Universidad Carlos III de Madrid
www.uc3m.es

TRABAJO FIN DE GRADO

Título: Integración de datos de imagen molecular y expresión génica

Autor: Carlos Porras Rodríguez

Titulación: Grado en Ingeniería Biomédica

Director: Javier Pascau González-Garzón

Co-director: José María Mateos Pérez

Fecha: 4/7/2014

Index

Introduction

Gene expression databases

What is a gene expression database?

The information that can be extracted from aspects like the location and the expression of certain genes is of great interest in many fields. Using this kind of information we can unravel the function of the genes in different conditions and diseases, or in embryonic development. Gene expression patterns in certain organs and developmental stages have been obtained from several species using microarrays or other kinds of sequencing studies. Data can be obtained by *in situ* visualization of the expression levels of different mRNAs, proteins or transgenic reporters.

Nevertheless, obtaining the information from microarray procedures only yields experimental data. Enormous quantities of these data are available in the literature, but they are not easily accessible, and it is highly problematic to retrieve. In order to put some order in this chaos of knowledge from very different sources, gene expression databases are one possible solution. A gene expression database can be defined as "a database containing anatomically annotated *in situ* gene expression information"(De Boer, Ruijter, Voorbraak, & Moorman, 2009). In this way, a gene expression database links gene expression data with predefined anatomical structures. In the past years there have been several initiatives which try to provide a solution to this problem, their own expression database.

Existing gene expression databases

Table 1 (below) shows several of the different databases which are currently available. Most of them contain the data for gene expression of mice, which are mammals just like Homo Sapiens, and which can be used for expression studies to determine the role of genes that function in different molecular pathways. In this way, a correlation between the expression of genes in mice and in humans can be found in many cases. As it can be observed in table 2 (also below), some of them use wild type mice whilst others use transgenic mice. The information which can be extracted from all of them, as it is also observable in the table, is not the same: they include the gene expression from different Theiler Stages (Stages of embryonic development in mice) where TS1 is the one-cell zygote and TS28 includes the information for adult mice. Also, the information is not organized in the same way: some of them use a vocabulary annotation, others use an ontology and EMAGE uses a spatial framework. The difference between the vocabulary annotation and the ontological annotation is that in ontological annotations there is a clearly established hierarchy among the structures, whilst in vocabulary annotation, the structures are just a list, without any kind of hierarchy. Among the most important gene expression databases we can find for mice are EMAGE, GXD, ArrayExpress, GENSAT, Allen Brain Atlas, EUREXPRESS and BioGPS. In the following paragraphs, a brief description of all these databases is going to be presenting, highlighting the most interesting features of each one.

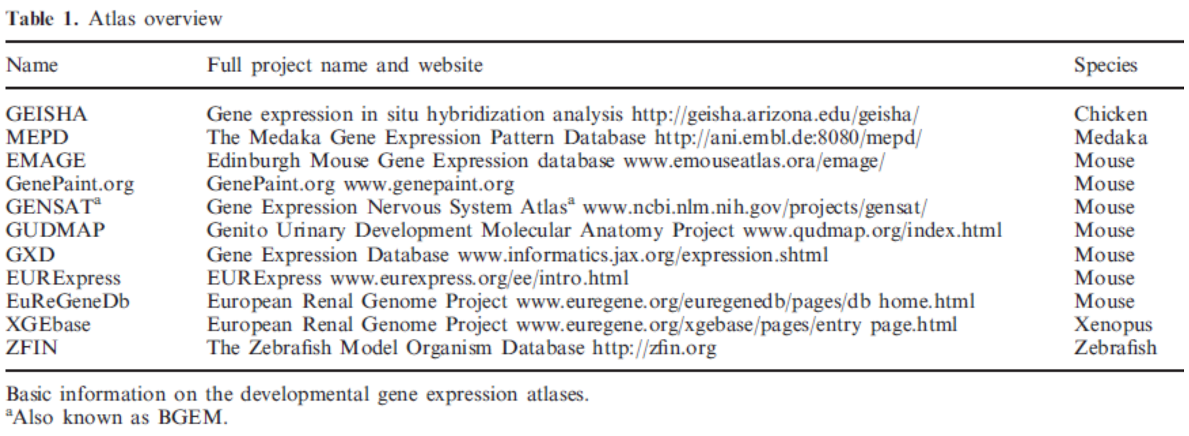


Table 1. Atlas overview

Name	Full project name and website	Species
GEISHA	Gene expression in situ hybridization analysis http://geisha.arizona.edu/geisha/	Chicken
MEPD	The Medaka Gene Expression Pattern Database http://ani.embl.de:8080/mepd/	Medaka
EMAGE	Edinburgh Mouse Gene Expression database www.emouseatlas.ora/emage/	Mouse
GenePaint.org	GenePaint.org www.genepaint.org	Mouse
GENSAT[a]	Gene Expression Nervous System Atlas[a] www.ncbi.nlm.nih.gov/projects/gensat/	Mouse
GUDMAP	Genito Urinary Development Molecular Anatomy Project www.qudmap.org/index.html	Mouse
GXD	Gene Expression Database www.informatics.jax.org/expression.shtml	Mouse
EURExpress	EURExpress www.eurexpress.org/ee/intro.html	Mouse
EuReGeneDb	European Renal Genome Project www.euregene.org/euregenedb/pages/db home.html	Mouse
XGEbase	European Renal Genome Project www.euregene.org/xgebase/pages/entry page.html	Xenopus
ZFIN	The Zebrafish Model Organism Database http://zfin.org	Zebrafish

Basic information on the developmental gene expression atlases.
[a]Also known as BGEM.

Table 1: Gene expression databases available, websites and species. Obtained from (De Boer et al., 2009)

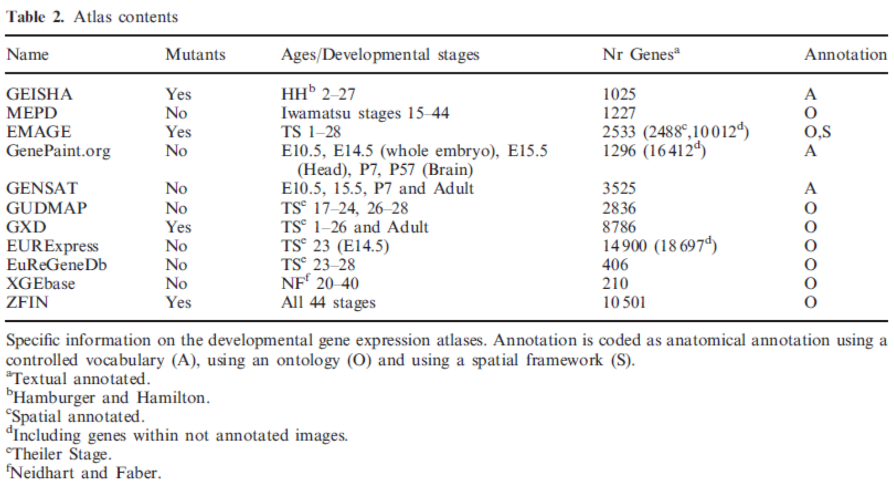


Table 2. Atlas contents

Name	Mutants	Ages/Developmental stages	Nr Genes[a]	Annotation
GEISHA	Yes	HH[b] 2–27	1025	A
MEPD	No	Iwamatsu stages 15–44	1227	O
EMAGE	Yes	TS 1–28	2533 (2488[c],10 012[d])	O,S
GenePaint.org	No	E10.5, E14.5 (whole embryo), E15.5 (Head), P7, P57 (Brain)	1296 (16 412[d])	A
GENSAT	No	E10.5, 15.5, P7 and Adult	3525	A
GUDMAP	No	TS[e] 17–24, 26–28	2836	O
GXD	Yes	TS[e] 1–26 and Adult	8786	O
EURExpress	No	TS[e] 23 (E14.5)	14 900 (18 697[d])	O
EuReGeneDb	No	TS[e] 23–28	406	O
XGEbase	No	NF[f] 20–40	210	O
ZFIN	Yes	All 44 stages	10 501	O

Specific information on the developmental gene expression atlases. Annotation is coded as anatomical annotation using a controlled vocabulary (A), using an ontology (O) and using a spatial framework (S).
[a]Textual annotated.
[b]Hamburger and Hamilton.
[c]Spatial annotated.
[d]Including genes within not annotated images.
[e]Theiler Stage.
[f]Neidhart and Faber.

Table 2: Details available in each of the atlases. Obtained from (De Boer et al., 2009)

EMAGE is a project from Heriott-Watt University in Edinburgh (Scotland, UK). It is a gene expression database which has been developed together with a mouse anatomical atlas called EMAP (Edinburgh Mouse Atlas Project). The two tools used together provide a result similar to the one expected from this project; nevertheless, it can only be used with Theiler Stages from 7 to 23, and only includes the fraction of genetic information which is spatially annotated and provided by EMAGE. The database includes more than 10000 genes, but only 2500 are spatially annotated (Richardson et al., 2009).

EUREXPRESS is an integrated project funded by the European Union, proposing an acquisition of expression patterns based on *in situ* hybridization of the whole transcriptome. The project comprises expression data of around 20000 genes which can be described in detail in developing mouse (Diez-Roux et al., 2011).

ArrayExpress is a public repository provided by the European Bioinformatics Institute (Hinxton, UK) which uses information from microarray experiments that supports the MIAME (Minimum Information About a Microarray Experiment) requirements. It integrates up to 12000 genes from 35 species, including mice and humans. The data for the integration is submitted online or from local databases and then is curated (checked that it fulfils MIAME) (Parkinson et al., 2005).

GENSAT (Gene Expression Nervous System Atlas) is a project of the Rockefeller University founded by NIH (National Institute of Health). It is a public database including gene expression data for two recombinant mouse lines: BAC-EGFP and BAC-Cre. It uses *in situ* hybridization and transgenic mouse techniques to obtain the data. In this database, structures are not annotated in an ontology, but in a vocabulary annotation (Wheeler et al., 2003).

BioGPS is an interface designed by researchers at Genomics Institute of the Novartis Research Foundation in San Diego (California, USA). BioGPS is based on a simple, unstructured plugin interface which allows integrating data from mice, rats and humans, and from either existing resources from papers or own resources from the developers (Wu et al., 2009).

The Allen Brain Atlas is a project which, as does EMAGE, includes also spatial information, together with the gene and the anatomical information. It is a project developed by the Allen Institute for Brain Science in Seattle (Washington, USA). It includes information from human, mice and non-human primates, obtained by in-situ hybridization of around 20000 genes, as well as the histological 3D images of the brain (Sunkin et al., 2013).

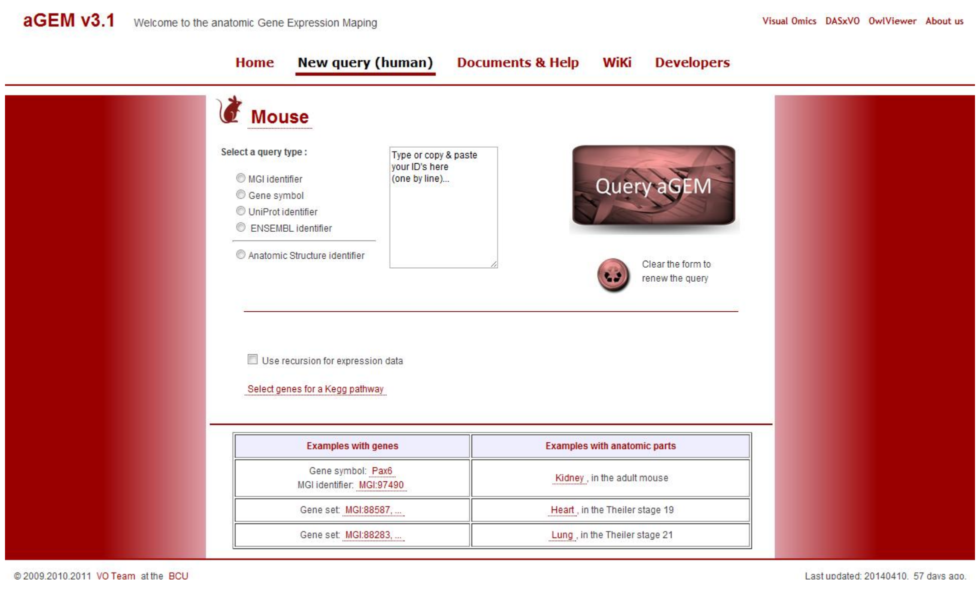
GXD (Gene eXpression Database) is a project from the Jackson Laboratory in Bar Harbor (Maine, USA). It collects data from scientific literature, laboratories and from large-data providers, which allows it to capture a broad spectrum of assay types, and covers all developmental stages and tissues. GXD is updated daily, providing an immediate response to the biomedical research community. It includes data from around 9000 genes, and integrates it with other Mouse Genome Informatics (MGI, another project which is taking place in this laboratory) resources and many other databases (Finger et al., 2011).

