSOLUTE';
232 elseif i == 2
233     fit_tbin = F;
234     tau_type = 'BINNED';
235 end
236
237 % Update log file with progress:
238 fprintf(1, '\n\tProcessing %s tau MSD data:\n', tau_type);
239 fprintf(1, '\t*****\n');
240 fprintf(1, '\tS0 guess supplied = %0.4f pixels/sec = %0.4f um/min\n', S0, S0*pixel_calib*60);
241 fprintf(1, '\tP0 guess supplied = %0.4f sec = %0.4f min\n', P0, P0/60);
242 fprintf(1, '\t*****\n');
243 fprintf(1, '\tsquared 2-norm of the residual (resnorm) at solution = %0.4E\n', resnorm);
244 fprintf(1, '\texit flag = %0.0f\n', exitflag);
245 fprintf(1, '\t*****\n');
246 fprintf(1, '\tS weighted fit returned = %0.4f pixels/sec = %0.4f um/min\n', S, Sout);
247 fprintf(1, '\tP weighted fit returned = %0.4f sec = %0.4f min\n', P, Pout);
248 fprintf(1, '\t*****\n');
249 fprintf(1, '\tRandom motility coefficient (mu) = %0.4f pixels^2/sec = %0.4f um^2/min\n', mu, muout);
250 fprintf(1, '\t*****\n');
251 fprintf(fid, '\n\tProcessing %s tau MSD data:\n', tau_type);

```

```

252 fprintf(fid, '\t*****\n');
253 fprintf(fid, '\tS0 guess supplied = %0.4f pixels/sec = %0.4f um/min\n', S0, S0*pixel_calib*60);
254 fprintf(fid, '\tP0 guess supplied = %0.4f sec = %0.4f min\n', P0, P0/60);
255 fprintf(fid, '\t*****\n');
256 fprintf(fid, '\tsquared 2-norm of the residual (resnorm) at solution = %.4E\n', resnorm);
257 fprintf(fid, '\texit flag = %.0f\n', exitflag);
258 fprintf(fid, '\t*****\n');
259 fprintf(fid, '\tS weighted fit returned = %0.4f pixels/sec = %0.4f um/min\n', S, Sout);
260 fprintf(fid, '\tP weighted fit returned = %0.4f sec = %0.4f min\n', P, Pout);
261 fprintf(fid, '\t*****\n');
262 fprintf(fid, '\tRandom motility coefficient (mu) = %0.4f pixels^2/sec = %0.4f um^2/min\n', mu, muout);
263 fprintf(fid, '\t*****\n');
264
265 end
266
267 % Ensure number of data entries in 'fit' arrays is consistent with MSD:
268 if length(fit_tabs(:,1))~=length(MSD_tabs(:,1))
269     fprintf(1, '\tWARNING: length(fit_tabs) ~= length(MSD_tabs)\n')
270     fprintf(fid, '\tWARNING: length(fit_tabs) ~= length(MSD_tabs)\n')
271     warn = 1;
272 end
273 if length(fit_tbin(:,1))~=length(MSD_tbin(:,1))
274     fprintf(1, '\tWARNING: length(fit_tbin) ~= length(MSD_tbin)\n')
275     fprintf(fid, '\tWARNING: length(fit_tbin) ~= length(MSD_tbin)\n')
276     warn = 1;
277 end
278
279 % Rename fit_tabs and fit_tbin to note that they are from the biased random
280 % walk model (BRW):
281 fit_BRW_tabs = fit_tabs;
282 fit_BRW_tbin = fit_tbin;
283
284 % Save 'fit' arrays in .mat and .txt format:
285 save 'fit_BRW_tabs.mat' fit_BRW_tabs
286 fid2_1 = fopen('fit_BRW_tabs.txt', 'wt');
287 fprintf(fid2_1, 'Weighted Fit MSD (pix^2)\tAbs. tau (sec)\n');
288 rows = length(fit_BRW_tabs);
289 for k = 1:rows
290     fprintf(fid2_1, '%f\t%.0f\n', fit_BRW_tabs(k,:));
291 end
292 fclose(fid2_1);
293
294 save 'fit_BRW_tbin.mat' fit_BRW_tbin
295 fid2_2 = fopen('fit_BRW_tbin.txt', 'wt');
296 fprintf(fid2_2, 'Weighted Fit MSD (pix^2)\tBin. tau (sec)\n');
297 rows = length(fit_BRW_tbin);
298 for k = 1:rows
299     fprintf(fid2_2, '%f\t%.0f\n', fit_BRW_tbin(k,:));
300 end
301 fclose(fid2_2);
302
303 % If no warnings generated report so in log file:
304 if warn == 0
305     fprintf(1, '\n\tFunction completed without errors/warnings\n');
306     fprintf(fid, '\n\tFunction completed without errors/warnings\n');
307 end
308
309 % Update log file that function is completed:
310 fprintf(1, '\n%s completed\n', func_name);
311 fprintf(fid, '\n%s completed\n', func_name);
312
313 end

```

Power_Law_v4.m

```

1 % Steven J. Henry
2 % 11/03/2011
23 %*****

```

```

24 % PURPOSE:
25 % The following function applies a power-law fit to MSD vs. tau data. It
26 % solves for the parameters 'A' and 'alpha' where: MSD = A*tau^alpha
27 %
28 % REMARKS:
29 % We can determine A and alpha using MATLAB's intrinsic 'lsqnonlin' routine
30 % which fits a linear function in the least squares sense. The goal is to
31 % identify constants 'A' and 'alpha' that fit empirical MSD vs. tau data
32 % such that:
33 % MSD = A*tau^alpha
34 %
35 % Using properties of logs:
36 % log10(MSD) = log10(A*tau^alpha)
37 % log10(MSD) = log10(A)+log10(tau^alpha)
38 % log10(MSD) = log10(A)+alpha*log10(tau)
39 % rearranging:
40 % log10(MSD) = alpha*log10(tau) + log(A)
41 % which has the familiar form:
42 % Y = mX + b
43 % where Y = log10(MSD), m = alpha, X = log10(tau), b = log10(A) => A = 10^b
44 %
45 % For compatibility with 'lsqnonlin' syntax we need to think of the Y = mX + b
46 % system in terms of a matrix representation.
47 %
48 % ASSUMPTIONS:
49 % Model is appropriate for mode of motility.
50 %
51 % INPUT:
52 % MSD_tabs = array containing mean squared displacements and corresponding
53 % lag time taus in terms of absolute differences with the following
54 % structure:
55 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
56 % col 2 = obsolete 'NaN'
57 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
58 % col 4 = Neff (number of independent observations)
59 % col 5 = tau absolute elapsed time (sec)
60 % col 6 = VAR(dx) (pix^2)
61 % col 7 = VAR(dy) (pix^2)
62 % MSD_tbin = array containing mean squared displacements and corresponding
63 % lag time taus in terms of binned differences with the following
64 % structure:
65 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
66 % col 2 = obsolete 'NaN'
67 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
68 % col 4 = Neff (number of independent observations)
69 % col 5 = tau binned elapsed time (sec)
70 % col 6 = VAR(dx) (pix^2)
71 % col 7 = VAR(dy) (pix^2)
72 % pixel_calib = user-supplied microns/pixel
73 % fid = file ID of log file to which progress is recorded
74 %
75 % OUTPUT:
76 % fit_PL_tabs = (PL = power law) array with length = length(MSD_tabs)
77 % containing fit data generated using best-fit 'A' and 'alpha' parameters
78 % having structure:
79 % col 1 = fit mean squared displacement (pixels^2)
80 % col 2 = tau absolute elapsed time (sec)
81 % fit_PL_tbin = (PL = power law) array with length = length(MSD_tbin)
82 % containing fit data generated using best-fit 'A' and 'alpha' parameters
83 % having structure:
84 % col 1 = fit mean squared displacement (pixels^2)
85 % col 2 = tau binned elapsed time (sec)
86 % Aout = best fit coefficient value (um^2/min^alpha)
87 % alphaout = best fit power value (unitless)
88 %*****
89
90 function [fit_PL_tabs fit_PL_tbin Aout alphaout] = Power_Law_v4(MSD_tabs, MSD_tbin, pixel_calib, fid)
91

```

```

92 % Get function name:
93 func_name = mfilename;
94
95 % Update log file that function is running:
96 fprintf(1, '\n%s running ... \n', func_name);
97 fprintf(fid, '\n%s running ... \n', func_name);
98
99 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
100 % warning dialog the log file records no errors/warnings generated:
101 warn = 0;
102
103 for i = 1:2
104
105     if i == 1
106         MSD = MSD_tabs;
107     elseif i == 2
108         MSD = MSD_tbin;
109     end
110
111     % Take logs of data:
112     Y = log10(MSD(:,1));
113     X = log10(MSD(:,5));
114
115     %*****
116     % Perform weighted fitting:
117
118     % Define a weight vector 'w' that is the # independent observations
119     % used to generate that data point (Neff)
120     w = MSD(:,4);
121     % w = ones(size(MSD(:,4))); % use for unweighted fit
122     % Use of 'lscov' requires considering the matrix form of the system of
123     % equations. We want to think in terms of a system Cp = Y.
124     % C = coefficient matrix. Col 1 = vector X. Col 2 = ones vector.
125     % p = parameter vector. p(1) = m. p(2) = b.
126     C = [X ones(size(X))];
127
128     % stdx, mse, and S are returned for possible use with polyval in future
129     % versions
130     [p] = lscov(C, Y, w);
131     m = p(1); % slope
132     b = p(2); % intercept
133
134     % Solve the weighted parameters A and alpha you actually care about:
135     alpha = m; % unitless
136     alphaout = alpha;
137     A = 10^alpha; % units of pix^2/sec^alpha
138     % Unit conversion:
139     % (pix^2/sec^alpha)*(um^2/pix^2)*(60^alpha sec^alpha/1 min^alpha)
140     Aunit = A*(pixel_calib^2)*(60^alpha);
141     Aout = Aunit;
142     %*****
143
144     % Reserve memory for an array that will hold the theoretical MSD values
145     % using returned parameters:
146     rows = length(MSD(:,1));
147     F = zeros(rows,3);
148
149     for j = 1:rows
150         tau = MSD(j,5);
151         F(j,1) = A*tau^alpha;
152         F(j,2) = tau;
153     end
154
155     if i == 1
156         fit_tabs = F;
157         tau_type = 'ABSOLUTE';
158     elseif i == 2
159         fit_tbin = F;

```



```

160     tau_type = 'BINNED';
161 end
162
163 % Update log file with progress:
164 fprintf(1, '\n\tProcessing %s tau MSD data:\n', tau_type);
165 fprintf(1, '\t*****\n');
166 fprintf(1, '\tWEIGHTED Fit Parameters:\n');
167 fprintf(1, '\t\tA = %0.4f (pix^2/sec^alpha)\n', A);
168 fprintf(1, '\t\tA = %0.4f (um^2/min^alpha)\n', Aunit);
169 fprintf(1, '\t\talpha = %0.4f (unitless)\n', alpha);
170 fprintf(1, '\t*****\n');
171 fprintf(fid, '\n\tProcessing %s tau MSD data:\n', tau_type);
172 fprintf(fid, '\t*****\n');
173 fprintf(fid, '\tWEIGHTED Fit Parameters:\n');
174 fprintf(fid, '\t\tA = %0.4f (pix^2/sec^alpha)\n', A);
175 fprintf(fid, '\t\tA = %0.4f (um^2/min^alpha)\n', Aunit);
176 fprintf(fid, '\t\talpha = %0.4f (unitless)\n', alpha);
177 fprintf(fid, '\t*****\n');
178
179 end
180
181 % Ensure number of data entries in 'fit' arrays is consistent with MSD:
182 if length(fit_tabs(:,1)) ~= length(MSD_tabs(:,1))
183     fprintf(1, '\tWARNING: length(fit_tabs) ~= length(MSD_tabs)\n')
184     fprintf(fid, '\tWARNING: length(fit_tabs) ~= length(MSD_tabs)\n')
185     warn = 1;
186 end
187 if length(fit_tbin(:,1)) ~= length(MSD_tbin(:,1))
188     fprintf(1, '\tWARNING: length(fit_tbin) ~= length(MSD_tbin)\n')
189     fprintf(fid, '\tWARNING: length(fit_tbin) ~= length(MSD_tbin)\n')
190     warn = 1;
191 end
192
193 % Rename fit_tabs and fit_tbin to denote that they are from the Power Law
194 % model (PL):
195 fit_PL_tabs = fit_tabs;
196 fit_PL_tbin = fit_tbin;
197
198 % Save 'fit' arrays in .mat and .txt format:
199 save 'fit_PL_tabs.mat' fit_PL_tabs
200 fid2_1 = fopen('fit_PL_tabs.txt', 'wt');
201 fprintf(fid2_1, 'Weighted Fit MSD (pix^2)\tAbs. tau (sec)\n');
202 rows = length(fit_PL_tabs);
203 for k = 1:rows
204     fprintf(fid2_1, '%f\t%.0f\n', fit_PL_tabs(k,:));
205 end
206 fclose(fid2_1);
207
208 save 'fit_PL_tbin.mat' fit_PL_tbin
209 fid2_2 = fopen('fit_PL_tbin.txt', 'wt');
210 fprintf(fid2_2, 'Weighted Fit MSD (pix^2)\tBin. tau (sec)\n');
211 rows = length(fit_PL_tbin);
212 for k = 1:rows
213     fprintf(fid2_2, '%f\t%.0f\n', fit_PL_tbin(k,:));
214 end
215 fclose(fid2_2);
216
217 % If no warnings generated report so in log file:
218 if warn == 0
219     fprintf(1, '\n\tFunction completed without errors/warnings\n');
220     fprintf(fid, '\n\tFunction completed without errors/warnings\n');
221 end
222
223 % Update log file that function is completed:
224 fprintf(1, '\n%s completed\n', func_name);
225 fprintf(fid, '\n%s completed\n', func_name);
226
227 end

```

Plot_SandP_Fit_v6.m

```
1 % Steven J. Henry
2 % 07/22/2011
35 %*****
36 % PURPOSE:
37 % This function overlays the theoretical fit data from 'SandP.m' on top
38 % of the empirical MSD data.
39 %
40 % REMARKS:
41 % n/a
42 %
43 % ASSUMPTIONS:
44 % n/a
45 %
46 % INPUT:
47 % MSD_tabs = array containing mean squared displacements and corresponding
48 % lag time taus in terms of absolute differences with the following
49 % structure:
50 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
51 % col 2 = obsolete 'NaN'
52 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
53 % col 4 = Neff (number of independent observations)
54 % col 5 = tau absolute elapsed time (sec)
55 % col 6 = VAR(dx) (pix^2)
56 % col 7 = VAR(dy) (pix^2)
57 % MSD_tbin = array containing mean squared displacements and corresponding
58 % lag time taus in terms of binned differences with the following
59 % structure:
60 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
61 % col 2 = obsolete 'NaN'
62 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
63 % col 4 = Neff (number of independent observations)
64 % col 5 = tau binned elapsed time (sec)
65 % col 6 = VAR(dx) (pix^2)
66 % col 7 = VAR(dy) (pix^2)
67 % fit_tabs = array with length = length(MSD_tabs) containing fit data
68 % generated using best-fit S and P parameters having structure:
69 % col 1 = fit mean squared displacement (pixels^2)
70 % col 2 = tau absolute elapsed time (sec)
71 % fit_tbin = array with length = length(MSD_tabs) containing fit data
72 % generated using best-fit S and P parameters having structure:
73 % col 1 = fit mean squared displacement (pixels^2)
74 % col 2 = tau binned elapsed time (sec)
75 % pixel_calib = user-supplied microns/pixel
76 % t_max = user-supplied imaging duration (min)
77 % run_title = user specified string description of experimental condition
78 % epsilon_flag = 0 if MSD data being filtered is not corrected for
79 % random noise or 1 if MSD data being filtered is corrected for random
80 % noise
81 % fid = file ID of log file to which progress is recorded
82 %
83 % OUTPUT:
84 %
85 %*****
86
87 function [] = Plot_SandP_Fit_v6(MSD_tabs, MSD_tbin, fit_tabs, fit_tbin, pixel_calib, t_max, run_title, epsilon_flag, fid)
88
89 % Get function name:
90 func_name = mfilename;
91
92 % Update log file that function is running:
93 fprintf(1, '\n%s running ...\n', func_name);
94 fprintf(fid, '\n%s running ...\n', func_name);
95
96 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
97 % warning dialog the log file records no errors/warnings generated:
```

```

98 warn = 0;
99
100 % Prepare the data for plotting:
101 t_abs = MSD_tabs(:,5)/60; % min
102 m_abs = MSD_tabs(:,1)*pixel_calib^2; % microns^2
103 sev_abs = MSD_tabs(:,3)*pixel_calib^2; % microns^2
104 f_t_abs = fit_tabs(:,2)/60; % min
105 f_m_abs = fit_tabs(:,1)*pixel_calib^2; % microns^2
106
107 t_bin = MSD_tbin(:,5)/60; % min
108 m_bin = MSD_tbin(:,1)*pixel_calib^2; % microns^2
109 sev_bin = MSD_tbin(:,3)*pixel_calib^2; % microns^2
110 f_t_bin = fit_tbin(:,2)/60; % min
111 f_m_bin = fit_tbin(:,1)*pixel_calib^2; % microns^2
112
113 %*****
114 % Plot mean +/- s.e.v. (linear-linear axes)
115
116 % Set ordinate upperbound as maximum mean squared displacement value:
117 temp = zeros(1,2);
118 temp(1,1) = max(MSD_tabs(:,1)+MSD_tabs(:,3));
119 temp(1,2) = max(MSD_tbin(:,1)+MSD_tbin(:,3));
120 d2_max_h4_h5 = max(temp)*pixel_calib^2; % microns^2
121
122 % Plot data:
123 h4 = figure;
124 h4_sub(1) = subplot(1,2,1);
125
errorbar(t_abs,m_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
126 hold on
127
plot(f_t_abs,f_m_abs,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
128 h4_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
129 set(h4_legend_handle,'FontName','Arial','FontSize',12);
130
131 h4_sub(2) = subplot(1,2,2);
132
errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
133 hold on
134
plot(f_t_bin,f_m_bin,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
135 h4_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
136 set(h4_legend_handle,'FontName','Arial','FontSize',12);
137
138 % Set axes properties:
139 set(h4_sub,'ylim',[0 d2_max_h4_h5]);
140 set(h4_sub,'xlim',[0 t_max]);
141 set(h4_sub,'FontName','Arial');
142 set(h4_sub,'FontSize',14);
143
144 h4_x_axis_handles = cell2mat(get(h4_sub,'xlabel'));
145 set(h4_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
146 set(h4_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);
147
148 h4_y_axis_handles = cell2mat(get(h4_sub,'ylabel'));
149 if epsilon_flag == 0
150     ord_label = '<math>\langle r^2 \rangle</math> \mu m s.e.v. (\mu m^2)';
151 else if epsilon_flag == 1
152     ord_label = '<math>\langle r^2 \rangle - 4 * \epsilon^2</math> \mu m s.e.v. (\mu m^2)';
153 end
154 set(h4_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);
155
156 h4_title_handles = cell2mat(get(h4_sub,'title'));
157 set(h4_title_handles,'String',{run_title,'Biased Random Walk Model'},'FontName','Arial','FontSize',16);

```

```

158
159 % Save figure window generated:
160 saveas(h4, 'Fit_BRW_sev.fig', 'fig');
161 %*****
162
163 %*****
164 % Plot mean +/- s.e.v. (log-log axes)
165 % On log-log axes negative values result in output warnings to user. To
166 % avoid this filter for msd - s.d. (lower bounds) that result in negative
167 % values.
168 L_sev_abs = sev_abs;
169 for i = 1:length(L_sev_abs)
170     if m_abs(i)-L_sev_abs(i) < 0
171         L_sev_abs(i) = 0;
172     end
173 end
174 L_sev_bin = sev_bin;
175 for i = 1:length(L_sev_bin)
176     if m_bin(i)-L_sev_bin(i) < 0
177         L_sev_bin(i) = 0;
178     end
179 end
180
181 % Plot data:
182 h5 = figure;
183 h5_sub(1) = subplot(1,2,1);
184
errorbar(t_abs,m_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerS
ize',5);
185 hold on
186
plot(f_t_abs,f_m_abs,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'Handl
eVisibility','on');
187 h5_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
188 set(h5_legend_handle,'FontName','Arial','FontSize',12);
189
190 h5_sub(2) = subplot(1,2,2);
191
errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSiz
e',5);
192 hold on
193
plot(f_t_bin,f_m_bin,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'Handl
eVisibility','on');
194 h5_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
195 set(h5_legend_handle,'FontName','Arial','FontSize',12);
196
197 % Set axes properties:
198 set(h5_sub,'ylim',[1 d2_max_h4_h5]);
199 set(h5_sub,'xlim',[1 t_max]);
200 set(h5_sub,'yscale','log');
201 set(h5_sub,'xscale','log');
202 set(h5_sub,'YMinorTick','on');
203 set(h5_sub,'XMinorTick','on');
204 set(h5_sub,'FontName','Arial');
205 set(h5_sub,'FontSize',14);
206
207 h5_x_axis_handles = cell2mat(get(h5_sub,'xlabel'));
208 set(h5_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
209 set(h5_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);
210
211 h5_y_axis_handles = cell2mat(get(h5_sub,'ylabel'));
212 if epsilon_flag == 0
213     ord_label = '<math>\rho</math> s.e.v. (

```

```

218
219 h5_title_handles = cell2mat(get(h5_sub,'title'));
220 set(h5_title_handles,'String',{run_title;'Biased Random Walk Model'},'FontName','Arial','FontSize',16);
221
222 % Save figure window generated:
223 saveas(h5,'Fit_BRW_sev_loglog.fig','fig');
224 %*****
225
226 %*****
227 % Plot mean +/- s.e.v. (log-log axes) using binned taus and fix the data
228 % aspect ratio to [1 1 1].
229
230 % Plot data:
231 h6 = figure;
232 h6_plot = axes;
233
errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
234 hold on
235
plot(f_t_bin,f_m_bin,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
236 h6_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
237 set(h6_legend_handle,'FontName','Arial','FontSize',12);
238
239 % Set axes properties:
240 if d2_max_h4_h5 >= t_max
241     axis_lim = d2_max_h4_h5;
242 else
243     axis_lim = t_max;
244 end
245
246 set(h6_plot,'ylim',[1 axis_lim]);
247 set(h6_plot,'xlim',[1 axis_lim]);
248 set(h6_plot,'yscale','log');
249 set(h6_plot,'xscale','log');
250 set(h6_plot,'DataAspectRatio',[1 1 1]);
251 set(h6_plot,'YMinorTick','on');
252 set(h6_plot,'XMinorTick','on');
253 set(h6_plot,'FontName','Arial');
254 set(h6_plot,'FontSize',14);
255 set(h6_plot,'box','on');
256
257 h6_plot_x_axis_handle = xlabel('\tau (min)');
258 set(h6_plot_x_axis_handle,'FontName','Arial','FontSize',16);
259
260 if epsilon_flag == 0
261     ord_label = '<math>\mu \text{ s.e.v. } (\mu \mu^2)</math>';
262 elseif epsilon_flag == 1
263     ord_label = '<math>\mu - 4 \epsilon^2 \mu \text{ s.e.v. } (\mu \mu^2)</math>';
264 end
265 h6_plot_y_axis_handle = ylabel(ord_label);
266 set(h6_plot_y_axis_handle,'FontName','Arial','FontSize',16);
267
268 h6_plot_title_handle = title({run_title;'Biased Random Walk Model'});
269 set(h6_plot_title_handle,'FontName','Arial','FontSize',16);
270
271 % Save figure window generated:
272 saveas(h6,'Fit_BRW_sev_loglog_final.fig','fig');
273 %*****
274
275 % If no warnings generated report so in log file:
276 if warn == 0
277     fprintf(1,'\n\tFunction completed without errors/warnings\n');
278     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
279 end
280
281 % Update log file that function is completed:

```

```

282 fprintf(1, '\n%s completed\n', func_name);
283 fprintf(fid, '\n%s completed\n', func_name);
284
285 end

```

Plot_Power_Law_Fit_v4.m

```

1  % Steven J. Henry
2  % 07/22/2011
23 %*****
24 % PURPOSE:
25 % This function overlays the theoretical fit data from 'Power_Law.m' on top
26 % of the empirical MSD data.
27 %
28 % REMARKS:
29 % n/a
30 %
31 % ASSUMPTIONS:
32 % n/a
33 %
34 % INPUT:
35 % MSD_tabs = array containing mean squared displacements and corresponding
36 % lag time taus in terms of absolute differences with the following
37 % structure:
38 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
39 % col 2 = obsolete 'NaN'
40 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
41 % col 4 = Neff (number of independent observations)
42 % col 5 = tau absolute elapsed time (sec)
43 % col 6 = VAR(dx) (pix^2)
44 % col 7 = VAR(dy) (pix^2)
45 % MSD_tbin = array containing mean squared displacements and corresponding
46 % lag time taus in terms of binned differences with the following
47 % structure:
48 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
49 % col 2 = obsolete 'NaN'
50 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
51 % col 4 = Neff (number of independent observations)
52 % col 5 = tau binned elapsed time (sec)
53 % col 6 = VAR(dx) (pix^2)
54 % col 7 = VAR(dy) (pix^2)
55 % fit_tabs = array with length = length(MSD_tabs) containing fit data
56 % generated using best-fit A and alpha parameters:
57 % col 1 = fit mean squared displacement (pixels^2)
58 % col 2 = tau absolute elapsed time (sec)
59 % fit_tbin = array with length = length(MSD_tabs) containing fit data
60 % generated using best-fit A and alpha parameters:
61 % col 1 = fit mean squared displacement (pixels^2)
62 % col 2 = tau binned elapsed time (sec)
63 % pixel_calib = user-supplied microns/pixel
64 % t_max = user-supplied imaging duration (min)
65 % run_title = user specified string description of experimental condition
66 % epsilon_flag = 0 if MSD data being filtered is not corrected for
67 % random noise or 1 if MSD data being filtered is corrected for random
68 % noise
69 % fid = file ID of log file to which progress is recorded
70 %
71 % OUTPUT:
72 %
73 %*****
74
75 function [] = Plot_Power_Law_Fit_v4(MSD_tabs, MSD_tbin, fit_tabs, fit_tbin, pixel_calib, t_max, run_title,
epsilon_flag, fid)
76
77 % Get function name:
78 func_name = mfilename;
79
80 % Update log file that function is running:

```

```

81 fprintf(1, '\n%s running ...'\n', func_name);
82 fprintf(fid, '\n%s running ...'\n', func_name);
83
84 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
85 % warning dialog the log file records no errors/warnings generated:
86 warn = 0;
87
88 % Prepare the data for plotting:
89 t_abs = MSD_tabs(:,5)/60; % min
90 m_abs = MSD_tabs(:,1)*pixel_calib^2; % microns^2
91 sev_abs = MSD_tabs(:,3)*pixel_calib^2; % microns^2
92 f_t_abs = fit_tabs(:,2)/60; % min
93 f_m_abs_uw = fit_tabs(:,1)*pixel_calib^2; % microns^2, unweighted fit
94
95 t_bin = MSD_tbin(:,5)/60; % min
96 m_bin = MSD_tbin(:,1)*pixel_calib^2; % microns^2
97 sev_bin = MSD_tbin(:,3)*pixel_calib^2; % microns^2
98 f_t_bin = fit_tbin(:,2)/60; % min
99 f_m_bin_uw = fit_tbin(:,1)*pixel_calib^2; % microns^2, unweighted fit
100
101 %*****
102 % Plot mean +/- s.e.v. (linear-linear axes)
103
104 % Set ordinate upperbound as maximum mean squared displacement value:
105 temp = zeros(1,2);
106 temp(1,1) = max(MSD_tabs(:,1)+MSD_tabs(:,3));
107 temp(1,2) = max(MSD_tbin(:,1)+MSD_tbin(:,3));
108 d2_max_h4_h5 = max(temp)*pixel_calib^2; % microns^2
109
110 % Plot data:
111 h4 = figure;
112 h4_sub(1) = subplot(1,2,1);
113
114 errorbar(t_abs,m_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
115 hold on
116 plot(f_t_abs,f_m_abs_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
117 h4_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
118 set(h4_legend_handle,'FontName','Arial','FontSize',12);
119 h4_sub(2) = subplot(1,2,2);
120
121 errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
122 hold on
123 plot(f_t_bin,f_m_bin_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
124 h4_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
125 set(h4_legend_handle,'FontName','Arial','FontSize',12);
126
127 % Set axes properties:
128 set(h4_sub,'ylim',[0 d2_max_h4_h5]);
129 set(h4_sub,'xlim',[0 t_max]);
130 set(h4_sub,'FontName','Arial');
131 set(h4_sub,'FontSize',14);
132
133 h4_x_axis_handles = cell2mat(get(h4_sub,'xlabel'));
134 set(h4_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
135 set(h4_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);
136
137 h4_y_axis_handles = cell2mat(get(h4_sub,'ylabel'));
138 if epsilon_flag == 0
139     ord_label = '<r^2> \mu m s.e.v. (\mu m^2)';
140 else if epsilon_flag == 1
141     ord_label = '<r^2> - 4*\epsilon^2 \mu m s.e.v. (\mu m^2)';

```

```

141 end
142 set(h4_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);
143
144 h4_title_handles = cell2mat(get(h4_sub,'title'));
145 set(h4_title_handles,'String',{run_title:'Power Law Model'},'FontName','Arial','FontSize',16);
146
147 % Save figure window generated:
148 saveas(h4,'Fit_PL_sev.fig','fig');
149 %*****
150
151 %*****
152 % Plot mean +/- s.e.v. (log-log axes)
153 % On log-log axes negative values result in output warnings to user. To
154 % avoid this filter for msd - s.d. (lower bounds) that result in negative
155 % values.
156 L_sev_abs = sev_abs;
157 for i = 1:length(L_sev_abs)
158     if m_abs(i)-L_sev_abs(i) < 0
159         L_sev_abs(i) = 0;
160     end
161 end
162 L_sev_bin = sev_bin;
163 for i = 1:length(L_sev_bin)
164     if m_bin(i)-L_sev_bin(i) < 0
165         L_sev_bin(i) = 0;
166     end
167 end
168
169 % Plot data:
170 h5 = figure;
171 h5_sub(1) = subplot(1,2,1);
172
173 errorbar(t_abs,m_abs,sev_abs,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
174
175 hold on
176
177 plot(f_t_abs,f_m_abs_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
178 h5_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
179 set(h5_legend_handle,'FontName','Arial','FontSize',12)
180
181 h5_sub(2) = subplot(1,2,2);
182
183 errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
184
185 hold on
186
187 plot(f_t_bin,f_m_bin_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
188 h5_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
189 set(h5_legend_handle,'FontName','Arial','FontSize',12)
190
191 % Set axes properties:
192 set(h5_sub,'ylim',[1 d2_max_h4_h5]);
193 set(h5_sub,'xlim',[1 t_max]);
194 set(h5_sub,'yscale','log');
195 set(h5_sub,'xscale','log');
196 set(h5_sub,'YMinorTick','on');
197 set(h5_sub,'XMinorTick','on');
198 set(h5_sub,'FontName','Arial');
199 set(h5_sub,'FontSize',14);
200
201 h5_x_axis_handles = cell2mat(get(h5_sub,'xlabel'));
202 set(h5_x_axis_handles(1),'String','Absolute \tau (min)','FontName','Arial','FontSize',16);
203 set(h5_x_axis_handles(2),'String','Binned \tau (min)','FontName','Arial','FontSize',16);
204
205 h5_y_axis_handles = cell2mat(get(h5_sub,'ylabel'));
206 if epsilon_flag == 0

```



