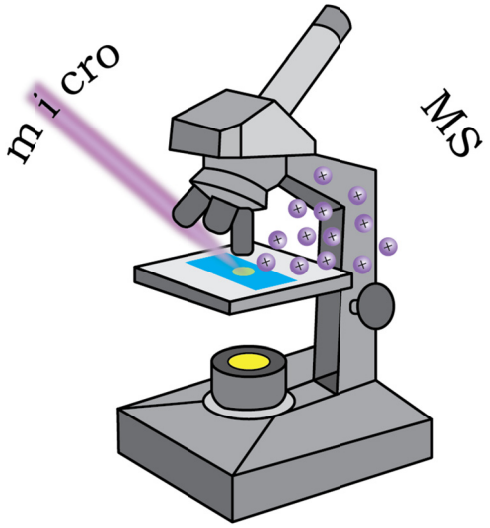
microMS User Guide



Software from the Sweedler research group at the University of Illinois, written by Troy Comi with support from the National Institutes of Health, Award Number P30 DA018310 from the National Institute on Drug Abuse, and from the National Institute of Mental Health, Award Number 1U01 MH109062; the National Science Foundation Graduate Research Fellowship Program; and the Training Program at Chemistry-Interface with Biology (T32 GM070421).

Contents

Introduction

microMS is a feature-rich GUI for performing basic image analysis and correlation of image positions into physical coordinate spaces. The overall goal of this package is to image and locate a field of cells or other objects dispersed across a microscope slide, allow subpopulations to be selected based on flexible and user defined criteria, convert the cell / object locations to a platform dependent set of positions to be used for follow-up assays such as selected cell collections or mass spectrometry profiling. While developed for single cell analysis by mass spectrometry, with few modifications, the underlying code is versatile enough for a variety of targets, image modalities, and follow-up analytical systems. microMS aims to simplify cell finding, improve coordinate registration, and provide an interface suitable for novice users. Several additional features are added to expand the repertoire of profiling experiments. For more advanced users, the addition of new, off-line instrument platforms and even direct instrument control are possible. This guide will introduce the most common usage of microMS, detail additional features, and provide a starting point for adding new instrument coordinate systems. The operation and use of this code has been described in doi: 10.1007/s13361-017-1704-1. Other applications include high throughput single cell profiling via SIMS (doi: 10.1021/acs.analchem.6b04819) and MALDI (doi: 10.1021/acschembio.6b00602).

If you use microMS in a scientific publication, you are welcome to link to http://neuroproteomics.scs.illinois.edu/microMS.htm, but please include the following citation in your article: Comi TJ; Neumann EK; Do TD; Sweedler JV. microMS: A Python Platform for Image-guided Mass Spectrometry Profiling, JASMS 2017, in press. DOI: 10.1007/s13361-017-1704-1

Installation and Startup Instructions

Refer to http://neuroproteomics.scs.illinois.edu/microMS.htm for the most recent instructions for installation and startup.

Windows Installation and Execution

Most dependencies of microMS are included in standard distribution packages of python 3. We have had success using anaconda which also includes an IDE (https://www.continuum.io/downloads). A

python 3.X (X >= 5) version is required to be installed. After installation, pyserial and openslide require separate installations. Pyserial is installed by opening Windows PowerShell and entering `pip install pyserial`. This should automatically install the most recent version. Openslide requires additional binaries, located here: http://openslide.org/download/ under Windows Binaries. Download and extract the folders, placing them in your documents or program files. The contained bin folder also must be added to the operating system path (e.g., C:\Program Files\Openslide\bin\): http://www.howtogeek.com/118594/how-to-edit-your-system-path-for-easy-command-line-access/ Once added correctly, the openslide python wrapper is simply installed by entering `pip install openslide-python` in PowerShell.

With these dependencies installed, microMS is installed by downloading and extracting the ZIP file from http://neuroproteomics.scs.illinois.edu/microMS.htm. The main script is run with `python microMS.py` in a command prompt in the script directory or by double clicking the microMS.bat file which runs the above command and waits for a key press.

Linux Installation and Execution

With an installation of python 3, the required packages are installed in the terminal with `pip3 install <package>`. Openslide is installed with:

```
apt-get install python-openslide
pip3 install openslide-python
```

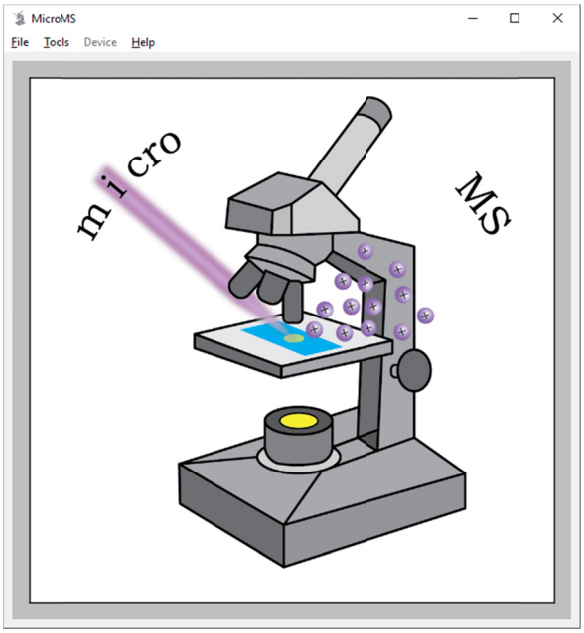
After unzipping the microMS file, the main method is run with either `python microMS.py` or `./microMS.py` after executable permission is granted (i.e. `chmod +x microMS.py`). Adding the microMS directory to the PATH variable will allow execution from any directory.