Integrating the gene expression databases: aGEM

Although the great amount of available gene expression databases might be beneficial for researchers, obtaining all the information regarding the function of a certain gene may constitute a difficulty and inconveniency. It is not only highly inefficient but very user-unfriendly, since the user will need to deal with several very different interfaces. It is also a problem for developers, because the various interfaces which will need to be programmed independently from one another. Also, there will be a problem in resolving synonyms, since certain anatomical structure or gene might be named in one way in one database and in another very distinct one in other databases.

aGEM stands for "anatomical Gene Expression Mapping" and it is a platform designed by the *Instituto Nacional de Bioinformática* and *Unidad de Biocomputación* of *the Centro Nacional de Biotecnología* (National Institute for Bioinformatics and the Biocomputing Unit of the National Centre for Biotechnology) in Madrid (Spain). The purpose of the platform is to overcome the problem which has been mentioned above. aGEM integrates information from EMAGE, GXD, GENSAT, Allen Brain Atlas, EUREXPRESS and BioGPS databases for mice, and HUDSEN, Human Protein Atlas and BioGPS databases for human information. It also includes OMIM (Online Mendelian Inheritance in Man) which is a NCBI tool which allows obtaining a correspondence among genes and related diseases in humans. The mapping among the different databases needed to be performed in most cases manually, due to the already mentioned problem of resolving synonyms. (Jiménez-Lozano, Segura, Macías, Vega, & Carazo, 2009, 2012)

aGEM allows the user to make queries by anatomical structure, returning all the genes that are expressed in the structure; and by gene, providing all the structures in which the gene is expressed. It can also find correlations among the findings of the two former query types.


aGEM v3.1
Welcome to the anatomic Gene Expression Maping
Visual Omics
DASxVO
OwlViewer
About us
Home
New query (human)
Documents & Help
WiKi
Developers
Mouse
Select a query type :
MGI identifier
Gene symbol
UniProt identifier
ENSEMBL identifier
Anatomic Structure identifier
Type or copy & paste your ID's here (one by line)...
Query aGEM
Clear the form to renew the query
Use recursion for expression data
Select genes for a Kegg pathway
© 2009.2010.2011 VO Team at the BCU
Last updated: 20140410. 57 days ago.

Examples with genes	Examples with anatomic parts
Gene symbol: Pax6 MGI identifier: MGI:97490	Kidney , in the adult mouse
Gene set: MGI:88587, ...	Heart , in the Theiler stage 19
Gene set: MGI:88283, ...	Lung , in the Theiler stage 21

Figure 1: aGEM website ("aGEM," 2014)

Preclinical imaging

The purpose of preclinical imaging

Preclinical imaging is defined as the use of imaging techniques on animals, for research purposes (that is, veterinary images are not considered preclinical imaging modalities). The aims include testing new imaging techniques which are in development before using them in humans, observing the effects of diseases on the physiology of the animals and also the effects of new drugs on them. Using these techniques, researchers can monitor changes at body, organ, tissue or even cellular and molecular level, and extract conclusions which may be really interesting for the field of study.

As happens with clinical imaging, there are several modalities which are of great interest in preclinical imaging purposes. The modalities can be discriminated into classes according to different features: the type of energy which is used for producing the image (X-ray, nuclear, ultrasound, optical, magnetic resonance) or the features we can observe in the images (anatomical structures in anatomical imaging modalities versus physiological processes in functional imaging modalities). Each of these techniques is useful for a different kind of study, depending on which issues we are interested in.

Atlases: a brief definition

As far as anatomical images of animals and humans are concerned, it is interesting to merge the information obtained from those images in atlases. An atlas is defined as a collection of maps. These maps can be related to geography, representing regions on Earth, but also to anatomy, representing the structures of the human body (or animals). In an anatomical atlas, a visual representation of a structure is tagged with text, in a way that one can observe the picture and relate it with some terms. An anatomical atlas can be elaborated using illustrations, photos, or medical images. The main methods which can be used to obtain images which are suitable for their use in an atlas elaboration will be described below.

Atlases elaboration process and imaging modalities

There are many ways an atlas can be obtained from a sample. They can be done by using ex-vivo techniques as autoradiography, histological maps, taking photographs of different slices of our tissue of interest, or even making a drawing which represents the tissue more or less accurately. Nevertheless, the most interesting techniques for elaborating an atlas are medical imaging modalities, since they allow in-vivo acquisition of images; and it is also easier, since they do not require the work of removing the structure of interest from the rest of the body and slicing it. There are many medical imaging modalities which could be considered in order to obtain a useful atlas for the brain. In the following paragraphs there shall be considered several options, as X-ray computer tomography (CT), ultrasound, nuclear medicine techniques, optical imaging or magnetic resonance imaging; considering their pros and cons and justifying the final decision.

X-ray modalities use high energy photons (in the range of energies from 100 eV to 100 keV) to produce an image. X-rays are irradiated through the body, interacting with internal structures. A portion of the energy in the photons will be deposited in the body as a different energy form and the rest will be received by a sensor which will convert it into an electrical signal. In an X-ray image, the parameter we can distinguish from the tissue is its density, which is directly related with the atomic number. The energetic spectrum of X-rays (the amount of photons with certain energy) determines the contrast of the image: the less energetic the spectrum is, the higher the contrast; but also the higher the dose to the patient. There are many ways to use X-rays to produce an image, and each of them has its applications. The simplest one is conventional radiography, in which the body densities are collapsed into one simple image, which shows the sum of densities of the tissue in one direction. From several conventional radiographies, and after processing, we can obtain other useful images like digital subtraction angiography, dual energy subtraction or tomosynthesis, which are of great interest for some applications, like blood flow monitorization or the enhancement of certain structures of interest. Nevertheless, the most interesting application of X-rays is probably X-ray computed tomography (CT). As its own name suggests, we can obtain slices of the sample and observe the properties it presents much more accurately than when done with a planar radiography. The most important disadvantages of X-ray imaging are fact that X-rays are ionizing radiation and produce little contrast among soft tissues due to their almost uniform density; that is why, in order to image soft tissues as in mammography, we need to compress it and use a lower energetic spectrum. Also, in small animal imaging, due to the smaller amount of tissue to be traversed, we need lower energy spectra in order to correctly appreciate differences in tissue. For elaborating a structural atlas of the brain, using X-ray images is not a good choice since they do not provide enough contrast in the soft tissue structures which are present inside the skull.

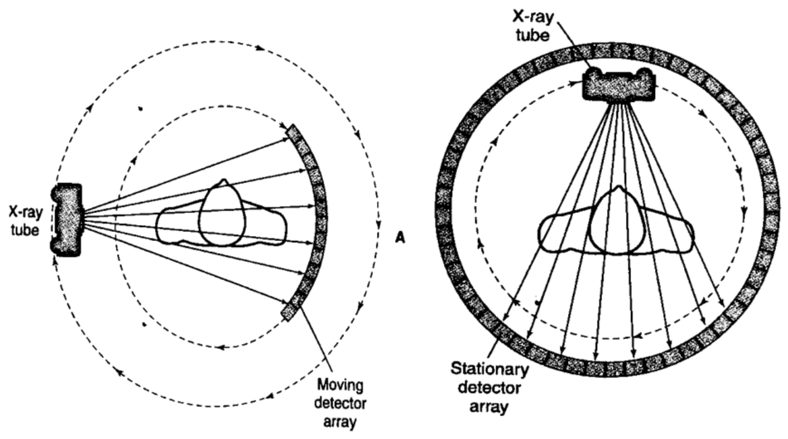


Figure 2: Third generation (left) and fourth generation (right) computed tomography machines. A third generation machine uses a rotating source and detector, while in a fourth generation one, the detector array is static and only the source rotates. Image extracted from ("Asian Radiology," 2014)

Another possibility is to use echography, which uses ultrasound pulses which are irradiated into the body, and then the echoes are detected after a certain time has gone on. Since the velocity of sound is constant, we can set exactly where the interfaces between the tissues are situated. In this way, we can also reconstruct tomographic slices of the body. The most interesting issues about ultrasound imaging are the fact that it is a non-ionizing source of energy, and hence it does not produce problems of DNA ionization due to radiation. It is also cheap and easy to use. Nevertheless, it is impossible to image through bone or air, and therefore it is impossible to image a brain using ultrasound, since it is totally enclosed by the skull. This modality also presents a really low signal to noise ratio, the lowest of all image modalities.

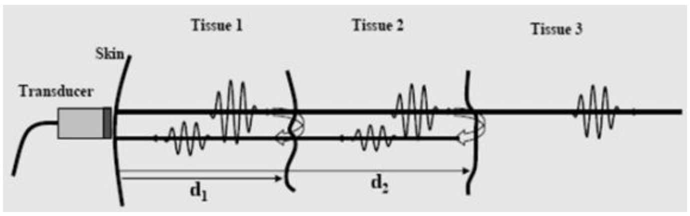


Figure 3: Ultrasound imaging basics.

A third possible modality is nuclear medicine, which detects the activity of a certain isotope which emits gamma radiation, either directly (SPECT: Single Photon Emission Computed Tomography) or by positron annihilation of an electron (PET: Positron Emission Tomography). The tracer is normally injected into the body combined with an organic molecule, and it is distributed by the bloodstream. Since the molecules present more affinity by some body receptors, they will concentrate in those regions and will cause the measured activity to be higher than in other areas. This image modality is a functional modality, since it detects a certain physiological activity, and it can be really interesting for purposes such as the monitorization of the glucose uptake in brain or tumours. However, since it does not determine anatomical structures (further away from the ones already mentioned), it is not very interesting for the elaboration of an atlas. It could be interesting to use ex-vivo autoradiography for this purpose, but needs convenient preparation of the sample.

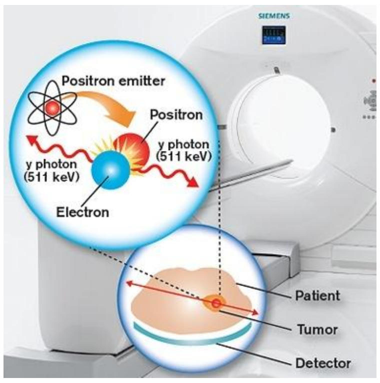


Figure 4: Mechanism of PET imaging technique (nuclear medicine) Extracted from ("Life Extension Magazine (LEF)," 2014)

Optical imaging uses photons in the range of visible light in order to produce a signal which can be measured and analysed. Most tissues are transparent up to a certain point to visible light, and also, we can use fluorescent molecules in order to produce an image, as it is done in SPIM. Optical techniques are of great interest since they use non-ionizing radiation, which avoids the animal (or the patient, in the case of clinical applications) problems caused by radiation. There are some techniques like ultramicroscopy, with which promising results have been obtaining in imaging transparent, small animals like Drosophila's brains. Nevertheless, these techniques for mammal imaging are already in development, and neither of them is so advanced to produce an atlas which can be used for our purpose. (Menzel, 2011)

For the acquisition of images in order to produce an atlas, it is also interesting to consider using Magnetic Resonance Imaging techniques. These techniques put the animal into a high intensity magnetic field, which aligns the spins of all the nuclei within it. Then, a pulse of radiofrequency is sent to the sample, and an output signal is produced. The output signals are detected by coils and transformed into an electric signal, which fills up the k-space (Fourier) of the image. Then, the k-space is processed and the image is obtained. There are several possible sequences which can be obtained with these techniques, T1, T2 and proton density images, which provide each of them different contrasts among the different tissues. More exactly, T1 images provide a great contrast in the soft tissues in the brain, so it would be interesting to build an anatomical atlas. Apart from the good levels of contrast resolution which can be obtained, it should be also bear in mind that magnetic resonance uses radiofrequency pulses, and hence in the range of non-ionizing radiation.

	Advantages	Disadvantages
Histological cuts	Great resolution (differences can be appreciated at cellular level).	Ex-vivo. Need for a technician to prepare the sample.
Autoradiography	Shows different slices of the tissue of interest.	Ex-vivo. Need for a technician to prepare the sample, who will get ionizing radiation.
Conventional radiography	Cheap and fast	Planar. Useless for an atlas.
X-ray computed tomography	Cheap, fast and tomographic	Low contrast in soft tissue structures like brain
Ultrasound imaging (echography)	Cheap and fast. Tomographic.	Impossible to image through bone. Low SNR.
Nuclear medicine	Tomographic.	Functional image (no anatomy is shown)
Optical image	Cheap. Tomographic.	Still under development for mammals
Magnetic resonance	Great contrast and spatial resolution. Tomographic.	Slow and expensive.