```

201     ord_label = '<math>\rho</math> s.e.v. ( $\mu^2$ )';
202 elseif epsilon_flag == 1
203     ord_label = '<math>\rho - 4\epsilon^2</math> \rho s.e.v. ( $\mu^2$ )';
204 end
205 set(h5_y_axis_handles,'String',ord_label,'FontName','Arial','FontSize',16);
206
207 h5_title_handles = cell2mat(get(h5_sub,'title'));
208 set(h5_title_handles,'String',{run_title;'Power Law Model'},'FontName','Arial','FontSize',16);
209
210 % Save figure window generated:
211 saveas(h5, 'Fit_PL_sev_loglog.fig', 'fig');
212 %*****
213
214 %*****
215 % Plot mean +/- s.e.v. (log-log axes) using binned taus and fix the data
216 % aspect ratio to [1 1 1].
217
218 % Plot data:
219 h6 = figure;
220 h6_plot = axes;
221
222 errorbar(t_bin,m_bin,sev_bin,'LineStyle','none','Color','k','Marker','o','MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',5);
223 hold on
224
225 plot(f_t_bin,f_m_bin_uw,'LineStyle','none','Color','r','Marker','d','MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',5,'HandleVisibility','on');
226
227 h6_legend_handle = legend('Empirical Data','Weighted Fit','Location','NorthWest');
228 set(h6_legend_handle,'FontName','Arial','FontSize',12);
229
230 % Set axes properties:
231 if d2_max_h4_h5 >= t_max
232     axis_lim = d2_max_h4_h5;
233 else
234     axis_lim = t_max;
235 end
236
237 set(h6_plot,'ylim',[1 axis_lim]);
238 set(h6_plot,'xlim',[1 axis_lim]);
239 set(h6_plot,'yscale','log');
240 set(h6_plot,'xscale','log');
241 set(h6_plot,'DataAspectRatio',[1 1 1]);
242 set(h6_plot,'YMinorTick','on');
243 set(h6_plot,'XMinorTick','on');
244 set(h6_plot,'FontName','Arial');
245 set(h6_plot,'FontSize',14);
246 set(h6_plot,'box','on');
247
248 h6_plot_x_axis_handle = xlabel('\tau (min)');
249 set(h6_plot_x_axis_handle,'FontName','Arial','FontSize',16);
250
251 if epsilon_flag == 0
252     ord_label = '<math>\rho</math> s.e.v. ( $\mu^2$ )';
253 elseif epsilon_flag == 1
254     ord_label = '<math>\rho - 4\epsilon^2</math> \rho s.e.v. ( $\mu^2$ )';
255 end
256 h6_plot_y_axis_handle = ylabel(ord_label);
257 set(h6_plot_y_axis_handle,'FontName','Arial','FontSize',16);
258
259 h6_plot_title_handle = title({run_title;'Power Law Model'});
260 set(h6_plot_title_handle,'FontName','Arial','FontSize',16);
261
262 % Save figure window generated:
263 saveas(h6, 'Fit_PL_sev_loglog_final.fig', 'fig');
264 %*****
265
266 % If no warnings generated report so in log file:
267 if warn == 0

```

```

265 fprintf(1,'\n\tFunction completed without errors/warnings\n');
266 fprintf(fid,'\n\tFunction completed without errors/warnings\n');
267 end
268
269 % Update log file that function is completed:
270 fprintf(1,'\n%s completed\n',func_name);
271 fprintf(fid,'\n%s completed\n',func_name);
272
273 end

```

Van_Hove_Analysis_v3.m

```

1 % Steven J. Henry
2 % 10/16/2013
17 %*****
18 % PURPOSE:
19 % This program computes all the displacements observed for a population of
20 % moving objects. Displacements are those between centroid observations of
21 % a given object's trajectory, not between objects. All objects contribute
22 % displacements to the ensemble (population) of displacements observed. The
23 % displacements computed ***have a sign*** and ***are not squared***. The
24 % displacements are overlapping such that all displacements corresponding
25 % to a given centroid's trajectory are not statistically independent.
26 %
27 % ASSUMPTIONS:
28 % 'data' array only posses data that had previously been utilized in
29 % 'Mean_Displacements.m'. As such this array has been processed via
30 % 'Post_IJ_Manual_Track.m' and possibly 'Filter_Exp_Data.m' such that only
31 % those centroid positions invoked in the previous computation of the MSD
32 % curve sit in the 'data' array presently.
33 %
34 % INPUT:
35 % pixel_calib = user specified objective calibration (um/pixel)
36 % data = an array containing all pertinent tracking information for each
37 % cell in the given experimental condition having the following structure
38 % (as a result of 'Post_IJ_Manual_Track.m'):
39 % col 1 = unique track number ID assigned to each cell
40 % col 2 = x coordinate (pixels) of centroid
41 % col 3 = y coordinate (pixels) of centroid
42 % col 4 = absolute time coresponding to frame in which cell is found
43 % (sec)
44 % col 5 = binned time corresponding to frame in which cell is found (sec)
45 % col 6 = area of cell in pixels
46 % col 7 = track change flag. Entry = 1 if start of new track (yes) or 0
47 % if no (i.e. continuation of anexisting track.
48 % MSD_tbin = array containing mean squared displacements and corresponding
49 % lag time taus in terms of binned differences with the following
50 % structure:
51 % col 1 = MSD(tau) = VAR(dr) (pixels^2)
52 % col 2 = obsolete 'NaN'
53 % col 3 = standard error of the variance (2* MSD(tau)/sqrt(Neff)) (pixels^2)
54 % col 4 = Neff (number of independent observations)
55 % col 5 = tau binned elapsed time (sec)
56 % col 6 = VAR(dx) (pix^2)
57 % col 7 = VAR(dy) (pix^2)
58 % epsilon_flag = 0 if MSD data being filtered is not corrected for
59 % random noise or 1 if MSD data being filtered is corrected for random
60 % noise
61 % run_title = user specified string description of experimental condition
62 % fid = file ID of log file to which progress is recorded
63 %
64 % OUTPUT:
65 % MATLAB figures (.fig)
66 % Van_Hove_dx.fig = van hove analysis of displacements in x direction
67 % Van_Hove_dy.fig = van hove analysis of displacements in y directions
68 % Variance.fig = variance as a function of tau
69 % Kurtosis.fig = kurtosis as a function of tau
70 %*****

```

```

71
72 function [] = Van_Hove_Analysis_v3(pixel_calib, data, MSD_tbin, epsilon_flag, run_title, fid)
73
74 % Get function name:
75 func_name = mfilename;
76
77 % Update log file that function is running:
78 fprintf(1, '\n%s running ... \n', func_name);
79 fprintf(fid, '\n%s running ... \n', func_name);
80
81 % Turn warning flag 'warn' off. If 'warn' is not activated by entry into a
82 % warning dialog the log file records no errors/warnings generated:
83 warn = 0;
84
85 % Retrieve number of tracks:
86 uniq_IDs = unique(data(:,1));
87 num_tracks = length(uniq_IDs);
88
89 % Reserve the variable name "all_disp" for the column array that will
90 % log all displacements computed from each track:
91 all_disp = [];
92
93 % Loop over each track
94 for i = 1:num_tracks
95
96     % Grap the track number
97     track = uniq_IDs(i);
98
99     % Find cooresponding row numbers for this track's entry in 'data'
100    % array:
101    track_ind = data(:,1)==track;
102
103    % Isolate track's x,y centroid positions
104    track_xy = data(track_ind,2:3);
105    % Isolate bin-time values corresponding to x,y centroid positions
106    % previously:
107    track_t = data(track_ind,5);
108
109    % What is the total number of centroid observations for this track?
110    num_obs = length(track_t);
111
112    % Determine the total number of displacements to be computed using a
113    % moving origin strategy. Example: if you have 5 observations of an
114    % object's centroid the total number of intervals you will compute is
115    % (5-1)+(5-2)+(5-3)+(5-4) = 4+3+2+1 = 10. More generally for N total
116    % observations of an object's centroid you will have summation(i = 1, i
117    % = N-1) intervals. Rather than having to evaluate "num_obs-1" with
118    % every iteration in the following for loop we can move the "-1" into
119    % the actual computation itself.
120    num_intervals = 0;
121    for ii = 2:num_obs
122        num_intervals = (ii-1) + num_intervals;
123    end
124
125    if isempty(all_disp)==1
126        all_disp = zeros(num_intervals,3);
127        print_row = 1;
128    else
129        all_disp_old = all_disp;
130        num_intervals_tot = size(all_disp_old,1);
131        clear all_disp
132        all_disp = zeros(num_intervals_tot+num_intervals,3);
133        all_disp(1:num_intervals_tot,:) = all_disp_old;
134        clear all_disp_old
135        print_row = num_intervals_tot+1;
136    end
137
138    for k = 1:num_obs-1

```

```

139
140     x_orig = track_xy(k,1);
141     y_orig = track_xy(k,2);
142     t_orig = track_t(k);
143
144     for kk = k+1:num_obs
145
146         % Load advance row
147         x_adv = track_xy(kk,1);
148         y_adv = track_xy(kk,2);
149         t_adv = track_t(kk);
150
151         % Compute displacements:
152         dx = x_adv - x_orig;
153         dy = y_adv - y_orig;
154         dt = t_adv - t_orig;
155
156         % Log displacements:
157         all_disp(print_row,1) = dx;
158         all_disp(print_row,2) = dy;
159         all_disp(print_row,3) = dt;
160
161         % Advance print row:
162         print_row = print_row+1;
163
164     end
165
166 end
167
168 end
169
170 % Save 'all_disp' array in .mat and .txt format:
171 save('all_disp.mat', 'all_disp');
172 fid2_1 = fopen('all_disp.txt', 'wt');
173 fprintf(fid2_1, 'dx (pix)\tdy (pix)\tdt (sec)\n');
174 rows = size(all_disp,1);
175 for k = 1:rows
176     fprintf(fid2_1, '%ft%f\t%f\n', all_disp(k,:));
177 end
178 fclose(fid2_1);
179
180 % *****
181 % Begin plotting: (Note this could be rolled into a separate function but
182 % the following suits the current needs satisfactorially.)
183
184 % Convert to physical units:
185 all_disp(:,1:2) = all_disp(:,1:2)*pixel_calib;
186 all_disp(:,3) = all_disp(:,3)/60;
187
188 % Identify total unique tau values to plot:
189 taus = unique(all_disp(:,3));
190 tot_taus = length(taus);
191
192 % The following are arbitrary but for the majority of cases identified so
193 % far (11/03/2011) data is only analyzed through 30 min and imaging is
194 % conducted at 1 min/frame:
195 m = 2;
196 n = 3;
197 tau_plot = [1 5 10 15 20 25]; %min
198
199 % Define figure handles
200 hist_fig_x = figure;
201 hist_fig_y = figure;
202 var_fig = figure;
203 kurt_fig = figure;
204
205 % Reserve memory:
206 V = zeros(tot_taus,4);

```

```

207 K = zeros(tot_taus,3);
208
209 % Generate counter:
210 counts = 1;
211
212 for i = 1:tot_taus
213     tau = taus(i);
214
215     ind = all_disp(:,3)==tau;
216
217     x_list = all_disp(ind,1);
218     V(i,1) = var(x_list);
219     K(i,1) = kurtosis(x_list);
220
221     y_list = all_disp(ind,2);
222     V(i,2) = var(y_list);
223     K(i,2) = kurtosis(y_list);
224
225     % delR = delX ihat + delY jhat
226     % VAR(delR) = VAR(delX)+VAR(delY)+2*COV(delX,delY);
227     % We anticipate delX and delY are independent and so COV = 0
228     xycov = cov(x_list,y_list);
229     % xycov is a 2X2 matrix with:
230     % xycov(1,1) = VAR(x_list)
231     % xycov(2,2) = VAR(y_list)
232     % xycov(1,2) = xycov(2,1) = COV(x_list,y_list);
233     V(i,3) = V(i,1)+V(i,2)+2*xycov(1,2);
234
235     V(i,4) = tau;
236     K(i,3) = tau;
237
238     % Round to the nearest tenth minute and evaluate to see if it resides
239     % in the list of tau_plot values for which a histogram should be
240     % constructed, but only do this if you haven't already plotted m*n
241     % histograms (the max for this figure).
242     if any(round(tau*10)/10==tau_plot)==1 && counts < m*n;
243
244         figure(hist_fig_x)
245         subplot(m,n,counts);
246         nbins = round(sqrt(length(x_list)));
247         [freq bin_loc] = hist(x_list,nbins);
248         bar(bin_loc,freq,1);
249         x_hist_fig_title = title(['\tau = ' num2str(round(tau)) ' min']);
250         set(x_hist_fig_title,'FontName','Arial','FontSize',16);
251         x_hist_fig_xlabel = xlabel('\Delta x (\mum)');
252         set(x_hist_fig_xlabel,'FontName','Arial','FontSize',16);
253         x_hist_fig_ylabel = ylabel('Counts');
254         set(x_hist_fig_ylabel,'FontName','Arial','FontSize',16);
255         axis([min(x_list) max(x_list) 0 max(freq)]);
256         set(gca,'yscale','log');
257         set(gca,'FontName','Arial');
258         set(gca,'FontSize',14);
259
260         figure(hist_fig_y)
261         subplot(m,n,counts);
262         nbins = round(sqrt(length(y_list)));
263         [freq bin_loc] = hist(y_list,nbins);
264         bar(bin_loc,freq,1);
265         y_hist_fig_title = title(['\tau = ' num2str(round(tau)) ' min']);
266         set(y_hist_fig_title,'FontName','Arial','FontSize',16);
267         y_hist_fig_xlabel = xlabel('\Delta y (\mum)');
268         set(y_hist_fig_xlabel,'FontName','Arial','FontSize',16);
269         y_hist_fig_ylabel = ylabel('Counts');
270         set(y_hist_fig_ylabel,'FontName','Arial','FontSize',16);
271         axis([min(y_list) max(y_list) 0 max(freq)]);
272         set(gca,'yscale','log');
273         set(gca,'FontName','Arial');
274

```

```

275     set(gca,'FontSize',14);
276
277     counts = counts+1;
278
279     end
280
281 end
282
283 % Save Van Hove figures generated:
284 saveas(hist_fig_x, 'Van_Hove_dx.fig', 'fig');
285 saveas(hist_fig_y, 'Van_Hove_dy.fig', 'fig');
286
287 % Re-plot MSD +/- s.e.v. data for easy visual comparison with variance
288 % method here:
289 t_bin = MSD_tbin(:,5)/60; % min
290 m_bin = MSD_tbin(:,1)*pixel_calib^2; % microns^2
291 sev_bin = MSD_tbin(:,3)*pixel_calib^2; % microns^2
292 % On log-log axes negative values result in output warnings to user. To
293 % avoid this filter for msd - s.d. (lower bounds) that result in negative
294 % values.
295 L_sev_bin = sev_bin;
296 for i = 1:length(L_sev_bin)
297     if m_bin(i)-L_sev_bin(i) < 0
298         L_sev_bin(i) = 0;
299     end
300 end
301 if epsilon_flag == 0
302     legend_label = '<math>\langle \Delta t^2 \rangle</math> \pm s.e.v. (

```

```

335 fid2_1 = fopen('Variance.txt','wt');
336 fprintf(fid2_1,'var(dx) (um^2)\tvar(dy) (um^2) \tvar(dr) (um^2) \ttau (min)\n');
337 rows = size(V,1);
338 for k = 1:rows
339     fprintf(fid2_1,'%ft%ft%ft%ft\n',V(k,:));
340 end
341 fclose(fid2_1);
342
343 figure(kurt_fig);
344
plot(K(:,3),K(:,1),'LineStyle','none','Marker','^','MarkerEdgeColor','k','MarkerFaceColor','b','MarkerSize',5,'HandleVisibility','o
n');
345 hold on
346
plot(K(:,3),K(:,2),'LineStyle','none','Marker','s','MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize',5,'HandleVisibility','o
n');
347 kurt_fig_xlabel = xlabel('\ttau (min)');
348 set(kurt_fig_xlabel,'FontName','Arial','FontSize',16);
349 kurt_fig_ylabel = ylabel('Kurtosis (unitless)');
350 set(kurt_fig_ylabel,'FontName','Arial','FontSize',16);
351 set(gca,'FontName','Arial');
352 set(gca,'FontSize',14);
353 set(gca,'box','on');
354 kurt_fig_legend = legend('\Deltax','\Deltay');
355 set(kurt_fig_legend,'FontName','Arial','FontSize',12);
356 kurt_fig_title = title(run_title);
357 set(kurt_fig_title,'FontName','Arial','FontSize',16);
358
359 % Save kurtosis figure window generated:
360 saveas(kurt_fig, 'Kurtosis.fig', 'fig');
361
362 % Save 'kurt' array in .mat and .txt format:
363 save('Kurtosis.mat','K');
364 fid2_1 = fopen('Kurtosis.txt','wt');
365 fprintf(fid2_1,'kurt(dx)\tkurt(dy)\ttau (min)\n');
366 rows = size(K,1);
367 for k = 1:rows
368     fprintf(fid2_1,'%ft%ft%ft\n',K(k,:));
369 end
370 fclose(fid2_1);
371
372
373 % If no warnings generated report so in log file:
374 if warn == 0
375     fprintf(1,'\n\tFunction completed without errors/warnings\n');
376     fprintf(fid,'\n\tFunction completed without errors/warnings\n');
377 end
378
379 % Update log file that function is completed:
380 fprintf(1,'\n%s completed\n',func_name);
381 fprintf(fid,'\n%s completed\n',func_name);
382
383 end

```

Tidy_Up_v1.m

```

1 % Steven J. Henry
2 % 05/01/2011
7 %*****
8 % PURPOSE:
9 % The following function places all files with .fig, .mat, and .txt
10 % extensions into folders called "figs", "mats", and "txts" respectively.
11 % Only the master log file is left outside these folders for easy
12 % navigation.
13 %
14 % REMARKS:
15 % n/a
16 %

```

```

17 % ASSUMPTIONS:
18 % n/a
19 %
20 % INPUT:
21 % logfile = string containing title of master log file
22 %
23 % OUTPUT:
24 % n/a
25 %*****
26
27 function [] = Tidy_Up_v1(logfile)
28
29 % Make a folder called 'txts':
30 mkdir('txts');
31 % Define the path to that folder:
32 txt_path = [pwd '\txts'];
33 % Move all .txt files in the current folder to destination 'txts'
34 movefile('*.txt',txt_path);
35 % Change the directory to the 'txts' folder
36 cd(txt_path);
37 % Move the master log file out of the 'txts' files and back up one
38 % directory:
39 movefile(logfile, '..')
40 % Change directory to original position:
41 cd('..')
42
43 % Make a folder called 'figs'
44 mkdir('figs');
45 % Define the path to that folder:
46 fig_path = [pwd '\figs'];
47 % Move all .fig files in the current folder to destination 'figs'
48 movefile('*.fig',fig_path);
49
50 % Make a folder called 'mats'
51 mkdir('mats');
52 % Define the path to that folder:
53 mat_path = [pwd '\mats'];
54 % Move all .mat files in the current folder to destination 'mats'
55 movefile('*.mat',mat_path);
56
57 end

```


Appendix B

Custom MATLAB Code for Analysis of Neutrophil Spreading

Introduction

The purpose of this appendix is to provide the reader with more detail regarding the data analysis workflow employed to measure neutrophil spreading statistics. Broadly speaking the workflow consisted of capturing timelapse images of neutrophil spreading on mPADs, identifying the posts in each image and computing geometric centroids, linking centroids into trajectories, positioning post trajectories relative to their resting lattice (undeflected) positions, and computing ensemble statistics. The appended code is original and custom built to interface with Maria Kilfoil's MATLAB version (1) of John Crocker and David Grier's particle tracking routines originally developed in the IDL programming language (2). When a Kilfoil function is called in the following code I direct the reader to consult her well-annotated user manual online (3). I only reproduce my custom MATLAB codes which were used to interface with the Kilfoil scripts and to perform *post hoc* ensemble averaging and statistical analysis.

A significant functionality of my code is to allow reconstruction of the resting mPADs lattice conformation from an image of the deflected lattice *via* a two dimensional Fourier filtering method (Fig. B.1). This functionality allows the user to work with data sets in which cell spreading and post engagement commenced prior to the start of data acquisition and therefore requires that a post trajectory be positioned relative to its resting lattice position which was not the trajectory position at the onset of imaging. The

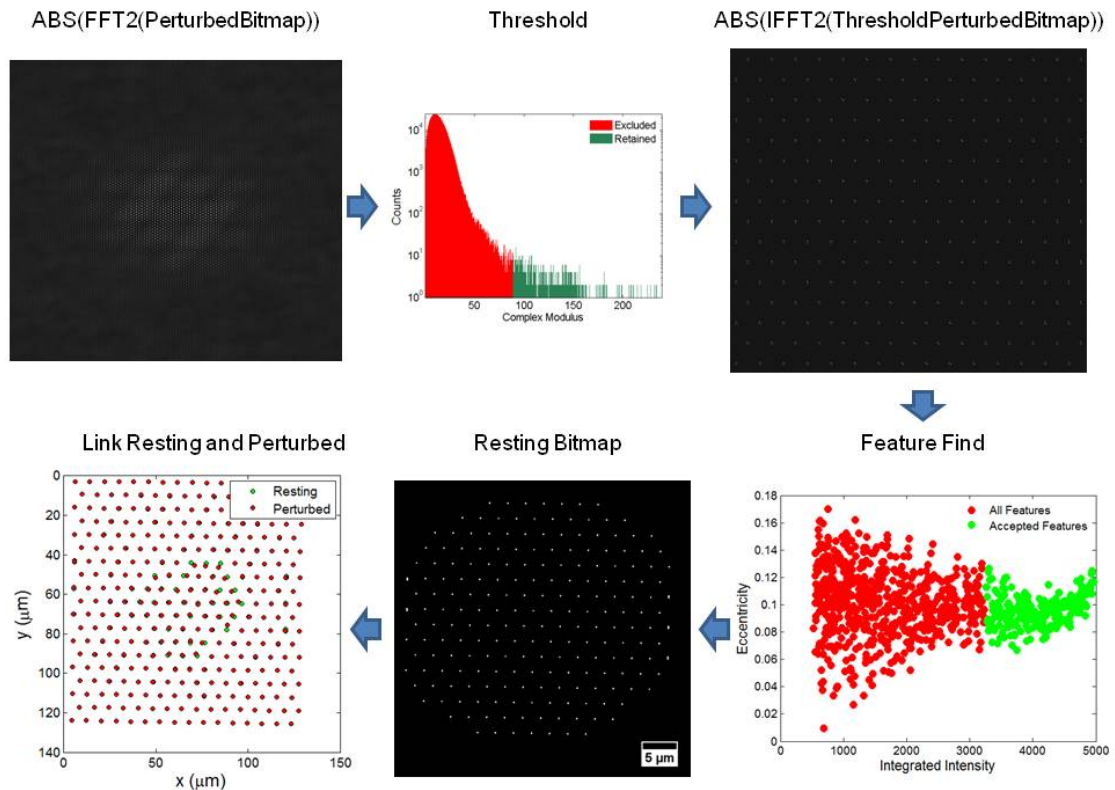


Figure B.1 Resting lattice bitmap reconstruction. This is the workflow associated with the reconstruction of a resting lattice bitmap from a perturbed bitmap. It consists of a two dimensional Fourier transform on the perturbed lattice bitmap followed by filtering on the histogram of complex moduli to retain the signal from the resting lattice. When a suitable threshold is empirically determined the inverse two dimensional Fourier transform is computed. Feature finding is performed to centroid the resting lattice positions and extraneous noise is excluded. Finally a nearest neighbours search is executed to position perturbed post positions relative to their resting lattice locations.

positioning of post trajectories relative to the resting lattice position is required to compute the deflection strain and therefore back calculate the force required to generate the deflection.

Methodology

1. Prepare data according to the nomenclature required by Kilfoil. A parent directory should contain a sequence of .tif images in a folder “fov#”. Images are labeled “fov#_####.tif”. For example “fov9” would refer to a folder corresponding to field of view 9 and “fov9_0037.tif” would be the filename of frame 37 at location 9. A time vector called “time” must be the same length as the number of images in the “fov#” folder and reside in the same parent directory as the “fov#” folder. The time vector is has a filename “fov#_times.mat”. Multiple “fov#” folders and “fov#_times.mat” vectors can reside within the same directory and can be processed simultaneously.
2. In MATLAB, run Kilfoil’s “mpretrack_init.m” to initialize the appropriate tracking parameters for the experimental data set such as particle shape, intensity, and size.
3. Using the tracking parameters established from “mpretrack_init.m” run Kilfoil’s “mpretrack.m”.
4. The output of “mpretrack.m” is a folder in the parent directory called “Feature_finding” containing a file called “MT_#_Feat_Size_#.mat”. For example “MT_9_Feat_Size_3.mat” would be the data array of all features of radius 3 pixels located in “fov9” images. The data array is the input of “fancytrack.m”.
5. In MATLAB, run Kilfoil’s “fancytrack.m”.
6. The output of “fancytrack.m” is a folder in the parent directory called “Bead_tracking” which contains a subfolder called “res_files.” This subfolder

contains a matrix called “res_fov#.mat” which holds the particle connectivities (i.e. trajectories) in successive frames.

7. In MATLAB, run “Post_Analysis_Driver_v9_thesis.m”
 - a. You will need to specify the following parameters:
 - i. Line 134: “exp_date” – 8 digit date of experiment (yyyymmdd)
 - ii. Line 137: “exp_donor” – donor ID string
 - iii. Line 140: “exp_cond” – string description of experimental condition
 - iv. Line 143: “fovn” – numeric field of view number
 - v. Line 146: “time_int” – time interval between frames in seconds
 - vi. Line 149: “max_num_frames” – maximum number of frames for fovn
 - vii. Line 152: “microntopix” – micron to pixel conversion in pixels/ μm
 - viii. Line 155: “kspring” – post spring constant in pN/nm
 - ix. Line 158: “basepath” – string path to parent directory containing fovn data
 - x. Line 166: “dilate” – numeric integer in pixels by which objects will be dilated to assist with visualizing post lattices
 - b. “Post_Analysis_Driver_v9_thesis.m” calls the following subroutines:
 - i. “SelectBackgroundPosts_v3.m” – You will be asked if you want to dedrift the data. You should always elect to dedrift or errors will be thrown because subsequent functions require the drift model. Select “New Model” and navigate to an image of the contractile cell. Draw a polygon around posts not beneath the cell. Double click when the polygon is closed to continue processing. The function will output a figure of the drift model in microns.
 - ii. “dedrifting_and_conversions_v3.m” (SJH adaptation of Kilfoil code)

1. “pixtomicro.m” (Kilfoil code) - The Kilfoil code does not receive the “microntopix” variable so you must adjust the hardcoded value manually.
2. “drift_loop_makedriftfrombkgr.m” (SJH adaptation of Kilfoil code)
 - a. From_8_columns_to_4 (Kilfoil code)
 - b. Motion.m (Kilfoil code)
3. “drift_loop_dedriftalldata.m” (SJH adaptation of Kilfoil code)
 - a. “dedrift.m (Kilfoil code)
 - b. “putting_in_missing_frames.m” (Kilfoil code)
- iii. “conversion_no_dd.m” (Kilfoil code)
- iv. “getting_individual_beads.m” (Kilfoil code)
- v. “Time_Avg_Centroids_v4.m” – This function will output plots of the time average centroid positions of each trajectory superimposed on the trajectories themselves. You should inspect these plots to see if trajectories must be merged.
- vi. “DedriftImageForOverlays_v1.m” – Navigate to a .tif image that will be used for subsequent superimposition of trajectories and post IDs.
- vii. “OverlayPostIDs_v5.m”
- viii. “Merge_Trajectories_v1.m” – If the plots from “Time_Avg_Centroids_v4.m” reveal trajectories that require merging specify which tracks should be joined. Repeat as necessary.
- ix. “PlotPostIDs_v1.m”
- x. “SetSuperResolutionRefinement_v1.m” – Set refinement value, this is an integer multiple by which you will increase the number of pixels.

- xi. “Construct_Bitmap_v4.m” -
- xii. “Find_Rotation_Angle_v1.m” – Specify the rotation angle or use the interactive feature to determine the angle.
- xiii. “Apply_Rotation_v3.m”
- xiv. “Crop_Bitmap_v2.m” – A user interface requires you to draw a rectangular ROI that achieves periodic boundary conditions. Double click inside ROI when finished to proceed with analysis.
- xv. “FilterForRestingLattice_v6.m” – An iterative process is run to determine a threshold on the histogram of complex moduli from a 2D Fourier transform of the perturbed lattice bitmap. The program pauses and allows you to probe the threshold value (“thres”) and manually change this value. You can assess the result of changing this value by executing lines 188-190, 204, 208-211, in the command window. The goal is to achieve a threshold setting (usually a value that retains the upper flat tail of the complex moduli histogram) so that the inverse 2D Fourier transform returns a resting lattice bitmap with bright and distinct features. When finished enter “return” in the command window.
- xvi. “UndoCrop_v1.m”
- xvii. “KilfoilInitialize_v3.m” – Iteratively supply parameters for feature identification of centroids in the resting lattice bitmap.
 - 1. “mpretrack_init.m” (Kilfoil code)
- xviii. “TrajectoriesRelativeToRestingLattice_v9.m”
- xix. “TrajectoriesInCellReferenceFrame_v5.m”- Double click on approximate geometric centroid of cell.

- xx. “IdentifyEngagedPosts_v4.m” – Draw a polygonal ROI around the diffuse cloud of high variance data points, excluding the tight cloud of low variance data points.
 - xxi. “ReviseCellRefTrajectories_v1.m”
 - xxii. “GeoSortEngagedPosts_v2.m” – Set the nearest neighbor distance (“<knn>”) in pixels to distinguish perimeter posts from core posts. This is the value that differentiates the first plateau of nearest neighbor distances from the second plateau.
 - xxiii. “RepopulatePostIDLists_v2.m”
 - xxiv. “PlotCellRefTrajectories_v7.m”
 - xxv. “PlotMetricsVsRadialDist_v7.m”
 - xxvi. “Plot_fvst_Strips_v1.m”
 - xxvii. “IndividualPostAutoCorrelation_v2.m”
 - xxviii. “Tidy_Up_v2.m”
8. The output from running “Post_Analysis_Driver_v9_thesis.m” is a “Post_Analysis_Driver_v9_thesis_fovn” folder. The numeric prefix to the folder is the ISO 8601 dateform (“yyymmddTHHMMSS”) on which the analysis was performed. The two subfolders of particular interest include “fig” which contains copies of all figures generated during the run and “lsx” which contains an Excel worksheet (MeanMetrics.xlsx) that summarizes the ensemble statistics computed during the run. A session workspace “SessionWorkspace.mat” is also saved which retains all the variables and arrays generated during the run.

Areas for Code Improvement

One area for improvement in the existing code is the localization of post trajectories relative to their resting lattice positions. Presently a simple nearest neighbors approach is employed. While this is satisfactory for small initial displacements relative to the resting lattice position when displacements are large the nearest neighbors approach leads to misidentification. A stronger strategy is to perform a seek analogous to an energy minimization scheme permuting all combinations of posts and resting lattice positions until the sum of the squared distances is minimized. This is the approach utilized in Crocker and Grier's particle tracking algorithm as well as Kilfoil's version of their routines. At a high level this improvement would involve a simple substitution of my "TrajectoriesRelativeToRestingLattice_v9.m" with Kilfoil's "fancytrack.m" with the necessary nomenclature and variable changes being satisfied.

An additional area for improvement is to move away from fixed data arrays and towards an object oriented programming scheme. In the former case great care must be taken to maintain array dimensionality and organization especially when passing arrays between functions. In the later case data is assigned properties and these properties can be used to retrieve data or aspects of data without as much overhead.

Code

Note: missing lines are version history annotation, removed for space considerations.

Post_Analysis_Driver_v9_thesis.m

```
1 % Steven J. Henry
2 % 02/17/2015
58 %*****
59 % PURPOSE:
60 % This driver calls a sequence of functions to dedrift post trajectories,
61 % locate post trajectories relative to their ideal resting lattice
62 % position, parses engaged from non-engaged posts, and computes associated
63 % statistics.
```