Image File Types

microMS utilizes openslide to read sections of whole slide images. Any openslide-supported format should be a suitable input, though only Hamamatsu ndpi and bigTiff images have been thoroughly tested and are accepted by default. For multichannel images, ndpi files should end with 'Brightfield' (for channel 1, brightfield) and 'Triple' (for channel 2, RGB fluorescence). Tiff images should *not* start with '8x' or '64x'. Multichannel tiff images should be indicated with the suffix 'c#.tif' where '#' is a digit between 1 and 9. By default, tiff images can only be zoomed out by 4x. Tiff images may be decimated to 8x and 64x, generating the files '8x<image name>' and '64x<image name>' with <image name> being the original file name.

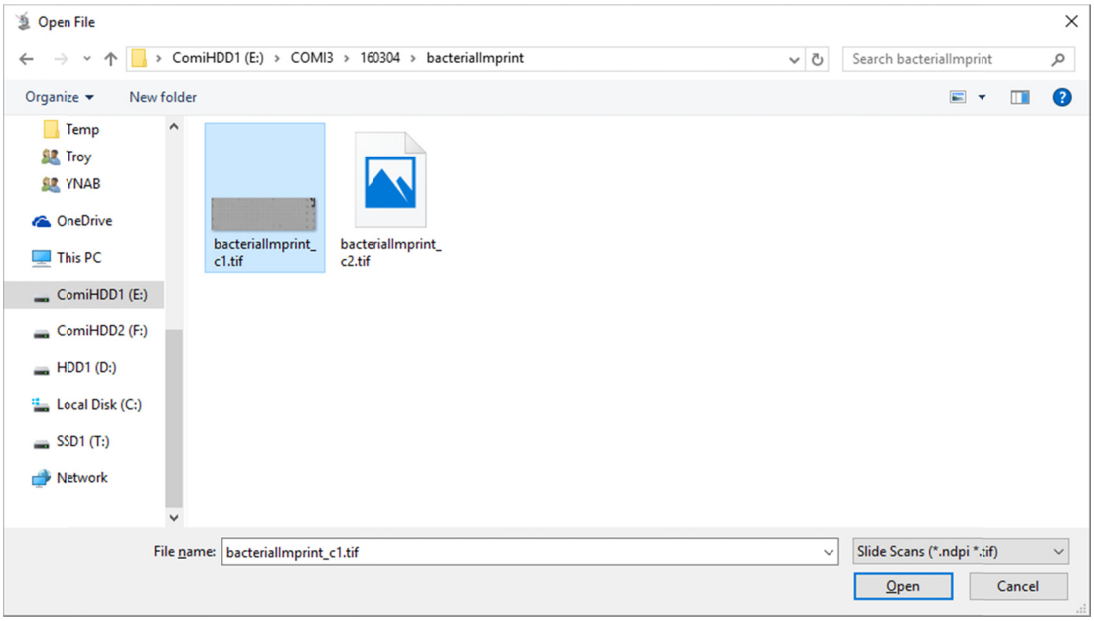
Opening an Image

On startup, the main GUI window is displayed with the program icon:



An image is opened by selecting *Open* in the *File* menu (or Ctrl + O). This will launch a file selection dialog which prompts the user for the desired file which will subsequently be loaded. Selecting any

image of a multichannel-data image set will open all channels with matching filenames as described above.



Multiple image channels (if they exist) are loaded at the same time and overlaid on top of each other:

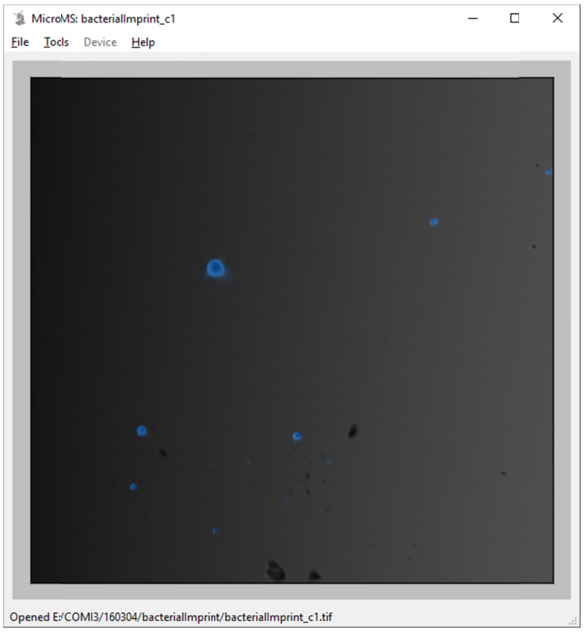
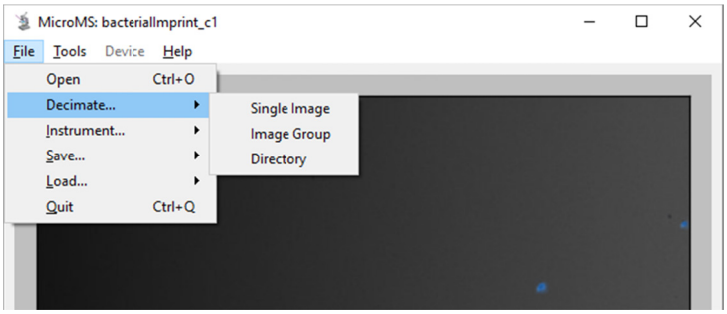


Image Decimation

Non-pyramidal tiff images will by default only be able to display at up to 1/4x. To enable further zoom levels from 1x to 1/256x (which is recommended), tiff image sets must be decimated by selecting the *Decimate* option in the *File* menu. Several options are available:



Both 'Single Image' and 'Image Group' options are for the selection of one tiff image. When an image of a multichannel-data image set is selected with the 'Image Group' option, all images of the set will be decimated. The 'Directory' option allows the selection of a single directory. All images cotained in sub-directories of the selected directory be decimated as shown schematically below:

Single Image

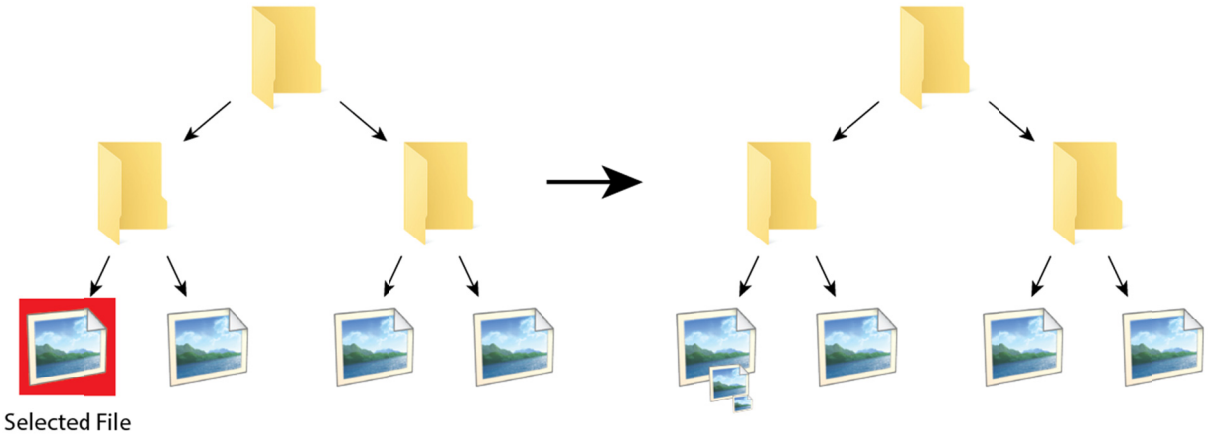
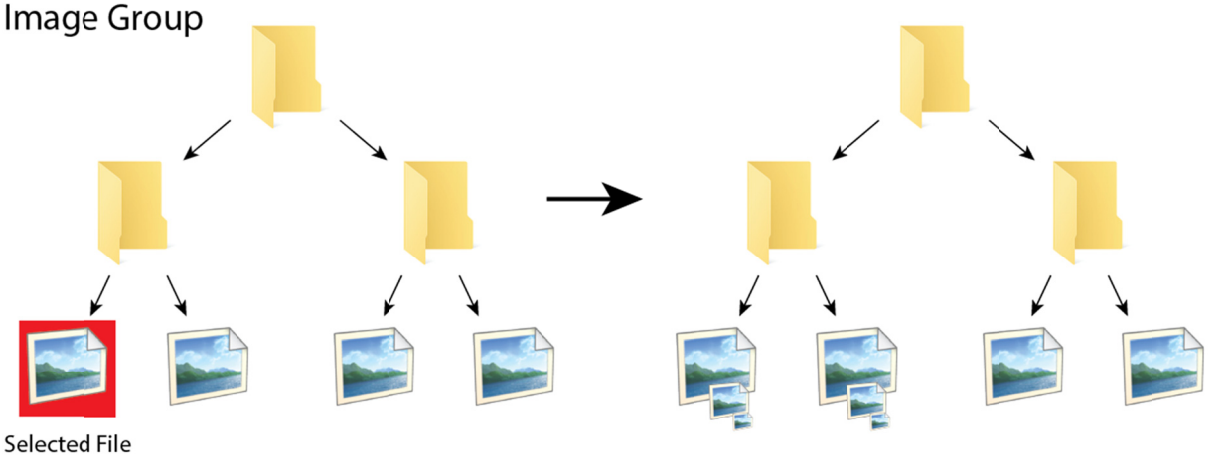
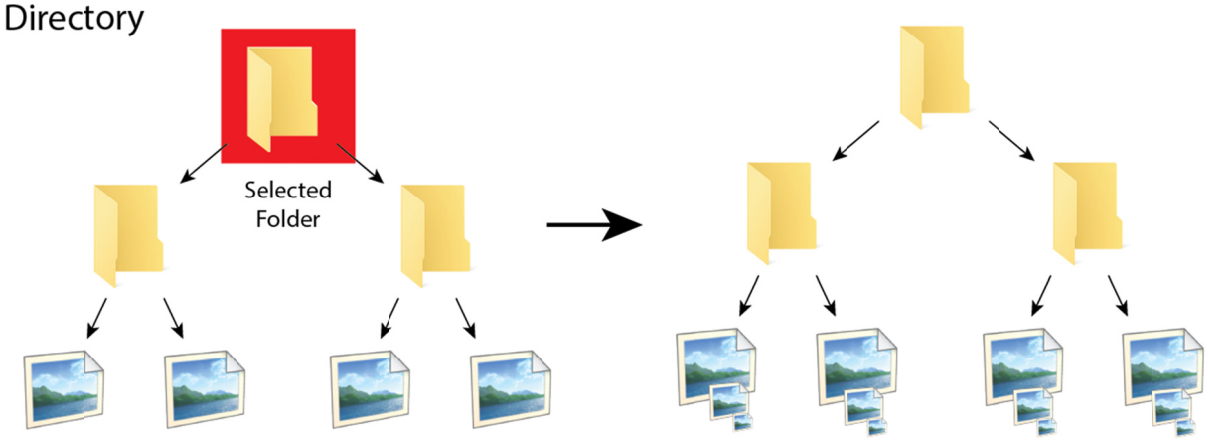