From images to the atlas: Image registration

The first step in an atlas generation operation is the image acquisition. The images can be acquired from a single subject or from several subjects. Acquiring different images from several subjects allows the compensation of the variations they might appear among them, and hence, a more accurate atlas can be obtained, since it would represent more generally the average individual. For this reason, it is necessary that all the images are modified in order to overlap as much as possible among all the subjects. This process is called image registration.

Registration is defined as the process of aligning images so that corresponding features in all of them can be easily related (Maintz & Viergever, 1996). There are several registration techniques, which can be classified in two groups: point-based registration (or feature-based registration) and volume-based registration (or intensity-based registration). In all the cases, the algorithm involves measuring the values of a function (called cost function) which expresses numerically how similar or how different two images are. One of the two images will undergo some kind of geometrical transformation, and then the value of the function is measured. If the cost function measures differences, it will be desirable to minimize it, and on the other hand, if it measures similarities, it will be interesting to maximize its value.

Point-based registration is defined as the selection of a set of points in both of the images which are correspondent among themselves. To register two images in this way, first of all, the centroids (centres of mass) of the point clouds are needed to be computed and then, they are aligned using a translation matrix in one of the images. After that, the analysis of the cost function starts. In the case of point-based registration, the cost function will be the sum of distances of the points within the two clouds. This cost function needs to be minimized, as it expresses the error of the registration process.

In intensity-based registration, the whole image volume is used for computing the cost function which will give us the transformation matrix. Whilst in point-based registration the issue is which point selection is optimal, in intensity-based registration the most important parameter defined by the cost function which best suits our images. Most common cost functions are correlation, sum of squared differences and mutual information (V. Hajnal & L.G. Hill, 2001). Correlation is a statistical measure of mutual dependence between two variables, which needs to be maximized in order to register the images. The sum of squared differences of intensity measures the squared differences in values of intensity between the two images; due to this fact, it is desirable to minimize it. Mutual information is a measurement which is taken from the entropies of each image individually and the joint entropies of the histogram. The larger mutual information is, the more accurate is the registration of the two images.

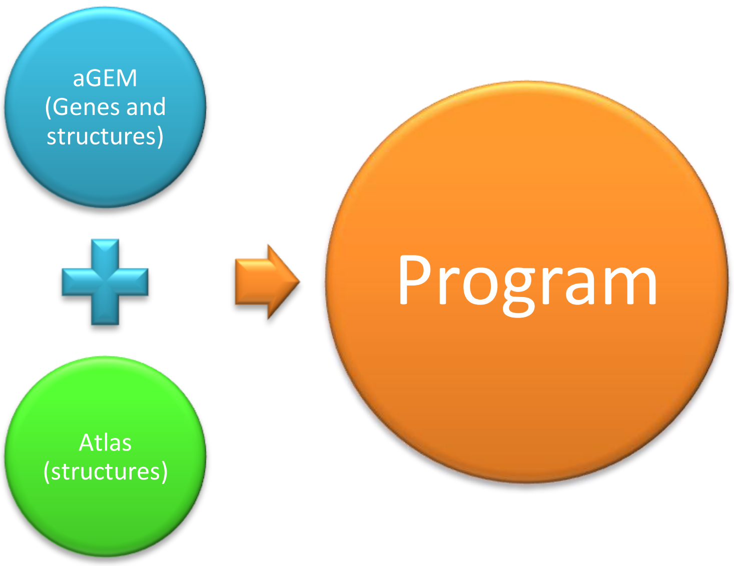
Registration techniques can be enhanced by using multi-resolution scheme approaches. For this approach, first, the two images reduce their resolution and are registered. Then, bigger resolution images are transformed using the matrix which was computed for the low resolution images and then a new registration procedure is performed, using the new transformed image as the starting point. In this way, registration will be much quicker and will avoid getting stuck in local minima.

Motivation and objectives

As far as all the background information about atlases and gene expression databases has already been analysed, now we need to define further the project, its objectives and facts of interest. Gene expression databases, in most cases, do not provide any kind of integration with anatomical information of where those genes are expressed. The characterization of the whole transcriptome for structures like the brain is of limited utility if we have no anatomical information. Combining the databases with the anatomical information provided by an anatomical atlas, we can have lots of advantages. First of all, the most immediate advantage that such integration would introduce would be user-friendliness. Part of this problem is solved with the aGEM tool already developed, that integrates different databases into a single user interface.Visual representation of the gene locations would improve user experience if it is integrated with aGEM. As a second advantage, this integration could facilitate the connection between imaging and gene expression information when defining or analysing results from preclinical experiments.

- The definition of an imaging protocol in order to study the phenotype of a transgenic animal model could benefit from the results of this project, since the researcher could look for anatomical structures related to the genes that have been manipulated.
- The results of image quantification are usually an statistical parametric map, that presents the statistical significance of a certain analysis for every voxel. Significant areas from this image could be related to the underlying genes by means of the proposed integration tool.

So, the main objective of the project is to connect all the information provided by aGEM and by the atlas. For the issue, it is needed to find how the information is stored and related in aGEM, in order to extract enough information of interest to program a first version of the tool. Also, it is required to study which atlases are available and which one is the most suitable for our purposes. Once all these steps have been done, the kind of program which is going to be developed needs to be analysed. There are several possibilities, like a program in Java, C++, Matlab or a plugin for its integration in ImageJ.



Once all the necessary information is extracted, an integration step must be done for the program to be operative. Then, when the information mapping is ready, the interface of the program can be written. The first version of program should be able to perform certain query types:

1. Anatomy query: Given a list of anatomy structures, the user should be able to select any of them and the program would launch a query showing the genes and their information expressed in the structure, which would be shown in the atlas images.
2. Gene query: The user selects a gene and the program would detect in which structures is the gene expressed and show them in the atlas images.

The purpose in a first stage is to integrate part of the information contained in aGEM in a beta trial version of the interface, in order to check its real utility.

Materials and methods

In this section, the steps which have been followed for the development of the project are going to be explained in detail, as well as the tools used for the purpose.

ImageJ

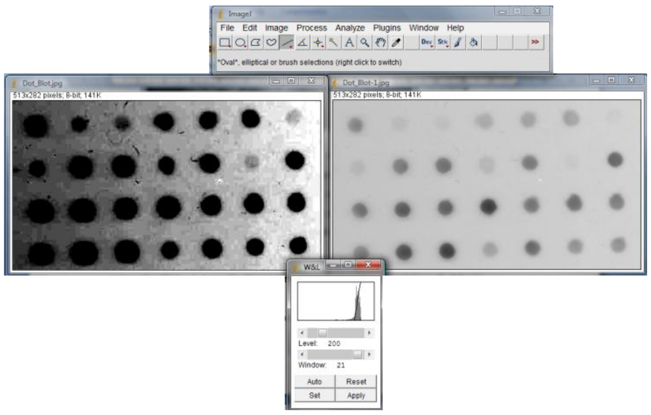
Although, as said, there are several possibilities to build such a program, in a final stage the chosen option was to build a plugin for ImageJ (http://imagej.nih.gov/ij/). This choice was made based on the fact that imageJ is a widely used tool by researchers and relatively easy to implement due to its base in Java programming language (which will be described later).

ImageJ is an image processing program developed by the National Institutes of Health (NIH). It is a public domain program which has a wide variety of functionalities, where probably the most interesting one for our purposes is its extensibility: it was designed with an open architecture, allowing programmers to increase its functionalities by adding plugins programmed in the Java programming language or by recording macros. ImageJ is a platform available for its use in Windows, Linux and Mac OS and it can be downloaded, used as an online applet or installed in any computer with a Java virtual machine 1.5 or later version. Since the ImageJ code is fully available online, programmers can develop their own distributions of ImageJ. One of the most popular of these distributions is the one called Fiji (Fiji Is Just ImageJ) which includes some of the most useful plugins already integrated.

There are several functionalities which make ImageJ a very versatile and desirable environment to work with images. First of all, it is compatible with a large collection of different image formats, among which are JPEG, GIF, PNG, TIFF, BMP, DICOM, FITS or even raw information images (just an array of numbers which can be ordered to build the matrix which will show up the image) notwithstanding the bit depth. Some other formats, as NIFTI or LEI, are available in ImageJ by installing the plugin "LOCI Bioformats Importer" (included in the Fiji distribution). Some of the basic operations which are available to be done in ImageJ are adjusting window and level of the image, applying lookup tables, thresholding, filtering... All these operations are performed in one or some regions of interest (ROI) which can be designed from scratch by the user ("Image Processing with ImageJ: José María Mateos Perez, Javier Pascau: 9781783283958: Amazon.com: Books," n.d.).

ImageJ functionalities

In the management of medical and microscopy images, one of the most basic and most interesting tools is the adjustment of window and level. This is especially important in the case of X-ray images, in which the grey level of the image corresponds to certain density of the tissue. The densities are mapped into units of image intensity called Hounsfield Units, needed to standardize all the images obtained by different X-ray machines of different developers. In the Hounsfield Unit scale, air is black and represented with values of -1000, while water obtains the grey level 0 and bones present values around 1000. Nevertheless, it could be interesting to enhance the contrast of soft tissue (with a density around that of water, that is, zero) and reduce the amount of detail in bone. For this purpose, one can select 0 as the central value of the scale and map the nearby values into others which are more distanced among them. In this way visualization is enhanced. In the lower images it can be appreciated how window and level settings can be adjusted in order to distinguish better some of the structures of the image.



"Oval", elliptical or brush selections (right click to switch)

513x282 pixels; 8-bit; 141K

513x282 pixels; 8-bit; 141K

Level: 200

Window: 21

Figure 5: Automatic Window and Level adjustment using ImageJ. Adjusted image is in the left side, the right one is the original.

Window and level processing does not, however, change the values of the pixels in the image. It is only a change which is performed for visualization. There are other possible changes similar to the one performed by window/level adjustment, which are called lookup tables (LUTs). LUTs change the way different pixel values are displayed, and they do not need to be linear, or to conserve greyscale (they can change greyscale values into RGB colour values)

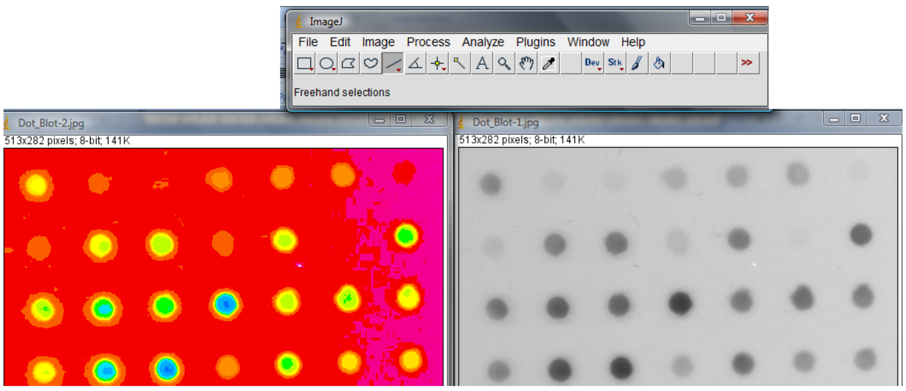

ImageJ
File Edit Image Process Analyze Plugins Window Help
Freehand selections
Dot_Blot-2.jpg
513x282 pixels; 8-bit; 141K
Dot_Blot-1.jpg
513x282 pixels; 8-bit; 141K

Figure 6: Application of a fake colour LUT to an image

Thresholding is an operation used to segment an image. Segmentation can be defined as the extraction of a portion of an image, which is of our interest. It takes into account also the grey level of the pixels, and establishes that pixels above (or below) a certain level of intensity are of interest for us and we want to extract them. ImageJ can perform single thresholding, by selecting only one intensity level, or multi-level thresholding, if there is an interesting band of intensities which is wanted to be segmented. In any case, thresholding produces a binary image which will have pixels with a value of 255 in the places in which those pixels of interest could be found in the original image, and pixels with value 0 in the places in which the value of the pixel in the original image was out of the range of interest.

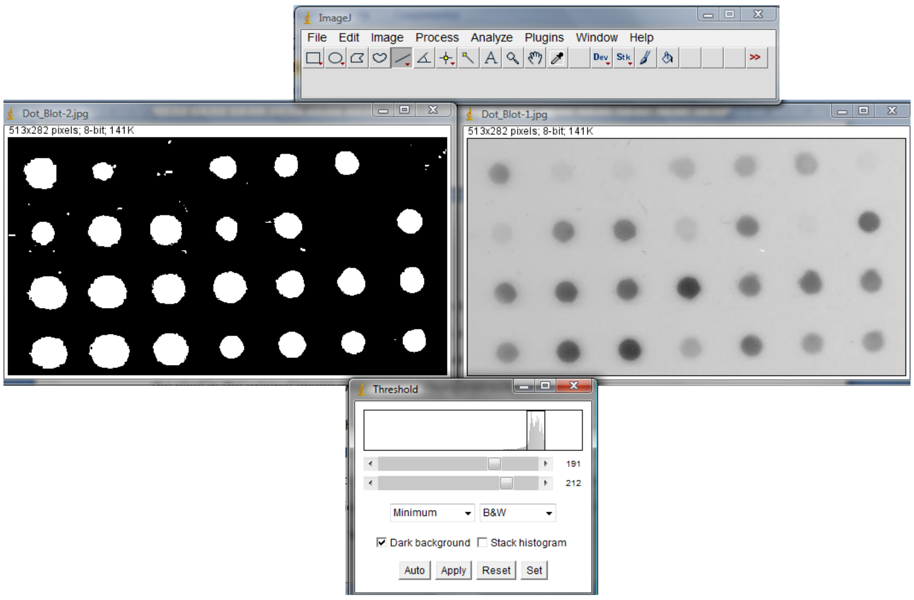


Figure 7: Double thresholding (left) of an image with ImageJ.