```

64 %
65 % ASSUMPTIONS:
66 % n/a
67 %
68 % INPUT:
69 % exp_date = 8 digit number date of experiment (yyyymmdd)
70 % exp_donor = donor ID string ('DXX')
71 % exp_cond = string describing experimental condition (e.g. 'Control')
72 % fovn = field of view number
73 % time_int = time interval between frames in seconds
74 % max_num_frames = maximum number of frames for fovn
75 % microntopix = microntopix conversion in pixels/um
76 % kspring = post spring constant in pN/um
77 % basepath = path to parent directory containing fovn data
78 % dilate = radius in pixels by which a single "on" pixel should be dilated
79 %   to assisted with visual inspection of lattice
80 %
81 % OUTPUT:
82 % A host of intermediate and final ensemble analysis metrics and figures.
83 %
84 % DRIVER/FUNCTION MAP:
85 % Level Name:
86 % 0   Post_Analysis_Driver_v9.m
87 % 1   SelectBackgroundPosts_v3.m
88 % 1   dedrifting_and_conversions_v3.m (SJH adaptation of Kilfoil)
89 % 2   pixtomicro.m (Kilfoil)
90 % 2   drift_loop_makedriftfrombkgr.m (SJH adaptation of Kilfoil)
91 % 3   from_8_columnsns_to_4.m (Kilfoil)
92 % 3   motion.m (Kilfoil)
93 % 2   drift_loop_dedriftingalldata (SJH adaptation of Kilfoil)
94 % 3   dedrift.m (Kilfoil)
95 % 3   putting_in_missing_frames.m (Kilfoil)
96 % 1   conversions_no_dd.m (Kilfoil)
97 % 1   getting_individual_beads.m (Kilfoil)
98 % 1   Time_Avg_Centroids_v4.m
99 % 1   DedriftImageForOverlays_v1.m
100 % 1  OverlayPostIDs_v5.m
101 % 1  Merge_Trajectories_v1.m
102 % 1  PlotPostIDs_v1.m
103 % 1  SetSuperResolutionRefinement_v1.m
104 % 1  Construct_Bitmap_v4.m
105 % 1  Find_Rotation_Angle_v1.m
106 % 1  Apply_Rotation_v3.m
107 % 1  Crop_Bitmap_v2.m
108 % 1  FilterForRestingLattice_v6.m
109 % 1  UndoCrop_v1.m
110 % 1  KilfoilInitialize_v3.m
111 % 2  mpretrack_init.m (Kilfoil)
112 % 1  TrajectoriesRelativeToRestingLattice_v9.m
113 % 1  TrajectoriesInCellReferenceFrame_v5.m
114 % 1  IdentifyEngagedPosts_v4.m
115 % 1  ReviseCellRefTrajectories_v1.m
116 % 1  GeoSortEngagedPosts_v2.m
117 % 1  RepopulatePostIDLists_v2.m
118 % 1  PlotCellRefTrajectories_v7.m
119 % 1  PlotMetricsVsRadialDis_v7.m
120 % 1  Plot_fvst_Strips_v1.m
121 % 1  IndividualPostAutoCorrelation_v2.m
122 % 1  Tidy_Up_v2.m
123 %*****
124
125 clc;
126 clear all;
127 close all;
128 pause(5);
129 tic;
130
131 %**** HARD CODED PARAMETERS START *****

```

```

132
133 % Set experiment data in YYYYMMDD format:
134 exp_date = #####;
135
136 % Set experimental donor in 'DXX' format:
137 exp_donor = 'DXX';
138
139 % Set experimental condition:
140 exp_cond = 'EnterConditionHere';
141
142 % Set fovn:
143 fovn = #;
144
145 % Set time between frames:
146 time_int = #; % (s/frame)
147
148 % Set maximum number of frames
149 max_num_frames = #;
150
151 % Have user specify the microntopix conversion:
152 microntopix = #; % pixels/um
153
154 % Set post spring constant
155 kspring = #; %pN/nm
156
157 % Set basepath to experimental folder
158 basepath = uigetdir(...
159     'EnterPathToDataHere',...
160     'Where does the data reside?');
161 basepath = [basepath '\'];
162
163 % Radius (pix) by which a single "on" pixel should be dilated using a
164 % 'disk' structuring element to assist with visual inspection of lattice
165 % manipulations
166 dilate = #;
167
168 %**** HARD CODED PARAMETERS END ****
169
170 % Have user specify where to save data
171 save_here = uigetdir(basepath,'Where should analysis be saved?');
172
173 % Determine date and time. Create a string in "dateform" "30" (ISO 8601)
174 % which has the format 'yyyymmddTHMMSS'.
175 dstr = datestr(now, 30);
176
177 % Create a folder to hold results of analysis:
178 func_name = mfilename;
179 results_folder_name = [dstr '_' func_name];
180 results_folder_path = [save_here '\ results_folder_name 'fov' ...
181     num2str(fovn)];
182 mkdir(results_folder_path);
183
184 % Set directory to analysis folder:
185 cd(results_folder_path);
186
187 % Start a log file. Save in new directory:
188 logfile = [results_folder_name '_Log.txt'];
189 fid = fopen(logfile,'wt');
190
191 % Update log file that function is running:
192 fprintf(1,'Analysis commenced: %s\n',dstr);
193 fprintf(fid,'Analysis commenced: %s\n',dstr);
194
195 fprintf(1,'\nResults folder path:\n%s\n',results_folder_path);
196 fprintf(fid,'\nResults folder path:\n%s\n',results_folder_path);
197
198 fprintf(1,'\n%s running ...'\n',func_name);
199 fprintf(fid,'\n%s running ...'\n',func_name);

```

```

200
201 % Record hard-coded values:
202 fprintf(1, '\nHard-coded values for this run:\n');
203 fprintf(fid, '\nHard-coded values for this run:\n');
204
205 fprintf(1, '\texp_date = %s \n', num2str(exp_date));
206 fprintf(fid, '\texp_date = %s \n', num2str(exp_date));
207
208 fprintf(1, '\texp_donor = %s \n', num2str(exp_donor));
209 fprintf(fid, '\texp_donor = %s \n', num2str(exp_donor));
210
211 fprintf(1, '\texp_cond = %s \n', num2str(exp_cond));
212 fprintf(fid, '\texp_cond = %s \n', num2str(exp_cond));
213
214 fprintf(1, '\tfovn = %s \n', num2str(fovn));
215 fprintf(fid, '\tfovn = %s \n', num2str(fovn));
216
217 fprintf(1, '\tmicrontopix = %s um/pix\n', num2str(microntopix));
218 fprintf(fid, '\tmicrontopix = %s um/pix\n', num2str(microntopix));
219
220 fprintf(1, '\ttime_int = %s s/frame\n', num2str(time_int));
221 fprintf(fid, '\ttime_int = %s s/frame\n', num2str(time_int));
222
223 fprintf(1, '\tkspring = %s pN/nm\n', num2str(kspring));
224 fprintf(fid, '\tkspring = %s pN/nm\n', num2str(kspring));
225
226 fprintf(1, '\tmax_num_frames = %s \n', num2str(max_num_frames));
227 fprintf(fid, '\tmax_num_frames = %s \n', num2str(max_num_frames));
228
229 fprintf(1, '\tdilate = %s pix (radius)\n', num2str(dilate));
230 fprintf(fid, '\tdilate = %s pix (radius)\n', num2str(dilate));
231 % end record hard-coded values
232
233 % Do you want to dedrift the data?
234 % If you elect not to dedrift you will produce errors later in functions
235 % that depend upon the output of dedrifting, namely the 'driftmodel_um' and
236 % 'Idedrift' variables.
237 dedrift_prechoice = menu('Dedrifting the data?', 'Yes', 'No');
238 if dedrift_prechoice == 1
239     % Dedrift data. It's critical that smoo = 0 or correlation will be
240     % introduced in the data.
241     dedrift_choice = menu('Dedrifting with:', 'Existing model', 'New model');
242     if dedrift_choice == 1
243         [drift_name, drift_path] = uigetfile([basepath '*.mat'], ...
244         'Select existing drift model in microns');
245         load([drift_path drift_name]);
246         [bkgr_name, bkgr_path] = uigetfile([basepath '*.mat'], ...
247         'Select existing list of background post IDs');
248         load([bkgr_path bkgr_name]);
249         [driftmodel_um] = dedrifting_and_conversions_v3(...
250         basepath, fovn, driftmodel_um);
251     elseif dedrift_choice == 2
252         [bkgr_post_ids] = SelectBackgroundPosts_v3(...
253         basepath, fovn, results_folder_path, fid);
254         [driftmodel_um] = dedrifting_and_conversions_v3(basepath, fovn, []);
255     end
256     save([results_folder_path filesep 'driftmodel_um.mat'], ...
257     'driftmodel_um');
258     save([results_folder_path filesep 'bkgr_post_ids.mat'], ...
259     'bkgr_post_ids');
260 elseif dedrift_prechoice == 2
261     conversions_no_dd( basepath, fovn);
262 end
263
264 % Construct individual bead trajectories
265 getting_individual_beads(basepath, fovn);
266
267 % Move 'ddposum_files' folder within results_folder_path:

```

```

268 source = [basepath 'Bead_tracking' filesep 'ddposum_files'];
269 destination = [results_folder_path filesep 'ddposum_files'];
270 movefile(source,destination);
271
272 % Copy 'individual_beads' and rename to 'beads'
273 source = [results_folder_path filesep 'ddposum_files' filesep ...
274   'individual_beads'];
275 bead_path = [results_folder_path filesep 'beads'];
276 copyfile(source,bead_path);
277
278 % Select bead.mat files to process and compute time-average centroids
279 [TAC,FOC] = Time_Avg_Centroids_v4(bead_path,microntopix,...
280   results_folder_path,fid);
281
282 % Select an image that will be used for subsequent overlays and to
283 % determine the cell geometric centroid. This image will require
284 % "dedrifting" so future overlays properly align.
285 [ldrift,raw_dim] = DedriftImageForOverlays_v1(...
286   basepath,fov,microntopix,driftmodel_um,results_folder_path,fid);
287
288 % Output an image with a post-ID overlay:
289 cat = 'allposts';
290 color = [0,0,0.8];
291 OverlayPostIDs_v5(bead_path,ldrift,cat,color,microntopix,...
292   results_folder_path,fid);
293
294 % Ask user if he wants to merge trajectories belonging to the same post:
295 merge_choice = menu('Merge trajectories?','Yes','No');
296
297 if merge_choice == 1
298
299   % Make a copy of the original results of getting_individual_beads.m
300   mkdir(bead_path,'premerge_beads');
301   copyfile(bead_path,[bead_path filesep 'premerge_beads']);
302
303   exitflag = 1;
304   while exitflag ~= 2
305
306     % Merge trajectories that belong to the same post:
307     Merge_Trajectories_v1(bead_path,fid);
308
309     % Generate an intermediate plot
310     PlotPostIDs_v1(bead_path,microntopix,results_folder_path,fid);
311
312     exitflag = menu('Merge again?','Merge again','Finished merging');
313
314   end
315
316   % Select bead.mat files to process and compute time-average centroids
317   [TAC,FOC] = Time_Avg_Centroids_v4(bead_path,microntopix,...
318     results_folder_path,fid);
319
320   % Output an image with a post-ID overlay:
321   cat = 'allposts_aftermerge';
322   OverlayPostIDs_v5(bead_path,ldrift,cat,color,microntopix,...
323     results_folder_path,fid);
324
325 end
326
327 close('all');
328
329 % Have user specify how much to super-sample the existing bitmap:
330 [refine] = SetSuperResolutionRefinement_v1(results_folder_path,fid);
331
332 % Construct a binary bitmap using time average centroids.
333 [Ap,Ap_LUT] = Construct_Bitmap_v4(TAC,raw_dim,refine,fid);
334 % Plot bitmap:
335 fig_handle_Ap_dilated = figure;

```

```

336 imshow(imdilate(Ap,strel('disk',dilate)), 'DisplayRange', []);
337 title_string = [num2str(dilate) ...
338 'X dilated bitmap of time averaged centroids positioned to '...
339 num2str(1/refine) ' pix'];
340 title(title_string, 'FontSize', 20, 'Interpreter', 'None');
341 %Save figure and arrays:
342 saveas(fig_handle_Ap_dilated, [results_folder_path ...
343 '\Bitmap_Perturbed_Dilated.fig'], 'fig');
344 save([results_folder_path filesep 'Bitmap_Perturbed.mat'], 'Ap');
345 save([results_folder_path filesep 'Bitmap_Perturbed_LUT.mat'], ...
346 'Ap_LUT');
347
348 % Rotate binary bitmap containing time average centroids so one row is
349 % (approximately) parallel with vertical axis.
350 rotate_choice = menu('Rotate by:', 'Entering rotation', 'Finding rotation');
351 if rotate_choice == 1
352     theta = input('Set theta (rotation angle and sign) = ');
353 elseif rotate_choice == 2
354     [theta] = Find_Rotation_Angle_v1(Ap, results_folder_path, fid);
355 end
356 save([results_folder_path filesep 'theta.mat'], 'theta');
357
358 [PXr, PYr] = Apply_Rotation_v3(TAC(:, 2), TAC(:, 3), theta, ...
359 results_folder_path, fid, raw_dim);
360
361 [Ap_rotated, Ap_rotated_LUT] = Construct_Bitmap_v4(horzcat(TAC(:, 1), ...
362 PXr, PYr), raw_dim, refine, fid);
363
364 % Plot bitmap:
365 fig_handle_Ap_rotated_dilated = figure;
366 imshow(imdilate(Ap_rotated, strel('disk', dilate)), 'DisplayRange', []);
367 title_string = [num2str(dilate) 'X dilated bitmap of rotated time averaged centroids positioned to ' num2str(1/refine)
' pix'];
368 title(title_string, 'FontSize', 20, 'Interpreter', 'None');
369 %Save figure and array:
370 saveas(fig_handle_Ap_rotated_dilated, [results_folder_path ...
371 '\Bitmap_Perturbed_Rotated_Dilated.fig'], 'fig');
372 save([results_folder_path filesep ...
373 'Bitmap_Perturbed_Rotated.mat'], 'Ap_rotated');
374 save([results_folder_path filesep ...
375 'Bitmap_Perturbed_Rotated_LUT.mat'], 'Ap_rotated_LUT');
376
377 % Crop rotated bitmap so boundaries are (approximately) periodic:
378 [Ap_cropped, rverts, pad] = Crop_Bitmap_v2(Ap_rotated, dilate, ...
379 results_folder_path, fid);
380
381 % Filter for resting lattice
382 pts = 5;
383 method = 'auto';
384 [Ar] = FilterForRestingLattice_v6(Ap_cropped, pts, method, ...
385 results_folder_path, fid);
386
387 % Undo crop:
388 [Ar_uncropped] = UndoCrop_v1(Ar, pad, dilate, results_folder_path, fid);
389
390 % Save Ar_uncropped in format necessary for Kilfoil feature finding:
391 mkdir([results_folder_path '\fov0']);
392 imwrite(Ar_uncropped, [results_folder_path '\fov0\fov0_0000.tif']);
393 time = 0;
394 save([results_folder_path '\fov0_times.mat'], 'time');
395
396 % Perform initialize.m to set necessary parameters for Kilfoil feature
397 % finding:
398 [M2, MT, KilfoilRestingLatticeParameters] = KilfoilInitialize_v3(0, 0, ...
399 results_folder_path, fid);
400 dim = size(Ar_uncropped);
401 [RX, RY] = Apply_Rotation_v3(MT(:, 1), MT(:, 2), -theta, ...
402 results_folder_path, fid, dim);

```

```

403
404 % Step down resolution:
405 RX = RX/refine;
406 RY = RY/refine;
407 [Ar,Ar_LUT] = Construct_Bitmap_v4(horzcat((1:size(MT(:,1),1)),RX,RY),...
408   raw_dim,refine,fid);
409 % Plot bitmap:
410 fig_handle_Ar_dilated = figure;
411 imshow(imdilate(Ar,strel('disk',dilate)), 'DisplayRange',[]);
412 title_string = [num2str(dilate) ...
413   'X dilated bitmap of resting lattice centroids positioned to ' ...
414   num2str(1/refine) ' pix'];
415 title(title_string,'FontSize',20,'Interpreter','None');
416 %Save figure and array:
417 saveas(fig_handle_Ar_dilated, [results_folder_path ...
418   '\Bitmap_Resting_Dilated.fig', 'fig');
419 save([results_folder_path filesep 'Bitmap_Resting.mat'], 'Ar');
420 save([results_folder_path filesep 'Bitmap_Resting_LUT.mat'], 'Ar_LUT');
421
422 [Apo,Apo_LUT] = Construct_Bitmap_v4(FOC,raw_dim,refine,fid);
423 % Plot bitmap:
424 fig_handle_Apo_dilated = figure;
425 imshow(imdilate(Apo,strel('disk',dilate)), 'DisplayRange',[]);
426 title_string = [num2str(dilate) ...
427   'X dilated bitmap of frame 1 perturbed lattice centroids ' ...
428   num2str(1/refine) ' pix'];
429 title(title_string,'FontSize',20,'Interpreter','None');
430 %Save figure and array:
431 saveas(fig_handle_Apo_dilated, [results_folder_path ...
432   '\Bitmap_Perturbed_Frame1_Dilated.fig', 'fig');
433 save([results_folder_path filesep ...
434   'Bitmap_Perturbed_Frame1.mat'], 'Apo');
435 save([results_folder_path filesep ...
436   'Bitmap_Perturbed_Frame1_LUT.mat'], 'Apo_LUT');
437
438 imwrite(imdilate(Ap,strel('disk',dilate)), 'Ap_tavg_dil.tif');
439 imwrite(imdilate(Apo,strel('disk',dilate)), 'Ap_frame1_dil.tif');
440 imwrite(imdilate(Ar,strel('disk',dilate)), 'Ar_dil.tif');
441
442 imwrite(Ap, 'Ap_tavg.tif');
443 imwrite(Apo, 'Ap_frame1.tif');
444 imwrite(Ar, 'Ar.tif');
445
446 close('all');
447
448 % Update bead.mat files so all trajectories are relative to their resting
449 % lattice position:
450 [shift_Ap,sbead_path] = TrajectoriesRelativeToRestingLattice_v9(...
451   Ar,Apo_LUT,microntopix,refine,bead_path,results_folder_path,fid);
452
453 % Translate trajectories into the cell reference frame
454 [manual_centroid,rsbead_manualcentroid_path]...
455   = TrajectoriesInCellReferenceFrame_v5(sbead_path,microntopix,...
456   ldedrift,results_folder_path,fid);
457
458 % Sort on post trajectories to distinguish cell-engaged from non-engaged
459 % posts:
460 [engaged_post_IDs,ens_manualcentroid_rsbead_path]...
461   = IdentifyEngagedPosts_v4(bead_path,rsbead_manualcentroid_path,...
462   microntopix,ldedrft,results_folder_path,fid);
463
464 % Pause to make manual adjustments?
465 pause_choice = menu('Make manual changes?:','Yes','No');
466 if pause_choice == 1
467   keyboard;
468 end
469
470 % Revise the cell reference frame trajectories using the true geometric

```

```

471 % centroid of the ensemble of engaged posts:
472 [centroid,rsbead_path,ens_rsbead_path] = ReviseCellRefTrajectories_v1(...
473     sbead_path,engaged_post_IDs,microntopix,results_folder_path,fid);
474
475 % Perform geometric sort of engaged posts to bin those that reside at the
476 % cell edge vs. those that reside at the core:
477 [core_post_IDs, core_rsbead_path, perim_post_IDs, perim_rsbead_path]...
478     = GeoSortEngagedPosts_v2(sbead_path,rsbead_path,microntopix,...
479     engaged_post_IDs,ldedrift,results_folder_path,fid);
480
481 % Pause to make manual adjustments?
482 pause_choice = menu('Make manual changes?:','Yes','No');
483 if pause_choice == 1
484     keyboard;
485     % If manual adjustments have been made you need to repopulate the list
486     % of post_IDs belonging to each category
487     [engaged_post_IDs, core_post_IDs, perim_post_IDs]...
488     = RepopulatePostIDLists_v2(ens_rsbead_path,core_rsbead_path,...
489     perim_rsbead_path,results_folder_path,fid);
490 end
491
492 cat = 'ens';
493 color = [0,0,0.8];
494 OverlayPostIDs_v5(bead_path,ldedrift,cat,color,microntopix,...
495     results_folder_path,fid,engaged_post_IDs);
496
497 transition_time = input(['Set ' cat ...
498     ' time (s) after Fmax to consider system "steady state" = ']);
499 fprintf(1,...
500     '\nFor %s category user said steady state is %s (s) post peak\n',...
501     cat,num2str(round(transition_time)));
502 fprintf(fid,...
503     '\nFor %s category user said steady state is %s (s) post peak\n',...
504     cat,num2str(round(transition_time)));
505
506 [ens_mpara,ens_sdpara,ens_cpara,ens_separa,...
507     ens_mperp,ens_sdperp,ens_cperp,ens_seperp,...
508     ens_mpara_F,ens_sdpara_F,ens_cpara_F,ens_separa_F,...
509     ens_mperp_F,ens_sdperp_F,ens_cperp_F,ens_seperp_F,...
510     ens_t,ens_Fmax,ens_Fss]...
511     = PlotCellRefTrajectories_v7(ens_rsbead_path,time_int,...
512     kspring,cat,color,results_folder_path,fid,transition_time);
513
514 [ens_radial_metrics] = PlotMetricsVsRadialDist_v7(...
515     exp_date,exp_donor,exp_cond,fovn,...
516     sbead_path,centroid,microntopix,...
517     transition_time,cat,color,results_folder_path,fid,...
518     ens_Fmax,ens_Fss,ens_t,ens_mpara_F);
519
520 Plot_fvst_Strips_v1(rsbead_path,ens_radial_metrics,kspring,...
521     time_int,cat,color,results_folder_path,fid);
522
523 IndividualPostAutocorrelation_v2(rsbead_path,engaged_post_IDs,...
524     max_num_frames,time_int,cat,color,results_folder_path,fid);
525
526 cat = 'core';
527 color = [0.17, 0.51, 0.34];
528 OverlayPostIDs_v5(bead_path,ldedrift,cat,color,microntopix,...
529     results_folder_path,fid,core_post_IDs);
530
531 choice = menu('Set new transition time?','Yes',...
532     ['Use current value (' num2str(transition_time) 's)']);
533 if choice == 1
534     transition_time = input(['Set ' cat ...
535     ' time (s) after Fmax to consider system "steady state" = ']);
536 end
537
538 fprintf(1,...

```

```

539     '\nFor %s category user said steady state is %s (s) post peak\n',...
540     cat,num2str(round(transition_time)));
541 fprintf(fid,...
542     '\nFor %s category user said steady state is %s (s) post peak\n',...
543     cat,num2str(round(transition_time)));
544
545 [core_mpara,core_sdpara,core_cpara,core_separa,...
546     core_mperp,core_sdperp,core_cperp,core_seperp,...
547     core_mpara_F,core_sdpara_F,core_cpara_F,core_separa_F,...
548     core_mperp_F,core_sdperp_F,core_cperp_F,core_seperp_F,...
549     core_t,core_Fmax,core_Fss]...
550 = PlotCellRefTrajectories_v7(core_rsbead_path,time_int,...
551     kspring,cat,color,results_folder_path,fid,transition_time);
552
553 [core_radial_metrics] = PlotMetricsVsRadialDist_v7(...
554     exp_date,exp_donor,exp_cond,fov, ...
555     sbead_path,centroid,microntopix,...
556     transition_time,cat,color,results_folder_path,fid,...
557     core_Fmax,core_Fss,core_t,core_mpara_F);
558
559 Plot_fvst_Strips_v1(rsbead_path,core_radial_metrics,kspring,...
560     time_int,cat,color,results_folder_path,fid);
561
562 IndividualPostAutocorrelation_v2(rsbead_path,core_post_IDs,...
563     max_num_frames,time_int,cat,color,results_folder_path,fid);
564
565 % Start Perim analysis
566 cat = 'perim';
567 color = 'r';
568 OverlayPostIDs_v5(bead_path,ldedrft,cat,color,microntopix,...
569     results_folder_path,fid,perim_post_IDs);
570
571 choice = menu('Set new transition time?','Yes',...
572     ['Use current value (' num2str(transition_time) 's)']);
573 if choice == 1
574     transition_time = input(['Set ' cat ...
575         ' time (s) after Fmax to consider system "steady state" = ']);
576 end
577
578 fprintf(1,...
579     '\nFor %s category user said steady state is %s (s) post peak\n',...
580     cat,num2str(round(transition_time)));
581 fprintf(fid,...
582     '\nFor %s category user said steady state is %s (s) post peak\n',...
583     cat,num2str(round(transition_time)));
584
585 [perim_mpara,perim_sdpara,perim_cpara,perim_separa,...
586     perim_mperp,perim_sdperp,perim_cperp,perim_seperp,...
587     perim_mpara_F,perim_sdpara_F,perim_cpara_F,perim_separa_F,...
588     perim_mperp_F,perim_sdperp_F,perim_cperp_F,perim_seperp_F,...
589     perim_t,perim_Fmax,perim_Fss]...
590 = PlotCellRefTrajectories_v7(perim_rsbead_path,time_int,...
591     kspring,cat,color,results_folder_path,fid,transition_time);
592
593 [perim_radial_metrics] = PlotMetricsVsRadialDist_v7(...
594     exp_date,exp_donor,exp_cond,fov, ...
595     sbead_path,centroid,microntopix,...
596     transition_time,cat,color,results_folder_path,fid,...
597     perim_Fmax,perim_Fss,perim_t,perim_mpara_F);
598
599 Plot_fvst_Strips_v1(rsbead_path,perim_radial_metrics,kspring,...
600     time_int,cat,color,results_folder_path,fid);
601
602 IndividualPostAutocorrelation_v2(rsbead_path,perim_post_IDs,...
603     max_num_frames,time_int,cat,color,results_folder_path,fid)
604
605 % Update log file that function is completed:
606 elapsed = toc/60; % minutes, default is seconds

```



```

607 dstr2 = datestr(now, 30);
608 fprintf(1, '\nAnalysis concluded: %s\n', dstr2);
609 fprintf(fid, '\nAnalysis commenced: %s\n', dstr2);
610
611 fprintf(1, '\nElapsed time = %s min\n', num2str(elapsed));
612 fprintf(fid, '\nElapsed time = %s min\n', num2str(elapsed));
613
614 fclose(fid);
615
616 % Sort all files generated into folders of .fig, .mats, and .txt files
617 % leaving the master log file residing outside the three folders:
618 cd(results_folder_path);
619 Tidy_Up_v2(logfile);
620
621 save([results_folder_path filesep 'SessionWorkspace.mat']);

```

SelectBackgroundPosts_v3.m

```

1 % Steven J. Henry
2 % 07/23/2014
14 %*****
15 % PURPOSE:
16 % This function establishes the set of posts in a FOV that are known
17 % background posts for use in constructing a drift model.
18 %
19 % ASSUMPTIONS:
20 % n/a
21 %
22 % INPUT:
23 % basepath = path to parent directory containing fovn data
24 % fovn = field of view number
25 % save_path = path to folder holding analysis results
26 % fid = log file identifier
27 %
28 % OUTPUT:
29 % bkgr_pos_ids = a vector of post IDs designated as "background" post
30 % because they reside within user's drawn polygon ROI.
31 %
32 % DRIVER/FUNCTION MAP:
33 % n/a (calls no subroutines)
34 %*****
35
36 function [bkgr_post_ids] = SelectBackgroundPosts_v3(basepath, fovn, ...
37     save_path, fid)
38
39
40 % Get function name:
41 func_name = mfilename;
42
43 % Update log file that function is running:
44 fprintf(1, '\n%s running ... \n', func_name);
45 fprintf(fid, '\n%s running ... \n', func_name);
46
47 % Open a frame and select region not containing cell
48 [img_FileName, img_PathName] = uigetfile([basepath '* .tif'], ...
49     'Select .tif image to outline background posts', 'MultiSelect', 'off');
50
51 % Draw a region on the cell image that EXCLUDES the cell (i.e. that
52 % includes known background posts)
53 fig_handle_impoly = figure;
54 imshow([img_PathName img_FileName]);
55 title('Outline region containing background posts, excluding cell(s)');
56 fprintf(1, ...
57     '\tOutline region containing background posts, excluding cell(s)\n');
58 fprintf(fid, ...
59     '\tOutline region containing background posts, excluding cell(s)\n');
60 h = impoly;
61 verts = wait(h);

```

```

62
63 %Make a copy of the figure for superimposing trajectories:
64 h1=gcf;
65 fig_handle_impoly2 = figure;
66 objects=allchild(h1);
67 copyobj(get(h1,'children'),fig_handle_impoly2);
68 colormap(gray);
69
70 % Open res file that corresponds to this iamge:
71 % Open a frame and select region containing cell
72 [res_FileName,res_PathName] = uigetfile([basepath '*.mat'],...
73 'Select res.mat file corresponding to image.','MultiSelect','off');
74 load([res_PathName,res_FileName]);
75
76 % Identify the bead ID #s in res.mat that belong to these background posts
77 % and create a bkgr_res.mat file:
78 post_list = unique(res(:,8)); %#ok<NODEF>
79 num_posts = length(post_list);
80 bkgr_keep_list = false(size(res,1),1);
81 for i = 1:num_posts
82     if rem(i,50)==0
83         fprintf(1,'\tProcessing post %s of %s\n',...
84             num2str(i),num2str(num_posts));
85         fprintf(fid,'\tProcessing post %s of %s\n',...
86             num2str(i),num2str(num_posts));
87     end
88     post = post_list(i);
89     post_id = res(:,8)==post;
90     x = res(post_id,1);
91     y = res(post_id,2);
92     IN = inpolygon(x(1),y(1),verts(:,1),verts(:,2));
93     if IN == 1
94         bkgr_keep_list(post_id) = true;
95     end
96 end
97 res_bkgr = res(bkgr_keep_list,:);
98 save([res_PathName,'res_fov',num2str(fov),'_bkgr.mat'],'res_bkgr');
99
100 bkgr_post_ids = unique(res_bkgr(:,8));
101
102 if exist('fig_handle_impoly','var')==1
103     saveas(fig_handle_impoly,[save_path filesep img_FileName(1:end-4) ...
104         '_user_impoly.fig'])
105 end
106
107 close(fig_handle_impoly,fig_handle_impoly2);
108
109 % Update log file that function is completed:
110 fprintf(1,'%s completed\n',func_name);
111 fprintf(fid,'%s completed\n',func_name);
112
113 end

```

dedrifting_and_conversions_v3.m (SJH adaptation of Kilfoil code)

```

1 % Steven J. Henry
2 % 02/17/2015
7 %*****
8 % PURPOSE:
9 % This is a modification of Kilfoil's "dedrifting_and_conversions.m" to
10 % establish the drift model (center of mass motion) of the ensemble of
11 % background (non-cell engaged) posts from all post trajectories.
12 %
13 % ASSUMPTIONS:
14 % n/a
15 %
16 % INPUT:
17 % basepath = path to parent directory containing fovn data

```

```

18 % take = field of view number
19 % save_path = path to folder holding analysis results
20 % driftmodel_um = received as an empty vector []
21 %
22 % OUTPUT:
23 % dr_um = drift model in microns of center of mass motion of ensemble of
24 % particles (posts)
25 %
26 % DRIVER/FUNCTION MAP:
27 % 0 dedrifting_and_conversions_v3.m (SJH adaptation of Kilfoil)
28 % 1 pxtomicro.m (Kilfoil)
29 % 1 drift_loop_makedriftfrombkgr.m (SJH adaptation of Kilfoil)
30 % 1 drift_loop_dedriftingalldata (SJH adaptation of Kilfoil)
31 %*****
32
33 function(dr_um) = dedrifting_and_conversions_v3(basepath,take,...
34 driftmodel_um)
35
36 pathout = ([basepath 'Bead_tracking\ddposum_files\']);
37 [status,message,messageid] = mkdir( pathout );
38
39 if isempty(driftmodel_um)==1
40 % Load res_bkgr.mat
41 load( [basepath 'Bead_Tracking\res_files\res_fov' num2str(take) ...
42 '_bkgr.mat'] );
43
44 % Convert to microns from pixels
45 res_bkgr_um = pxtomicro(res_bkgr);
46
47 % Create drift vectors in microns
48 dr_um = drift_loop_makedriftfrombkgr(res_bkgr_um);
49
50 print('-dpng','-r150', [pathout 'dedrift_run' num2str(take) '.png'])
51
52 else
53 dr_um = driftmodel_um;
54 end
55
56 % Load res.mat
57 load( [basepath 'Bead_Tracking\res_files\res_fov' num2str(take) '.mat'] );
58
59 % Convert to microns from pixels
60 res_um = pxtomicro(res);
61
62 % Apply drift model to all posts in res
63 ddposum = drift_loop_dedriftingalldata(res_um,dr_um);
64
65 % Save dedrift data
66 save( [pathout 'ddposum_run' num2str(take) '.mat'], 'ddposum' );
67
68 end

```

drift_loop_makedriftfrombkgr.m (SJH adaptation of Kilfoil code)