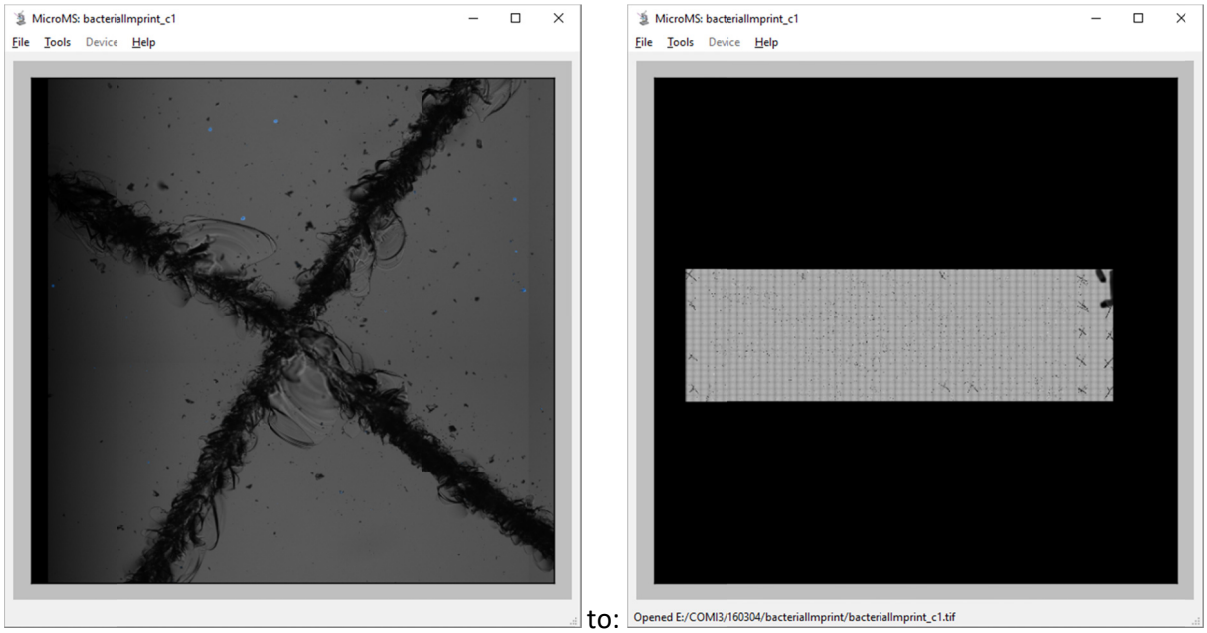
Image Group



Directory



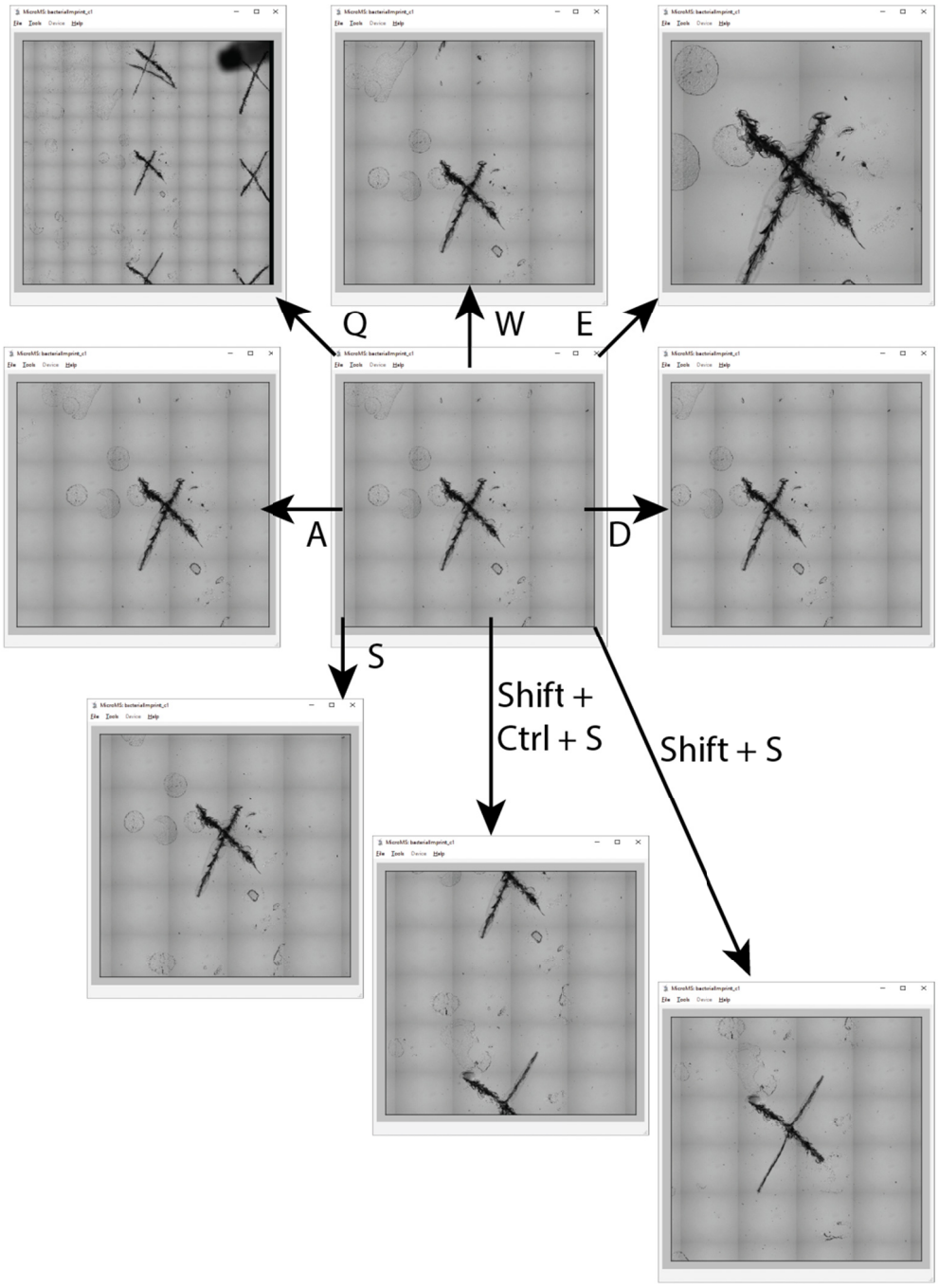
Following decimation of a single image or a group of multichannel images, the image(s) will be automatically opened. The max zoom level increases from:



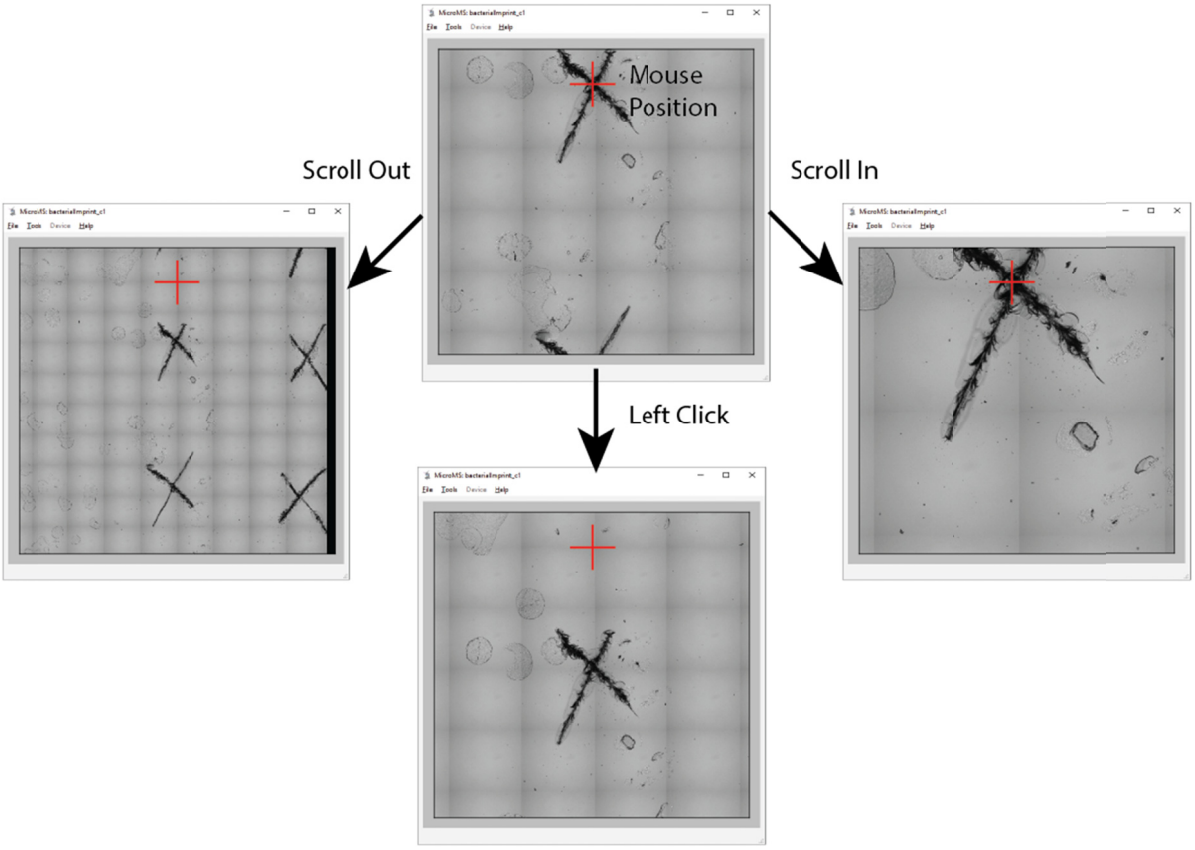
Notice that since a single image (brightfield) was decimated, only the specified channel is displayed at higher zoom levels. After decimation is complete, the process does not need to be repeated the next time the image set is opened. Selecting the non-decimated image will automatically load the full image stack.

Image Navigation

Images may be navigated with a combination of keyboard and mouse controls. With keyboard movements, the keys W, A, S, D move the view frame up, left, down, and right, respectively. Each step moves 1/10 of the frame size. Combining each key with *Ctrl*+*Shift* moves 1/2 the frame, and *Shift* moves a full frame. The keys Q and E zoom out and in:


Q
W
E
A
D
S
Shift +
Ctrl + S
Shift + S

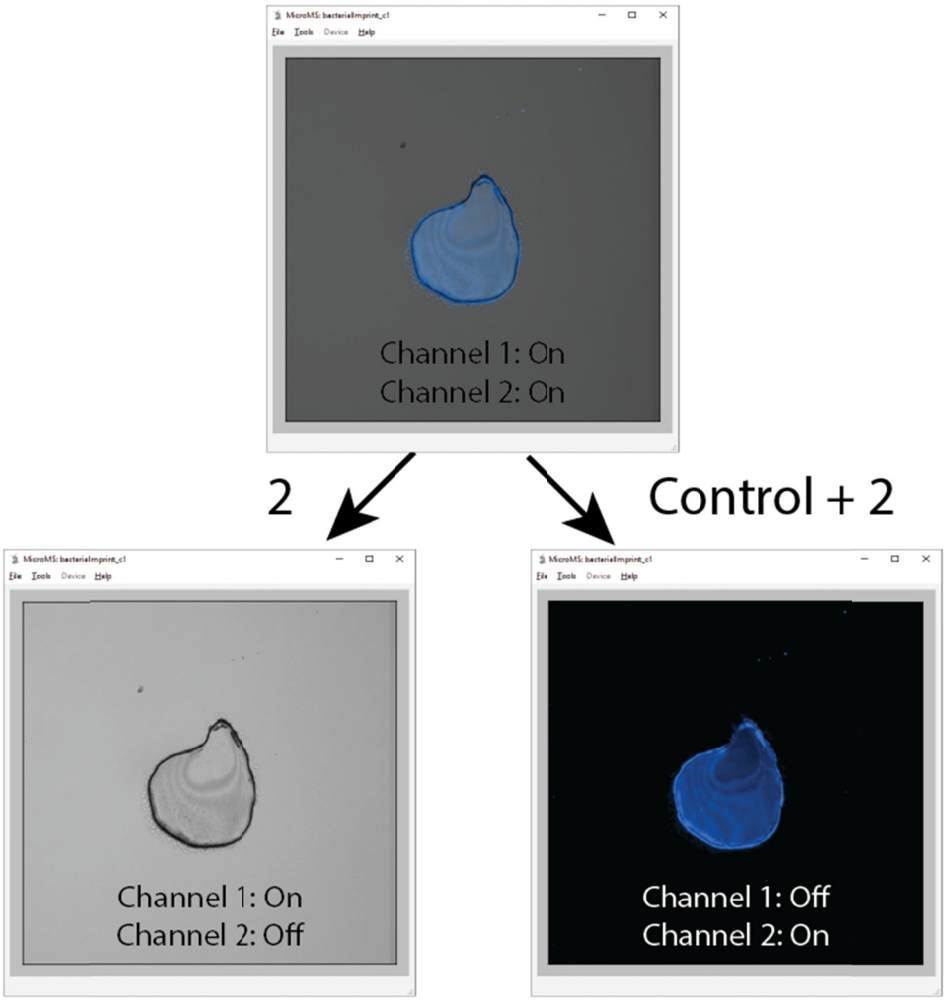
With the mouse, the scroll wheel is used to zoom in (wheel up) and out (wheel down). When zooming in, the center of the frame also moves to keep the mouse location in the same position. Left clicking a position moves that spot to the center of the frame:



The key R will reset the field of view to the top left corner at full zoom. This operation is useful if the field of view moves too far from the sample area. Moving around an image data set should be smooth, but due to the size of images, some lag between input and display may occur. Also note that images are read from disk (not stored into memory/RAM). This design choice was made to allow microMS to run on a variety of systems without requiring large amounts of memory. However, reading images on a network drive or external hard drive (USB 2) is very slow!

Switching Image Channels

Displaying different image channels is controlled through the numeric keys and is cycled with the T or Z key. Each image is enumerated based on its name. For ndpi files, “Brightfield” is number 1 and “Triple” is number 2. With multichannel tiff images, the number in “c#.tif” is parsed and assigned to the corresponding image number. Pressing the number of an image will toggle that corresponding channel on and off. The combination of *Crtl* + ‘image number’ will turn all channels off except for the selected channel. It is possible to turn off all image channels, which will display a black background.



Up to 9 different channels may be utilized in one experiment, but only one color (red, green or blue) is taken from each fluorescent channel during image overlay. The color channel with the highest intensity is utilized as the "color" for that channel. The brightfield image must be channel one. Since each image is read from disk, displaying more channels will slow the response time proportionally.