Filtering an image allows to reduce noise, or to enhance edges in an image, among other possible modifications. ImageJ allows the user to filter the images with Gaussian filters (low pass isotropic non-adaptive filters) which will blur the image causing a net noise reduction, but also causing edges to be lost. It has also implemented the median filter (which substitutes the value of the pixel by the median of the values of the neighbourhood), and it allows to use personalized filters.

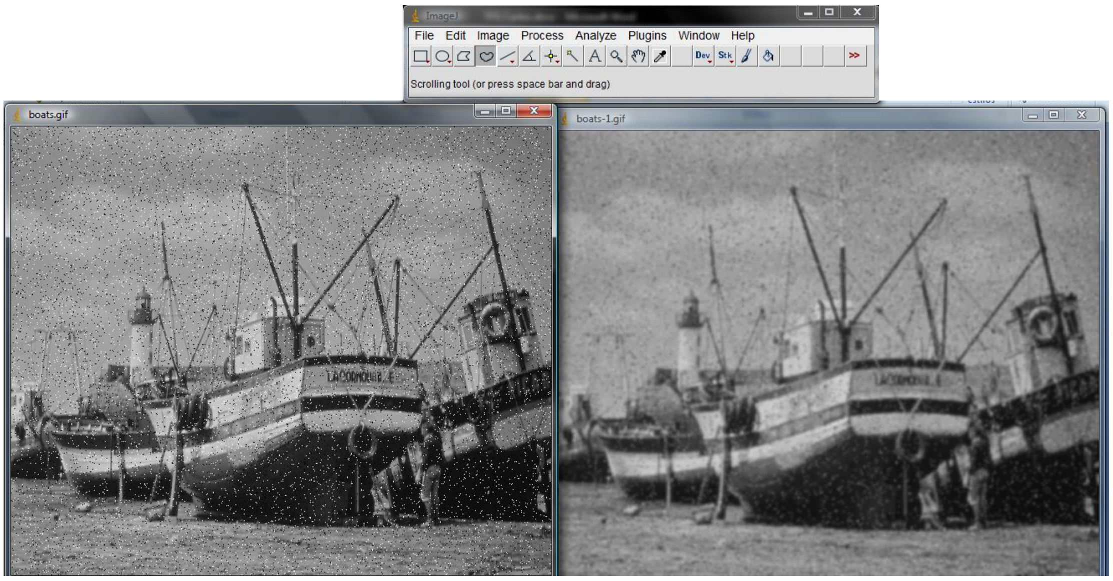
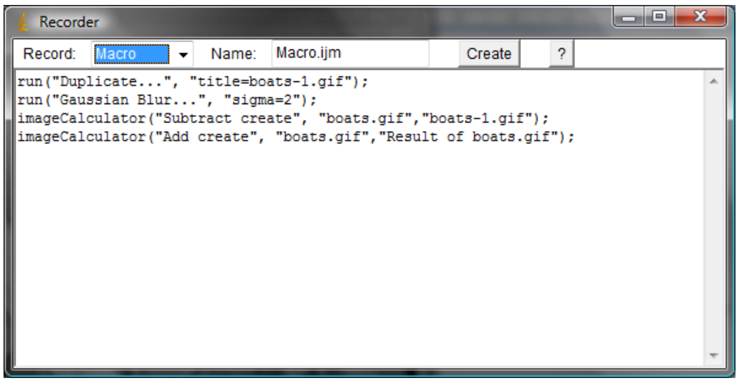


Figure 8: Application of a Gaussian Low Pass Filter to an image with noise.

Macros and plugins in ImageJ

A macro is a set of instructions which are stored in order to be executed sequentially using a single call or instruction. ImageJ has several implemented functions, but in many cases, performing a certain task in image processing does not only require one instruction, but several, one after the other. By storing them in a macro, one can repeat them all the times they are desired without need of going one by one. ImageJ has enclosed a macro recorder which records all the instructions the user has performed in the program and saves them in a macro. In this way, then the macro can be executed and ImageJ will be able to repeat exactly the same process which was executed, step by step. For example, the operation of edge enhancement that was performed for the image below can be stored in a macro in this way:



```
run("Duplicate...", "title=boats-1.gif");
run("Gaussian Blur...", "sigma=2");
imageCalculator("Subtract create", "boats.gif","boats-1.gif");
imageCalculator("Add create", "boats.gif","Result of boats.gif");
```

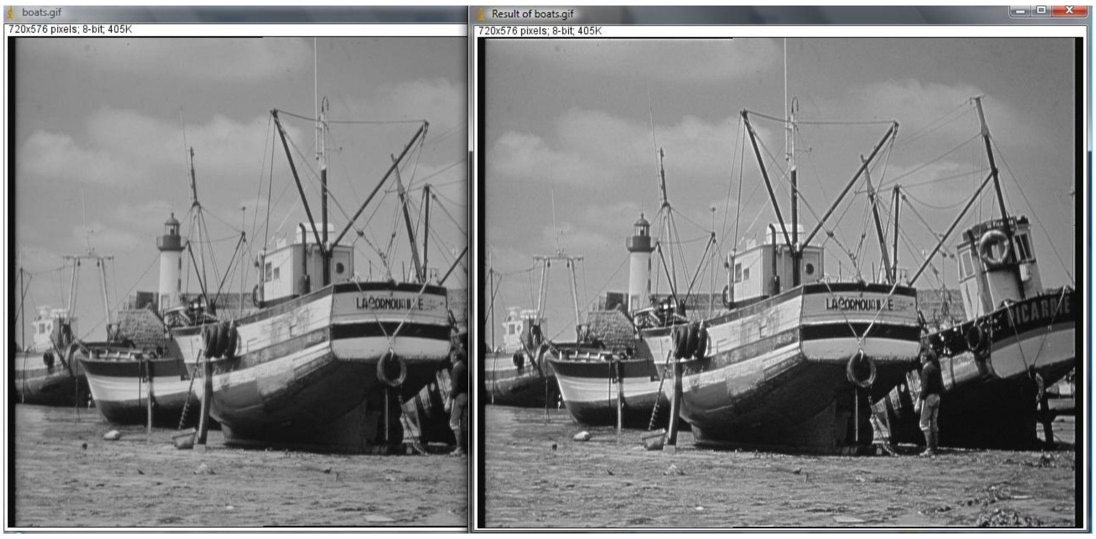


Figure 9: Application of an edge enhancement macro (upper image) to a sample image

Nevertheless, the weak points of macros are that they do not really add any new functionality to ImageJ, but use functionalities which are already implemented. Also, they are quite slow compared to plugins.

A plugin is defined as an application which is implemented to add new functionalities to another which already exists. They are not needed for the overall workflow of the main application, but the new functionalities they add to it may be vital to perform certain tasks. Plugins are a way to allow programmers to collaborate with the development of an application and to split certain functions from the main code which would increase too much the size of the application. In some cases, they are also used to restrict the access to the root code of an application to avoid software licensing conflicts. Plugins, as macros, can use the functionalities which are already present in ImageJ to perform some of their functions, but they do not limit to these functionalities exclusively. Plugins in ImageJ are programmed in the Java programming language.

Integrating the tool planned in this project into ImageJ as a plugin seems natural since all of the image processing tasks which need to be used for the project are already integrated into ImageJ, so there is no need to program them again. Also, programming a plugin is of great interest in case of possible future distributions of the tool or its expansions, since it will be then fully compatible with any terminal in which ImageJ is installed. The source code for the application which was worked out in this project is fully available in a public GitHub repository (Porras Rodriguez, 2014).

Java language

Java is a computer programming language which was originally developed by Sun Microsystems (later, this company would be acquired by Oracle) from 1991 to 1995, when its first version was released to the public. It is a concurrent and object-oriented programming language. This means that Java allows the execution of several computations at the same time, and these computations operate in objects which are distributed into classes. A class in Java is a set of objects which share certain defining parameters. Objects in classes are encapsulated, which means that they cannot go away the invariants of the class; in other words, if an object belongs to certain class it cannot acquire parameters which are out of the definition of the class. In this way, errors are avoided in computation, and objects are coherent among them. Java is a compiled language, meaning that the code needs to be translated first into machine language by means of a compiler before executing. Classes include certain predefined methods which can be used to make modifications in some objects of the class. The methods and definitions of each class can be found in its corresponding Javadoc document.

Java programs can be written in several ways: in bare text processors, or in more sophisticated programs like Integrated Development Environments (IDEs). IDEs are software applications which provide comprehensive facilities to computer software development. IDEs integrate source code editors, build automatizators and debuggers. In the case of Java, among the most widely spread IDEs are BlueJ, NetBeans and Eclipse. The use of Eclipse IDE was straightforward because it is the one most used at the Laboratorio de Imagen Médica (Laboratory of Medical Imaging) at Hospital Gregorio Marañón. It is highly convenient also since it can debug and try directly the plugin code in ImageJ. All the processes of programming in Java that are performed in this project use the Eclipse IDE.

Java allows the programming of interfaces using Swing, which is also compatible with ImageJ. Swing is the main Java graphical user interface (GUI) toolkit, and uses separated classes for each of the windows which are generated by the program. Since it is extensible, Swing permits the possibility to custom all of its implementations using Java inheritance mechanisms.

Understanding aGEM: databases

aGEM is a platform which integrates lots of information from many different sources. This information should be stored in a database to allow interactions as accession, modification and update. There are several standards of databases which can be used. In the case of aGEM, the information is stored in a MySQL database, which contains multiple tables.

MySQL is a relational database management system (RDBMS) which is based on the SQL language. This is a programming language specifically designed to manage data in a database in a comprehensive way (SQL stands for Structured Query Language). MySQL organizes information in tables, to which the user can access using the queries. In any database in MySQL, the user may retrieve information (SELECT queries), delete information (DELETE queries), modify the information in a given field (UPDATE queries) or introduce new information (INSERT queries).

In aGEM, the information is contained in 112 tables, each one storing information of a different database, or different datasets involving the same database. For this very first version of the program, the only tables which are going to be used are the tables of the database GXD, since they provide the most complete information of the genes and structures. The information on structures was obtained from the table *tbvg_mgi_gxd_structure*, extracting all the structures which are enclosed in the brain and discarding the others. Genes, on the other hand, are contained in the table *tbvg_mgi_gxd_full*. The other tables for GXD contain either repeated information or information which is not of interest for the application like the method used to determine the gene presence or absence in a certain structure.

Atlas

As explained before, it is necessary an atlas which represents the mouse brain in a way that many structures can be recognised and mapped into it. It has to be an atlas which includes an image obtained by magnetic resonance imaging techniques in order to be able to differentiate the structures. The atlas has also to contain comprehensive tags which are easily extracted by means of simple image processing techniques such as thresholding or window and level adjustment. The number of tags has to be average: a very big number of tags would make the mapping of the atlas with the structures of aGEM unbearable, whilst a low one would limit the usefulness of the project. Also, the information of the tags must stand on a format such as XML, which is compatible with an ontological distribution of the tags and also parseable with Java.

Available atlases

6 atlases were under consideration for this project. Some of them are found in the supporting bibliography, whilst others required more in-depth investigation.

Name of the atlas	Number of tags	Dimensions	MR Image available	Tag format	Other information
Toronto	43/62	210x274x141	Yes	.txt	Information unavailable in documentation
Ma	20	192x96x256	Yes (8 bits)	Unavailable	Only atlas image available
LONI MRM	799	256x256x256	Yes	.xml	
Duke	38	1024x512x512	Yes	.xml	RGB tag image
Allen	1205		No (only histological)	.xml	Highest number of tags and resolution
LONI MDA	53	346x346x346	Yes	.xml	Final choice

Table 3: Summary of the different atlases considered in the project.