```

1 % Steven J. Henry
2 % 02/17/2015
4 %*****
5 % PURPOSE:
6 % This is a modification of Kilfoil's "drift_loop.m" to establish drift
7 % model of background (non-cell engaged) posts only.
8 %
9 % ASSUMPTIONS:
10 % n/a
11 %
12 % INPUT:
13 % res_bkgr = res file from track.m that only contains user-specified
14 % background posts. Filter was done prior to passing to this function.
15 %

```

```

16 % OUTPUT:
17 % dr_um = drift model in microns of center of mass motion of ensemble of
18 % particles (posts)
19 %
20 % DRIVER/FUNCTION MAP:
21 % 0 drift_loop_makedriftfrombkgr.m (SJH adaptation of Kilfoil)
22 % 1 from_8_columns_to_4.m (Kilfoil)
23 % 1 motion.m (Kilfoil)
24 %*****
25
26 function[dr_um] = drift_loop_makedriftfrombkgr(res_bkgr)
27
28 % This part makes a matrix of 4 columns that is more convenient.
29 poss = from_8_columns_to_4(res_bkgr);
30
31 % Finds the center of mass motion for each frame.
32 drift=motion(poss, [1 0], 2);
33 npts=length(drift(:,1));
34 t=drift(:,1);
35 dr(:,1)=mot_eintegrate(drift(:,3));
36 dr(:,2)=mot_eintegrate(drift(:,4));
37
38 figure
39 plot( t, dr )
40 xlabel('frame','fontsize',16)
41 ylabel('drift (blue x, green y) (\mum)','fontsize',16)
42 set(gca,'fontsize',16,...
43 'TickLength',[0.025 0.06]);
44
45 % Pass out a variable called dr_um = drift in microns SJH 03/06/2014
46 dr_um = dr;

```

drift_loop_dedrftalldata.m (SJH adaptation of Kilfoil code)

```

1 % Steven J. Henry
2 % 02/17/2015
4 %*****
5 % PURPOSE:
6 % This is a modification of Kilfoil's "drift_loop.m" to perform subtraction
7 % of driftmodel established from background (non-cell engaged) posts to all
8 % posts.
9 %
10 % ASSUMPTIONS:
11 % n/a
12 %
13 % INPUT:
14 % res = res file from track.m that only contains all post data in this
15 % fovn.
16 % dr = drift model in microns of center of mass motion of ensemble of
17 % particles (posts)
18 %
19 % OUTPUT:
20 % rdf2 = dedrifted res matrix for all data with interpolated missing
21 % frames. JCC does not like this inclusion and says interpolation is not
22 % necessary. It appears you could just pass out 'rdf' instead of 'rdf2'.
23 %
24 % DRIVER/FUNCTION MAP:
25 % 0 drift_loop_dedrftalldata.m (SJH adaptation of Kilfoil)
26 % 1 from_8_columns_to_4.m (Kilfoil)
27 % 1 dedrift.m (Kilfoil)
28 % 1 putting_in_missing_frames.m (Kilfoil)
29 % 1 motion.m (Kilfoil)
30 %*****
31
32 function[rdf2] = drift_loop_dedrftalldata(res,dr)
33
34 poss = from_8_columns_to_4(res);
35

```

```

36 rdf=dedrift(poss, dr);
37
38 rdf2 = putting_in_missing_frames(rdf);
39
40 end

```

Time_Avg_Centroids_v4.m

```

1  % Steven J. Henry
2  % 09/20/2014
23 %*****
24 % PURPOSE:
25 % This function asks the user to select bead/particle/post trajectories
26 % .mat files and computes the time average centroid of each post in lab
27 % reference frame.
28 %
29 % ASSUMPTIONS:
30 % n/a
31 %
32 % INPUT:
33 % basepath = string pointing to location of experimental data and analysis
34 % microntopix = objective calibration for conversion of microns to pixels
35 % save_path = string pointing to folder containing analysis results
36 % fid = file ID of log file to which progress is recorded
37 %
38 % OUTPUT:
39 % TAC = three column array with total number of rows equal to the number of
40 % user specified trajectory .mat files
41 % col 1 = post number (an integer value)
42 % col 2 = time average x-coordinate of centroid (pix)
43 % col 3 = time average y-coordinate of centroid (pix)
44 % FOC = three column array with total number of rows equal to the number of
45 % user specified trajectory .mat files
46 % col 1 = post number (an integer value)
47 % col 2 = frame 1 x-coordinate of centroid (pix)
48 % col 3 = frame 1 y-coordinate of centroid (pix)
49 %
50 % DRIVER/FUNCTION MAP:
51 % n/a (calls no subroutines)
52 %*****
53
54 function [TAC,FOC] = Time_Avg_Centroids_v4(bead_path,microntopix,...
55     save_path,fid)
56
57 % Get function name:
58 func_name = mfilename;
59
60 % Update log file that function is running:
61 fprintf(1,'\n%s running ...\n',func_name);
62 fprintf(fid,'\n%s running ...\n',func_name);
63
64 % Retrieve all contents that reside inside 'bead_path' folder:
65 contents = dir(bead_path);
66
67 num_items = size(contents,1);
68 FileName = cell(num_items,1);
69 ID = NaN(num_items,1);
70 keep_ind = false(num_items,1);
71
72 for i = 1:num_items
73     item_name = contents(i).name;
74     if length(item_name) >= 10 && strcmp(item_name(1:5),'bead_')
75         FileName(i) = item_name;
76         ID(i) = str2double(item_name(6:end-4));
77         keep_ind(i) = true;
78     end
79 end
80

```

```

81 FileName = FileName(keep_ind);
82 ID = ID(keep_ind);
83
84 % How many post files did user select?
85 num_posts = length(FileName);
86
87 % Reserve memory
88 TAC = zeros(num_posts,3);
89 FOC = TAC;
90
91 % Images in Matlab have a reversed y-axis. The origin (0,0) is located in
92 % the NorthWest corner, not the SoutWest corner. As such if we plot the
93 % y-values directly the plot will be upside down. Note: This is only the
94 % case when you're plotting on empty axes. When superimposing trajectories
95 % onto an image the coordinate axes are correct and therefore the mapping
96 % of the trajectories onto them is fine.
97
98 fig_handle_trajectories = figure;
99 labels = cell(num_posts,1);
100 for i = 1:num_posts
101
102     if i == 1 || rem(i,50)==0 || i==num_posts
103         fprintf(1,'\tEvaluating post trajectory %s of %s\n',...
104             num2str(i),num2str(num_posts));
105         fprintf(fid,'\tEvaluating post trajectory %s of %s\n',...
106             num2str(i),num2str(num_posts));
107     end
108
109     load([bead_path filesep FileName(i)]);
110
111     x = bsec(:,1)*microntopix; %#ok<NODEF>
112     y = bsec(:,2)*microntopix;
113
114     plot(x,y,'color',[0.17, 0.51, 0.34]);
115     hold all;
116     if i == 1
117         axis('ij');
118         xlabel('x coordinate (pix)');
119         ylabel('y coordinate (pix)');
120     end
121     labels{i} = ['post' num2str(ID(i))];
122
123     TAC(i,1) = ID(i);
124     TAC(i,2) = mean(x);
125     TAC(i,3) = mean(y);
126
127     FOC(i,1) = ID(i);
128     FOC(i,2) = x(1);
129     FOC(i,3) = y(1);
130 end
131
132 %Make a copy of the first figure for superimposing means:
133 fig_handle_trajectories2 = figure;
134 copyobj(get(fig_handle_trajectories,'children'),fig_handle_trajectories2);
135 figure(fig_handle_trajectories2);
136 plot(TAC(:,2),TAC(:,3),'LineStyle','none','Marker','+',...
137     'MarkerFaceColor','r','MarkerEdgeColor','r','MarkerSize',4);
138
139 % % Add legends, for some reason if this is done before the means are
140 % % superimposed you can't superimpose the means:
141 % figure(fig_handle_trajectories)
142 % legend(labels);
143 % legend('hide');
144 % figure(fig_handle_trajectories2)
145 % legend(labels);
146 % legend('hide');
147
148 % Save figures:

```

```

149 fprintf(1,'\tSaving figures\n');
150 fprintf(fid,'\tSaving figures\n');
151 saveas(fig_handle_trajectories, [save_path ...
152     '\Post_Trajectories.fig'], 'fig');
153 saveas(fig_handle_trajectories2, [save_path ...
154     '\Post_Trajectories_Centroids.fig'], 'fig');
155
156 % Save 'TAC' array in .mat and .txt form:
157 fprintf(1,'\tSaving variable(s)\n');
158 fprintf(fid,'\tSaving variable(s)\n');
159 save 'TimeAveragedCentroids.mat' TAC;
160 save 'FrameOneCentroids.mat' FOC;
161 fid2 = fopen([save_path '\Time_Avg_Centroids.txt'],'wt');
162 fid3 = fopen([save_path '\Frame_One_Centroids.txt'],'wt');
163 fprintf(fid2,'Post#\t<x(t)> (pix)\t<y(t)> (pix)\n');
164 fprintf(fid3,'Post#\tx(0) (pix)\ty(0) (pix)\n');
165 rows = size(TAC,1);
166 for k = 1:rows
167     fprintf(fid2,'%f\t%f\t%f\n',TAC(k,:));
168     fprintf(fid3,'%f\t%f\t%f\n',FOC(k,:));
169 end
170 fclose(fid2);
171 fclose(fid3);
172
173 % Update log file that function is completed:
174 fprintf(1,'%s completed\n',func_name);
175 fprintf(fid,'%s completed\n',func_name);
176 end

```

DedriftImageForOverlays_v1.m

```

1 % Steven J. Henry
2 % 09/19/2014
3 %*****
4 %
5 % PURPOSE: This function "dedrifts" a single image by translating its
6 % content while preserving its size so that future superimpositions are
7 % properly registered.
8 %
9 % ASSUMPTIONS:
10 % n/a
11 %
12 % INPUT:
13 % basepath = string pointing to location of experimental data
14 % fovn = field of view number
15 % microntopix = objective calibration for conversion of microns to pixels
16 % driftmodel_um = model of ensemble drift in microns
17 % savep_path = path to location of stored analysis
18 % fid = file ID of log file to which progress is recorded
19 %
20 % OUTPUT:
21 % Idedrift = image matrix "dedrifted" so future superposition of
22 % trajectories is properly aligned. The iamge is dedrifted by performing
23 % translation while preserving the original image size.
24 % raw_dim = dimensions of raw image before translation
25 %
26 % DRIVER/FUNCTION MAP:
27 % n/a (calls no subroutines)
28 %*****
29
30 function [Idedrift,raw_dim] = DedriftImageForOverlays_v1(basepath,fovn,...
31     microntopix,driftmodel_um,save_path,fid)
32
33 % Get function name:
34 func_name = mfilename;
35
36 % Update log file that function is running:
37 fprintf(1,'\n%s running ...\n',func_name);

```

```

42 fprintf(fid, '\n%s running ... \n', func_name);
43
44 [ImName, ImPath] = uigetfile([basepath '*.tif']; 'Select .tif file in raw data folder to overlay post IDs and set
centroid', 'MultiSelect', 'off');
45 I = imread([ImPath ImName]);
46
47 % Determine the frame number of this image:
48 frame_str = ImName(5+length(num2str(fov_n)):end-4);
49 frame = str2double(frame_str);
50
51 Idedrift = I;
52 raw_dim = size(I);
53
54 % If user selected a frame other than the first, retrieve the associated
55 % drift from the model:
56 if frame > 1
57 dx_um = driftmodel_um(frame-1, 1);
58 dy_um = driftmodel_um(frame-1, 2);
59 % convert to pix
60 dx_pix = dx_um * microntopix;
61 dy_pix = dy_um * microntopix;
62 % round to nearest whole pixel value:
63 dx_pix = round(dx_pix);
64 dy_pix = round(dy_pix);
65 % Subtract off the drift but preserve the original Image size so add on an
66 % equivalent zero margin:
67 [r, c] = size(I);
68
69 if dx_pix > 0
70 Idedrift = horzcat(Iidedrift(:, dx_pix+1:end), zeros(r, dx_pix));
71 elseif dx_pix < 0
72 Idedrift = horzcat(zeros(r, abs(dx_pix)), Idedrift(:, 1:end-abs(dx_pix)));
73 end
74
75 if dy_pix > 0
76 Idedrift = vertcat(Iidedrift(dy_pix+1:end,:), zeros(dy_pix, c));
77 elseif dy_pix < 0
78 Idedrift = vertcat(zeros(abs(dy_pix), c), Idedrift(1:end-abs(dy_pix), :));
79 end
80
81 end
82
83 % scalefactor = 3;
84 % I = imresize(I, scalefactor);
85 % Idedrift = imresize(Iidedrift, scalefactor); % make I scalefactorX as big
86
87 fig_handle_Iidedrift = figure;
88 subplot(1, 2, 1);
89 imshow(I);
90 hold all;
91 axis('ij');
92 title(ImName, 'Interpreter', 'none', 'FontSize', 14);
93 subplot(1, 2, 2);
94 imshow(Iidedrift);
95 hold all;
96 axis('ij');
97 title('Dedrifted', 'FontSize', 14);
98
99 fprintf(1, '\tSaving variable(s) and figure(s)\n');
100 fprintf(fid, '\tSaving variable(s) and figure(s)\n');
101
102 save([save_path filesep ImName '_dedrifted.mat'], 'Iidedrift');
103 save([save_path filesep 'EmpiricalDataDimensions'], 'raw_dim');
104 saveas(fig_handle_Iidedrift, [save_path filesep ImName '_dedrifted.fig'], ...
105 'fig');
106
107 end

```


OverlayPostIDs_v5.m

```
1 % Steven J. Henry
2 % 09/20/2014
25 %*****
26 % PURPOSE: This function overlays Post IDs as text on an image at the
27 % post positions in the image.
28 %
29 % ASSUMPTIONS:
30 % n/a
31 %
32 % INPUT:
33 % bead_path = path to location of bead.mat data
34 % I = image matrix
35 % cat = string denoting category
36 % color = text color
37 % microntopix = objective calibration for conversion of microns to pixels
38 % save_path = path to location of stored analysis
39 % fid = file ID of log file to which progress is recorded
40 % subsetIDs = post IDs to plot
41 %
42 % OUTPUT:
43 % n/a
44 %
45 % DRIVER/FUNCTION MAP:
46 % n/a (calls no subroutines)
47 %*****
48
49 function [] = OverlayPostIDs_v5(bead_path,I,cat,color,microntopix,...
50     save_path,fid,subsetIDs)
51
52 % Get function name:
53 func_name = mfilename;
54
55 % Update log file that function is running:
56 fprintf(1,'\n%s running ...\n',func_name);
57 fprintf(fid,'\n%s running ...\n',func_name);
58
59 % Retrieve all contents that reside inside 'bead_path' folder:
60 contents = dir(bead_path);
61
62 num_items = size(contents,1);
63 FileName = cell(num_items,1);
64 ID = NaN(num_items,1);
65 keep_ind = false(num_items,1);
66
67 for i = 1:num_items
68     item_name = contents(i).name;
69     if length(item_name) >= 10 && strcmp(item_name(1:5),'bead_')
70         FileName(i) = item_name;
71         ID(i) = str2double(item_name(6:end-4));
72         keep_ind(i) = true;
73     end
74 end
75
76 FileName = FileName(keep_ind);
77 ID = ID(keep_ind);
78
79 if nargin < 8
80     subsetIDs = ID; %#ok<NASGU>
81 else
82     % Further filter 'FileName' to include only those ID's that are present
83     % in the user-supplied 'subsetIDs' vector:
84     num_files = length(FileName);
85     keep_ind2 = false(num_files,1);
86     for j = 1:num_files
87
88         insubset_test = ID(j)==subsetIDs;
```

```

89
90     if sum(insubset_test)>0
91         keep_ind2(j)=true;
92     end
93 end
94 FileName = FileName(keep_ind2);
95 ID = ID(keep_ind2);
96 end
97
98 % How many post files did user select?
99 num_posts = length(FileName);
100 xpix = zeros(num_posts,1);
101 ypix = zeros(num_posts,1);
102
103 for i = 1:num_posts
104
105     if i == 1
106         fig_handle_overlay = figure;
107         imshow(l,'DisplayRange',[]);
108         axis('ij');
109         hold all;
110     end
111
112     load([bead_path filesep FileName(i)]);
113
114     xpix(i) = bsec(1,1)*microntopix; %pix
115     ypix(i) = bsec(1,2)*microntopix; %pix
116
117     plot(xpix(i),ypix(i),'LineStyle','none','LineWidth',1,'Marker','o',...
118         'MarkerEdgeColor','k','MarkerFaceColor',color,'MarkerSize',15);
119     hold all;
120     % text(xpix(i)+2,ypix(i)+2,num2str(ID(i)),'Color','w','FontSize',...
121     % 8,'Background',color,'Margin',1);
122
123 end
124
125 xlabel('x coord (\mum)','FontSize',14);
126 ylabel('y coord (\mum)','FontSize',14);
127
128 % A figure with text only, no image underneath:
129 fig_handle_textonly = figure;
130 plot(xpix,ypix,'LineStyle','none','Marker','+', 'MarkerEdgeColor','r',...
131     'MarkerFaceColor','r','MarkerSize',6);
132 hold all;
133 axis('ij');
134 axis('square');
135
136 for i = 1:num_posts
137
138     text(xpix(i),ypix(i),num2str(ID(i)),'Color','k','FontSize',8);
139
140 end
141
142 xlabel('x coord (\mum)','FontSize',14);
143 ylabel('y coord (\mum)','FontSize',14);
144
145 % Save figure:
146 fprintf(1,'\tSaving variable(s) and figure(s)\n');
147 fprintf(fid,'\tSaving variable(s) and figure(s)\n');
148
149 saveas(fig_handle_overlay,[save_path filesep 'PostID_Overlay_' cat ...
150     '.fig']);
151 saveas(fig_handle_textonly,[save_path filesep 'PostID_TextOnly_' cat ...
152     '.fig']);
153
154 % Update log file that function is completed:
155 fprintf(1,'%s completed\n',func_name);
156 fprintf(fid,'%s completed\n',func_name);

```

157
158 end

Merge_Trajectories_v1.m

```
1 % Steven J. Henry
2 % 07/21/2014
7 %*****
8 % PURPOSE: This function merges the bead.mat files that all belong to the
9 % same post as manually dictated by the user. It is a function that allows
10 % clean-up of data if posts were dropped and picked up again with a new
11 % post ID.
12 %
13 % ASSUMPTIONS:
14 % n/a
15 %
16 % INPUT:
17 % bead_path = path to location of bead.mat data
18 % fid = file ID of log file to which progress is recorded
19 %
20 % OUTPUT:
21 % n/a
22 %
23 % DRIVER/FUNCTION MAP:
24 % n/a (calls no subroutines)
25 %*****
26
27 function [] = Merge_Trajectories_v1(bead_path,fid)
28
29 % Get function name:
30 func_name = mfilename;
31
32 % Update log file that function is running:
33 fprintf(1,'\n%s running ...\n',func_name);
34 fprintf(fid,'\n%s running ...\n',func_name);
35
36 % Initialize an exitflag to off:
37 exitflag = 0;
38
39 % Whiel exitflag is off:
40 while exitflag ~= 1
41
42     % Retrieve input from user. 'v' is either a numeric vector or the
43     % string 'DONE'
44     v = input('\tSupply vector of bead.mat IDs (e.g. [1 2 3]) to merge; -1 when done\n');
45
46     % If 'v' is -1
47     if v==-1
48         % User is finished entering vectors of bead IDs:
49         fprintf(1,'\tYou have finished entering bead IDs to merge\n');
50         fprintf(fid,'\tYou have finished entering bead IDs to merge\n');
51         exitflag = 1;
52
53     else
54         % Does the cell array to hold user-supplied vectors exist?
55         if exist('m','var')==1
56             % If so concatenate the existing array with a new cell to hold
57             % the new vector:
58             m_new = vertcat(m,cell(1,1));
59             clear('m');
60             m_new{end} = v;
61             m = m_new;
62             clear('m_new');
63         else
64             % Otherwise this is the first time the user supplied a vector
65             % so create a cell to hold it:
66             m = cell(1,1);
67             m{1} = v;
```

```

68     end
69
70     % User supplied vector
71     fprintf(1,'\tUser supplied vector of IDs to merge: %s\n',...
72           num2str(v));
73     fprintf(fid,'\tUser supplied vector of IDs to merge: %s\n',...
74           num2str(v));
75     end
76 end
77
78 % How many posts are being handled?
79 num_posts = size(m,1);
80
81 % Loop over the posts
82 for i = 1:num_posts
83
84     % Extract the user-supplied vector
85     v = m(i);
86
87     % How many merges are going to occur?
88     num_merges = length(v);
89
90     % Loop over the merges to take place:
91     for j = 1:num_merges
92
93         % Load the bead.mat file, the variable name is 'bsec'
94         bead_name = ['bead_' num2str(v(j)) '.mat'];
95         load([bead_path filesep bead_name]);
96
97         if j == 1
98             % If this is the first time write bsec to a master array:
99             bsec_master = bsec;
100            clear('bsec');
101        else
102            % Otherwise append this bsec to the existing master array:
103            bsec_master = vertcat(bsec_master,bsec); %#ok<AGROW>
104            clear('bsec');
105        end
106
107    end
108
109    % Once concatenated sort data by frame# and reassign ID at all frame
110    % numbers with the ID in the first frame:
111    [~,sort_index] = sort(bsec_master(:,3),'ascend');
112    bsec_master_sorted = bsec_master(sort_index,:);
113    id = bsec_master_sorted(1,4);
114    bsec_master_sorted(:,4) = id;
115    bsec = bsec_master_sorted;
116    clear('bsec_master','bsec_master_sorted');
117
118    % Search for duplicate occurrences of a frame # in the concatenated
119    % bsec. This was found to occur in a few cases and implies that the
120    % merged trajectories were considered distinct objects for some
121    % portion of their tracking (usually 1-2 frames at most). My
122    % hypothesis is that this occurs when a post is highly likely from a
123    % highly deflected such that some of the sidewall becomes visible in
124    % addition to the tip.
125    % Retrieve the list of frame #s. This should be a sorted list.
126    frames = bsec(:,3);
127    % If the number of unique entries in 'frames' is less than the length
128    % of 'frames', there are duplicates.
129    if length(unique(frames))<length(frames)
130        % Compute the consecutive differences (differences between adjacent
131        % rows). This vector is length(frames)-1 long.
132        dframes = diff(frames);
133        % A difference of zero implies that the adjacent entries have the
134        % same value.
135        dup_rows = find(dframes==0);

```

```

136     if isempty(dup_rows)==0
137         num_dups = length(dup_rows);
138         % Setup a deletion index vector. This will hold ones
139         % corresponding to rows that need to be deleted from 'bsec'.
140         del_rows = false(size(bsec,1),1);
141         for k = 1:num_dups
142             % Row 5 of bsec contains the length of the trajectory at
143             % that frame #. We want to retain the row corresponding to
144             % the previous trajectory.
145             % For example imagine to trajectories that belong to the
146             % same object. Trajectory 1 is dropped at frame 1400:
147             % bsec(1400,3) = 1400, frame #
148             % bsec(1400,4) = 1, ID
149             % bsec(1400,5) = 1400, length of trajectory at that frame
150             % Trajectory 2 is started at frame 1400:
151             % bsec(1400,3) = 1400, frame #
152             % bsec(1400,4) = 2, ID
153             % bsec(1400,5) = 1, length of trajectory at that frame
154             % The following logic would retain frame 1400 belonging to
155             % trajectory 1 and eliminate frame 1400 belonging to
156             % trajectory 2
157             a = bsec(dup_rows(k),5);
158             b = bsec(dup_rows(k)+1,5);
159             if a>b % keep row containing a, discard row containing b
160                 del_rows(dup_rows(k)+1)=true;
161             else
162                 del_rows(dup_rows(k))=true;
163             end
164         end
165         bsec(del_rows,:) = [];
166     end
167     % Sanity check:
168     frames = bsec(:,3);
169     if length(unique(frames))~=length(frames)
170         fprintf(1,'\tDuplicate frame# present in bead_%.mat after filtering\n',num2str(id));
171         fprintf(fid,'\tDuplicate frame# present in bead_%.mat after filtering\n',num2str(id));
172     end
173 end
174
175
176 % Now save bsec under 'id'
177 bead_name = ['bead_' num2str(id) '.mat'];
178 delete([bead_path filesep bead_name]);
179 save([bead_path filesep bead_name], 'bsec');
180
181 % Eliminate this 'id' from 'v'
182 del_index = v==id;
183 v(del_index)=[];
184
185 % Delete bead files that were merged into bead_id.mat:
186 num_deletions = length(v);
187 for k = 1:num_deletions
188     bead_name = ['bead_' num2str(v(k)) '.mat'];
189     delete([bead_path filesep bead_name]);
190 end
191
192 end
193
194 % Update log file that function is completed:
195 fprintf(1, '%s completed\n', func_name);
196 fprintf(fid, '%s completed\n', func_name);
197
198 end

```

PlotPostIDs_v1.m

```

1 % Steven J. Henry
2 % 07/21/2014

```

```

8 %*****
9 % PURPOSE: This function overlays Post IDs on an image.
10 %
11 % ASSUMPTIONS:
12 % n/a
13 %
14 % INPUT:
15 % bead_path = path to location of bead.mat data
16 % I = image matrix
17 % microntopix = objective calibration for conversion of microns to pixels
18 % savep_path = path to location of stored analysis
19 % fid = file ID of log file to which progress is recorded
20 %
21 % OUTPUT:
22 % n/a
23 %
24 % DRIVER/FUNCTION MAP:
25 % n/a (calls no subroutines)
26 %*****
27
28 function [] = PlotPostIDs_v1(bead_path,microntopix,save_path,fid)
29
30 % Get function name:
31 func_name = mfilename;
32
33 % Update log file that function is running:
34 fprintf(1,'\n%s running ...\n',func_name);
35 fprintf(fid,'\n%s running ...\n',func_name);
36
37 % Select bead.mat files to be processed
38 listing = dir(bead_path);
39
40 num_listings = length(listing);
41
42 fig_handle_ids = figure;
43
44 for i = 1:num_listings
45
46     if listing(i).isdir == 0
47
48         bead_name = listing(i).name;
49
50         if strcmp(bead_name(1:5),'bead')==1
51
52             load([bead_path filesep listing(i).name]);
53             xpix = bsec(1,1)*microntopix; %pix
54             ypix = bsec(1,2)*microntopix; %pix
55             ID = bead_name(6:end-4);
56             plot(xpix,ypix,'LineStyle','none','Marker','+',...
57                 'MarkerEdgeColor','r','MarkerFaceColor','r',...
58                 'MarkerSize',6);
59             hold on;
60             text(xpix,ypix,ID,'Color','k','FontSize',8);
61         end
62     end
63 end
64
65 end
66
67 axis('ij');
68
69 % Save figure:
70 fprintf(1,'\tSaving figure\n');
71 fprintf(fid,'\tSaving figure\n');
72 if exist('fig_handle_ids','var')==1
73     saveas(fig_handle_ids,[save_path filesep 'PostID_TextOnly.fig']);
74 end
75

```

```

76 % Update log file that function is completed:
77 fprintf(1, '%s completed\n', func_name);
78 fprintf(fid, '%s completed\n', func_name);
79
80 end

```

SetSuperResolutionRefinement_v1.m

```

1 % Steven J. Henry
2 % 06/12/2014
7 %*****
8 % PURPOSE: This function asks the user to supply a superresolution multiple
9 % ('refine') and tests if the supplied value is allowable.
10 %
11 % ASSUMPTIONS:
12 % n/a
13 %
14 % INPUT:
15 % save_path = path to location of stored analysis
16 % fid = file ID of log file to which progress is recorded
17 %
18 % OUTPUT:
19 % refine = user-set refinement value
20 %
21 % DRIVER/FUNCTION MAP:
22 % n/a (calls no subroutines)
23 %*****
24
25
26 function [refine] = SetSuperResolutionRefinement_v1(save_path, fid)
27
28 % Get function name:
29 func_name = mfilename;
30
31 % Update log file that function is running:
32 fprintf(1, '\n%s running ... \n', func_name);
33 fprintf(fid, '\n%s running ... \n', func_name);
34
35 % Have user specify refinement value:
36 escape_flag = 0;
37 acceptable = [1, 2, 4, 5, 8, 10, 16, 20, 25, 32, 50, 64, 100];
38 while escape_flag == 0;
39     set_refinement = input('Set refinement value {1,2,4,5,8,10,16,20,25,32,50,64,100}:');
40     test = set_refinement == acceptable;
41     if sum(test) == 1
42         refine = set_refinement;
43         fprintf(1, '\tUser selected a refinement of %s\n\tCentroids will be plotted to nearest %s\n', num2str(refine), num2str(1/refine));
44         fprintf(fid, '\tUser selected a refinement of %s\n\tcentroids will be plotted to nearest %s\n', num2str(refine), num2str(1/refine));
45         % Turn escape flag on:
46         escape_flag = 1;
47     else
48         fprintf(1, '\tUser requested refine = %s which is not an option.\n', num2str(set_refinement));
49         fprintf(fid, '\tUser requested refine = %s which is not an option.\n', num2str(set_refinement));
50     end
51 end
52
53 % Save refine:
54 save_name = 'refine.mat';
55 fprintf(1, '\tSaving %s\n', save_name);
56 fprintf(fid, '\tSaving %s\n', save_name);
57 if exist('refine', 'var') == 1
58     save([save_path filesep save_name], 'refine');
59 end
60
61 % Update log file that function is completed:
62 fprintf(1, '%s completed\n', func_name);

```

```

63 fprintf(fid, '%s completed\n', func_name);
64
65 end

```

Construct_Bitmap_v4.m

```

1  % Steven J. Henry
2  % 06/12/2014
25 %*****
26 % PURPOSE:
27 % This function asks the user to select a refinement value and construct a
28 % binary bitmap with "on" ("1") pixels at the location of time averaged
29 % centroids. Centroid positions are placed to the nearest refinement
30 % multiple.
31
32 % For example if a refinement of 2 is specified then centroids can
33 % be placed to the nearest 1/2 = 0.5 pixel location. If a refinement of 10
34 % is specified then centroids can be positioned to the nearest 1/10 = 0.1
35 % pixels.
36 %
37 % ASSUMPTIONS:
38 % n/a
39 %
40 % INPUT:
41 % M = three column matrix
42 %   col1 = postID
43 %   col2 = x-coordinate of centroid (pix)
44 %   col3 = y-coordinate of centroid (pix)
45 % dim = dimensions of empirical data bitmap [#rows, #cols]
46 % refine = refinement value to construct super-resolution bitmap
47 % fid = file ID of log file to which progress is recorded
48 %
49 % OUTPUT:
50 % A = binary array with on (1) pixels located at centroid positions
51 % LUT = look-up-table mapping postID of each centroid in A
52 %
53 % DRIVER/FUNCTION MAP:
54 % n/a (calls no subroutines)
55 %*****
56
57 function [A,LUT] = Construct_Bitmap_v4(M,dim,refine,fid)
58
59 % Get function name:
60 func_name = mfilename;
61
62 % Update log file that function is running:
63 fprintf(1, '\n%s running ...\n', func_name);
64 fprintf(fid, '\n%s running ...\n', func_name);
65
66 % How many post files did user pass?
67 num_posts = size(M,1);
68
69 % Round centroid locations to nearest multiple of 'refine':
70 ID = M(:,1);
71 X = round(M(:,2)*refine);
72 Y = round(M(:,3)*refine);
73
74 % Reserve memory
75 A = zeros(dim*refine);
76 [ylim,xlim] = size(A);
77 LUT = nan(size(M));
78
79 for k = 1:num_posts
80     i = Y(k);
81     j = X(k);
82     if 1<=i & i<=ylim & 1<=j & j<=xlim %#ok<AND2>
83         A(i,j) = 1;
84         LUT(k,1) = ID(k);

```