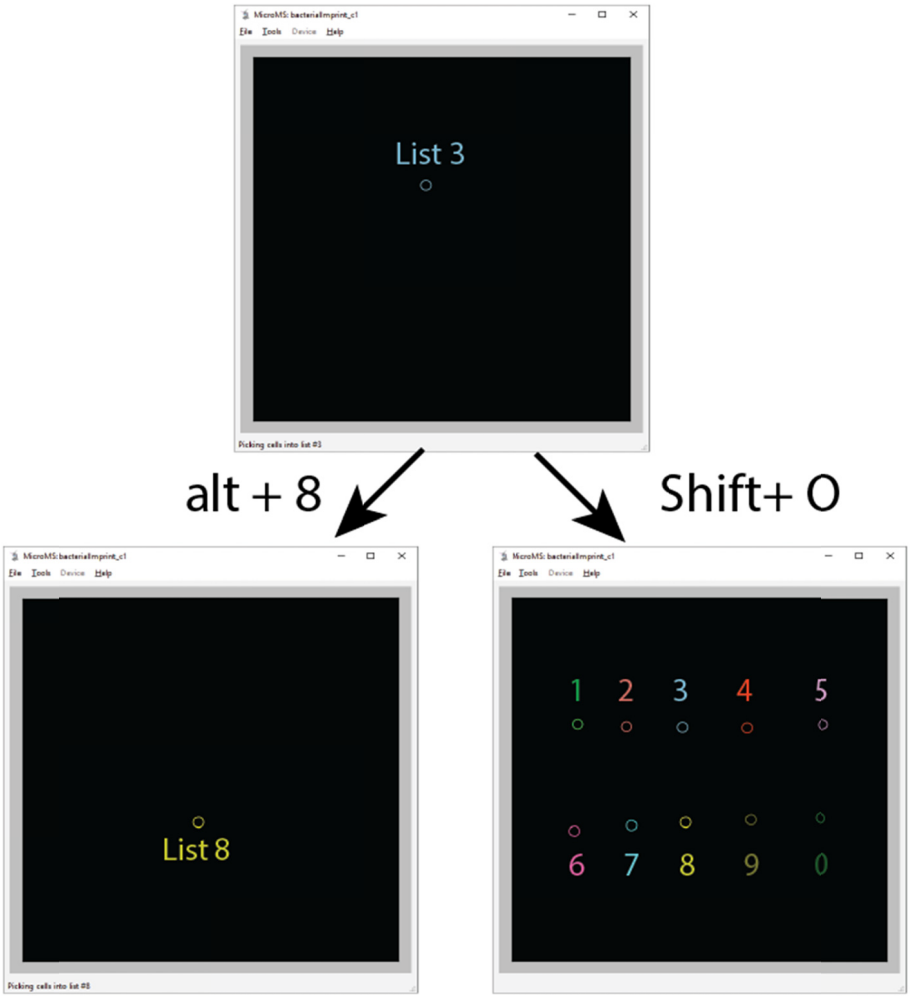
Blobs

All points of interest in microMS are represented as objects called blobs. Each blob consists of an x and y coordinate, in pixels, an effective radius and the object's circularity. The blob area (used to determine size and circularity) is taken as the number of pixels above the fluorescence threshold. A blob's effective radius is then $\sqrt{\frac{area}{\pi}}$ and it's circularity is $\frac{4\pi \times area}{perimeter} \in [0,1]$. Due to the calculation of perimeter with pixels, the circularity is generally larger than would be expected for a non-pixelated object. The x and y position is the center of mass for the Boolean image of pixels above the specified threshold. Note that this does not account for the intensity of the underlying image. When blobs are added manually, the resulting circularity is always 1. Blobs may optionally have a group assigned to them as part of packing routines explained below.

Collections of blobs are stored in lists. Up to 10 different blob lists can be utilized in one image set. Each list is treated as an independent set of targets and by default only one list is displayed at a time. Lists spawn new child lists during packing and filtering procedures. Generally, the first empty list (in increasing order) will be filled with automatically generated target sets. Loading found cells or instrument positions will populate the currently selected list, possibly overwriting its contents.

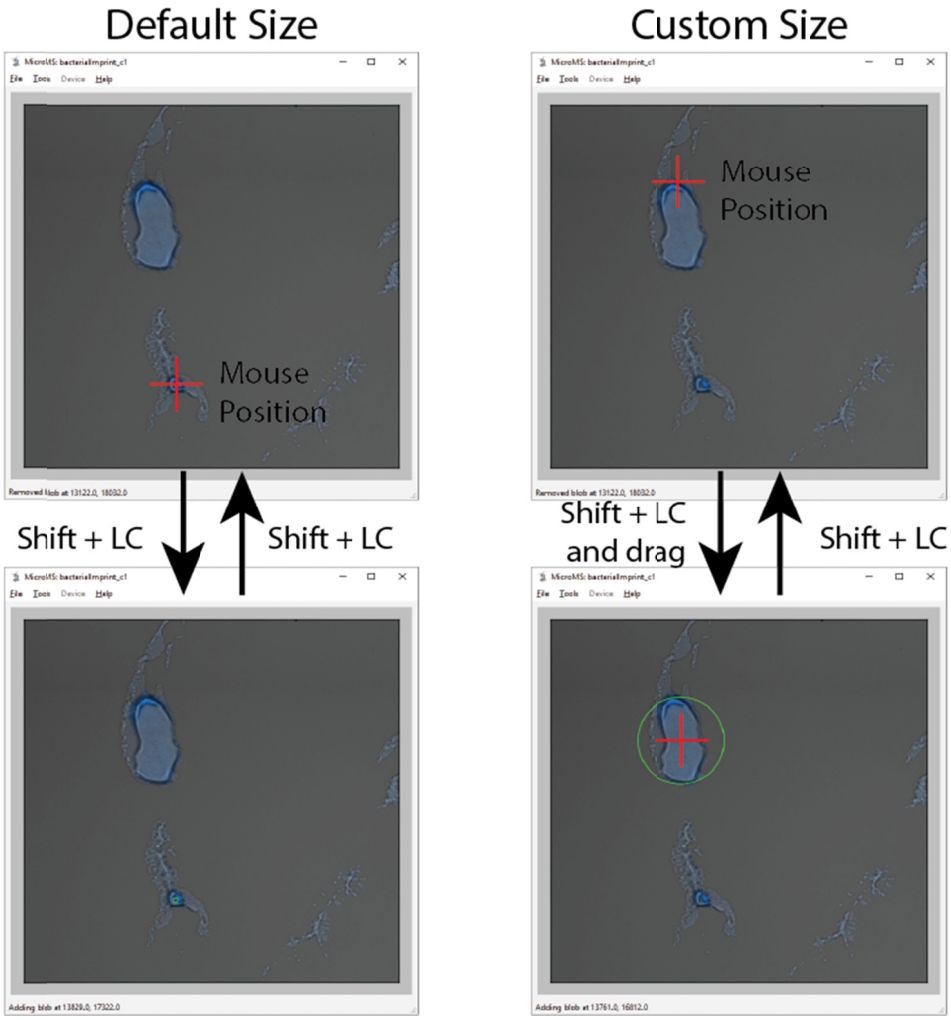
Displaying target collections

Each list of blobs is designated with a unique color. The currently selected list is displayed and is changed by the combination of *Alt* + 'the list number'. By default, only the currently selected list will be displayed. All subsequent discussions regarding changes to the blob list will only affect the current list. It is occasionally useful to see multiple lists simultaneously. Drawing all blob lists is toggled with *Shift* + O. Blobs which overlap will show a blended color, though generally it is difficult to see which combination is overlaid.