Toronto Atlas is a project of the Clinical Integrative Biology department of the Sunnybrook Health Sciences Centre in cooperation with the Mouse Imaging Centre at the Hospital for Sick Children of the Toronto Centre for Phenogenomics. The atlas which was of interest was the so-called Variational Mouse Brain Atlas, which is defined as *“a three-dimensional atlas of the mouse brain, manually segmented into 62 structures based on an average of 32 µm isotropic resolution T2-weighted, within skull images of forty 12 week old C57Bl/6J mice, scanned on a 7T scanner”*(Dorr, Lerch, Spring, Kabani, & Henkelman, 2008). Nevertheless, the atlas which is available for downloading and free use is not this one, since it presents only 43 tags instead of 62. When information on the atlas of 43 tags was looked for, the paper is unavailable for the public.

Regarding the Ma Atlas, it is a project of the Mount Sinai School of Medicine in New York, the University of Florida, the Stony Brook University (New York), Brookhaven National Laboratory, and The National High Magnetic Field Laboratory. The project described by the paper consists in the development of “*A comprehensive three-dimensional digital atlas database of the C57BL/6J mouse brain was developed based on magnetic resonance microscopy images acquired on a 17.6-T superconducting magnet. By using both manual tracing and an atlas-based semi-automatic segmentation approach, T2*-weighted magnetic resonance microscopy images of 10 adult male formalin-fixed, excised C57BL/6J mouse brains were segmented into 20 anatomical structures*.” In order to access to the information, registration was needed in the website of the project.(Y Ma et al., 2005; Yu Ma et al., 2008)

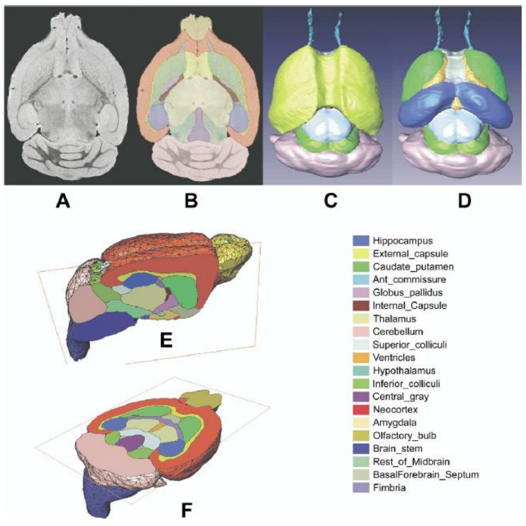


Figure 10: Ma atlas. It can be observed the magnetic resonance image (A) and the structures tag image (F). In F, each colour represents one structure in the list. The two images can be overlapped (B)

The third atlas which was considered was the atlas developed in at the Center for In Vivo Microscopy of Duke University, in collaboration with the Laboratory for Bioimaging and Anatomical Informatics of Drexel University. This atlas is developed in Waxholm Space, which is defined as the standardized coordinate system defined for rodent brain atlases (mice and rats). Waxholm space was promoted by the Digital Atlasing Task Force committee in May 2008. It is a high resolution atlas which contains images not only of magnetic resonance, but also histological images of the brains. It is defined as *"multi-specimen, multi-spectral space: 3 different MR acquisitions, delineations of 37 structures, and a Nissl volume for each of 7 male adult C57BL/6J mice."* (Johnson et al., 2009)

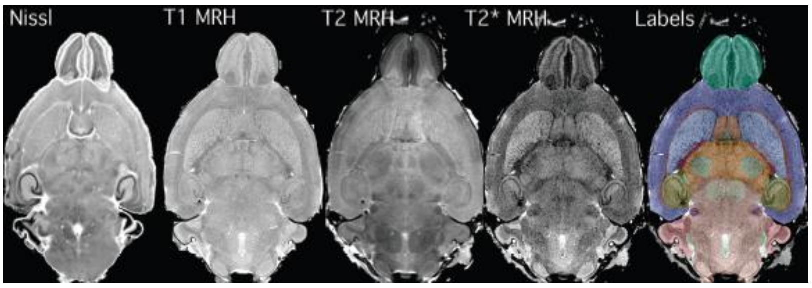


Figure 11: Duke University atlas. It can be appreciated the three magnetic resonance images (T1, T2 and T2) and the labels overlapped with them.*

Finally, the two atlases developed by the Laboratory of Neuro Imaging (LONI) of the University of California in Los Angeles (UCLA) were also examined. These two atlases are the Magnetic Resonance Microscopy atlas (MRM) and the Minimum Deformation Atlas (MDA). Both atlases are built during the evolution of a project committed *"to determine whether brain atrophy occurs in the mouse model, experimental autoimmune encephalomyelitis".* For the purpose, the atlases were necessary to evaluate changes in the brains of the mice affected by the induced disease. To build up the MRM atlas, "*magnetic resonance imaging was performed using an 89 mm vertical bore 11.7 T Bruker Avance imaging spectrometer with a micro-imaging gradient insert with a maximum gradient strength of 100 G/cm and 30 mm birdcage RF coil (Bruker Instruments).*" Then, after semiautomatic segmentation (automatic, but manually corrected for inaccuracies) the MRM atlas was obtained. Using image processing techniques on MRM atlas (mainly several constrained affine transformations) the developers obtained the MDA atlas.(MacKenzie-Graham et al., 2004, 2006)

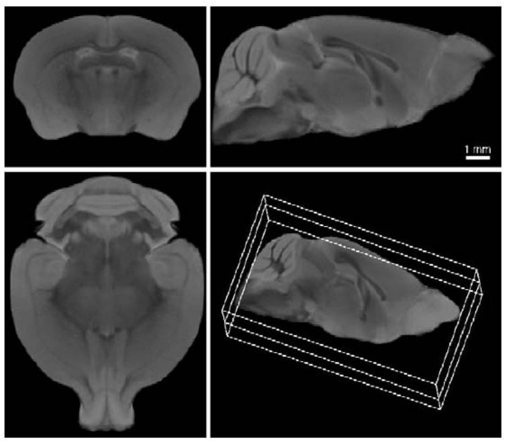


Figure 12: Magnetic resonance image of the LONI MDA atlas.

Atlas Selection

In order to select the atlas, several features were considered. First of all, the number of tags should not be too small or too large. Ma atlas was supposed to contain only 20 tags (according to the bibliography) which are too few, and it did not even have a separate file containing tags, so it was discarded because of this at first. LONI MRM atlas contained 799 tags, a number which was too large for the pretensions of the beta version. Also, the number of tags available in the Allen Brain Atlas is huge. Atlases like Toronto, Duke or LONI MDA were found to be an interesting choice regarding this particular aspect.

The second requirement was that the atlas should contain a magnetic resonance image of the brain of the mouse, for the reasons detailed above. Only the Allen Brain Atlas included an image which was not a magnetic resonance, but histological images of different cuts of the brain. Nevertheless, this atlas had already been discarded because of the huge number of tags it presents.

Another requirement is that the atlas images should present appropriate dimensions. If the dimensions of the image are too small, then it means that the pixel size is too big, and hence, less spatial resolution. But, on the other hand, if the image size is too big, the program will be delayed in its execution because of that. Duke Atlas presents a huge image, so although its tag image was a user-friendly RGB image, the execution using this atlas would have been too slow. Also, the number of tags it contains (38) is too low for the great spatial resolution which is achieved in the atlas. For this number of tags, it is not needed such a big image to show them correctly.

It is also essential that the atlas has a tag list which can easily be recognized and parsed using a Java program. Among these atlases, only Ma atlas did not present such a file. All the information of the tags of this atlas is fully available online upon registration in the webpage of the institute which developed the atlas, but this is not enough for the purposes of the application: it is a highly inconvenient issue for the user, and also for the programmer. All the rest of the atlases include attached a file with the information about their tags, either in .txt format or in .xml format. Text files can be easily read by simple Java programs. XML files require more advanced techniques as the use of parsers like DOM, SAX or JAXB, but are relatively easy to handle.

Last, but not least, it is necessary that all the documentation regarding the atlas and its development should be public, fully accessible, and clear enough. All the atlases have this information available; it was only in the case of the Toronto atlas where the information was obscure. The team which developed the Toronto atlas had several atlas projects published, and it was not really clear to which one of the atlases belonged each piece of documentation. So, it was finally discarded too.

All these steps leaded the programmer to choose the LONI MDA atlas, since it was the one which best fulfilled all the requirements of the project, which have been described above: It provides a magnetic resonance image which is neither too big nor to small, a number of tags which is also adequate, and all the process which lead to its development can be found clearly in the bibliography.

Structure mapping

Atlas parsing

The mapping of the structures is probably the most important step of the whole project. As already said, it cannot be performed by any automatic algorithm due to the different number and level of detail of structures which is present in aGEM and in the atlas. aGEM contains 276 different structures which need to be mapped into the 38 tags which are defined in the atlas and are related to brain (the other 15 tags represent the spinal cord and other structures of the Central Nervous System which are not related to brain).

The first step to perform the structure mapping was to download the atlases and obtain the structures contained in them. For this purpose the files including the tags were to be parsed. For parsing, two options were considered: DOM and JAXB. JAXB seems immediate since it allows automatic extraction of the information contained in the .xml file. It generates automatically a class in Java which encloses the values that are extracted from the .xml file as internal parameters. In this way, an object of the new class will be created for each object that the parser reads from the .xml file.

Although the use of JAXB might seem quite straightforward, it was not the option finally chosen. Due to irregularities in some of the .xml files, the JAXB parser did not work properly. After several trials with the different tag files from the different atlases, finally, DOM was the parser of choice. Unlike JAXB, DOM needs to be programmed manually and in a different way for each case. Although the programmer would normally need also to create a class for the objects contained in the .xml files, in this case the class which was automatically generated by JAXB for the .xml file containing information from the Allen Brain Atlas was to be used, with some modifications, in order to include the fields that appear in the other atlases and did not in Allen's. This class was named "Structure" and is one of the main pillars on top of which the whole application stands.

After all the atlases were parsed, then the number of tags could be quantized and the importance of the different fields could be contrasted. In this way, it could be observed that apart from the advantages already described for the use of the LONI MDA atlas, it could be found that the correspondence between the structures and the tag image was immediately implementable: one of the fields in the .xml file included the grey level of the structure in the image containing the volume information of the tags. So, the use of this atlas was even more encouraging, since the programming is simpler and less prone to errors.

The class which reads the atlas and stores the list of structures in it, including all the information which would be enclosed in them, is called "MDAtoSTR" class. When an object of the MDAtoSTR class is generated, the information of the .xml file containing the tags is read and stored in a list in the variable *MDATags*.

aGEM parsing

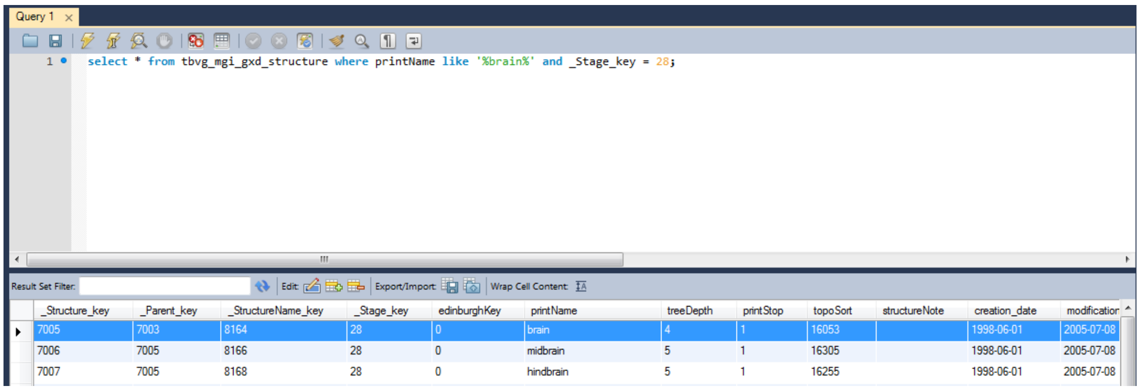
As already mentioned, information in aGEM was not contained in an .xml file, but in a MySQL database. Nevertheless, this was not in any case an obstacle for the integration of the information into the application. Since the database containing aGEM is stored in the server of the National Centre for Biotechnology, and for the execution of the project, the information was needed both there and at Gregorio Marañón Hospital, the database was dumped into a file. Then, the file was loaded in the server of the hospital in order to rebuild the database.

The second step was to determine which tables contained all the information which was needed to be read by the plugin. For that purpose, the tables were examined one by one using queries which extracted all the fields contained in all of them. The queries were launched by means of MySQL Workbench 6.0 for Windows 7. In this way, it was determined that the information which was of interest was contained in the tables t*bvg_mgi_gxd_structure* and *tbvg_mgi_gxd_full*.

The information in the table *tbvg_mgi_gxd_structure* contains, among others, the following fields of interest, which were added to the original "Structure" class that had already been defined:

1. *_Structure_key*: It contains a number which is used as a unique identifier for each structure. It contains a different key number not only for each structure but for each Theiler Stage (homologous structures in different Theiler Stages are treated as different)
2. *_Parent_key*: It stores the number corresponding to the identifier of the structure in the immediate upper level.
3. *_Stage_key*: It is the number of the Theiler Stage to which the information regarding the structure is related.
4. *printName*: It describes the name of the structure.
5. *treeDepth*: It is a number containing the number of levels of depth of the tree needed to go down in order to get to the structure.