```

85     LUT(k,2) = j;
86     LUT(k,3) = i;
87     end
88 end
89
90 del_id = isnan(LUT(:,1));
91 LUT(del_id,:)=[];
92
93 % A couple of sanity checks:
94 if sum(sum(A))==num_posts
95     fprintf(1,'\tsum(sum(A)) = %s which = %s centroids supplied\n',...
96             num2str(sum(sum(A))),num2str(num_posts));
97     fprintf(fid,'\tsum(sum(A)) = %s which = %s centroids supplied\n',...
98             num2str(sum(sum(A))),num2str(num_posts));
99 else
100    fprintf(1,'\t**WARNING**: sum(sum(A)) = %s which ~= %s centroids
101    supplied\n',num2str(sum(sum(A))),num2str(num_posts));
102    fprintf(fid,'\t**WARNING**: sum(sum(A)) = %s which ~= %s centroids
103    supplied\n',num2str(sum(sum(A))),num2str(num_posts));
104 end
105
106 edges = zeros(4,1);
107 edges(1) = sum(A(:,1));
108 edges(2) = sum(A(:,end));
109 edges(3) = sum(A(1,:));
110 edges(4) = sum(A(end,:));
111 empty_edges = edges==0;
112 if sum(empty_edges)==0
113     fprintf(1,'\tAn "on" pixel resides at each edge of A\n');
114     fprintf(fid,'\tAn "on" pixel resides at each edge of A\n');
115 else
116     fprintf(1,'\t %s edges of bitmap have "on" pixels\n',...
117             num2str(sum(~empty_edges)));
118     fprintf(fid,'\t %s edges of bitmap have "on" pixels\n',...
119             num2str(sum(~empty_edges)));
120 end
121
122 % Update log file that function is completed:
123 fprintf(1,'%s completed\n',func_name);
124 fprintf(fid,'%s completed\n',func_name);
125
126 end

```

Find_Rotation_Angle_v1.m

```

1  % Steven J. Henry
2  % 05/07/2014
7  %*****
8  % PURPOSE:
9  % This function assists the user in finding the rotation required so
10 % features are parallel with the vertical axis.
11 %
12 % ASSUMPTIONS:
13 % n/a
14 %
15 % INPUT:
16 % A = an image array
17 % save_path = string pointing to save location
18 % fid = file ID of log file to which progress is recorded
19 %
20 % OUTPUT:
21 % theta = rotation in degrees, (+) counter clockwise, (-) clockwise
22 %
23 % DRIVER/FUNCTION MAP:
24 % n/a (calls no subroutines)
25 %*****
26
27 function [theta] = Find_Rotation_Angle_v1(A,save_path,fid)

```

```

28
29 % Get function name:
30 func_name = mfilename;
31
32 % Update log file that function is running:
33 fprintf(1, '\n%s running ... \n', func_name);
34 fprintf(fid, '\n%s running ... \n', func_name);
35
36 % Plot array:
37 fig_handle_input = figure;
38 imshow(A, 'DisplayRange', []);
39 title('Draw a line along the lattice direction you want parallel with the vertical axis', 'FontSize', 14, 'Interpreter', 'None');
40
41 % Have user draw line along lattice row that should be parallel with the
42 % vertical axis:
43 h_line = imline;
44 coords = wait(h_line);
45
46 % If user drew a perfectly vertical line (Y coordinates equivalent),
47 % than rotation is zero
48 if coords(1,2)==coords(2,2)
49     theta = 0;
50     % If user drew a perfectly horizontal line (X coordinates equivalent),
51     % than the rotation is 90 deg and direction doesn't matter:
52 elseif coords(1,1)==coords(1,2)
53     theta = 90;
54     % Otherwise the line is neither vertical or horizontal so set P2 (head
55     % point) as coordinate with larger Y value and P1 (tail point) as
56     % coordinate with smaller Y value:
57 else
58
59     if coords(1,2)>coords(2,2)
60         P2 = coords(1,:);
61         P1 = coords(2,:);
62     else
63         P2 = coords(2,:);
64         P1 = coords(1,:);
65     end
66
67     % Now compute unit vector from P1->P2:
68     % v = <X2-X1, Y2-Y1>;
69     % |v| = sqrt((X2-X1)^2+(Y2-Y1)^2);
70     % vhat = v/|v|;
71     vmag = sqrt((P2(1)-P1(1))^2+(P2(2)-P1(2))^2);
72     vhat = [P2(1)-P1(1), P2(2)-P1(2)]/vmag;
73
74     % In axis 'ij' system with origin in NorthWest corner unit vector
75     % 'ihat' in positive y direction is [0 1]. The angle between 'ihat' and
76     % 'vhat' is the arccos of the dot product:
77     ihat = [0,1];
78     theta = acosd(ihat(1)*vhat(1)+ihat(2)*vhat(2));
79
80     % Now compute slope to make sure you have the direction of rotation
81     % correct:
82     m = (P2(2)-P1(2))/(P2(1)-P1(1));
83     if m>0 % we need a clockwise rotation so set theta to negative itself:
84         theta = -theta;
85     end
86
87 end
88
89 fprintf(1, '\ttheta = %s\n deg', num2str(theta));
90 fprintf(fid, '\ttheta = %s\n deg', num2str(theta));
91
92 % Save figure:
93 fprintf(1, '\tSaving figure(s)\n');
94 fprintf(fid, '\tSaving figure(s)\n');
95 saveas(fig_handle_input, [save_path '\PreRotated_Bitmap.fig'], 'fig');

```

```

96
97 % Save 'theta' value in .mat form:
98 fprintf(1, '\tSaving theta value\n');
99 fprintf(fid, '\tSaving theta value\n');
100 save([save_path '\theta.mat'], 'theta');
101
102 close(fig_handle_input);
103
104 % Update log file that function is completed:
105 fprintf(1, '%s completed\n', func_name);
106 fprintf(fid, '%s completed\n', func_name);
107
108 end

```

Apply_Rotation_v3.m

```

1 % Steven J. Henry
2 % 05/27/2014
16 %*****
17 % PURPOSE:
18 % Performs a rotation by constructing a rotation matrix and applying that
19 % rotation matrix to the x,y coordinate pairs of the centroids supplied.
20 %
21 % ASSUMPTIONS:
22 % n/a
23 %
24 % INPUT:
25 % X = x-coordinate of centroid (pix)
26 % Y = y-coordinate of centroid (pix)
27 % theta = rotation in degrees, (+) counter clockwise, (-) clockwise
28 % save_path = string pointing to save location
29 % fid = file ID of log file to which progress is recorded
30 % dim = dimensions of raw data iamge
31 %
32 % OUTPUT:
33 % Xr = rotated x-coordinate of centroid (pix)
34 % Yr = rotated y-coordinate of centroid (pix)
35 %
36 % DRIVER/FUNCTION MAP:
37 % n/a (calls no subroutines)
38 %*****
39
40 function [Xr,Yr] = Apply_Rotation_v3(X,Y,theta,save_path,fid,dim)
41
42 % Get function name:
43 func_name = mfilename;
44
45 % Update log file that function is running:
46 fprintf(1, '\n%s running ... \n', func_name);
47 fprintf(fid, '\n%s running ... \n', func_name);
48
49 % Did user supply a 'dim' variable:
50 if nargin < 6
51     % If not, assume supplied X and Y coordinates are already translated to
52     % viewing window origin at center of image:
53     yo = 0;
54     xo = 0;
55 else
56     if sum(dim<0)>0
57         fprintf(1, '\t**WARNING** Passed dimensions are not positive\n');
58         fprintf(fid, '\t**WARNING** Passed dimensions are not positive\n');
59     end
60     ymax = dim(1);
61     xmax = dim(2);
62     yo = ymax/2;
63     xo = xmax/2;
64     fprintf(1, '\tUser-dictated viewing window is %s X %s pix^2\n', ...
65         num2str(ymax), num2str(xmax));

```

```

66     fprintf(1, '\tOrigin (xo,yo) = (%s,%s) pix\n',...
67             num2str(xo),num2str(yo));
68     fprintf(fid, '\tUser-dictated viewing window is %s X %s pix^2\n',...
69             num2str(ymax),num2str(xmax));
70     fprintf(fid, '\tOrigin (xo,yo) = (%s,%s) pix\n',...
71             num2str(xo),num2str(yo));
72 end
73
74 % Sanity checks
75 [xrow,xcol] = size(X);
76 [yrow,ycol] = size(Y);
77 if xcol ~= 1 || ycol ~= 1
78     fprintf(1, '\t**WARNING** X and Y coordinates should be supplied in separate column vectors\n');
79     fprintf(fid, '\t**WARNING** X and Y coordinates should be supplied in separate column vectors\n');
80 end
81 if xrow ~= yrow
82     fprintf(1, '\t**WARNING** # x-coordinates (%s) ~= # y-coordinates (%s)\n',num2str(xrow),num2str(yrow));
83     fprintf(1, '\t**WARNING** # x-coordinates (%s) ~= # y-coordinates (%s)\n',num2str(xrow),num2str(yrow));
84 end
85
86 % Translate coordinates relative to image center:
87 X = X-xo;
88 Y = Y-yo;
89
90 % Reserve memory
91 Xr = zeros(size(X));
92 Yr = Xr;
93
94 num_centroids = length(X);
95 R = [cosd(theta), sind(theta);
96     -sind(theta), cosd(theta)];
97 for i = 1:num_centroids
98     v = [X(i);Y(i)];
99     vr = R*v;
100    Xr(i) = vr(1);
101    Yr(i) = vr(2);
102 end
103
104 Xr = Xr+xo;
105 Yr = Yr+yo;
106
107 % Update log file that function is completed:
108 fprintf(1, '%s completed\n',func_name);
109 fprintf(fid, '%s completed\n',func_name);
110
111 end

```

Crop_Bitmap_v2.m

```

1  % Steven J. Henry
2  % 05/01/2014
13 %*****
14 % PURPOSE:
15 % This function crops a supplied image array by an amount calculated from a
16 % user-specified rectangle. The goal is to set the crop boundary in such a
17 % way that the boundaries are periodic.
18 %
19 % ASSUMPTIONS:
20 % n/a
21 %
22 % INPUT:
23 % A = an image array
24 % dilate = integer value by which to expand pixels for purpose of
25 % visualization, (cropping is performed on undilated image)
26 % save_path = string pointing to save location
27 % fid = file ID of log file to which progress is recorded
28 %
29 % VARIABLE OUTPUT:

```

```

30 % C = cropped image array A
31 % verts = vertices of cropping rectangle [xmin ymin width height]
32 % pad = [ypre,xpre,ypost,xpost]
33 %
34 % DRIVER/FUNCTION MAP:
35 % n/a (calls no subroutines)
36 %*****
37
38 function [C,verts,pad] = Crop_Bitmap_v2(A,dilate,save_path,fid)
39
40 % Get function name:
41 func_name = mfilename;
42
43 % Update log file that function is running:
44 fprintf(1,'\n%s running ...\n',func_name);
45 fprintf(fid,'\n%s running ...\n',func_name);
46
47 % Dilate supplied image and plot:
48 fig_handle_dilated_input = figure;
49 imshow(imdilate(A,strel('disk',dilate)), 'DisplayRange',[]);
50 title_string = 'Draw cropping rectangle, double click when done';
51 title(title_string,'FontSize',14,'Interpreter','None');
52
53 % Position rectangle for cropping:
54 h_rect = imrect;
55 verts = wait(h_rect);
56 verts = round(verts);
57
58 % Crop
59 C = imcrop(A,verts);
60
61 % Determine pad size to undo crop later:
62 [r,c] = size(A);
63 xpre = verts(1)-1;
64 ypre = verts(2)-1;
65 xpost = c-(xpre+verts(3)+1); %width does not include terminal point
66 ypost = r-(ypre+verts(4)+1); %height does not include terminal point
67 pad = [ypre,xpre,ypost,xpost];
68
69 % Plot cropped image:
70 fig_handle_dilated_output = figure;
71 imshow(imdilate(C,strel('disk',dilate)), 'DisplayRange',[]);
72 title_string = 'Dilated cropped image';
73 title(title_string,'FontSize',20,'Interpreter','None');
74
75 % Save figures:
76 fprintf(1,'\tSaving figures\n');
77 fprintf(fid,'\tSaving figures\n');
78 saveas(fig_handle_dilated_input, [save_path ...
79     '\Bitmap_Perturbed_PreCropped.fig'], 'fig');
80 saveas(fig_handle_dilated_output, [save_path ...
81     '\Bitmap_Perturbed_PostCropped.fig'], 'fig');
82
83 % Save 'C' array in .mat and .txt form:
84 fprintf(1,'\tSaving C array\n');
85 fprintf(fid,'\tSaving C array\n');
86 save('C.mat','C','verts');
87 dlmwrite([save_path '\C.txt'],C,'\t');
88
89 % Update log file that function is completed:
90 fprintf(1,'%s completed\n',func_name);
91 fprintf(fid,'%s completed\n',func_name);
92 end

```

FilterForRestingLattice_v6.m

```

1 % Steven J. Henry
2 % 07/23/2014

```

```

29 %*****
30 % PURPOSE:
31 % This function filters a bitmap of post centroids to return the resting
32 % (i.e unperturbed) lattice by filtering out "noise" (i.e. perturbation) in
33 % Fourier space.
34 %
35 % ASSUMPTIONS:
36 %
37 % INPUT:
38 % P = user supplied bitmap of perturbed lattice
39 % pts = # probe points
40 % method = 'auto' or 'manual'
41 % save_path = string pointing to save location
42 % fid = file ID of log file to which progress is recorded
43 %
44 % OUTPUT:
45 % R2 = filtered resting lattice of same type and size as P.
46 %
47 % DRIVER/FUNCTION MAP:
48 % n/a (calls no subroutines)
49 %*****
50
51 function [R2] = FilterForRestingLattice_v6(P,pts,method,save_path,fid)
52
53 % Get function name:
54 func_name = mfilename;
55
56 % Update log file that function is running:
57 fprintf(1,'\n%s running ...\n',func_name);
58 fprintf(fid,'\n%s running ...\n',func_name);
59
60 iterate_flag = 1;
61 iteration = 1;
62
63 while iterate_flag == 1
64
65     %If this is the first pass through the loop set up threshold values
66     if iteration == 1
67
68         % Perform FFT on P:
69         fprintf(1,'\n\tComputing FFT of perturbed bitmap...\n');
70         fprintf(fid,'\n\tComputing FFT of perturbed bitmap...\n');
71         P_tilda = fft2(P);
72
73         % Plot the FFT of P:
74         fig_handle_FFT_P = figure;
75         imshow(abs(fftshift(P_tilda)), 'DisplayRange', []);
76         title('abs(fftshift(fft2(P)))', 'FontSize', 20, 'Interpreter', 'None');
77
78         % Compute complex modulus to determine threshold value:
79         abs_P_tilda = abs(P_tilda);
80
81         % Reshape the matrix of complex moduli into a column vector to run
82         % through 'hist' function:
83         [r,c] = size(abs_P_tilda);
84         abs_P_tilda_v = reshape(abs_P_tilda,r*c,1);
85
86         % Compute histogram of pixel intensities:
87         nbins = sqrt(length(abs_P_tilda_v));
88         [counts,loc] = hist(abs_P_tilda_v,nbins);
89
90         % Plot histogram before threshold
91         fig_handle_hist_FFT_P = figure;
92         bar(loc,counts,1, 'FaceColor', 'b', 'EdgeColor', 'b');
93         title('Histogram of Complex Moduli', 'FontSize', 20, ...
94             'Interpreter', 'None');
95         axis([min(abs_P_tilda_v) max(abs_P_tilda_v) 0 max(counts)]);
96         set(gca, 'FontSize', 14, 'yscale', 'log');

```

```

97     xlabel('Pixel Intensity','FontSize',16);
98     ylabel('Counts','FontSize',16);
99     [r,c] = size(P);
100    lower = 1;
101    upper = floor(log10(r*c))-1;
102    thresholds = round(logspace(lower,upper,pts));
103
104    % Reserve a figure to plot resolution metric results
105    fig_handle_resolution = figure;
106    labels = cell(1,1);
107
108    else
109        if pos == 1
110            lower = 10^floor(log10(thresholds(pos)))-1;
111            upper = thresholds(pos+1);
112        elseif pos == length(thresholds)
113            lower = thresholds(pos-1);
114            upper = 10^floor(log10(thresholds(pos)))+1;
115        else
116            lower = thresholds(pos-1);
117            upper = thresholds(pos+1);
118        end
119
120        ldecade = floor(log10(lower));
121        udecade = floor(log10(upper));
122        thres_max = thresholds(pos);
123
124        if ldecade == udecade || udecade-ldecade == 1
125            thresholds = round(linspace(lower,upper,pts));
126        else
127            thresholds = round(logspace(log10(lower),log10(upper),pts));
128        end
129
130        % If threshold value of previous peak is not present in this new
131        % thresholds list add it:
132        existance = thresholds==thres_max;
133        if sum(existance)==0
134            thresholds = sort([thresholds thres_max]);
135        end
136
137    end
138
139    fprintf(1,...
140        '\tThreshold (# retained pixels) range to be tested: %s\n',...
141        num2str(thresholds));
142    fprintf(fid,...
143        '\tThreshold (# retained pixels) range to be tested: %s\n',...
144        num2str(thresholds));
145
146    resolutions = zeros(size(thresholds));
147
148    for j = 1:length(thresholds)
149
150        % Working from the last (highest pixel intensity bin) backwards,
151        % determine how many bins you need to use to accumulate
152        % 'thresholds' pixels:
153        flag = 0;
154        tally = 0;
155        i = 1;
156        while flag == 0
157            % Shift the counts row vector by 'i' positions
158            shifted_counts = circshift(counts,i,2);
159            % Add the corresponding number of pixels residing in this bin
160            % to the total tally
161            tally = tally+shifted_counts(1);
162            % If the total tally equals or exceeds the original number of
163            % centroids, exit
164            if tally >= thresholds(j)

```

```

165     flag = 1;
166     else
167         % If not, increase the amount the circle shift occurs and
168         % repeat the tally
169         i = i+1;
170     end
171 end
172
173 % Knowing the last 'i' bins contain 'thresholds(j)' pixels we
174 % determine the intensity threshold. 'hist' returns 'loc' a column
175 % vector of bin centers. We will determine the bin edge that
176 % demarks the start of the 'ith' bin and use that edge as our
177 % threshold value.
178
179 % This is half the distance b/n the two bin centers
180 delta = (loc(end-i+1)-loc(end-i))/2;
181 % This is the start edge of bin 'end-i+1', also the end edge of bin
182 % 'end-i'
183 thres = loc(end-i)+delta;
184
185 % Filter P_tilda only retaining pixels that have a complex modulus
186 % greater or equal to 'thres' and turning off (with complex zero)
187 % pixels that have modulus less than 'thres'
188 makecomplexzero = abs_P_tilda<thres;
189 P_tilda_filtered = P_tilda;
190 P_tilda_filtered(makecomplexzero) = complex(0,0);
191
192 % Plot P_tilda_filtered:
193 if strcmp(method,'manual')==1
194     fig_handle_P_tilda = figure;
195     imshow(imdilate(abs(fftshift(P_tilda_filtered)),...
196         strel('disk',3)), 'DisplayRange', []);
197     title('dilated Complex Moduli of Thresholded P_tilda',...
198         'FontSize',20,'Interpreter','None');
199 end
200
201 % Perform inverse FFT to take filtered P matrix back to cartesian
202 % space:
203 fprintf(1, '\nComputing IFFT of filtered bitmap...\n');
204 R = ifft2(P_tilda_filtered);
205
206 % Plot R:
207 if strcmp(method,'manual')==1
208     fig_handle_R = figure;
209     imshow(R, 'DisplayRange', []);
210     title('ifft2(Thresholded P_tilda)', 'FontSize',20,...
211         'Interpreter','None');
212 end
213
214 % Compute "resolution" metric
215 Rmax = max(max(R));
216 Rmin = min(min(R));
217 contrast = (Rmax-Rmin)/(Rmax+Rmin);
218 resolutions(j) = 1/contrast;
219
220 % Plot histogram and cut off point
221 if strcmp(method,'manual')==1
222     fig_handle_hist_thres_FFT_P = figure;
223     bar(loc(1:end-i), counts(1:end-i), 1, 'FaceColor', 'r', ...
224         'EdgeColor', 'k');
225     hold all;
226     bar(loc(end-i+1:end), counts(end-i+1:end), 1, ...
227         'FaceColor', [0.17, 0.51, 0.34], 'EdgeColor', 'k');
228     title(['Complex Moduli of Perturbed Bitmap, Threshold = ' ...
229         num2str(thres) 'intensity counts'], 'FontSize',20,...
230         'Interpreter','None');
231     axis([min(abs_P_tilda_v) max(abs_P_tilda_v) 0 max(counts)]);
232     set(gca, 'FontSize', 14);

```



```

233     legend('Excluded','Retained');
234     legend('boxoff');
235     xlabel('Pixel Intensity','FontSize',16);
236     ylabel('Counts','FontSize',16);
237
238     close(fig_handle_P_tilda,fig_handle_R,fig_handle_hist_thres_FFT_P);
239     end
240
241 end
242
243 figure(fig_handle_resolution);
244 plot(thresholds,resolutions,'Marker','o','MarkerEdgeColor','k',...
245     'MarkerFaceColor','k');
246 hold all;
247 if iteration>1
248     oldlabels = labels;
249     newlabel = {'Iteration' num2str(iteration)};
250     labels = vertcat(oldlabels,newlabel);
251 else
252     labels = {'Iteration' num2str(iteration)};
253 end
254 set(gca,'xscale','log','FontSize',12);
255 xlabel('# pixels retained in fft2(pertrubed bitmap)','FontSize',14);
256 ylabel("resolution" metric','FontSize',14);
257 legend(labels);
258
259 pos = find(resolutions==max(resolutions));
260 fprintf(1,'\t\tOptimal threshold for this iteration = %s\n',...
261     num2str(thresholds(pos)));
262 fprintf(fid,'\t\tOptimal threshold for this iteration = %s\n',...
263     num2str(thresholds(pos)));
264 fprintf(1,'\t\tResolution = %s\n',num2str(max(resolutions)));
265 fprintf(fid,'\t\tResolution = %s\n',num2str(max(resolutions)));
266
267 if strcmp(method,'manual')==1
268     choice = menu('Iterate again or stop?','Another Iteration',...
269         'Finished');
270 elseif strcmp(method,'auto')==1
271     if length(thresholds(pos))>1
272         choice = 2;
273     else choice = 1;
274     end
275 end
276
277 if choice == 1
278     iteration = iteration+1;
279 elseif choice == 2
280     iterate_flag = 0;
281 end
282
283 end
284
285 keyboard;
286
287 % Plot histogram and cut off point
288 fig_handle_hist_thres_FFT_P = figure;
289 bar(loc(1:end-i),counts(1:end-i),1,'FaceColor','r','EdgeColor','r');
290 hold all;
291 bar(loc(end-i+1:end),counts(end-i+1:end),1,...
292     'FaceColor',[0.17, 0.51, 0.34],'EdgeColor',[0.17, 0.51, 0.34]);
293 title(['Complex Moduli of Perturbed Bitmap, Threshold = ' ...
294     num2str(thres) 'intensity counts'],'FontSize',20,'Interpreter','None');
295 axis([min(abs_P_tilda_v) max(abs_P_tilda_v) 0 max(counts)]);
296 set(gca,'FontSize',14,'yscale','log');
297 legend('Excluded','Retained');
298 legend('boxoff');
299 xlabel('Pixel Intensity','FontSize',16);
300 ylabel('Counts','FontSize',16);

```

```

301
302 % Save R as an image rescaled so 0 = min(min(R)) and 255 = max(max(R))
303 R2 = R-min(min(R));
304 R2 = R2./max(max(R2));
305 fig_handle_R2 = figure;
306 imshow(R2);
307 saveas(fig_handle_R2, [save_path '\IFFT_RestingLattice_Cropped.fig'],...
308     'fig');
309
310 % Save figures:
311 fprintf(1, '\tSaving figures\n');
312 fprintf(fid, '\tSaving figures\n');
313 saveas(fig_handle_FFT_P, [save_path '\FFT_PerturbedLattice.fig'], 'fig');
314 saveas(fig_handle_hist_FFT_P, ...
315     [save_path '\Histogram_FFT_PerturbedLattice.fig'], 'fig');
316 saveas(fig_handle_resolution, ...
317     [save_path '\ResolutionMetric.fig'], 'fig');
318 saveas(fig_handle_hist_thres_FFT_P, ...
319     [save_path '\Histogram_FFT_PerturbedLattice_Thresholded.fig'], 'fig');
320
321 % Save 'RestingLattice' array in .mat and .txt form:
322 RL = R2;
323 fprintf(1, '\tSaving Resting Lattice array\n');
324 fprintf(fid, '\tSaving Resting Lattice array\n');
325 save('RestingLattice.mat', 'RL');
326 dlmwrite([save_path '\RestingLattice.txt'], RL, '\t');
327
328 % Update log file that function is completed:
329 fprintf(1, '%s completed\n', func_name);
330 fprintf(fid, '%s completed\n', func_name);
331
332 end

```

UndoCrop_v1.m

```

1 % Steven J. Henry
2 % 05/06/2014
7 %*****
8 % PURPOSE:
9 % This function "uncrops" an array given the array vertices. It effectively
10 % does this by padding the supplied array.
11
12 % ASSUMPTIONS:
13 % 'pad' is supplied in the form of [ypre,xpre,ypost,xpost]
14 %
15 % INPUT:
16 % A = input cropped array
17 % pad = padding dimensions [ypre,xpre,ypost,xpost]
18 % dilate = integer value by which to expand pixels for purpose of
19 % visualization, (rotation is performed on undilated image)
20 % save_path = string pointing to save location
21 % fid = file ID of log file to which progress is recorded
22 %
23 % OUTPUT:
24 % B = padded cropped array A
25 %
26 % DRIVER/FUNCTION MAP:
27 % n/a (calls no subroutines)
28 %*****
29
30 function [B] = UndoCrop_v1(A,pad,dilate,save_path,fid)
31
32 % Get function name:
33 func_name = mfilename;
34
35 % Update log file that function is running:
36 fprintf(1, '\n%s running ... \n', func_name);
37 fprintf(fid, '\n%s running ... \n', func_name);

```

```

38
39 % 'pad' is supplied in the form of [ypre,xpre,ypost,xpost]
40 B = padarray(A,[pad(1) pad(2)],'pre');
41 B = padarray(B,[pad(3) pad(4)],'post');
42
43 % Plot dilated bitmap
44 fig_handle_dilated_bitmap = figure;
45 imshow(imdilate(B,strel('disk',dilate)), 'DisplayRange',[]);
46 title_string = [num2str(dilate) ...
47   'X dilated uncropped bitmap of resting lattice centroids'];
48 title(title_string,'FontSize',20,'Interpreter','None');
49
50 % Save figure:
51 saveas(fig_handle_dilated_bitmap, [save_path ...
52   '\UncroppedDilatedRestingLatticeBitmap.fig'], 'fig');
53
54 % Save parameters:
55 UncroppedRestingLattice = B;
56 save([save_path '\UncroppedRestingLattice.mat'],'UncroppedRestingLattice');
57
58 % Update log file that function is completed:
59 fprintf(1, '%s completed\n', func_name);
60 fprintf(fid, '%s completed\n', func_name);
61
62 end

```

KilfoilInitialize_v3.m

```

1 % Steven J. Henry
2 % 05/08/2014
17 %*****
18 % PURPOSE:
19 % This function runs Maria Kilfoil's particle tracking routines to identify
20 % the resting lattice positions recovered from the Fourier filtering
21 % performed earlier.
22
23 % ASSUMPTIONS:
24 % User supplies non-meaningful fovn = 0 and frame = 0 values.
25 %
26 % INPUT:
27 % fovn = field of view number (should be zero)
28 % frame = frame # (should be zero)
29 % save_path = string pointing to save location
30 % fid = file ID of log file to which progress is recorded
31 %
32 % OUTPUT:
33 % M2 - All the features found from calling 'feature2D.m'
34 % MT - All the features from 'feature2D.m' which were accepted given the
35 % criteria in 'KilfoilRestingLatticeParameters'
36 % KilfoilRestingLatticeParameters = cell array storing parameters used to
37 % filter particles
38 %
39 % DRIVER/FUNCTION MAP:
40 % 0 KilfoilInitialize_v3.m
41 % 1 mpretrack_init.m (Kilfoil)
42 %*****
43
44 function [M2,MT,KilfoilRestingLatticeParameters]...
45 = KilfoilInitialize_v3(fovn,frame,save_path,fid)
46
47 % Get function name:
48 func_name = mfilename;
49
50 % Update log file that function is running:
51 fprintf(1, '\n%s running ... \n', func_name);
52 fprintf(fid, '\n%s running ... \n', func_name);
53
54 % Initialize parameters:

```

```

55 basepath = [save_path '\'];
56 featsize = 5; %radius of features in pixels
57 barint = 1; %minimum integrated intensity of feature below mask
58 barrg = 20; %maximum radius of gyration squared to be accepted
59 barcc = 1; %maximum eccentricity accepted, 0 = perfect circle
60 ldivRg = barint/barrg; %minimum ratio of integrated intensity to
61 %      radius of gyration squared
62 lmin = 0; % minimum intensity of local maximum to be considered
63 masscut = 0; % parameter which defines a threshold for integrated
64 %      intensity of features before position refinement,
65 %      to speed up the code
66 field = 2; % 0,1 if interlaced, 2 if full frame
67
68 prompt={'featsize';...
69 'barint';...
70 'barrg';...
71 'barcc'};
72 name='Input Parameters for initialize.m';
73 numlines=1;
74
75 % Turn iterate flag on:
76 iterate_flag = 1;
77
78 while iterate_flag == 1
79
80     defaultanswer=(num2str(featsize),...
81     num2str(barint),...
82     num2str(barrg),...
83     num2str(barcc));
84     options.Resize='on';
85     options.WindowStyle='normal';
86     options.Interpreter='tex';
87     answer=inputdlg(prompt,name,numlines,defaultanswer);
88
89     %radius of features in pixels
90     featsize = str2num(answer{1});
91     %#ok<*ST2NM> %minimum integrated intensity of feature below mask
92     barint = str2num(answer{2});
93     %maximum radius of gyration squared to be accepted
94     barrg = str2num(answer{3});
95     %maximum eccentricity accepted, 0 = perfect circle
96     barcc = str2num(answer{4});
97
98     fig_handle_temp = figure;
99     [M2, MT] = mpretrack_init(basepath, featsize, barint,...
100     barrg, barcc, ldivRg, fovn, frame, lmin, masscut, field);
101
102     % Compute histograms of mod(x,1) and mod(y,1) to screen for pixel bias:
103     nbins = sqrt(length(MT(:,1)));
104     [xcounts,xloc] = hist(mod(MT(:,1),1),nbins);
105     [ycounts,yloc] = hist(mod(MT(:,2),1),nbins);
106
107     % Plot mod(x,1) and mod(y,1) to screen for pixel bias:
108     fig_handle_modhist = figure;
109     subplot(1,2,1)
110     bar(xloc,xcounts,1,'FaceColor','b','EdgeColor','k');
111     title('x coordinate bias','FontSize',20,'Interpreter','None');
112     xlabel('mod(x,1) (pix)','FontSize',14);
113     ylabel('counts','FontSize',14);
114     subplot(1,2,2)
115     bar(yloc,ycounts,1,'FaceColor','b','EdgeColor','k');
116     title('y coordinate bias','FontSize',20,'Interpreter','None');
117     xlabel('mod(y,1) (pix)','FontSize',14);
118     ylabel('counts','FontSize',14);
119
120     % Radius of gyration (col 4) vs. Integrated intensity (col 3)
121     fig_handle_Rg = figure;
122     plot(M2(:,3),M2(:,4),'LineStyle','none','Marker','o',...

```