To extract the brain structure, a query was launched. The constraints were that the value of the field *_Stage_key* should be 28, since in the atlas, the samples were adult mice in that stage; and also a constraint in *printName*, which should be similar to "brain". After the query was launched, the result found was that the *_Structure_key* for the structure representing the whole brain in the 28th Theiler Stage was 7005.



```
select * from tbvg_mgi_gxd_structure where printName like '%brain%' and _Stage_key = 28;
```

_Structure_key	_Parent_key	_StructureName_key	_Stage_key	edinburghKey	printName	treeDepth	printStop	topoSort	structureNote	creation_date	modification
7005	7003	8164	28	0	brain	4	1	16053		1998-06-01	2005-07-08
7006	7005	8166	28	0	midbrain	5	1	16305		1998-06-01	2005-07-08
7007	7005	8168	28	0	hindbrain	5	1	16255		1998-06-01	2005-07-08

Figure 13: MySQL workbench. Query example and results.

Then, after knowing the root structure (brain), it was needed to extract all the structures below it in the structures tree. For that purpose, it was highly unpractical to use MySQL Workbench, since many queries would have been necessary. The procedure that was followed was to extract directly the list into the program using Java code that can automatically generate each query and would also be generating the list. For this issue, it was needed to add to Eclipse an external jar library: MySQL Java Connector 5.1.29.

The program follows a recursive readout method: It first reads the structure "brain" and adds it to the result set. Then, the result set is iterated by going one by one, getting the field *_Structure_key* of each of the elements in the result set and adding to it all the structures found in the table which contain the very same number in the *_Parent_key* field. In this way, the total amount of structures is increased in each iteration due to the addition of more children. When the structures of the last level are reached, the set of structures containing as *_Parent_key* the *_Structure_key* of any of them is empty, so nothing else is added and the program finishes its execution. In this way, a plain list of all the structures is built independently of the level of the tree of structures that can be found. Each of the structures in the aGEM table generates an object of the Structure class in Java.

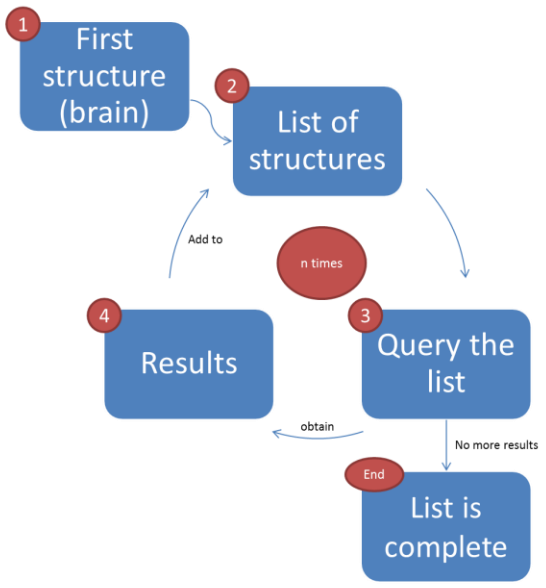


Figure 14: Workflow of the readout of structures from aGEM.

It was also observed that the gene information which was needed should be extracted in a very similar way. The information regarding the genes is found in the table *tbvg_mgi_gxd_full*. This table contains, among others, the following interesting fields:

1. *index*: It includes a single gene identifier in an analogous way as the field *_Structure_key* did in *tbvg_mgi_gxd_structure*
2. *symbol*: A short code of letters that can also be used to identify uniquely each gene.
3. *name*: The name of the gene.
4. *printName*: The name of the structure which contains the gene. There is one entry per gene and per structure.
5. *_Structure_key*: It contains the field *_Structure_key* from the table tbvg_mgi_gxd_structure corresponding to the structure which contains the gene.
6. *_Stage_key*: Theiler Stage.
7. *_Strength_key*: Strength of the gene expression. It adopts a numerical integer value between -2 and 8 (not including the value 0). The meaning of the value can be found by performing a query in the table *tbvg_mgi_gxd_strength*, and corresponds to:
    - -2: Not Applicable
    - -1: Not Specified
    - 1: Absent
    - 2: Present
    - 3: Ambiguous
    - 4: Trace

- 5: Weak
- 6: Moderate
- 7: Strong
- 8: Very strong

Another class was programmed for the acquisition of the gene list, analogous to the “Structure” class, which was called “Gene” class. It includes a field for each of the fields of interest given by the table. In order to extract only the genes of interest, the gene extraction needs to be performed after the structure extraction. The list of structures which was extracted from aGEM is iterated again, and all the genes regarding each of the structures in the list are added to another list. As in the aGEM table, the layout which was chosen to store the information regarding genes was to store each gene expressed in each different structure in a different object of the “Gene” class, which means that there can be several objects for the very same gene. This fact will be of great relevance later, since it makes the programming of the interface more complex.

All the methods which are used to read the information from aGEM (both from structures and from genes) and the lists which contain the information which has been already read are enclosed in the class “aGEMtaglist”. When initialized, the class aGEMtaglist generates both lists storing them in its variables *tagidentifiers* for structures and *allgenes* for genes.

Mapping of the structures

When the lists of structures contained in both aGEM and the atlas were already extracted, then the correspondences among them were established. There are two main problems which must be solved during the mapping process. The first one is the already mentioned problem of resolving synonyms: in anatomical terms, there are structures which can be named in different ways depending on the reference which is quoted. For example, in some anatomy textbook the same structure can be referred as “brain” or “encephalon”. Since this kind of differences can be given also in the tags from the atlas and the aGEM database, it must be taken into account when doing the mapping. This is in part the reason why the mapping has to be done manually and not automatically. The second main problem in the mapping process is the difference between the number of tags in one and in the other list. aGEM has 276 tags versus the 38 tags which are available for the atlas; this means that more than one structure in aGEM will point to the one single structure in the atlas. These relationships among tags should be determined in order to achieve a convenient mapping.

The mapping was performed manually in an .xls file. In one column, the file contains the name of the aGEM tag and in the right one it contains the name of the corresponding tag in the atlas. For using the file within the program, it was needed to add a new external jar for Eclipse, to allow it to read and interpret the file: jxl jar (Java Excel Api). The methods required for reading the file and importing the results to the program are contained in a new class called “mapeo”. Objects of this class contain as subsidiary objects an object of the class aGEMtaglist and another of the class MDAtoSTR, and the methods that relate them.

Plugin development

Interface

The next step is the programming of the interface. Inside the interface there are two steps which should be clearly differentiated. The first step is the programming of the different menus which should lead the user to the different types of queries that can be performed. These menus should be clear enough to avoid confusion. The second step is the programming of the image processing part: what should be done with the atlas image and the tags image in order to show the user the result required by the query.

The layout chosen for the menus was the following one:

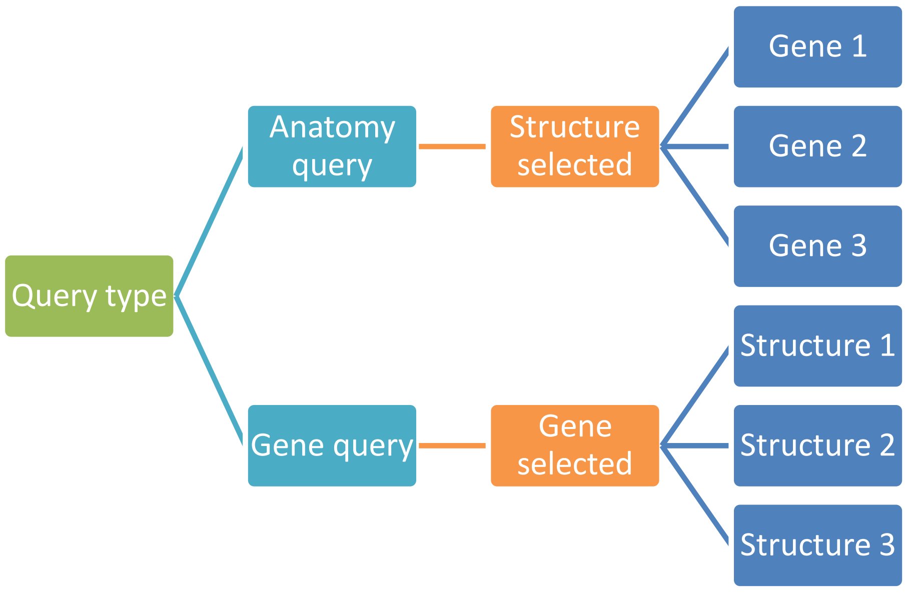


Figure 15: General workflow of the plugin.

For the first decision (Query Type) a generic ImageJ dialog was used. This dialog included two Radio Buttons, to be checked whether the query wants to be performed anatomically or by gene. This kind of menu was chosen due to its simplicity to implement and to be used by the user.

When the user selects any of the options and presses OK, another window pops up. The window should include either the list of available anatomical structures, or the list of genes which are present in the brain according to the GXD database. Nevertheless, some conflicts in the information appeared, both in the anatomical list for the menu and in the gene list. In the case of the anatomical list, there is a conflict in the fact that either the list from the atlas could be used or also the list from aGEM, or both combined. Due to the fact that the structures which will be shown in the images are those in the atlas, it would be nonsense to use the structure tag list provided by aGEM, since it could not be shown up to that level of detail. In the case of the gene list, the problem is given by the fact that the genes in the list contained in the aGEMtaglist object are namely repeated (recall: each gene in each structure generates a different object of the Gene class and hence will be repeated in the list). Because it is not desirable that the genes in the menu are shown repeatedly, in order to be shown, the gene names should be extracted in parallel to a set instead of a list, since a set in Java does not allow repetitions. In this way, the new set would contain the names of all the genes only once, so that it can be used for the menu generation.

Once the conflicts are solved, it is needed to choose a window layout for the generation of the menus. The ImageJ interface dialogs are too simple for this purpose, so they were performed in Swing. The layout chosen in both cases was a plain scrollable window containing all the genes or all the structures the user can query about. The user selects one element and then clicks in “Proceed” button.

To continue with the process, a binary image should be obtained, using it as a mask to overlap with the atlas image. To identify which are the pixels of the image of tags that should be segmented, first we need to extract the list of structures of interest. In the case of an anatomy query, it is straightforward since there will only be a structure of interest, but in the case of a gene query, a list of structures where the gene is expressed is elaborated. The objects of class Structure which are selected contain a field, called ID, which is just the value of the grey level that each structure have in the atlas image.

Once the wanted grey levels are known, binary masks are produced for each one by double thresholding around the value of interest. If there is only one structure of interest, then performing the double thresholding once is enough, but on the other hand, when several binary masks are produced, they need to be combined. They are combined using the ImageJ image calculator, using a logical OR operator.

After the binary mask containing all the pixels of interest is computed, it should be merged with the atlas image. It is an operation which is performed by means of the ImageJ image compositor. The atlas is set to the greyscale channel of the image, while the binary mask is set as the input in the green channel. The result is an image of the magnetic resonance of the atlas in which the regions of interest are shaded green.

Together with the composite, a window containing the information pops up. It is just a plain text, scrollable window which shows the information regarding the query performed. In the case of an anatomical query, the window will show all the genes expressed in the structure queried, and their strength; while in the case of a gene query, it will show all the occurrences of the queried gene in the main gene list, with the information regarding location and strength.

Classes

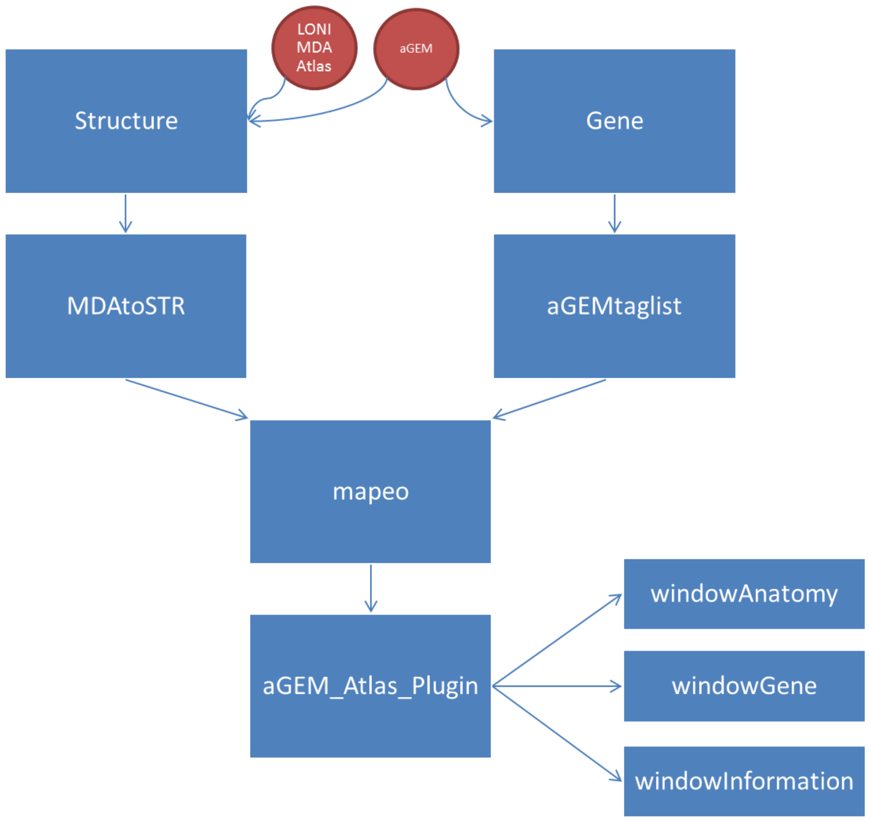


Figure 20: Class diagram of the plugin.

For the correct working of the application, there are 9 core classes. These classes have been already mentioned, and are included in the diagram above. In any case, it is interesting to perform an individual analysis of all of them in order to understand better the workflow of the program.

Class Structure

It is a class which works as a container to include all the data of the structures which are read in the atlases and in the database. The class Structure contains all the variables necessary to import information for the fields of the structures of all the atlases which were used, and the information present in the aGEM table. That is the reason why some of the variables it contains are unused in the program: they are not needed for this version, but are ready to be used in following versions the can potentially include more atlases. It also contains the getter methods (which retrieve the value in the field) and setter methods (which modify the value in the field) for all of the variables. The variables, unless otherwise said, are all of the String class:

- id: It contains the value extracted from the ID field in the MDA atlas .xml file. This id field, as already said, includes the grey level of the structure in the tags image.

- acronym: It includes the abbreviation of the name of the structure. It is used both for the MDA structures and the aGEM structures.
- name: It includes the name of the structure. It is also used in both MDA and aGEM structures.
- children: It is an ArrayList of objects of the class Structure, which contain as objects inside the list, the substructures of the immediate lower level. It is also used in MDA and in aGEM, although the procedure to fill the list is not the same in both cases.

Class Gene

It is a class analogous to the Structure class in the sense that it also works as a container for the data read from the genes in the database. It contains all the variables needed to read the data from aGEM, even though they might not be used for the actual version of the program. It also includes all the getter and setter methods for all of them. The following variables are the ones which are of major importance, and, unless otherwise said, are all of the String class:

- id: It stores the identifier of the gene
- symbol: It includes an abbreviated name for the gene.
- name: It includes the name of the gene
- structureName: It is the name of the structure to which the gene belongs.
- structureKey: It is the key of the structure to which the gene belongs
- strength: It is a string which expresses the value of the strength the gene presents for the given structure.

Class MDAtoSTR

This class includes the variables and methods used for extracting the information from the atlas and storing them. It includes as a variable the ArrayList MDAtags, which will store all the information regarding the structures in the atlas. The class contains a constructor, which includes the steps of connecting to the .xml file, reading it and storing the data read in MDAtags. Apart from the constructor, it includes the methods necessary to retrieve information from the list. The methods which are enclosed in the class are:

- FillFields: This is a private method which is called in the constructor. It builds an object of class Structure and fills all the variables in it with the fields included in the atlas .xml file. This method works in a recursive way: this is required in order to field the ArrayList of the variable “children”.
- strtoString: It converts the list MDAtags into an array of strings containing the names of all the structures. This method is needed to build the list to be included in the menu

of the interface, since Swing is compatible with arrays of strings but not with ArrayLists. It returns the list ordered alphabetically.

- PrintNames: Shows in screen the names of all the structures contained in MDAtags.
- getList: Returns the list MDAtags, acting as a getter.
- printChildren: It prints the children (substructures) which are enclosed in the children ArrayList of each structure of the list MDAtags, in order of occurrence. It does not permit to determine to which structure belongs each substructure.
- getTags: Similar to strtoString, but the list is not returned alphabetically.
- showKinder: It prints in screen the children (substructures) of the first level of the k-th structure in MDAtags.
- searchTag: It takes an string as an input and returns the Structure object which contains such a string in the field "name"
- getChildTags: Analogous to showKinder, but uses the method searchTag to allow to return the substructures of a given name.
- getAllDescendants: Analogous to getChildTags, but it uses a recursive method in order to return all the descendants of the queried tag, not only the first level ones.
- showAllDescendants: Prints in screen the result of a getAllDescendants query.
- getSize: Returns the number of tags in MDATags.

The most important methods are searchTag and getAllDescendants, since they will be used in further operations in the program.

Class aGEMtaglist

This class performs all the operations of interaction with the aGEM database. It includes two lists, one regarding structures ("tagidentifiers") and another one regarding genes ("genelist"). Inside it there can be found all the methods needed to query both lists, and the constructor, which is used to build up the lists using the connection to the MySQL database. The methods which are defined in the class are:

- FillFields: It is a private method (it is only called by the constructor or other classes, it cannot be called by the user) analogous to the method FillFields of the class MDAtoSTR. This means that this method is used to read the structures from the aGEM database and stores them in the "tagidentifiers" list. Nevertheless, in this case, this method is not recursive by itself since the children (substructures) are extracted by another method.

- extractChildren: It is another private method which is used to fill in the list “children” contained in each Structure before adding it to the list (it is called by the FillFields method) by comparing the ParentStructureKey variable of the new structure with the ID of the former ones. Since it also calls FillFields, between both methods, a recursive loop is established.
- setGenes: It is another private method, called by the constructor after the recursive loop of FillFields and extractChildren is finished. It iterates the list “tagidentifiers” and looks in the database for the genes contained in those structures, adding them to the list “genelist”.
- geneToString: It returns an array of strings containing the names of the different genes which are in “genelist”. As said before, this step must be performed in order to obtain a list usable by Swing for the interface.
- strToString: It returns an array of strings containing the names of the different structures which are in “tagidentifiers”, for the same purpose as the former method.
- getGXDtags: It is the getter for the list “tagidentifiers”.
- getGenes: It is the getter for the list “allgenes”.
- printer: It prints in screen all the names of the structures in “tagidentifiers”, as well as the total number of structures at the end of the list.
- geneprinter: Homologous to printer method, but for the list “allgenes”.
- search: It obtains an string containing the name of a structure as an input and returns the object of class Structure which corresponds to such name.
- searchbyID: It obtains an string containing the ID of a structure as an input and returns the object of class Structure which corresponds to such name.
- findstructuresbygene: It obtains a list of structures where a given gene is expressed.
- searchgene: Homologous to search, but works for genes instead of structures.
- searchgenesbystructure: It obtains a list of genes expressed in a given structure.
- showstructurequeryresult: It prints in screen the result of a searchgenesbystructure query.
- getChildren: It returns all the children (substructures) of the first level of a given structure.
- showChildren: It shows the results of a getChildren query in screen.
- getAllDescendants: It returns all the children (substructures) of a given structure, notwithstanding the level they are. It is a recursive method.
- showAllDescendants: It prints in screen the result of a getAllDescendants query.

- getFather: It returns the structure of the immediate upper level which contains another one.
- getAllAncestors: It returns all the structures which contain the queried one notwithstanding the level they are found.
- showAllAncestors: It shows in screen the result of a getAllAncestors query.

Class mapeo

This class builds an object of both aGEMtaglist class and MDAtoSTR class as a variable in its constructor. It contains methods to query the .xls file containing the mapping of the structures, retrieving the correspondences among the lists of structures within them and with the gene list. These correspondences are stored in two hashtables which are the ones which are queried by the user and the other classes of the program. It is the core class of this project, since it performs all the operations of mapping. The class contains the following methods:

- FindCorrespondences: It receives as an input the name of the structure in the atlas, and returns the list of structures of aGEM to which the structure of the atlas maps.
- ShowCorrespondences: It prints in screen the result of one "FindCorrespondences" query.
- FindAllDownstreamCorrespondences: It receives as an input the name of the structure in the atlas, and returns the list of structures of aGEM to which the structure of the atlas maps, and also all their substructures (children)
- showDownstreamCorrespondences: It prints in screen the result of one "FindAllDownstreamCorrespondences" query.
- FindInverseCorrespondence: It receives as an input the name of a structure in aGEM and returns the atlas structure to which it maps.
- ShowInverseCorrespondence: It shows up the result of a FindInverseCorrespondence query.
- FindInverseDownstreamCorrespondences: It receives as an input the name of a structure in aGEM and returns the atlas structures to which it and its substructures map.
- ShowInverseDownstreamCorrespondences: It shows the result of a FindInverseDownstreamCorrespondences query.

Class aGEM_atlas_plugin

This is the executable class which will be run by ImageJ. It includes three methods, all of them private (which means that a user cannot perform the operations contained in the methods by writing commands in the console):

- run: It contains the sequences which will be performed by ImageJ when the plugin is executed. It shows sequentially the different menus and stores the user's choices.
- IllegalQuery: It includes the commands to be executed when there is an error in some step of the class execution.
- makeMask: It is the method which is used to compute the mask which is to be overlapped with the atlas image. As said before, this method uses double thresholding to perform such a task.

Classes windowAnatomy, windowGene and windowInformation

These are Swing window classes, used in order to generate the different menus and pop-ups with the information queried by the user. They are called in the run method of the class aGEM_atlas_plugin.

Results

At first, when the plugin is executed, the query selection dialog appears. The program was tested with all the possibilities. Anatomy queries are faster in execution then gene queries, because they do not require any mask merging, while the gene queries require merging all the masks containing the structures where those genes are expressed. In any case, the execution is fluent, and the average query takes about 90 seconds to be solved.

As said, when execution is started, this dialog appears, which allows the user to choose between the two query types:

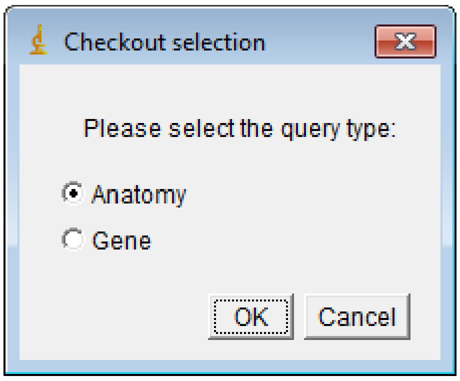


Figure 16: Query type selection window.

Then, when the user selects which query type is wanted, the second window pops up, requiring the user to select a gene or a structure of interest:

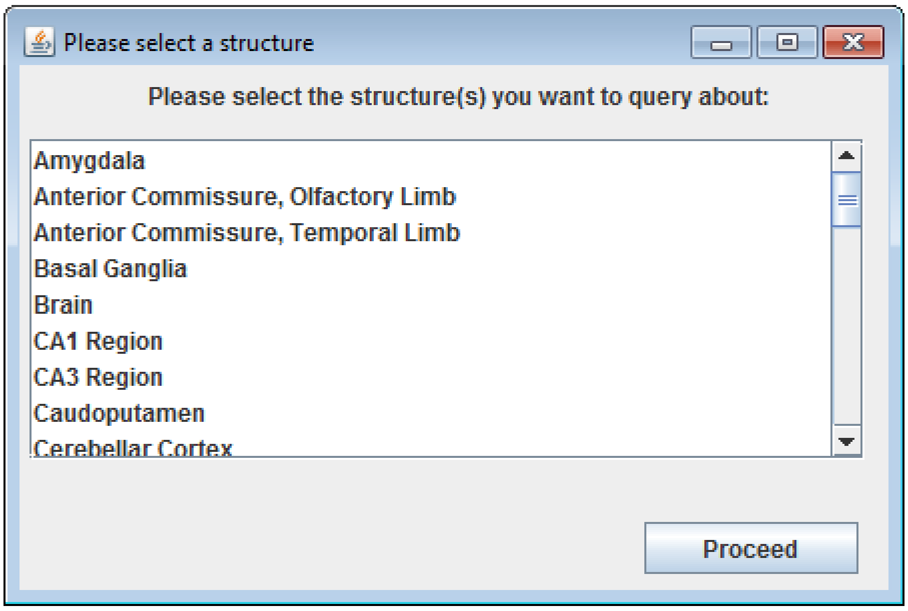

Please select a structure
Please select the structure(s) you want to query about:
Amygdala
Anterior Commissure, Olfactory Limb
Anterior Commissure, Temporal Limb
Basal Ganglia
Brain
CA1 Region
CA3 Region
Caudoputamen
Cerebellar Cortex
Proceed

Figure 17: Structure selection window. The gene selection window is similar to this one.