```

123     'MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',8);
124     hold on;
125     xlabel('Integrated Intensity');
126     ylabel('Rg');
127     plot(MT(:,3),MT(:,4), 'LineStyle','none','Marker','o',...
128         'MarkerEdgeColor','g','MarkerFaceColor','g','MarkerSize',8);
129     legend('All Features','Accepted Features','Location','NorthWest');
130
131     % Eccentricity (col 5) vs. Integrated intensity (col 3)
132     fig_handle_eccentricity = figure;
133     plot(M2(:,3),M2(:,5), 'LineStyle','none','Marker','o',...
134         'MarkerEdgeColor','r','MarkerFaceColor','r','MarkerSize',8);
135     hold on
136     xlabel('Integrated Intensity');
137     ylabel('Eccentricity');
138     plot(MT(:,3),MT(:,5), 'LineStyle','none','Marker','o',...
139         'MarkerEdgeColor','g','MarkerFaceColor','g','MarkerSize',8);
140     legend('All Features','Accepted Features','Location','NorthWest');
141
142     choice = menu('Satisfied with initialization?','Yes',...
143         'No, re-initialize');
144     if choice == 1
145         iterate_flag = 0;
146     else
147         close(fig_handle_temp,fig_handle_modhist,fig_handle_Rg,...
148             fig_handle_eccentricity);
149     end
150
151 end
152
153 %Save initialization parameters:
154 fprintf(1,'\n\tThe final initialization parameters were:\n');
155 fprintf(1,'\t\tbasepath = %s\n',basepath);
156 fprintf(1,'\t\tfeatsize = %s\n',answer{1});
157 fprintf(1,'\t\tbarint = %s\n',answer{2});
158 fprintf(1,'\t\tbarrg = %s\n',answer{3});
159 fprintf(1,'\t\tbarcc = %s\n',answer{4});
160 fprintf(1,'\t\tldivRg = %s\n',num2str(ldivRg));
161 fprintf(1,'\t\tfovn = %s\n',num2str(fovn));
162 fprintf(1,'\t\tframe = %s\n',num2str(frame));
163 fprintf(1,'\t\tlmin = %s\n',num2str(lmin));
164 fprintf(1,'\t\tmasscut = %s\n',num2str(masscut));
165 fprintf(1,'\t\tfield = %s\n',num2str(field));
166
167 fprintf(fid,'\n\tThe final initialization parameters were:\n');
168 fprintf(fid,'\t\tbasepath = %s\n',basepath);
169 fprintf(fid,'\t\tfeatsize = %s\n',answer{1});
170 fprintf(fid,'\t\tbarint = %s\n',answer{2});
171 fprintf(fid,'\t\tbarrg = %s\n',answer{3});
172 fprintf(fid,'\t\tbarcc = %s\n',answer{4});
173 fprintf(fid,'\t\tldivRg = %s\n',num2str(ldivRg));
174 fprintf(fid,'\t\tfovn = %s\n',num2str(fovn));
175 fprintf(fid,'\t\tframe = %s\n',num2str(frame));
176 fprintf(fid,'\t\tlmin = %s\n',num2str(lmin));
177 fprintf(fid,'\t\tmasscut = %s\n',num2str(masscut));
178 fprintf(fid,'\t\tfield = %s\n',num2str(field));
179
180 % Save figures:
181 fprintf(1,'\tSaving figure(s)\n');
182 fprintf(fid,'\tSaving figure(s)\n');
183 saveas(fig_handle_modhist, [save_path ...
184     '\Kilfoil_Initialize_PixelBiasScreen.fig'], 'fig');
185 saveas(fig_handle_Rg, [save_path '\Kilfoil_Initialize_Rg.fig'], 'fig');
186 saveas(fig_handle_eccentricity, [save_path ...
187     '\Kilfoil_Initialize_Eccentricity.fig'], 'fig');
188
189 % Save parameters:
190 fprintf(1,'\tSaving variable(s)\n');

```

```

191 fprintf(fid,'\tSaving variable(s)\n');
192 KilfoilRestingLatticeParameters = horzcat(prompt,answer);
193 save([save_path 'KilfoilRestingLatticeParameters.mat'],...
194     'KilfoilRestingLatticeParameters');
195 save([save_path 'MT.mat'],'MT');
196 save([save_path 'M2.mat'],'M2');
197
198 % Update log file that function is completed:
199 fprintf(1,'%s completed\n',func_name);
200 fprintf(fid,'%s completed\n',func_name);
201
202 end

```

TrajectoriesRelativeToRestingLattice_v9.m

```

1 % Steven J. Henry
2 % 09/22/2014
65 %*****
66 % PURPOSE: This function (1) searches for nearest neighbors in a perturbed
67 % lattice bitmap and corresponding resting lattice bitmap, (2) applies the
68 % shift so r(0) is relative to resting lattice (3) if a resting position
69 % can not be identified (e.g. b/c of cropping) the post has it's
70 % time-averaged centroid position subtracted
71 %
72 % ASSUMPTIONS:
73 % n/a
74 %
75 % INPUT:
76 % R = binary array with on (1) pixels located at centroid positions of
77 % resting lattice bitmap
78 % LUT = look-up-table mapping postID of each centroid in R
79 % microntopix = objective calibration for conversion of microns to pixels
80 % refine = refinement value to construct super-resolution bitmap
81 % bead_path = path to bead.mat files
82 % save_path = string pointing to save location
83 % fid = file ID of log file to which progress is recorded
84 %
85 % OUTPUT:
86 % filtered_shift = array containing distance from resting lattice position
87 % to pertured position of each post.
88 % sbead_path = path to folder created which contains sbead.mat files which
89 % are bead.mat files with an additional entry denoting the resting
90 % lattice position of each post.
91 %
92 % DRIVER/FUNCTION MAP:
93 % n/a (calls no subroutines)
94 %*****
95
96 function [filtered_shift,sbead_path]...
97     = TrajectoriesRelativeToRestingLattice_v9(R,P_LUT,microntopix,refine,...
98     bead_path,save_path,fid)
99
100 % Get function name:
101 func_name = mfilename;
102
103 % Update log file that function is running:
104 fprintf(1,'\n%s running ...\n',func_name);
105 fprintf(fid,'\n%s running ...\n',func_name);
106
107 % Establish list of coordinates:
108 [ypos_R,xpos_R] = find(R==1);
109 %concatenate so col1 = x-coord, col2 = y-coord
110 Rcoords = horzcat(xpos_R,ypos_R);
111 Pcoords = P_LUT(:,2:3);
112
113 % Perform nearest neighbors search on resting "R" and perturbed "P"
114 % lattices:
115 [idR,dist] = knnsearch(Rcoords,Pcoords);

```

```

116
117 unique_idR = unique(idR);
118 num_unique = length(unique_idR);
119 cull = zeros(length(idR),1);
120
121 % Sort through idR for duplicate hits:
122 for i = 1:num_unique
123     test_id = unique_idR(i);
124     test_indices = idR == test_id;
125     min_dist = min(dist(test_indices));
126     keep = (idR==test_id & dist==min_dist);
127     if sum(keep)==1
128         cull(keep) = 1;
129     else
130         new_keep = find(keep==1,1,'first');
131         cull(new_keep) = 1;
132     end
133 end
134
135 culled_idR = zeros(length(idR),1);
136 for i = 1:length(idR)
137     if cull(i) == 1
138         culled_idR(i) = idR(i);
139     else
140         culled_idR(i) = NaN;
141     end
142 end
143
144 shift = P_LUT;
145 x_tail = NaN(size(shift,1),1);
146 y_tail = NaN(size(shift,1),1);
147 u = NaN(size(shift,1),1);
148 v = NaN(size(shift,1),1);
149
150 for i = 1:length(culled_idR)
151     if isnan(culled_idR(i))==0
152
153         x_tail(i) = Rcoords(culled_idR(i),1);
154         y_tail(i) = Rcoords(culled_idR(i),2);
155
156         u(i) = Pcoords(i,1)-x_tail(i);
157         v(i) = Pcoords(i,2)-y_tail(i);
158
159         shift(i,2) = u(i);
160         shift(i,3) = v(i);
161
162     else %otherwise entry in FOC did not have nearest neighbor in R
163         shift(i,2) = NaN;
164         shift(i,3) = NaN;
165     end
166
167 end
168
169 % Make a new directory to store sbead.mat files:
170 newfolder = 'sbeads';
171 mkdir(save_path,newfolder);
172 sbead_path = [save_path filesep newfolder];
173
174 % Retrieve all contents that reside inside 'bead_path' folder:
175 contents = dir(sbead_path);
176
177 num_items = size(contents,1);
178 FileName = cell(num_items,1);
179 ID = NaN(num_items,1);
180 keep_ind = false(num_items,1);
181
182 for i = 1:num_items
183     item_name = contents(i).name;

```

```

184     if length(item_name) >= 10 && strcmp(item_name(1:5),'bead_')
185         FileName{i} = item_name;
186         ID(i) = str2double(item_name(6:end-4));
187         keep_ind(i) = true;
188     end
189 end
190
191 FileName = FileName(keep_ind);
192 ID = ID(keep_ind);
193
194 % How many post files did user select?
195 num_posts = length(FileName);
196
197 for i = 1:num_posts
198
199     if i == 1 || rem(i,50)==0 || i==num_posts
200         fprintf(1,'\tShifting post trajectory %s of %s\n',...
201             num2str(i),num2str(num_posts));
202         fprintf(fid,'\tShifting post trajectory %s of %s\n',...
203             num2str(i),num2str(num_posts));
204     end
205
206     r = find(shift(:,1)==ID(i));
207
208     if isempty(r)==0 && sum(isnan(shift(r,:)))==0
209
210         load([bead_path filesep FileName{i}], 'bsec');
211
212         x = bsec(:,1); %um
213         y = bsec(:,2); %um
214
215         % These are the coordinates of the resting lattice position or the
216         % time-average-centroid in the event no corresponding resting
217         % lattice position was located.
218         xshift = x(1)-shift(r,2)/(microntopix*refine);
219         yshift = y(1)-shift(r,3)/(microntopix*refine);
220
221         new_row = [xshift, yshift, 0, ID(i), 0];
222
223         bsec_old = bsec;
224         clear('bsec');
225         bsec = vertcat(new_row,bsec_old);
226
227         % Save file
228         save([sbead_path filesep 's' FileName{i}], 'bsec');
229
230     end
231
232 end
233
234 % Prepare a histogram of distances between perturbed and resting positions:
235 % Filter out 'dist' values not relevant:
236 del_idx = isnan(shift(:,2));
237 filtered_shift = shift(~del_idx,:);
238 filtered_dist = sqrt(filtered_shift(:,2).^2+filtered_shift(:,3).^2);
239 [dcounts,dloc] = hist(filtered_dist,sqrt(length(filtered_dist)));
240
241 fig_handle_hist = figure;
242 subplot(1,2,1)
243 bar(dloc,dcounts,1,'FaceColor','b','EdgeColor','b');
244 title('Pixel Basis','FontSize',20,'Interpreter','None');
245 axis([min(dloc) max(dloc) 0 max(dcounts)]);
246 set(gca,'FontSize',14);
247 xlabel('Distance Perturbed to Resting (pix)','FontSize',16);
248 ylabel('Counts','FontSize',16);
249
250 subplot(1,2,2)
251 dloc_um = dloc./(microntopix*refine);

```



```

252 bar(dloc_um,dcounts,1,'FaceColor','b','EdgeColor','b');
253 title(['microntopix*refine = ' num2str(microntopix) ' pix/um X ' ...
254   num2str(refine)], 'FontSize',20,'Interpreter','None');
255 axis([min(dloc_um) max(dloc_um) 0 max(dcounts)]);
256 set(gca,'FontSize',14);
257 xlabel('Distance Perturbed to Resting (\mum)','FontSize',16);
258 ylabel('Counts','FontSize',16);
259
260 % Prepare a quiver plot of vectors pointing from resting lattice
261 % (x_tail(i), y_tail(i)) to perturbed position in frame one (u(i),v(i)).
262 x_tail(del_idx)=[];
263 y_tail(del_idx)=[];
264 u(del_idx)=[];
265 v(del_idx)=[];
266 fig_handle_quiv = figure;
267 plot(Rcoords(:,1)/microntopix,Rcoords(:,2)/microntopix,...
268   'LineStyle','none','Marker','o','MarkerFaceColor','g',...
269   'MarkerEdgeColor','k','MarkerSize',4);
270 hold all
271 plot(Pcoords(:,1)/microntopix,Pcoords(:,2)/microntopix,...
272   'LineStyle','none','Marker','o','MarkerFaceColor','r',...
273   'MarkerEdgeColor','k','MarkerSize',4)
274 quiver(x_tail/microntopix,y_tail/microntopix,u/(microntopix*10),...
275   v/(microntopix*10));
276 axis('ij');
277 axis('square');
278 xlabel('x (\mum)','FontSize',16);
279 ylabel('y (\mum)','FontSize',16);
280 title('tail = resting lattice, head = frame 1 perturbed lattice',...
281   'FontSize',18);
282
283 % Prepare a scatter plot of differences in resting and perturbed lattice
284 % positions:
285 fig_handle_scatter = figure;
286 plot(shift(:,2),shift(:,3),'LineStyle','none','Marker','o',...
287   'MarkerFaceColor','b','MarkerSize',6);
288 axis('square');
289 xlabel(['\Deltax (superpix = pix/' num2str(refine) ')'],'FontSize',14);
290 ylabel(['\Deltay (superpix = pix/' num2str(refine) ')'],'FontSize',14);
291 title('Difference Resting to Perturbed','FontSize',16);
292
293 %Save plots if the figure handle exists as a variable:
294 fprintf(1,'\tSaving figure(s)...\n');
295 fprintf(fid,'\tSaving figure(s)...\n');
296 if exist('fig_handle_hist','var')==1
297   saveas(fig_handle_hist,[save_path ...
298     '\Histogram_DistancePerturbedToResting.fig']);
299 end
300 if exist('fig_handle_quiv','var')==1
301   saveas(fig_handle_quiv,[save_path '\Quiver_RestingToPerturbed.fig']);
302 end
303 if exist('fig_handle_scatter','var')==1
304   saveas(fig_handle_scatter,[save_path ...
305     '\Scatter_RestingToPerturbed.fig']);
306 end
307
308 % Update log file that function is completed:
309 fprintf(1,'%s completed\n',func_name);
310 fprintf(fid,'%s completed\n',func_name);
311
312 end

```

TrajectoriesInCellReferenceFrame_v5.m

```

1 % Steven J. Henry
2 % 09/22/2014
27 %*****
28 % PURPOSE: This function computes the position of post trajectories

```

```

29 % relative to a cell reference frame formed by the unit vectors parallel
30 % and orthogonal to the line that connects the user-defined centroid with
31 % that post. This is essentially translating the post position (x,y) from
32 % the lab reference frame (cartesian space) into a cell reference frame
33 % (still cartesian space but rotated and translated).
34 %
35 % ASSUMPTIONS:
36 % The user-supplied centroid position is supplied in the same frame of
37 % reference as the post trajectories were constructed. In MATLAB images are
38 % by default processed with the origin (0,0) located in the North West
39 % corner of the image, not the South West corner. This is natural when you
40 % consider that an image is an array of pixels and indexing into/out of
41 % that array requires specifying an (i,j) pair relative to the North
42 % West corner of the array.
43 %
44 % INPUT:
45 % sbead_path = path to folder which contains sbead.mat
46 % microntopix = calibration in units of pixels/um
47 % I = image that will be used for user to select approximate geometric
48 % centroid of cell
49 % save_path = string pointing to save location
50 % fid = file ID of log file to which progress is recorded
51 %
52 % OUTPUT:
53 % manual_centroid = user defined approximate geometric centroid
54 % rsbead_manualcentroid_path = path to folder containing rsbead.mat files
55 % files of same size as sbead.mat files but with positions translated
56 % into the cell reference frame (radial and tangential deflections)
57 %
58 % DRIVER/FUNCTION MAP:
59 % n/a (calls no subroutines)
60 %*****
61
62 function [manual_centroid,rsbead_manualcentroid_path]...
63     = TrajectoriesInCellReferenceFrame_v5(sbead_path,microntopix,I,...
64     save_path,fid)
65
66 % Get function name:
67 func_name = mfilename;
68
69 % Update log file that function is running:
70 fprintf(1,'\n%s running ...\n',func_name);
71 fprintf(fid,'\n%s running ...\n',func_name);
72
73 % Open an image and select approximate geometry centroid of cell:
74 fig_handle_manual_centroid = figure;
75 imshow(I,'DisplayRange',[]);
76 title('Click on approximate geometric centroid of cell','FontSize',14,...
77     'Interpreter','None');
78 h_point = impoint;
79 coords = wait(h_point);
80
81 xc = coords(1);
82 yc = coords(2);
83 manual_centroid = [xc,yc];
84
85 fprintf(1,'\tUser set (xc,yc) = (%s,%s) pix\n',num2str(xc),num2str(yc));
86 fprintf(fid,'\tUser set (xc,yc) = (%s,%s) pix\n',num2str(xc),num2str(yc));
87
88 % Retrieve all contents that reside inside 'sbead_path' folder:
89 contents = dir(sbead_path);
90
91 num_items = size(contents,1);
92 FileName = cell(num_items,1);
93 keep_ind = false(num_items,1);
94
95 for i = 1:num_items
96     item_name = contents(i).name;

```

```

97     if length(item_name) >= 11 && strcmp(item_name(1:6),'sbead_')
98         FileName{i} = item_name;
99         keep_ind(i) = true;
100     end
101 end
102
103 FileName = FileName(keep_ind);
104
105 % Make a new directory to store rsbead.mat files:
106 newfolder = 'rsbeads_manualcentroid';
107 mkdir(save_path,newfolder);
108 rsbead_manualcentroid_path = [save_path filesep newfolder];
109
110 % How many files:
111 num_posts = length(FileName);
112
113 for i = 1:num_posts
114     load([rsbead_path filesep FileName{i}], 'bsec');
115     r = bsec;
116     xlab = bsec(:,1)*microntopix;
117     ylab = bsec(:,2)*microntopix;
118     % Zero out old positions:
119     r(:,1) = 0;
120     r(:,2) = 0;
121     % Retrieve first observation of post:
122     xp1 = bsec(1,1)*microntopix;
123     yp1 = bsec(1,2)*microntopix;
124     % Construct unit vectors:
125     rmag = sqrt((xp1-xc)^2+(yp1-yc)^2);
126     r_para_hat = [xp1-xc,yp1-yc]/rmag;
127     r_perp_hat = [-yp1-yc,xp1-xc]/rmag;
128     % Translate coordinates:
129     r(:,1) = (xlab-xp1)*r_para_hat(1)+(ylab-yp1)*r_para_hat(2);
130     r(:,2) = (xlab-xp1)*r_perp_hat(1)+(ylab-yp1)*r_perp_hat(2);
131     % Back to microns:
132     r(:,1) = r(:,1)/microntopix;
133     r(:,2) = r(:,2)/microntopix;
134     % Clear original 'bsec'
135     clear('bsec');
136     bsec = r;
137     % Save file
138     save([rsbead_manualcentroid_path filesep 'r' FileName{i}], 'bsec');
139 end
140
141 fprintf(1, '\tSaving variable(s) and figure(s)\n');
142 fprintf(fid, '\tSaving variable(s) and figure(s)\n');
143
144 save([save_path filesep 'UserSetCentroid_pix.mat'], 'manual_centroid');
145 saveas(fig_handle_manual_centroid, [save_path filesep ...
146     'UserSetCentroid.fig'], 'fig');
147
148 % Update log file that function is completed:
149 fprintf(1, '%s completed\n', func_name);
150 fprintf(fid, '%s completed\n', func_name);
151
152 end

```

IdentifyEngagedPosts_v4.m

```

1 % Steven J. Henry
2 % 09/22/2014
21 %*****
22 % PURPOSE: A function that assists the user in selecting cell-engaged posts
23 % by plotting the variance of all posts. After dedrifting, non-cell engaged
24 % posts have extremely small variances compared to cell-engaged posts.
25 %
26 % ASSUMPTIONS:
27 % n/a

```

```

28 %
29 % INPUT:
30 % bead_path = path to bead.mat files
31 % rsbead_manualcentroid_path = path to folder containing rsbead.mat files
32 % files of same size as sbead.mat files but with positions translated
33 % into the cell reference frame (radial and tangential deflections)
34 % microntopix = calibration in units of pixels/um
35 % I = image that will be used for user to select approximate geometric
36 % centroid of cell
37 % save_path = string pointing to save location
38 % fid = file ID of log file to which progress is recorded
39 %
40 % OUTPUT:
41 % ID_in = list of post IDs declared cell-engaged
42 % ens_manualcentroid_rsbead_path = path to folder containing rsbead.mat
43 % files belonging to cell-engaged posts.
44 %
45 % DRIVER/FUNCTION MAP:
46 % n/a (calls no subroutines)
47 %*****
48
49 function [ID_in,ens_manualcentroid_rsbead_path]...
50     = IdentifyEngagedPosts_v4(bead_path,rsbead_manualcentroid_path,...
51     microntopix,I,save_path,fid)
52
53 % Get function name:
54 func_name = mfilename;
55
56 % Update log file that function is running:
57 fprintf(1,'\n%s running ...\n',func_name);
58 fprintf(fid,'\n%s running ...\n',func_name);
59
60 % Retrieve all contents that reside inside 'rsbead_manualcentroid_path'
61 % folder:
62 contents = dir(rsbead_manualcentroid_path);
63
64 num_items = size(contents,1);
65 FileName = cell(num_items,1);
66 ID = NaN(num_items,1);
67 keep_ind = false(num_items,1);
68
69 for i = 1:num_items
70     item_name = contents(i).name;
71     if length(item_name) >= 12 && strcmp(item_name(1:7),'rsbead_')
72         FileName{i} = item_name;
73         ID(i) = str2double(item_name(8:end-4));
74         keep_ind(i) = true;
75     end
76 end
77
78 FileName = FileName(keep_ind);
79 ID = ID(keep_ind);
80
81 % How many post files did user select?
82 num_posts = length(FileName);
83 x_var = zeros(num_posts,1);
84 y_var = x_var;
85
86 satisfied = 0;
87
88 while satisfied ~= 1
89
90     % If this is the first time in this loop have the user select the
91     % rsbead.mat files you're filtering:
92     if satisfied == 0
93
94         for i = 1:num_posts
95

```

```

96     if i == 1 || rem(i,50)==0 || i==num_posts
97         fprintf(1,'\tEvaluating post trajectory %s of %s\n',...
98             num2str(i),num2str(num_posts));
99     end
100
101     load([rsbead_manualcentroid_path filesep FileName{i}]);
102
103     x = bsec(2:end,1)*microntopix; %#ok<NODEF>
104     y = bsec(2:end,2)*microntopix;
105
106     x_var(i) = var(x);
107     y_var(i) = var(y);
108
109     end
110
111 end
112
113 fig_handle_var_ids = figure;
114 plot(x_var,y_var,'LineStyle','none','Marker','o',...
115     'MarkerEdgeColor','k','MarkerFaceColor','b');
116 hold all;
117 text(x_var,y_var,num2str(ID),'Color','k','FontSize',8);
118 axis('square');
119 set(gca,'FontSize',12);
120 xlabel('VAR(r(t)_||) (pix^2)','FontSize',14);
121 ylabel('VAR(r(t)_\perp) (pix^2)','FontSize',14);
122
123 fig_handle_var = figure;
124 plot(x_var,y_var,'LineStyle','none','Marker','o',...
125     'MarkerEdgeColor','k','MarkerFaceColor','b');
126 hold all;
127 axis('square');
128 set(gca,'FontSize',12);
129 xlabel('VAR(r(t)_||) (pix^2)','FontSize',14);
130 ylabel('VAR(r(t)_\perp) (pix^2)','FontSize',14);
131
132 h = impoly;
133 verts = wait(h);
134 data_in = inpolygon(x_var,y_var,verts(:,1),verts(:,2));
135
136 fig_handle_engaged_overlay = figure;
137 imshow(I);
138 axis('ij');
139 axis('square');
140 hold all;
141
142 num_in = sum(data_in);
143 ID_in = ID(data_in);
144
145 for j = 1:num_in
146
147     loadID = ID_in(j);
148
149     load([bead_path filesep 'bead_' num2str(loadID) '.mat']);
150
151     x = bsec(:,1)*microntopix; %pix
152     y = bsec(:,2)*microntopix; %pix
153
154     plot(x,y,'Color',[0.17, 0.51, 0.34]);
155     text(x(1)-5,y(1)-5,num2str(loadID),'Color','k','FontSize',6,...
156         'Background','r','Margin',1);
157
158 end
159
160 choice = menu('Satisfied with filter?','No, repeat','Yes, done');
161
162 if choice == 1
163     close(fig_handle_var,fig_handle_engaged_overlay);

```

```

164     % -1 avoids us having to reload the sbead.mat and bead.mat files
165     satisfied = -1;
166     else
167         satisfied = 1;
168     end
169
170 end
171
172 % Make a new directory to store a copy of the rsbead.mat files:
173 newfolder = 'ens_manualcentroid';
174 mkdir(rsbead_manualcentroid_path,newfolder);
175 ens_manualcentroid_rsbead_path = [rsbead_manualcentroid_path filesep ...
176     newfolder];
177
178 for k = 1:num_in
179
180     file = ['rsbead_' num2str(ID_in(k)) '.mat'];
181     source = [rsbead_manualcentroid_path filesep file];
182     copyfile(source,ens_manualcentroid_rsbead_path);
183
184 end
185
186 %Save plots:
187 fprintf(1,'\tSaving variable(s) and figure(s)\n');
188 fprintf(fid,'\tSaving variable(s) and figure(s)\n');
189
190 save([save_path filesep 'EngagedPostIDs.mat'],'ID_in');
191 saveas(fig_handle_var_ids,[save_path ...
192     '\VarianceFilterScatterPlot_IDs.fig']);
193 saveas(fig_handle_var,[save_path '\VarianceFilterScatterPlot.fig']);
194 saveas(fig_handle_engaged_overlay,[save_path ...
195     '\CellEngagedPostsOverlay.fig']);
196
197 end

```

ReviseCellRefTrajectories_v1.m

```

1  % Steven J. Henry
2  % 09/22/2014
9  %*****
10 % PURPOSE: After the set of cell-engaged posts has been determined using an
11 % approximate user-selected geometric centroid the true geometric centroid
12 % of the set of engaged posts is determined. Then the cell ref trajectories
13 % ('rsbead.mat') files are revised.
14 %
15 % ASSUMPTIONS:
16 % n/a
17 %
18 % INPUT:
19 % sbead_path = path to sbead.mat files
20 % engaged_post_IDs = list of post IDs declared cell-engaged
21 % microntopix = calibration in units of pixels/um
22 % save_path = string pointing to save location
23 % fid = file ID of log file to which progress is recorded
24 %
25 % OUTPUT:
26 % centroid = revised geometric centroid based-upon set of cell-engaged post
27 % rsbead_path = path to rsbead.mat files (now revised)
28 % ens_rsbead_path = path to cell-engaged rsbead.mat files (now revised)
29 %
30 % DRIVER/FUNCTION MAP:
31 % n/a (calls no subroutines)
32 %*****
33
34 function [centroid,rsbead_path,ens_rsbead_path]...
35     = ReviseCellRefTrajectories_v1(sbead_path,engaged_post_IDs,...
36     microntopix,save_path,fid)
37

```

```

38 % Get function name:
39 func_name = mfilename;
40
41 % Update log file that function is running:
42 fprintf(1, '\n%s running ...\n', func_name);
43 fprintf(fid, '\n%s running ...\n', func_name);
44
45 % Compute the actual geometric centroid of the set of engaged posts using
46 % their associated resting lattice positions:
47 num_engaged = length(engaged_post_IDs);
48 xp = zeros(num_engaged, 1);
49 yp = xp;
50 for i = 1:num_engaged
51     ID = engaged_post_IDs(i);
52     load([sbead_path filesep 'sbead_' num2str(ID) '.mat'], 'bsec');
53     xp(i) = bsec(1, 1) * microntopix; %pix
54     yp(i) = bsec(1, 2) * microntopix; %pix
55 end
56 xc = mean(xp);
57 yc = mean(yp);
58 centroid = [xc, yc];
59
60 fprintf(1, ...
61     '\tGeometric centroid of engaged posts (xc,yc) = (%s,%s) pix\n', ...
62     num2str(xc), num2str(yc));
63 fprintf(fid, ...
64     '\tGeometric centroid of engaged posts (xc,yc) = (%s,%s) pix\n', ...
65     num2str(xc), num2str(yc));
66
67 % Retrieve all contents that reside inside 'sbead_path' folder:
68 contents = dir(sbead_path);
69
70 num_items = size(contents, 1);
71 FileName = cell(num_items, 1);
72 keep_ind = false(num_items, 1);
73
74 for i = 1:num_items
75     item_name = contents(i).name;
76     if length(item_name) >= 11 && strcmp(item_name(1:6), 'sbead_')
77         FileName(i) = item_name;
78         keep_ind(i) = true;
79     end
80 end
81
82 FileName = FileName(keep_ind);
83
84 % Make a new directory to store rsbead.mat files:
85 newfolder = 'rsbeads';
86 mkdir(save_path, newfolder);
87 rsbead_path = [save_path filesep newfolder];
88
89 % How many files:
90 num_posts = length(FileName);
91
92 for i = 1:num_posts
93     load([sbead_path filesep FileName{i}], 'bsec');
94     r = bsec;
95     xlab = bsec(:, 1) * microntopix;
96     ylab = bsec(:, 2) * microntopix;
97     % Zero out old positions:
98     r(:, 1) = 0;
99     r(:, 2) = 0;
100    % Retrieve first observation of post:
101    xp1 = bsec(1, 1) * microntopix;
102    yp1 = bsec(1, 2) * microntopix;
103    % Construct unit vectors:
104    rmag = sqrt((xp1 - xc)^2 + (yp1 - yc)^2);
105    r_para_hat = [xp1 - xc, yp1 - yc] ./ rmag;

```

```

106 r_perp_hat = [-(yp1-yc),xp1-xc]/rmag;
107 % Translate coordinates:
108 r(:,1) = (xlab-xp1)*r_para_hat(1)+(ylab-yp1)*r_para_hat(2);
109 r(:,2) = (xlab-xp1)*r_perp_hat(1)+(ylab-yp1)*r_perp_hat(2);
110 % Back to microns:
111 r(:,1) = r(:,1)/microntopix;
112 r(:,2) = r(:,2)/microntopix;
113 % Clear original 'bsec'
114 clear('bsec');
115 bsec = r;
116 % Save file
117 save([rsbead_path filesep 'r' FileName{i}], 'bsec');
118 end
119
120 % Now retrieve the revised 'rsbead.mat' files that belong to the set of
121 % engaged posts:
122 newfolder = 'ens';
123 mkdir(rsbead_path,newfolder);
124 ens_rsbead_path = [rsbead_path filesep newfolder];
125 for i = 1:num_engaged
126     ID = engaged_post_IDs(i);
127     file = ['rsbead_' num2str(ID) '.mat'];
128     source = [rsbead_path filesep file];
129     destination = [ens_rsbead_path filesep file];
130     copyfile(source,destination);
131 end
132
133 fprintf(1, '\tSaving variable(s) and figure(s)\n');
134 fprintf(fid, '\tSaving variable(s) and figure(s)\n');
135
136 save([save_path filesep 'GeometricCentroid_pix.mat'], 'centroid');
137
138 % Update log file that function is completed:
139 fprintf(1, '%s completed\n', func_name);
140 fprintf(fid, '%s completed\n', func_name);
141
142 end

```

GeoSortEngagedPosts_v2.m