After all the choices are made, then the program should process the images in order to show the user a visual result. The image containing the anatomical data is shown, and then, the information regarding the genes and structures queried should be popped up in another window. The window contains the information on the gene name, the name of the structure where it is expressed, and the strength.

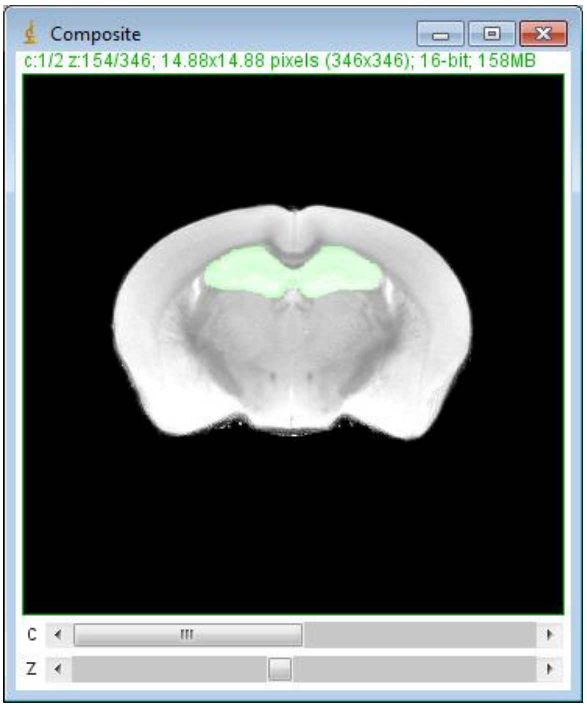
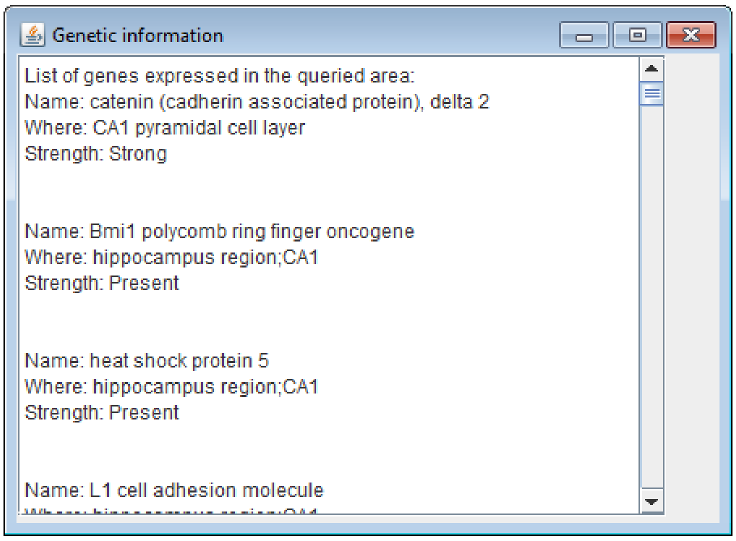

Composite
c:1/2 z:154/346; 14.88x14.88 pixels (346x346); 16-bit; 158MB
c
z

Figure 18: The result image. It can be appreciated in green the queried structure.



```
List of genes expressed in the queried area:
Name: catenin (cadherin associated protein), delta 2
Where: CA1 pyramidal cell layer
Strength: Strong


Name: Bmi1 polycomb ring finger oncogene
Where: hippocampus region;CA1
Strength: Present


Name: heat shock protein 5
Where: hippocampus region;CA1
Strength: Present


Name: L1 cell adhesion molecule
```

Figure 19: Gene information window. It contains information for the genes present in the queried region, or for the genes queried in the case of a gene query.

Discussion and conclusions

As demonstrated, a functional plugin for ImageJ was developed, which is able to merge the information provided in one of the tables of aGEM with one atlas. It is a tool which can be used by any ImageJ user and it is open-source, which means that anyone can place modifications and improvements. It allows the projected queries (anatomical and gene queries) to be performed in a clear and user-friendly way.

The results of this project show that the programming of such a tool integrating data from genes and from anatomical atlas is feasible, and can be done by simple procedures. Although it is a initial version which has a lot of features to be further developed, it can be a good starting point for future developments.

To extend further this tool, it could be a good idea to change the format of the aGEM databases from MySQL to SQLite format. This could not be done due to difficulties when converting the tables from one format to another one. Also, it might be interesting to save the information in an easier accessible way, so that the only time it needs to go to the database or to the .xml or .xls files would be the first one. This would make the program much quicker, which is one of the main problems of the current implementation. A complete execution of a simple anatomical query might last for more than a minute, while the execution of gene queries is much longer. Furthermore, in this way, the plugin would work properly even though the required files of the databases, atlas and mapping are moved or deleted.

Future lines

The potential of the tool is quite high; however, it admits further improvement and added functionalities which would make it much more interesting than the beta version that was proposed by this project. Some of these possible improvements are the addition of maps for different atlases, in order to see the structures in different levels of detail, because different atlases might show more accuracy in different areas of the brain, or they might show more tags which indeed increase the resolution of the queries. As far as this possibility is concerned, the tool is implemented in a way that it allows to add up new atlases in an easy way.

One of the main issues which should be covered in further versions is the use of data from other databases contained in aGEM, different from GXD. aGEM is, as said, a novel idea which is really useful, since it allows the user to not having to query each one of the databases one by one, but to query them at the same time. The tool developed in the project was not included at first by aGEM, due to the fact that the developers did not have the knowledge in medical imaging processing which was required to build up the tool. Nevertheless, in this initial version, it does not take advantage of all the potential of aGEM. In future developments, it would be necessary to include more data from aGEM in order to increase the interest of the project.

Also, it would be quite interesting to include more analysis tools apart from a simple list of expressed genes. It would be really interesting to include as a query type or as a result the statistical data of expression of the different genes. In such way, patterns of expression might be identified, or the correlation among several genes. The result of such an statistical analysis could be thresholded, discarding the structures which are not relevant or do not correlate properly with the statistical parameter of interest and taking only those which provide significative results. Another possibility could be multi-gene query, which would allow the user to identify related structures in which the same set of genes is expressed.

Another improvement which the tool requires is to be made quicker, since it is now quite inefficient in some of its processes. The menus for the queries could be modified to allow multiple selection, in order to query multiple structures (which should be represented in different colours) and also multiple genes.

References

aGEM. (2014). Retrieved from http://agem.cnb.csic.es/VisualOmics/aGEM/

Asian Radiology. (2014). Retrieved from http://asian.radiology.web.id/wp-content/uploads/2013/03/image41.png http://asian.radiology.web.id/wp-content/uploads/2013/03/image41.png

De Boer, B. a., Ruijter, J. M., Voorbraak, F. P. J. M., & Moorman, A. F. M. (2009). More than a decade of developmental gene expression atlases: Where are we now? *Nucleic Acids Research*, *37*(22), 7349–7359. doi:10.1093/nar/gkp819

Diez-Roux, G., Banfi, S., Sultan, M., Geffers, L., Anand, S., Rozado, D., … Ballabio, A. (2011). A high-resolution anatomical atlas of the transcriptome in the mouse embryo. *PLoS Biology*, *9*(1), e1000582. doi:10.1371/journal.pbio.1000582

Dorr, a E., Lerch, J. P., Spring, S., Kabani, N., & Henkelman, R. M. (2008). High resolution three-dimensional brain atlas using an average magnetic resonance image of 40 adult C57Bl/6J mice. *NeuroImage*, *42*(1), 60–9. doi:10.1016/j.neuroimage.2008.03.037

Finger, J. H., Smith, C. M., Hayamizu, T. F., McCright, I. J., Eppig, J. T., Kadin, J. a., … Ringwald, M. (2011). The mouse Gene Expression Database (GXD): 2011 update. *Nucleic Acids Research*, *39*(SUPPL. 1), D835–41. doi:10.1093/nar/gkq1132

Image Processing with ImageJ: José María Mateos Perez, Javier Pascau: 9781783283958: Amazon.com: Books. (n.d.). Retrieved June 18, 2014, from http://www.amazon.com/Image-Processing-ImageJ-María-Mateos/dp/1783283955

Jiménez-Lozano, N., Segura, J., Macías, J. R., Vega, J., & Carazo, J. M. (2009). aGEM: An integrative system for analyzing spatial-temporal gene-expression information. *Bioinformatics*, *25*(19), 2566–2572. doi:10.1093/bioinformatics/btp422

Jiménez-Lozano, N., Segura, J., Macías, J. R., Vega, J., & Carazo, J. M. (2012). Integrating human and murine anatomical gene expression data for improved comparisons. *Bioinformatics (Oxford, England)*, *28*(3), 397–402. doi:10.1093/bioinformatics/btr639

Johnson, A. G., Badea, A., Burstein, P., Nissanov, J., Gustafson, C., Brandenberg, J., & Liu, S. (2009). Waxholm Space: Target Volumes for a Standard Coordinate System for the Mouse Brain. *Frontiers in Neuroinformatics*. doi:10.3389/conf.neuro.11.2009.08.004

Life Extension Magazine (LEF). (2014). Retrieved from http://www.lef.org/magazine/mag2012/images/jul2012_Value-Of-PET_03.jpg

Ma, Y., Hof, P. R., Grant, S. C., Blackband, S. J., Bennett, R., Slatest, L., … Benveniste, H. (2005). A three-dimensional digital atlas database of the adult C57BL/6J mouse brain by magnetic resonance microscopy. *Neuroscience*, *135*(4), 1203–15. doi:10.1016/j.neuroscience.2005.07.014

Ma, Y., Smith, D., Hof, P. R., Foerster, B., Hamilton, S., Blackband, S. J., … Benveniste, H. (2008). In Vivo 3D Digital Atlas Database of the Adult C57BL/6J Mouse Brain by Magnetic Resonance Microscopy. *Frontiers in Neuroanatomy*, *2*(April), 1. doi:10.3389/neuro.05.001.2008

MacKenzie-Graham, A., Lee, E.-F., Dinov, I. D., Bota, M., Shattuck, D. W., Ruffins, S., … Toga, A. W. (2004). A multimodal, multidimensional atlas of the C57BL/6J mouse brain. *Journal of Anatomy*, *204*(2), 93–102. doi:10.1111/j.1469-7580.2004.00264.x

MacKenzie-Graham, A., Tinsley, M. R., Shah, K. P., Aguilar, C., Strickland, L. V., Boline, J., … Toga, A. W. (2006). Cerebellar cortical atrophy in experimental autoimmune encephalomyelitis. *NeuroImage*, *32*, 1016–1023. doi:10.1016/j.neuroimage.2006.05.006

Maintz, J. B. A., & Viergever, M. A. (1996). An Overview of Medical Image Registration Methods. *Nature*, *12*, 1–22. doi:10.1.1.39.4417

Menzel, R. (2011). Ultramicroscopy - imaging a whole animal or a whole brain with micron resolution. *Frontiers in Neuroscience*, *5*, 11. doi:10.3389/fnins.2011.00011

Parkinson, H., Sarkans, U., Shojatalab, M., Abeygunawardena, N., Contrino, S., Coulson, R., … Brazma, a. (2005). ArrayExpress--a public repository for microarray gene expression data at the EBI. *Nucleic Acids Research*, *33*(Database issue), D553–D555. doi:10.1093/nar/gki056

Porras Rodriguez, C. (2014). *TheProject, GitHub repository*. Retrieved from https://github.com/cpr16992/ThePlugin

Richardson, L., Venkataraman, S., Stevenson, P., Yang, Y., Moss, J., Graham, L., … Armit, C. (2009). EMAGE mouse embryo spatial gene expression database: 2014 update. *Nucleic Acids Research*, *42*(D1), D703–9. doi:10.1093/nar/gkp763

Sunkin, S. M., Ng, L., Lau, C., Dolbeare, T., Gilbert, T. L., Thompson, C. L., … Dang, C. (2013). Allen Brain Atlas: An integrated spatio-temporal portal for exploring the central nervous system. *Nucleic Acids Research*, *41*(D1), D996–D1008. doi:10.1093/nar/gks1042

V. Hajnal, J., & L.G. Hill, D. (2001). *Medical Image Registration*. *CRC Press Book*. Retrieved June 21, 2014, from http://www.crcpress.com/product/isbn/9780849300646

Wheeler, D. L., Church, D. M., Federhen, S., Lash, A. E., Madden, T. L., Pontius, J. U., … Wagner, L. (2003). Database resources of the national center for biotechnology. *Nucleic Acids Research*, *31*(1), 28–33. doi:10.1093/nar/gkq1172

Wu, C., Orozco, C., Boyer, J., Leglise, M., Goodale, J., Batalov, S., … Su, A. I. (2009). BioGPS: an extensible and customizable portal for querying and organizing gene annotation resources. *Genome Biology*, *10*(11), R130. doi:10.1186/gb-2009-10-11-r130