```

1 % Steven J. Henry
2 % 09/22/2014
3 %*****
14 %
15 % PURPOSE: This has the user set a threshold in nearest-neighbor distance
16 % to sort peripheral posts from core posts.
17 %
18 % ASSUMPTIONS:
19 % n/a
20 %
21 % INPUT:
22 % sbead_path = path to sbead.mat files
23 % rsbead_path = path to rsbead.mat files
24 % microntopix = calibration in units of pixels/um
25 % engaged_IDs = list of cell-engaged post IDs
26 % I = image for superposition of geometrically sorted post IDs
27 % save_path = string pointing to save location
28 % fid = file ID of log file to which progress is recorded
29 %
30 % OUTPUT:
31 % core_IDs = list of posts residing in geometric core
32 % core_rsbead_path = path to rsbead.mat files belonging to core posts
33 % perim_IDs = list of posts residing at geometric periphery
34 % perim_rsbead_path = path to rsbead.mat files belonging to periphery posts
35 %
36 % DRIVER/FUNCTION MAP:
37 % n/a (calls no subroutines)
38 %*****
39

```



```

40 function [core_IDs, core_rsbead_path, perim_IDs, perim_rsbead_path]...
41 = GeoSortEngagedPosts_v2(sbead_path,rsbead_path,microntopix,...
42 engaged_IDs,l,save_path,fid)
43
44 % Get function name:
45 func_name = mfilename;
46
47 % Update log file that function is running:
48 fprintf(1,'\n%s running ...\n',func_name);
49 fprintf(fid,'\n%s running ...\n',func_name);
50
51 % Retrieve all contents that reside inside 'sbead_path' folder:
52 contents = dir(sbead_path);
53
54 num_items = size(contents,1);
55 FileName = cell(num_items,1);
56 ID = NaN(num_items,1);
57 keep_ind = false(num_items,1);
58
59 for i = 1:num_items
60     item_name = contents(i).name;
61     if length(item_name) >= 11 && strcmp(item_name(1:6),'sbead_')
62         FileName(i) = item_name;
63         ID(i) = str2double(item_name(7:end-4));
64         keep_ind(i) = true;
65     end
66 end
67
68 FileName = FileName(keep_ind);
69 ID = ID(keep_ind);
70
71
72 % Loop over all the files in 'FileName' and retain only those that have and
73 % ID that is also found within the 'engaged_IDs' list:
74 num_files = length(FileName);
75 resting_coords = NaN(num_files,6);
76 keep_ind = false(num_files,1);
77 for i = 1:num_files
78
79     engaged_test = ID(i) == engaged_IDs;
80
81     if sum(engaged_test)>0
82         keep_ind(i) = true;
83         load([sbead_path filesep FileName{i}]);
84         x = bsec(1,1)*microntopix; %pix
85         y = bsec(1,2)*microntopix; %pix
86         resting_coords(i,1) = ID(i);
87         resting_coords(i,2) = x;
88         resting_coords(i,3) = y;
89     end
90
91 end
92
93 % Eliminate rows without data as these rows were reserved for files within
94 % 'FileName' that did not belong to engaged posts.
95 resting_coords = resting_coords(keep_ind,:);
96
97 % For each entry in 'resting_coords' compute the mean distance to the six
98 % nearest neighbors:
99 num_posts = size(resting_coords,1);
100
101 for i = 1:num_posts
102
103     xy = resting_coords(i,2:3);
104
105     keep_ind = true(num_posts,1);
106     keep_ind(i) = false;
107

```

```

108 xy_subset = resting_coords(keep_ind,2:3);
109
110 [~,dist] = knnsearch(xy_subset,xy,'k',6);
111
112 resting_coords(i,4) = mean(dist);
113 resting_coords(i,5) = std(dist);
114 resting_coords(i,6) = nnz(dist);
115
116 end
117
118 % Plot a histogram of the average 6 nearest neighbor distances for each
119 % post
120 [dcounts,dloc] = hist(resting_coords(:,4),sqrt(num_posts));
121
122 fig_handle_dist = figure;
123 subplot(1,2,1)
124 bar(dloc,dcounts,1,'FaceColor','b','EdgeColor','b');
125 axis([min(dloc) max(dloc) 0 max(dcounts)]);
126 set(gca,'FontSize',14);
127 xlabel('<knn> dist (pix), k = 6','FontSize',16);
128 ylabel('Counts','FontSize',16);
129
130 % Plot scatter plot of the average 6 nearest neighbor distances for each
131 % post with the associated standard deviation in rank order from smallest
132 % to largest
133 [~,sort_ind] = sort(resting_coords(:,4),'ascend');
134 sorted_resting_coords = resting_coords(sort_ind,:);
135
136 subplot(1,2,2)
137 errorbar(sorted_resting_coords(:,4),sorted_resting_coords(:,5),...
138 'LineStyle','none','Color',[0.83,0.82,0.78],'Marker','o',...
139 'MarkerFaceColor','b','MarkerEdgeColor','b','MarkerSize',8);
140 set(gca,'FontSize',14);
141 xlabel('Rank Order','FontSize',16);
142 ylabel('<knn> dist (pix) \pm SD, k = 6','FontSize',16);
143
144 % Have user set a threshold above which will be declared peripheral posts,
145 % below which will be declared core posts:
146 thres = input('Set nearest neighbor distance (pix) threshold below which posts are "core", above which posts are
"perim" = \n');
147
148 fprintf(1,'\tUser set core vs. peripheral nearest neighbor distance threshold = %s pix\n',num2str(thres));
149 fprintf(fid,'\tUser set core vs. peripheral nearest neighbor distance threshold = %s pix\n',num2str(thres));
150
151 core_ind = sorted_resting_coords(:,4)<thres;
152 core_IDs = sorted_resting_coords(core_ind,1);
153 perim_IDs = sorted_resting_coords(~core_ind,1);
154
155 % Overlay the post IDs for each category (core vs. perim):
156 fig_handle_ids = figure;
157
158 % Make a new directory to store a copy of the rsbead.mat files belonging to
159 % the core posts:
160 newfolder = 'core';
161 mkdir(rsbead_path,newfolder);
162 core_rsbead_path = [rsbead_path filesep newfolder];
163 num_core = length(core_IDs);
164 for k = 1:num_core
165 file = ['rsbead_' num2str(core_IDs(k)) '.mat'];
166 source = [rsbead_path filesep file];
167 copyfile(source,core_rsbead_path);
168
169 load([rsbead_path filesep 'sbead_' num2str(core_IDs(k)) '.mat'],'bsec');
170 xo = bsec(1,1)*microntopix; %pix
171 yo = bsec(1,2)*microntopix; %pix
172 subplot(1,2,1);
173 if k == 1
174 imshow(l,'DisplayRange',[]);

```

```

175     axis('ij');
176     hold all;
177     title('Core Posts',FontSize,16);
178     xlabel('x coord (pix)',FontSize,14);
179     ylabel('y coord (pix)',FontSize,14);
180     end
181     text(xo,yo,num2str(core_IDs(k)),'Color','k','FontSize',8,...
182         'Background',[0.17, 0.51, 0.34],'Margin',1);
183 end
184
185 % Make a new directory to store a copy of the rsbead.mat files belonging to
186 % the peripheral posts:
187 newfolder = 'perim';
188 mkdir(rsbead_path,newfolder);
189 perim_rsbead_path = [rsbead_path filesep newfolder];
190 num_perim = length(perim_IDs);
191 for k = 1:num_perim
192     file = ['rsbead_' num2str(perim_IDs(k)) '.mat'];
193     source = [rsbead_path filesep file];
194     copyfile(source,perim_rsbead_path);
195
196     load([sbead_path filesep 'sbead_' num2str(perim_IDs(k)) '.mat'],...
197         'bsec');
198     xo = bsec(1,1)*microntopix; %pix
199     yo = bsec(1,2)*microntopix; %pix
200     subplot(1,2,2);
201     if k == 1
202         imshow(I,'DisplayRange',[]);
203         axis('ij');
204         hold all;
205         title('Perim Posts',FontSize,16);
206         xlabel('x coord (pix)',FontSize,14);
207         ylabel('y coord (pix)',FontSize,14);
208     end
209     text(xo,yo,num2str(perim_IDs(k)),'Color','k','FontSize',8,...
210         'Background','r','Margin',1);
211 end
212
213 % Save figure:
214 fprintf(1,'\tSaving variable(s) and figure(s)\n');
215 fprintf(fid,'\tSaving variable(s) and figure(s)\n');
216
217 save([save_path filesep 'Core_PostIDs.mat'],core_IDs);
218 save([save_path filesep 'Perim_PostIDs.mat'],perim_IDs);
219 saveas(fig_handle_dist,[save_path filesep 'PerimVsCoreKnnDist.fig']);
220 saveas(fig_handle_ids,[save_path filesep 'PerimVsCoreIDsOverlay.fig']);
221
222 % Update log file that function is completed:
223 fprintf(1,'%s completed\n',func_name);
224 fprintf(fid,'%s completed\n',func_name);
225
226 end

```

RepopulatePostIDLists_v2.m

```

1 % Steven J. Henry
2 % 09/22/2014
15 %*****
16 % PURPOSE: User selects folder containing rsbead.mat files for three
17 % categories: ensemble, core, and peripheral. The function records
18 % the IDs of the rsbead.mat files within those respective folders.
19 %
20 % ASSUMPTIONS:
21 % n/a
22 %
23 % INPUT:
24 % ens_rsbead_path = path to rsbead.mat files of cell-engaged posts
25 % core_rsbead_path = path to rsbead.mat files of core cell-engaged posts

```

```

26 % perim_rsbead_path = path to rsbead.mat files of peripheral cell-engaged
27 % posts
28 % save_path = string pointing to save location
29 % fid = file ID of log file to which progress is recorded
30 %
31 % OUTPUT:
32 % engaged_post_IDs = list of cell-engaged posts
33 % core_post_IDs = list of core cell-engaged posts
34 % perim_post_IDs = list of peripheral cell-engaged posts
35 %
36 % DRIVER/FUNCTION MAP:
37 % n/a (calls no subroutines)
38 %*****
39
40 function [engaged_post_IDs, core_post_IDs, perim_post_IDs]...
41     = RepopulatePostIDLists_v2(ens_rsbead_path,core_rsbead_path,...
42     perim_rsbead_path,save_path,fid)
43
44 % Get function name:
45 func_name = mfilename;
46
47 % Update log file that function is running:
48 fprintf(1,'\n%s running ...\n',func_name);
49 fprintf(fid,'\n%s running ...\n',func_name);
50
51 for i = 1:3
52
53     if i == 1
54         % Set path to folder containing ensemble of engaged posts:
55         path = ens_rsbead_path;
56     elseif i == 2
57         % Set path to folder containing ensemble of engaged posts:
58         path = core_rsbead_path;
59     elseif i == 3
60         % Set path to folder containing ensemble of engaged posts:
61         path = perim_rsbead_path;
62     end
63
64     contents = dir(path);
65
66     num_items = size(contents,1);
67     ID = NaN(num_items,1);
68
69     for j = 1:num_items
70         item_name = contents(j).name;
71         if length(item_name) >= 12 && strcmp(item_name(1:7),'rsbead_')
72             ID(j) = str2double(item_name(8:end-4));
73         end
74     end
75
76     del_ind = isnan(ID);
77     ID(del_ind) = [];
78
79     if i == 1
80         engaged_post_IDs = ID;
81     elseif i == 2
82         core_post_IDs = ID;
83     elseif i == 3
84         perim_post_IDs = ID;
85     end
86
87     clear('ID');
88
89 end
90
91 % Save figure:
92 fprintf(1,'\tSaving variable(s) and figure(s)\n');
93 fprintf(fid,'\tSaving variable(s) and figure(s)\n');

```

```

94
95 save([save_path filesep 'ManualEdit_Ens_PostIDs.mat'],'engaged_post_IDs');
96 save([save_path filesep 'ManualEdit_Core_PostIDs.mat'],'core_post_IDs');
97 save([save_path filesep 'ManualEdit_Perim_PostIDs.mat'],'perim_post_IDs');
98
99 % Update log file that function is completed:
100 fprintf(1,'%s completed\n',func_name);
101 fprintf(fid,'%s completed\n',func_name);
102
103 end

```

PlotCellRefTrajectories_v7.m

```

1 % Steven J. Henry
2 % 09/26/2014
37 %*****
38 % PURPOSE: This function plots the post trajectories that were previously
39 % translated to a cell-reference frame.
40 %
41 % ASSUMPTIONS:
42 % n/a
43 %
44 % INPUT:
45 % cat_rsbead_path = path to rsbead.mat files for category 'cat'
46 % time_int = interval between frames in seconds
47 % kspring = spring constant in pN/nm
48 % cat = string denoting category being analyzed
49 % color = plotting color a string or vector
50 % save_path = string pointing to save location
51 % fid = file ID of log file to which progress is recorded
52 % transition_time = time (s) after Fmax to consider system at steady state
53 %
54 % OUTPUT:
55 % mpara = ensemble average of radial deflections (nm)
56 % sdpara = std of ensemble of radial deflections (nm)
57 % cpara = count of contributors to ensemble of radial deflections (#)
58 % separa = standard error of ensemble of radial deflections (nm)
59 % mperp = ensemble average of tangential deflections (nm)
60 % sdperp = std of ensemble of tangential deflections (nm)
61 % cperp = count of contributors to ensemble of tangential deflections (#)
62 % seperp = standard error of ensemble of tangential deflections (nm)
63 % mpara_F = mpara*kspring (pN)
64 % sdpara_F = sdpara*kspring (pN)
65 % cpara_F = cpara (#)
66 % separa_F = separa*kspring (pN)
67 % mperp_F = mperp*kspring (pN)
68 % sdperp_F = sdperp*kspring (pN)
69 % cperp_F = cperp (#)
70 % seperp_F = seperp*kspring (pN)
71 % t = time vector
72 % Fmax = radial maximum force observed for each post
73 % Fss = radial mean steady state force observed for each post
74 %
75 % DRIVER/FUNCTION MAP:
76 % n/a (calls no subroutines)
77 %*****
78
79 function [mpara,sdpara,cpara,separa,...
80          mperp,sdperp,cperp,seperp,...
81          mpara_F,sdpara_F,cpara_F,separa_F,...
82          mperp_F,sdperp_F,cperp_F,seperp_F,...
83          t,Fmax,Fss]...
84 = PlotCellRefTrajectories_v7(cat_rsbead_path,time_int,kspring,...
85 cat,color,save_path,fid,transition_time)
86
87 % Get function name:
88 func_name = mfilename;
89

```

```

90 % Update log file that function is running:
91 fprintf(1,'\n%s running ...\n',func_name);
92 fprintf(fid,'\n%s running ...\n',func_name);
93
94 % Retrieve all contents that reside inside 'cat_rsbead_path' folder:
95 contents = dir(cat_rsbead_path);
96
97 num_items = size(contents,1);
98 FileName = cell(num_items,1);
99 keep_ind = false(num_items,1);
100
101 for i = 1:num_items
102     item_name = contents(i).name;
103     if length(item_name) >= 12 && strcmp(item_name(1:7),'rsbead_')
104         FileName{i} = item_name;
105         keep_ind(i) = true;
106     end
107 end
108
109 FileName = FileName(keep_ind);
110
111 % How many files:
112 n = length(FileName);
113 ID = NaN(n,1);
114
115 max_frame = [];
116
117 ymin = 0;
118 ymax = 0;
119
120 for i = 1:n
121     load([cat_rsbead_path filesep FileName{i}], 'bsec');
122     rpara = bsec(:,1)*1000; %#ok<NODEF> % nm
123     rperp = bsec(:,2)*1000; % nm
124     frames = bsec(:,3);
125     % Exclude row 1 which contains the distance to the resting lattice
126     % position from the resting lattice position (i.e. 0).
127     rpara(1) = [];
128     rperp(1) = [];
129     frames(1) = [];
130
131     ID(i) = bsec(1,4);
132
133     if isempty(max_frame)
134         max_frame = max(frames);
135     elseif max(frames)>max_frame
136         max_frame = max(frames);
137     end
138
139     if min(rpara)<ymin
140         ymin = min(rpara);
141     end
142     if min(rperp)<ymin
143         ymin = min(rperp);
144     end
145     if max(rpara)>ymax;
146         ymax = max(rpara);
147     end
148     if max(rperp)>ymax;
149         ymax = max(rperp);
150     end
151
152     t = frames*time_int;
153
154     if i == 1
155         fig_handle_r = figure;
156         fig_handle_f = figure;
157         labels = cell(n,1);

```

```

158 end
159
160 % Plot r vs. t trajectory:
161 figure(fig_handle_r)
162 subplot(1,2,1);
163 plot(t,rpara,'LineStyle','-','Color',color,'Marker',none);
164 if i == 1
165     title(['Individual r(t)_||', ' cat'],'FontSize',20,...
166           'Interpreter','Tex');
167     set(gca,'FontSize',14);
168     xlabel('time (s)','FontSize',16);
169     ylabel('r(t)_|| (nm)','FontSize',16);
170 end
171 hold all;
172 if i == n
173     axis([0 max_frame*time_int ymin ymax]);
174 end
175
176 subplot(1,2,2);
177 plot(t,rperp,'LineStyle','-','Color',color,'Marker',none);
178 if i == 1
179     title(['Individual r(t)_\perp, ' cat'],'FontSize',20,...
180           'Interpreter','Tex');
181     set(gca,'FontSize',14);
182     xlabel('time (s)','FontSize',16);
183     ylabel('r(t)_\perp (nm)','FontSize',16);
184 end
185 hold all;
186 if i == n
187     axis([0 max_frame*time_int ymin ymax]);
188 end
189 labels{i} = ['post' num2str(ID(i))];
190
191 % Plot f vs. t trajectory:
192 figure(fig_handle_f)
193 subplot(1,2,1);
194 plot(t,rpara*kspring,'LineStyle','-','Color',color,'Marker',none);
195 if i == 1
196     title(['Individual f(t)_||, ' cat'],'FontSize',20,...
197           'Interpreter','Tex');
198     set(gca,'FontSize',14);
199     xlabel('time (s)','FontSize',16);
200     ylabel('f(t)_|| (pN)','FontSize',16);
201 end
202 hold all;
203 if i == n
204     axis([0 max_frame*time_int ymin*kspring ymax*kspring]);
205 end
206
207 subplot(1,2,2);
208 plot(t,rperp*kspring,'LineStyle','-','Color',color,'Marker',none);
209 if i == 1
210     title(['Individual f(t)_\perp, ' cat'],'FontSize',20,...
211           'Interpreter','Tex');
212     set(gca,'FontSize',14);
213     xlabel('time (s)','FontSize',16);
214     ylabel('f(t)_\perp (pN)','FontSize',16);
215 end
216 hold all;
217 if i == n
218     axis([0 max_frame*time_int ymin*kspring ymax*kspring]);
219 end
220 labels{i} = ['post' num2str(ID(i))];
221
222 end
223
224 for k = 1:2
225     if k == 1

```

```

226     figure(fig_handle_r);
227     elseif k == 2
228         figure(fig_handle_f);
229     end
230
231     subplot(1,2,1);
232     legend(labels);
233     legend('hide');
234     subplot(1,2,2);
235     legend(labels);
236     legend('hide');
237 end
238
239 % Create array for averaging:
240 Rpara = NaN(max_frame,n);
241 Rperp = Rpara;
242
243 for j = 1:n
244     load([cat_rsbead_path filesep FileName{j}], 'bsec');
245     rpara = bsec(:,1);% um
246     rperp = bsec(:,2);% um
247     frames = bsec(:,3);
248     % Exclude row 1 which contains the distance to the resting lattice
249     % position from the resting lattice position (i.e. 0).
250     rpara(1) = [];
251     rperp(1) = [];
252     frames(1) = [];
253
254     n_frames = length(frames);
255     for k = 1:n_frames
256         print_row = frames(k);
257         Rpara(print_row,j) = rpara(k);
258         Rperp(print_row,j) = rperp(k);
259     end
260 end
261
262 % Convert to nanometers
263 Rpara = Rpara*1000;
264 Rperp = Rperp*1000;
265
266 % Convert to piconewtons
267 Rpara_F = Rpara*kspring;
268 Rperp_F = Rperp*kspring;
269
270 [rows,cols] = size(Rpara);
271 mpara = NaN(rows,1);
272 sdpara = mpara;
273 cpara = mpara;
274 mperp = mpara;
275 sdperp = mpara;
276 cperp = mpara;
277
278 mpara_F = mpara;
279 sdpara_F = mpara;
280 cpara_F = mpara;
281 mperp_F = mpara;
282 sdperp_F = mpara;
283 cperp_F = mpara;
284
285 for k = 1:rows
286     keep_id1 = ~isnan(Rpara(k,:));
287     mpara(k) = mean(Rpara(k,keep_id1));
288     sdpara(k) = std(Rpara(k,keep_id1));
289     cpara(k) = sum(keep_id1);
290
291     keep_id2 = ~isnan(Rperp(k,:));
292     mperp(k) = mean(Rperp(k,keep_id2));
293     sdperp(k) = std(Rperp(k,keep_id2));

```



```

294 cperp(k) = sum(keep_id2);
295
296 keep_id3 = ~isnan(Rpara_F(k,:));
297 mpara_F(k) = mean(Rpara_F(k,keep_id3));
298 sdpara_F(k) = std(Rpara_F(k,keep_id3));
299 cpara_F(k) = sum(keep_id3);
300
301 keep_id4 = ~isnan(Rperp_F(k,:));
302 mperp_F(k) = mean(Rperp_F(k,keep_id4));
303 sdperp_F(k) = std(Rperp_F(k,keep_id4));
304 cperp_F(k) = sum(keep_id4);
305 end
306
307 separa = sdpara./sqrt(cpara);
308 seperp = sdperp./sqrt(cperp);
309 separa_F = sdpara_F./sqrt(cpara_F);
310 seperp_F = sdperp_F./sqrt(cperp_F);
311
312 t = (0:size(Rpara,1)-1)*time_int;
313
314 % Retrieve force of each post at Fmax of mpara_F and save to array for
315 % future plotting.
316 [~,row_of_max] = max(mpara_F);
317 Fmax_vec = Rpara_F(row_of_max,:);
318 keep_id = ~isnan(Fmax_vec);
319 Fmax_vec = Fmax_vec(keep_id);
320 ID_vec = ID(keep_id);
321 Fmax = horzcat(ID_vec,Fmax_vec);
322
323 % Retrieve steady state force of each post after 'transition_time' past
324 % t_max
325 t_Fmax = t(row_of_max);
326 Fss = NaN(size(Fmax,1),3);
327 for k = 1:cols
328     f_trajec = Rpara(:,k);
329     keep_ind = ~isnan(f_trajec);
330     t_trajec = t(keep_ind);
331     f_trajec = f_trajec(keep_ind);
332     avg_ind = t_trajec > t_Fmax+transition_time;
333     Fss(k,1) = ID(k);
334     Fss(k,2) = mean(f_trajec(avg_ind));
335     Fss(k,3) = std(f_trajec(avg_ind));
336 end
337
338 % Displacements plots standard deviation bars
339 ymin = zeros(1,2);
340 ymax = ymin;
341 ymin(1) = min(mpara-sdpara);
342 ymin(2) = min(mperp-sdperp);
343 ymin = min(ymin);
344 ymax(1) = max(mpara+sdpara);
345 ymax(2) = max(mperp+sdperp);
346 ymax = max(ymax);
347
348 fig_handle_rsdbars = figure;
349 subplot(1,2,1);
350 errorbar(t,mpara,sdpara,'LineStyle','none','Color',[0.83,0.82,0.78],...
351     'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,...
352     'MarkerSize',8);
353 title(['Ensemble Average r(t)_|_> ', ' cat'],'FontSize',20,...
354     'Interpreter','Tex');
355 set(gca,'FontSize',14);
356 xlabel('time (s)','FontSize',16);
357 ylabel('<r(t)_|_> \pm SD (nm)','FontSize',16);
358 axis([0 t(end) ymin ymax]);
359
360 subplot(1,2,2);
361 errorbar(t,mperp,sdperp,'LineStyle','none','Color',[0.83,0.82,0.78],...

```

```

362 'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,...
363 'MarkerSize',8);
364 title(['Ensemble Average  $r(t)_{\perp}$ , ' cat],FontSize,20,...
365 'Interpreter','Tex');
366 set(gca,FontSize,14);
367 xlabel('time (s)',FontSize,16);
368 ylabel('< $r(t)_{\perp}$ > \pm SD (nm)',FontSize,16);
369 axis([0 t(end) ymin ymax]);
370
371 % Displacement plots standard error bars
372 ymin = zeros(1,2);
373 ymax = ymin;
374 ymin(1) = min(mpara-separa);
375 ymin(2) = min(mperp-seperp);
376 ymin = min(ymin);
377 ymax(1) = max(mpara+separa);
378 ymax(2) = max(mperp+seperp);
379 ymax = max(ymax);
380
381 fig_handle_rsebars = figure;
382 subplot(1,2,1);
383 errorbar(t,mpara,separa,'LineStyle','none','Color',[0.83,0.82,0.78],...
384 'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,...
385 'MarkerSize',8);
386 title(['Ensemble Average  $r(t)_{\perp}$ , ' cat],FontSize,20,...
387 'Interpreter','Tex');
388 set(gca,FontSize,14);
389 xlabel('time (s)',FontSize,16);
390 ylabel('< $r(t)_{\perp}$ > \pm SE (nm)',FontSize,16);
391 axis([0 t(end) ymin ymax]);
392
393 subplot(1,2,2);
394 errorbar(t,mperp,seperp,'LineStyle','none','Color',[0.83,0.82,0.78],...
395 'Marker','o','Marker','o','MarkerFaceColor',color,...
396 'MarkerEdgeColor',color,'MarkerSize',8);
397 title(['Ensemble Average  $r(t)_{\perp}$ , ' cat],FontSize,20,...
398 'Interpreter','Tex');
399 set(gca,FontSize,14);
400 xlabel('time (s)',FontSize,16);
401 ylabel('< $r(t)_{\perp}$ > \pm SE (nm)',FontSize,16);
402 axis([0 t(end) ymin ymax]);
403
404 % Displacement plots no error bars
405 ymin = zeros(1,2);
406 ymax = ymin;
407 ymin(1) = min(mpara);
408 ymin(2) = min(mperp);
409 ymin = min(ymin);
410 ymax(1) = max(mpara);
411 ymax(2) = max(mperp);
412 ymax = max(ymax);
413
414 fig_handle_rmeans = figure;
415 subplot(1,2,1);
416 plot(t,mpara,'LineStyle','none','Marker','o','MarkerFaceColor',color,...
417 'MarkerEdgeColor',color,'MarkerSize',8);
418 title(['Ensemble Average  $r(t)_{\perp}$ , ' cat],FontSize,20,...
419 'Interpreter','Tex');
420 set(gca,FontSize,14);
421 xlabel('time (s)',FontSize,16);
422 ylabel('< $r(t)_{\perp}$ > (nm)',FontSize,16);
423 axis([0 t(end) ymin ymax]);
424
425 subplot(1,2,2);
426 plot(t,mperp,'LineStyle','none','Marker','o','Marker','o',...
427 'MarkerFaceColor',color,'MarkerEdgeColor',color,'MarkerSize',8);
428 title(['Ensemble Average  $r(t)_{\perp}$ , ' cat],FontSize,20,...
429 'Interpreter','Tex');

```

```

430 set(gca,'FontSize',14);
431 xlabel('time (s)','FontSize',16);
432 ylabel('<r(t)_perp> (nm)','FontSize',16);
433 axis([0 t(end) ymin ymax]);
434
435 % Force plots standard deviation bars
436 ymin = zeros(1,2);
437 ymax = ymin;
438 ymin(1) = min(mpara_F-sdpara_F);
439 ymin(2) = min(mperp_F-sdperp_F);
440 ymin = min(ymin);
441 ymax(1) = max(mpara_F+sdpara_F);
442 ymax(2) = max(mperp_F+sdperp_F);
443 ymax = max(ymax);
444
445 fig_handle_fsdbars = figure;
446 subplot(1,2,1);
447 errorbar(t,mpara_F,sdpara_F,'LineStyle','none','Color',[0.83,0.82,0.78],...
448     'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,...
449     'MarkerSize',8);
450 title(['Ensemble Average f(t)_||, ' cat'],'FontSize',20,...
451     'Interpreter','Tex');
452 set(gca,'FontSize',14);
453 xlabel('time (s)','FontSize',16);
454 ylabel('<f(t)_||> \pm SD (pN)','FontSize',16);
455 axis([0 t(end) ymin ymax]);
456
457 subplot(1,2,2);
458 errorbar(t,mperp_F,sdperp_F,'LineStyle','none','Color',[0.83,0.82,0.78],...
459     'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,...
460     'MarkerSize',8);
461 title(['Ensemble Average f(t)_perp, ' cat'],'FontSize',20,...
462     'Interpreter','Tex');
463 set(gca,'FontSize',14);
464 xlabel('time (s)','FontSize',16);
465 ylabel('<f(t)_perp> \pm SD (pN)','FontSize',16);
466 axis([0 t(end) ymin ymax]);
467
468 % Force plots standard error bars
469 ymin = zeros(1,2);
470 ymax = ymin;
471 ymin(1) = min(mpara_F-separa_F);
472 ymin(2) = min(mperp_F-seperp_F);
473 ymin = min(ymin);
474 ymax(1) = max(mpara_F+separa_F);
475 ymax(2) = max(mperp_F+seperp_F);
476 ymax = max(ymax);
477
478 fig_handle_fsebars = figure;
479 subplot(1,2,1);
480 errorbar(t,mpara_F,separa_F,'LineStyle','none','Color',[0.83,0.82,0.78],...
481     'Marker','o','MarkerFaceColor',color,'MarkerEdgeColor',color,...
482     'MarkerSize',8);
483 title(['Ensemble Average f(t)_||, ' cat'],'FontSize',20,...
484     'Interpreter','Tex');
485 set(gca,'FontSize',14);
486 xlabel('time (s)','FontSize',16);
487 ylabel('<f(t)_||> \pm SE (pN)','FontSize',16);
488 axis([0 t(end) ymin ymax]);
489
490 subplot(1,2,2);
491 errorbar(t,mperp_F,seperp_F,'LineStyle','none','Color',[0.83,0.82,0.78],...
492     'Marker','o','Marker','o','MarkerFaceColor',color,...
493     'MarkerEdgeColor',color,'MarkerSize',8);
494 title(['Ensemble Average f(t)_perp, ' cat'],'FontSize',20,...
495     'Interpreter','Tex');
496 set(gca,'FontSize',14);
497 xlabel('time (s)','FontSize',16);

```

```

498 ylabel('<f(t)_\perp> \pm SE (pN)',FontSize,16);
499 axis([0 t(end) ymin ymax]);
500
501 % Force plots no errorbars
502 ymin = zeros(1,2);
503 ymax = ymin;
504 ymin(1) = min(mpara_F);
505 ymin(2) = min(mperp_F);
506 ymin = min(ymin);
507 ymax(1) = max(mpara_F);
508 ymax(2) = max(mperp_F);
509 ymax = max(ymax);
510
511 fig_handle_fmeans = figure;
512 subplot(1,2,1);
513 plot(t,mpara_F,'LineStyle','none','Marker','o','MarkerFaceColor',color,...
514      'MarkerEdgeColor',color,'MarkerSize',8);
515 title(['Ensemble Average f(t)_\_|', ' cat'],FontSize,20,...
516      'Interpreter','Tex');
517 set(gca,'FontSize',14);
518 xlabel('time (s)',FontSize,16);
519 ylabel('<f(t)_\_|> (pN)',FontSize,16);
520 axis([0 t(end) ymin ymax]);
521
522 subplot(1,2,2);
523 plot(t,mperp_F,'LineStyle','none','Marker','o','Marker','o',...
524      'MarkerFaceColor',color,'MarkerEdgeColor',color,'MarkerSize',8);
525 title(['Ensemble Average f(t)_\perp', ' cat'],FontSize,20,...
526      'Interpreter','Tex');
527 set(gca,'FontSize',14);
528 xlabel('time (s)',FontSize,16);
529 ylabel('<f(t)_\perp> (pN)',FontSize,16);
530 axis([0 t(end) ymin ymax]);
531
532 %Save plots
533 fprintf(1,'\tSaving variable(s) and figure(s)\n');
534 fprintf(fid,'\tSaving variable(s) and figure(s)\n');
535
536 saveas(fig_handle_r,[save_path '\rvst_' cat '.fig']);
537 saveas(fig_handle_rsdbars,[save_path '\rvst_meanSDerrorbars_' cat '.fig']);
538 saveas(fig_handle_rsebars,[save_path '\rvst_meanSEerrorbars_' cat '.fig']);
539 saveas(fig_handle_rmeans,[save_path '\rvst_mean_' cat '.fig']);
540 saveas(fig_handle_f,[save_path '\fvst_' cat '.fig']);
541 saveas(fig_handle_fsdbars,[save_path '\fvst_mean_SDerrorbars_' cat ...
542      '.fig']);
543 saveas(fig_handle_fsebars,[save_path '\fvst_mean_SEerrorbars_' cat ...
544      '.fig']);
545 saveas(fig_handle_fmeans,[save_path '\fvst_mean_' cat '.fig']);
546
547 % Update log file that function is completed:
548 fprintf(1,'%s completed\n',func_name);
549 fprintf(fid,'%s completed\n',func_name);
550
551 end

```

PlotMetrics VsRadialDist_v7.m

```

1 % Steven J. Henry
2 % 09/25/2014
49 % *****
50 % PURPOSE: To plot time at Fmax, Fmax, and <F(t>thres)> vs radial distance
51 % of post from centroid.
52 %
53 % ASSUMPTIONS:
54 % n/a
55 %
56 % INPUT:
57 % exp_date = 8 digit number date of experiment (yyyymmdd)

```

```

58 % exp_donor = donor ID string ('DXX')
59 % exp_cond = string describing experimental condition (e.g. 'Control')
60 % fovn = field of view number
61 % sbead_path = path to sbead.mat files
62 % centroid = geometric centroid of cell
63 % microntopix = microntopix conversion in pixels/um
64 % transition_thres = time (s) after Fmax to consider system @ steady state
65 % cat = string denoting category being analyzed
66 % color = plotting color a string or vector
67 % save_path = string pointing to save location
68 % fid = file ID of log file to which progress is recorded
69 % Fmax = radial maximum force observed for each post
70 % Fss = radial mean steady state force observed for each post
71 % t = time vector
72 % mpara_F = ensemble average of radial deflections * kspring (pN)
73 %
74 % OUTPUT:
75 % radialmetrics = array of metrics as a function of radial distance of post
76 %   from cell centroid
77 %
78 % DRIVER/FUNCTION MAP:
79 % n/a (calls no subroutines)
80 %*****
81
82 function [radialmetrics] = PlotMetricsVsRadialDist_v7(...
83     exp_date,exp_donor,exp_cond,fovn,...
84     sbead_path,centroid,microntopix,...
85     transition_thres,cat,color,save_path,fid,...
86     Fmax,Fss,t,mpara_F)
87
88 % Get function name:
89 func_name = mfilename;
90
91 % Update log file that function is running:
92 fprintf(1,'\n%s running ...\n',func_name);
93 fprintf(fid,'\n%s running ...\n',func_name);
94
95 num_posts = size(Fmax,1);
96
97 % Array to hold relevant metrics:
98 % col1 = ID
99 % col2 = x coordinate of resting post in lab reference frame (um)
100 % col3 = y coordinate of resting post in lab reference frame (um)
101 % col4 = radial distance from resting post position to cell centroid (um)
102 % col5 = Fmax_para (pN)
103 % col6 = <F(t>transition_time)_para> (pN)
104 % col7 = std of F(tau>transition_time)_para (pN)
105 r = NaN(num_posts,7);
106 xc = centroid(1)/microntopix;
107 yc = centroid(2)/microntopix;
108
109 for i = 1:num_posts
110
111     ID = Fmax(i,1);
112
113     if(Fmax(i,1)~=Fss(i,1))
114         fprintf(1,'\t\nWARNING: Post ID of row %s in Fmax = %s but in Fss =
115             %s\n',num2str(i),num2str(Fmax(i,1)),num2str(Fss(i,1)));
116         fprintf(fid,'\t\nWARNING: Post ID of row %s in Fmax = %s but in Fss =
117             %s\n',num2str(i),num2str(Fmax(i,1)),num2str(Fss(i,1)));
118     end
119
120 % Load sbead.mat file for post# "ID" and compute the radial distance.
121 % This is the distance from the cell centroid to the resting lattice
122 % position of the post:
123 load([sbead_path filesep 'sbead_' num2str(ID) '.mat'],'bsec');
124 xpost = bsec(1,1); % um
125 ypost = bsec(1,2); % um

```

```

124 r(i,1) = ID;
125 r(i,2) = xpost; %um
126 r(i,3) = ypost; %um
127 r(i,4) = sqrt((xpost-xc)^2+(ypost-yc)^2); %um
128 r(i,5) = Fmax(i,2); % pN
129 r(i,6) = Fss(i,2); % pN
130 r(i,7) = Fss(i,3); % pN
131
132 end
133
134 % Report per post mean:
135 keep_ind = ~isnan(r(:,5));
136 ppm_Fmax = mean(r(keep_ind,5));
137 ppsd_Fmax = std(r(keep_ind,5));
138 ppc_Fmax = length(r(keep_ind,5));
139 fprintf(1,'\tPer post <Fmax> +/- SD (pN) = %s +/- %s, (n = %s)\n',...
140 num2str(ppm_Fmax),num2str(ppsd_Fmax),num2str(ppc_Fmax));
141 fprintf(fid,'\tPer post <Fmax> +/- SD (pN) = %s +/- %s, (n = %s)\n',...
142 num2str(ppm_Fmax),num2str(ppsd_Fmax),num2str(ppc_Fmax));
143
144 keep_ind = ~isnan(r(:,6));
145 ppm_Fss = mean(r(keep_ind,6));
146 ppsd_Fss = std(r(keep_ind,6));
147 ppc_Fss = length(r(keep_ind,6));
148 fprintf(1,'\tPer post <F(tau>%s)> +/- SD (pN) = %s +/- %s, (n = %s)\n',...
149 num2str(transition_thres),num2str(ppm_Fss),num2str(ppsd_Fss),...
150 num2str(ppc_Fss));
151 fprintf(fid,...
152 '\tPer post <F(tau>%s)> +/- SD (pN) = %s +/- %s, (n = %s)\n',...
153 num2str(transition_thres),num2str(ppm_Fss),num2str(ppsd_Fss),...
154 num2str(ppc_Fss));
155
156 % Plot metrics vs. radial distance without text overlay:
157 fig_handle_radial = figure;
158 % Fmax vs. R
159 subplot(1,2,1)
160 plot(r(:,4),r(:,5),'LineStyle','none','Color',color,'Marker','o',...
161 'MarkerFaceColor',color,'MarkerEdgeColor','k','MarkerSize',6);
162 hold all;
163 tit_str = ['Fmax vs. R,' cat];
164 title(tit_str,'FontSize',16,'Interpreter','Tex');
165 set(gca,'FontSize',12);
166 xlabel('Radial Distance (\mum)','FontSize',14);
167 ylabel('Fmax (pN)','FontSize',14);
168 xlim([0,max(r(:,4))]);
169
170 % <F(t>transition_thres)> vs. R
171 subplot(1,2,2)
172 errorbar(r(:,4),r(:,6),r(:,7),'LineStyle','none',...
173 'Color',[0.83,0.82,0.78],'Marker','o','MarkerFaceColor',color,...
174 'MarkerEdgeColor','k','MarkerSize',6);
175 hold all;
176 tit_str = ['<F(tau>' num2str(transition_thres) '> vs. R,' cat];
177 title(tit_str,'FontSize',16,'Interpreter','Tex');
178 set(gca,'FontSize',12);
179 xlabel('Radial Distance (\mum)','FontSize',14);
180 ylabel(['<F(tau>' num2str(transition_thres) '> (pN)'],'FontSize',14);
181 xlim([0,max(r(:,4))]);
182
183 % Plot metrics vs. radial distance with text overlay:
184 fig_handle_radial2 = figure;
185 % Fmax vs. R
186 subplot(1,2,1)
187 plot(r(:,4),r(:,5),'LineStyle','none','Color',color,'Marker','o',...
188 'MarkerFaceColor',color,'MarkerEdgeColor','k','MarkerSize',6);
189 hold all;
190 text(r(:,4),r(:,5),num2str(r(:,1)),'Color','k');
191 tit_str = ['Fmax vs. R,' cat];

```

```

192 title(tit_str,'FontSize',16,'Interpreter','Tex');
193 set(gca,'FontSize',12);
194 xlabel('Radial Distance (\mum)','FontSize',14);
195 ylabel('Fmax (pN)','FontSize',14);
196 xlim([0,max(r(:,4))]);
197
198 % <F(t>transition_thres)> vs. R
199 subplot(1,2,2)
200 errorbar(r(:,4),r(:,6),r(:,7),'LineStyle','none',...
201         'Color',[0.83,0.82,0.78],'Marker','o','MarkerFaceColor',color,...
202         'MarkerEdgeColor','k','MarkerSize',6);
203 hold all;
204 text(r(:,4),r(:,6),num2str(r(:,1)),'Color','k');
205 tit_str = ['<F(tau>' num2str(transition_thres) 's)> vs. R, ' cat];
206 title(tit_str,'FontSize',16,'Interpreter','Tex');hold all;
207 set(gca,'FontSize',12);
208 xlabel('Radial Distance (\mum)','FontSize',14);
209 ylabel(['<F(tau>' num2str(transition_thres) ')> (pN)'],'FontSize',14);
210 xlim([0,max(r(:,4))]);
211
212 % Plot heatmaps. Becasue MATLAB does not allow you to associate different
213 % colormaps with different axes within the same figure the various metrics
214 % are divided among different figures unlike the previous radial plots
215 % which were all subplots within a single figure.
216
217 % Find spatial plotting limits:
218 xmin = min(floor(r(:,2)-xc))-1;
219 xmax = max(ceil(r(:,2)-xc))+1;
220 ymin = min(floor(r(:,3)-yc))-1;
221 ymax = max(ceil(r(:,3)-yc))+1;
222 if xmax>ymin
223     lmax = xmax;
224 else
225     lmax = ymax;
226 end
227 if xmin<ymin
228     lmin = xmin;
229 else
230     lmin = ymin;
231 end
232
233 % Fmax
234 % Only generate the figure if there are at least two data points:
235 num_data = sum(~isnan(r(:,5)));
236 if num_data > 1
237
238     fig_handle_heat_Fmax = figure;
239     colormap('jet');
240
241     subplot(1,2,1);
242     scatter(r(:,2)-xc,r(:,3)-yc,75,r(:,5),'fill');
243     hold all;
244     tit_str = ['Fmax, ' cat];
245     title(tit_str,'FontSize',16,'Interpreter','Tex');
246     set(gca,'FontSize',12);
247     xlabel('x-x_c (\mum)','FontSize',14);
248     ylabel('y-y_c (\mum)','FontSize',14);
249     axis([lmin lmax lmin lmax]);
250     axis('square');
251     axis('ij');
252     caxis([min(r(:,5)) max(r(:,5))]);
253     c = colorbar('EastOutside');
254     set(c,'FontSize',12);
255     ylabel(c,'pN','FontSize',12);
256
257     subplot(1,2,2);
258     scatter(r(:,2)-xc,r(:,3)-yc,75,r(:,5),'fill');
259     hold all;

```

```

260 text(r(:,2)-xc,r(:,3)-yc,num2str(r(:,1)), 'Color', 'k');
261 tit_str = ['IDs, Fmax, ' cat];
262 title(tit_str, 'FontSize', 16, 'Interpreter', 'Tex');
263 set(gca, 'FontSize', 12);
264 xlabel('x-x_c (\mum)', 'FontSize', 14);
265 ylabel('y-y_c (\mum)', 'FontSize', 14);
266 axis([lmin lmax lmin lmax]);
267 axis('square');
268 axis('ij');
269 caxis([min(r(:,5)) max(r(:,5))]);
270 c = colorbar('EastOutside');
271 set(c, 'FontSize', 12);
272 ylabel(c, 'pN');
273 end
274
275 % <F(t>transition_thres)
276 % Only generate the figure if there are at least two data points:
277 num_data = sum(~isnan(r(:,6)));
278 if num_data > 1
279
280 fig_handle_heat_meanFss = figure;
281 colormap(flipud(colormap('jet')));
282
283 subplot(1,2,1);
284 scatter(r(:,2)-xc,r(:,3)-yc,75,r(:,6), 'fill');
285 hold all;
286 tit_str = ['<F(\tau>' num2str(transition_thres) 's)>, ' cat];
287 title(tit_str, 'FontSize', 16, 'Interpreter', 'Tex');
288 set(gca, 'FontSize', 12);
289 xlabel('x-x_c (\mum)', 'FontSize', 14);
290 ylabel('y-y_c (\mum)', 'FontSize', 14);
291 axis([lmin lmax lmin lmax]);
292 axis('square');
293 axis('ij');
294 caxis([min(r(:,6)) max(r(:,6))]);
295 c = colorbar('EastOutside');
296 set(c, 'FontSize', 12);
297 ylabel(c, 'pN', 'FontSize', 12);
298
299 subplot(1,2,2);
300 scatter(r(:,2)-xc,r(:,3)-yc,75,r(:,6), 'fill');
301 hold all;
302 text(r(:,2)-xc,r(:,3)-yc,num2str(r(:,1)), 'Color', 'k');
303 tit_str = ['IDs, <F(\tau>' num2str(transition_thres) 's)>, ' cat];
304 title(tit_str, 'FontSize', 16, 'Interpreter', 'Tex');
305 set(gca, 'FontSize', 12);
306 xlabel('x-x_c (\mum)', 'FontSize', 14);
307 ylabel('y-y_c (\mum)', 'FontSize', 14);
308 axis([lmin lmax lmin lmax]);
309 axis('square');
310 axis('ij');
311 caxis([min(r(:,6)) max(r(:,6))]);
312 c = colorbar('EastOutside');
313 set(c, 'FontSize', 12);
314 ylabel(c, 'pN', 'FontSize', 12);
315 end
316
317 % Compute FWHM for this category's ensemble average curve:
318 width = FWHM_NoDisplay_PosOnly(t,mpara_F); % s or NaN
319
320 fprintf(1, '\tEnsemble FWHM (s) = %s , (n = %s)\n', ...
321 num2str(width), num2str(ppc_Fmax));
322 fprintf(fid, '\tEnsemble FWHM (s) = %s , (n = %s)\n', ...
323 num2str(width), num2str(ppc_Fmax));
324
325 % Record values in an excel spreadsheet for easier compiling across
326 % multiple experiments:
327 fprintf(1, '\n\tSaving mean metrics in Excel spreadsheet..\n');

```



```

328 fprintf(fid,'\n\tSaving mean metrics in Excel spreadsheet..\n');
329
330 header_info = {'Date','Donor','Condition','fovn'...
331   [cat ' <Fmax> (pN)],[cat ' SD'],[cat ' n'],...
332   [cat ' <F(t> num2str(transition_thres) '> (pN)],[cat ' SD'],...
333   [cat ' n],[cat ' <FWHM> (s)],[cat ' n']};
334 xls_name = [save_path filesep 'MeanMetrics.xlsx'];
335 xlswrite(xls_name,header_info,cat,'A1');
336 data_to_log = (exp_date,exp_donor,exp_cond,fovn,...
337   ppm_Fmax, ppsd_Fmax, ppc_Fmax...
338   ppm_Fss, ppsd_Fss, ppc_Fss...
339   width, ppc_Fmax);
340 xlswrite(xls_name,data_to_log,cat,'A2');
341
342 % Give 'r' a more descriptive variable name when saving:
343 radialmetrics = r;
344
345 %Save plots
346 fprintf(1,'\tSaving variable(s) and figure(s)\n');
347 fprintf(fid,'\tSaving variable(s) and figure(s)\n');
348
349 save([save_path filesep 'RadialMetrics_' cat '.mat'],'radialmetrics');
350 saveas(fig_handle_radial,[save_path '\RadialPlots_' cat '.fig']);
351 saveas(fig_handle_radial2,[save_path '\RadialPlots_IDoverlay_' cat ...
352   '.fig']);
353 if exist('fig_handle_heat_Fmax','var')
354 saveas(fig_handle_heat_Fmax,[save_path '\HeatMaps_Fmax_' cat '.fig']);
355 end
356 if exist('fig_handle_heat_meanFss','var')
357 saveas(fig_handle_heat_meanFss,[save_path '\HeatMaps_meanFss_' cat ...
358   '.fig']);
359 end
360
361 % Update log file that function is completed:
362 fprintf(1,'%s completed\n',func_name);
363 fprintf(fid,'%s completed\n',func_name);
364
365 end

```

Plot_fvst_Strips_v1.m

```

1 % Steven J. Henry
2 % 02/17/2015
7 %*****
8 % PURPOSE: To plot force vs time for each post vertically shifting each
9 % force trajectory so they are individually distinguishable.
10 %
11 % ASSUMPTIONS:
12 % n/a
13 %
14 % INPUT:
15 % rsbead_path = path to rsbead.mat files for category 'cat'
16 % radialmetrics = array of metrics as a function of radial distance of post
17 % from cell centroid
18 % kspring = post spring constant (pN/nm)
19 % time_int = time interval between frames (sec)
20 % cat = string denoting category being analyzed
21 % color = string or vector to specify plotting color
22 % save_path = path to location where analysis is being saved
23 % fid = logfile handle
24 %
25 % OUTPUT:
26 % n/a
27 %
28 % DRIVER/FUNCTION MAP:
29 % n/a (calls no subroutines)
30 %*****
31 function [] = Plot_fvst_Strips_v1(rsbead_path,radial_metrics,...

```

```

32     kspring,time_int,cat,color,save_path,fid)
33
34 % Get function name:
35 func_name = mfilename;
36
37 % Update log file that function is running:
38 fprintf(1,'\n%s running ...\n',func_name);
39 fprintf(fid,'\n%s running ...\n',func_name);
40
41 [~,sort_ind] = sort(radial_metrics(:,4),'ascend');
42
43 ID = radial_metrics(sort_ind,1);
44
45 num_posts = length(ID);
46
47 fig_handle_strip = figure;
48 tick = 0;
49 step = 150;
50 ticks = NaN(num_posts,1);
51 labels = cell(num_posts,1);
52
53 for i = 1:num_posts
54
55     load([rsbead_path filesep 'rsbead_' num2str(ID(i)) '.mat'],'bsec');
56     rpara = bsec(:,1)*1000; % % nm
57     rperp = bsec(:,2)*1000; % nm
58     frames = bsec(:,3);
59     % Exclude row 1 which contains the distance to the resting lattice
60     % position from the resting lattice position (i.e. 0).
61     rpara(1) = [];
62     rperp(1) = [];
63     frames(1) = [];
64
65     t = frames*time_int;
66
67     ticks(i) = tick;
68
69     s1 = subplot(1,2,1);
70     plot(t,rpara*kspring+tick,'LineStyle','-','Color',color,...
71         'Marker','none');
72     hold all;
73
74     s2 = subplot(1,2,2);
75     plot(t,rperp*kspring+tick,'LineStyle','-','Color',color,...
76         'Marker','none');
77     hold all;
78
79     tick = tick+step;
80
81     labels(i) = ['post' num2str(ID(i))];
82
83 end
84
85 subplot(s1)
86 title(['f(t)_|| (pN), ' cat'],'FontSize',20,'Interpreter','Tex');
87 set(s1,'FontSize',14);
88 xlabel('time (s)','FontSize',16);
89 ylabel('Radial distance from centroid (\rmm)','FontSize',16);
90 set(s1,'YTick',ticks);
91 set(s1,'YTickLabel',num2str(radial_metrics(sort_ind,4)),'FontSize',8);
92 legend(labels);
93 legend('hide');
94
95 subplot(s2)
96 title(['f(t)_\perp (pN), ' cat'],'FontSize',20,'Interpreter','Tex');
97 set(s2,'FontSize',14);
98 xlabel('time (s)','FontSize',16);
99 ylabel('Radial distance from centroid (\rmm)','FontSize',16);

```

```

100 set(s2,'YTick',ticks);
101 set(s2,'YTickLabel',num2str(radial_metrics(sort_ind,4)), 'FontSize',8);
102 legend(labels);
103 legend('hide');
104
105 % Save figures:
106 fprintf(1,'\n\tSaving variable(s) and figure(s)\n');
107 fprintf(fid,'\n\tSaving variable(s) and figure(s)\n');
108 saveas(fig_handle_strip,[save_path filesep 'fvst_strip_' cat '.fig']);
109
110 % Update log file that function is completed:
111 fprintf(1,'%s completed\n',func_name);
112 fprintf(fid,'%s completed\n',func_name);
113
114 end

```

IndividualPostAutoCorrelation_v2.m

```

1 % Steven J. Henry
2 % 10/06/2014
14 %*****
15 % PURPOSE: This function performs autocorrelations of individual posts.
16 %
17 % ASSUMPTIONS:
18 % n/a
19 %
20 % INPUT:
21 % rsbead_path = path to rsbead.mat files for category 'cat'
22 % subsetIDs = list of post IDs belonging to this category 'cat'
23 % max_num_frames = maximum number of frames for this fovn
24 % time_int = time interval between frames (sec)
25 % cat = string denoting category being analyzed
26 % color = string or vector to specify plotting color
27 % save_path = path to location where analysis is being saved
28 % fid = logfile handle
29 %
30 % OUTPUT:
31 % n/a
32 %
33 % DRIVER/FUNCTION MAP:
34 % n/a (calls no subroutines)
35 %*****
36
37 function [] = IndividualPostAutocorrelation_v2(rsbead_path,subsetIDs,...
38     max_num_frames,time_int,cat,color,save_path,fid)
39
40 % Get function name:
41 func_name = mfilename;
42
43 % Update log file that function is running:
44 fprintf(1,'\n%s running ...\n',func_name);
45 fprintf(fid,'\n%s running ...\n',func_name);
46
47 % How many post files did user select?
48 num_posts = length(subsetIDs);
49
50 % Initialize plot that will hold all autocorrelation functions:
51 fig_handle_corr = figure;
52 subplot(1,2,1);
53 xlabel('\tau (s)', 'FontSize', 14, 'Interpreter', 'Tex');
54 ylabel('Normalized Autocorrelation', 'FontSize', 14, 'Interpreter', 'Tex');
55 title(['Individual r(\tau)_|_|, ' cat], 'FontSize', 16, 'Interpreter', 'Tex');
56 hold all;
57 subplot(1,2,2);
58 xlabel('\tau (s)', 'FontSize', 14, 'Interpreter', 'Tex');
59 ylabel('Normalized Autocorrelation', 'FontSize', 14, 'Interpreter', 'Tex');
60 title(['Individual r(\tau)_\perp, ' cat], 'FontSize', 16, ...
61     'Interpreter', 'Tex');

```

```

62 hold all;
63 labels = cell(num_posts,1);
64
65 max_length = 2*max_num_frames-1;
66 C_para = NaN(max_length,num_posts);
67 C_perp = C_para;
68
69 for i = 1:num_posts
70
71     load([rsbead_path filesep 'rsbead_' num2str(subsetIDs(i)) '.mat'],...
72         'bsec');
73
74     r_para = bsec(:,1); %#ok<NODEF> % um
75     r_perp = bsec(:,2); % um
76     % Exclude row 1 which contains the distance to the resting lattice
77     % position from the resting lattice position (i.e. 0).
78     r_para(1) = [];
79     r_perp(1) = [];
80
81     [c_para, lags_para] = xcorr(r_para,'coeff');
82     [c_perp, lags_perp] = xcorr(r_perp,'coeff');
83
84     % These fancy indices are to ensure that lag = 0 row is aligned for all
85     % autocorrelations to be averaged:
86     L = length(c_para);
87     half_pad = (max_length - L)/2;
88     C_para(half_pad+1:half_pad+L,i) = c_para;
89     C_perp(half_pad+1:half_pad+L,i) = c_perp;
90
91     figure(fig_handle_corr)
92     subplot(1,2,1)
93     plot(lags_para*time_int,c_para,'LineStyle','-','Color',color);
94     hold all;
95
96     subplot(1,2,2)
97     plot(lags_perp*time_int,c_perp,'LineStyle','-','Color',color);
98     hold all;
99
100    labels{i} = ['post' num2str(subsetIDs(i))];
101
102 end
103
104 figure(fig_handle_corr)
105 subplot(1,2,1);
106 legend(labels);
107 legend('hide');
108 subplot(1,2,2);
109 legend(labels);
110 legend('hide');
111
112 m_para = NaN(max_length,1);
113 sd_para = m_para;
114 m_perp = m_para;
115 sd_perp = m_para;
116
117 % Compute means of autocorrelation of all trajectories
118 for i = 1:max_length
119     nanid = isnan(C_para(i,:));
120     m_para(i) = mean(C_para(i,~nanid));
121     sd_para(i) = std(C_para(i,~nanid));
122     m_perp(i) = mean(C_perp(i,~nanid));
123     sd_perp(i) = std(C_perp(i,~nanid));
124 end
125
126 fig_handle_mean = figure;
127 m_lags = (-max_num_frames+1:max_num_frames-1)*time_int;
128 subplot(1,2,1);
129 plot(m_lags,m_para,'LineStyle','none','Color',color,'Marker','o',...

```

```

130     'MarkerFaceColor',color,'MarkerEdgeColor',color,'MarkerSize',6);
131 xlabel('\tau (s)','FontSize', 14,'Interpreter','Tex');
132 ylabel('Mean Normalized Autocorrelation','FontSize', 14,...
133     'Interpreter','Tex');
134 title(['Mean r(\tau)_|_', ' cat'],'FontSize',16,'Interpreter','Tex');
135
136 subplot(1,2,2);
137 plot(m_lags,m_perp,'LineStyle','none','Color',color,'Marker','o',...
138     'MarkerFaceColor',color,'MarkerEdgeColor',color,'MarkerSize',6);
139 xlabel('\tau (s)','FontSize', 14,'Interpreter','Tex');
140 ylabel('Mean Normalized Autocorrelation','FontSize', 14,...
141     'Interpreter','Tex');
142 title(['Mean r(\tau)_perp, ' cat'],'FontSize',16,'Interpreter','Tex');
143
144 fig_handle_mean_error = figure;
145 subplot(1,2,1);
146 errorbar(m_lags,m_para,sd_para,'LineStyle','none',...
147     'Color',[0.83,0.82,0.78],'Marker','o','MarkerFaceColor',color,...
148     'MarkerEdgeColor',color,'MarkerSize',6);
149 xlabel('\tau (s)','FontSize', 14,'Interpreter','Tex');
150 ylabel('Mean Normalized Autocorrelation \pm SD','FontSize', 14,...
151     'Interpreter','Tex');
152 title(['Mean r(t)_|_', ' cat'],'FontSize',16,'Interpreter','Tex');
153
154 subplot(1,2,2);
155 errorbar(m_lags,m_perp,sd_perp,'LineStyle','none',...
156     'Color',[0.83,0.82,0.78],'Marker','o','MarkerFaceColor',color,...
157     'MarkerEdgeColor',color,'MarkerSize',6);
158 xlabel('\tau (s)','FontSize', 14,'Interpreter','Tex');
159 ylabel('Mean Normalized Autocorrelation \pm SD','FontSize', 14,...
160     'Interpreter','Tex');
161 title(['Mean r(t)_perp, ' cat'],'FontSize',16,'Interpreter','Tex');
162
163 % Find tau (s) at which mean normalized autocorrelation = 1/e (~0.3679)
164 mid = ((length(m_para)-1)/2)+1;
165 m_para_trunc = m_para(mid:end);
166 m_perp_trunc = m_perp(mid:end);
167 m_lags_trunc = m_lags(mid:end);
168 [~,row_para] = min(abs(m_para_trunc-1/exp(1)));
169 characteristic_tau_para = m_lags_trunc(row_para);
170 [~,row_perp] = min(abs(m_perp_trunc-1/exp(1)));
171 characteristic_tau_perp = m_lags_trunc(row_perp);
172
173 % Record values in an excel spreadsheet for easier compiling across
174 % multiple experiments:
175 fprintf(1,'\n\tSaving mean metrics in Excel spreadsheet..\n');
176 fprintf(fid,'\n\tSaving mean metrics in Excel spreadsheet..\n');
177
178 header_info = {[cat ' tau@1/e para (s)],[cat ' tau@1/e perp (s)']};
179 xls_name = [save_path filesep 'MeanMetrics.xlsx'];
180 xlswrite(xls_name,header_info,cat,'Q1');
181 data_to_log = {characteristic_tau_para,characteristic_tau_perp};
182 xlswrite(xls_name,data_to_log,cat,'Q2');
183
184 % Save figures:
185 fprintf(1,'\n\tSaving variable(s) and figure(s)\n');
186 fprintf(fid,'\n\tSaving variable(s) and figure(s)\n');
187 save([save_path filesep 'Autocorrelation_Variables_' cat '.mat'],...
188     'C_para','C_perp','m_para','sd_para','m_perp','sd_perp');
189 saveas(fig_handle_corr,[save_path filesep ...
190     'Autocorrelation_AllTrajectories_' cat '.fig']);
191 saveas(fig_handle_mean,[save_path filesep ...
192     'Autocorrelation_Mean_' cat '.fig']);
193 saveas(fig_handle_mean_error,[save_path filesep ...
194     'Autocorrelation_Mean_Errorbars_' cat '.fig']);
195
196 % Update log file that function is completed:
197 fprintf(1, '%s completed\n',func_name);

```

```

198 fprintf(fid,'%s completed\n',func_name);
199
200 end

```

Tidy_Up_v2.m

```

1  % Steven J. Henry
2  % 05/20/2014
12 %*****
13 % PURPOSE:
14 % The following function places all files with .fig, .mat, and .txt
15 % extensions into folders called "figs", "mats", and "txts" respectively.
16 % Only the master log file is left outside these folders for easy
17 % navigation.
18 %
19 % ASSUMPTIONS:
20 % n/a
21 %
22 % INPUT:
23 % logfile = string containing title of master log file
24 %
25 % OUTPUT:
26 % n/a
27 %
28 % DRIVER/FUNCTION MAP:
29 % n/a (calls no subroutines)
30 %*****
31
32 function [] = Tidy_Up_v2(logfile)
33
34 dir_info = dir(pwd);
35 num_entities = length(dir_info);
36 num_files = 0;
37
38 for i = 1:num_entities
39     if dir_info(i).isdir == 0
40         num_files = num_files+1;
41     end
42 end
43
44 extensions = cell(num_files,1);
45 k = 1;
46 for i = 1:num_entities
47     if dir_info(i).isdir == 0
48         extensions{k,1} = dir_info(i).name(end-3:end);
49         k = k+1;
50     end
51 end
52
53 unique_extensions = unique(extensions);
54 num_unique = size(unique_extensions,1);
55
56 for i = 1:num_unique
57
58     % 4 character extension (e.g. .txt or .mat or .fig)
59     ext = unique_extensions{i};
60
61     % Make a folder using extension name without period (e.g. txt or mat or
62     % fig)
63     mkdir(ext(end-2:end));
64
65     % Define path to this new folder
66     ext_path = [pwd filesep ext(end-2:end)];
67
68     % Move all applicable files
69     movefile(['*' ext],ext_path);
70
71     pause(3);

```

```
72
73 % If you're moving .txt files undo logfile move:
74 if strcmp(ext,'.txt')==1
75     % Go into txt folder
76     cd(ext_path);
77     % Move logfile up one level
78     movefile(logfile,'..');
79     % Reset directory up one level
80     cd('..');
81 end
82
83 end
84
85 end
```

References

1. Pelletier, V., N. Gal, P. Fournier, and M. L. Kilfoil. 2009. Microrheology of Microtubule Solutions and Actin-Microtubule Composite Networks. *Physical Review Letters* 102:188303.
2. Crocker, J. C., and D. G. Grier. 1996. Methods of Digital Video Microscopy for Colloidal Studies. *Journal of Colloid and Interface Science* 179:298-310.
3. Kilfoil, M. L. 2014. *Biological Physics Kilfoil Lab*. Web Page. 04 November 2014. <http://people.umass.edu/kilfoil/downloads.html>.