Manual addition of targets

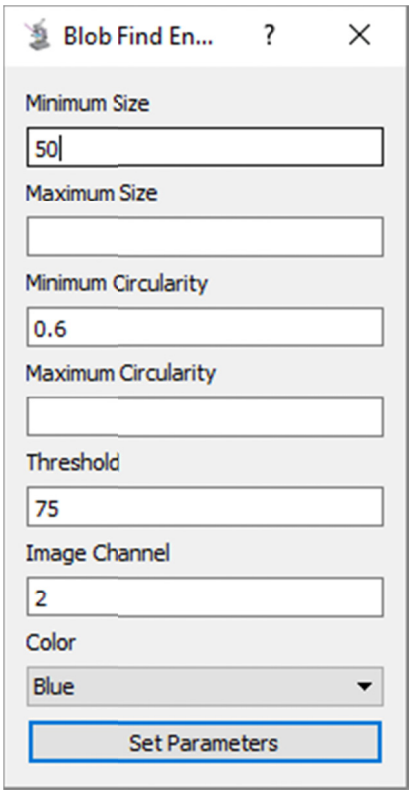
New blobs are added to the current list with a left mouse button click with *Shift* held. The default, minimum radius of a manually added blob is 10 pixels. Larger blobs can also be added by performing a left click and drag. The circumference is set when the mouse button is depressed and the center upon its release. During the dragging, all other features will disappear and the resulting blob will be dynamically drawn. Blobs are removed by holding *Shift* and left mouse click anywhere inside of them. Releasing a custom drawn blob inside of an existing blob will remove the existing blob without drawing the custom blob.



Automatic blob finding

Input blob finding parameters

Automatic blob finding is at the core of performing high throughput analysis. Currently, blobs are found with a threshold and group algorithm, which is sufficient to locate bright objects which have high contrast and low background. Briefly, the intensity of each pixel in the image is compared with the supplied intensity threshold value. Adjacent pixels with intensities above the threshold are part of the same, putative blob. Next, the number of pixels in a putative blob (size) and its shape (circularity) are examined. If these characteristics are within the supplied ranges the collection of pixels is considered a blob.



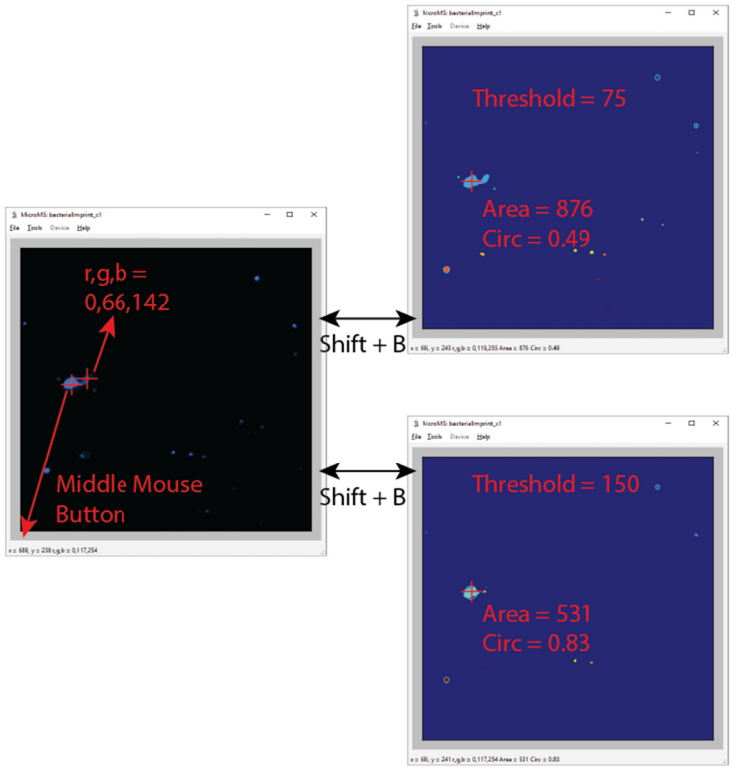
The parameters for cell finding must be chosen carefully to ensure only targets of interest are selected, though filtering after blob finding is possible as discussed later. The parameters for blob finding are shown to the left and are accessed by selecting *Blob Options* under the *Tools* menu (or *Ctrl* + B). The size corresponds to the number of adjacent pixels which are above the intensity threshold. Circularity is defined above; more circular objects will have circularity closer to 1. Threshold is the intensity threshold to discriminate objects from background. Image channel is the image within the dataset to consider for cell finding, with image channel 1 corresponding to brightfield. Finally, color is the RGB channel to extract for performing thresholding. The screenshot above is of an analysis of blue fluorescence in the image channel 2.

Size, threshold and channel should be integers, circularity can be floating point. Maximum size and circularity may be left blank to indicate there is no upper boundary to these values. Any invalid inputs

will be reverted to the last valid input on clicking "*Set Parameters"*. In addition to setting the cell finding parameters, "*Set Parameters*" performs an initial cell finding of the current position at the maximum zoom level. If the current field of view is zoomed out, it will be automatically set to the max zoom level. Any found objects will be highlighted with a turquoise circle. Performing any other action will clear these found blobs. Test blob finding can also be performed by pressing B.

Pixel Information and Threshold View

An easy way to determine suitable blob finding parameters, or identify why some blobs are excluded, is by examining the Pixel Information and utilizing Threshold View. At any time, clicking the middle mouse button will display information about the current pixel. This includes the pixel position relative to the maximum zoom image and the RGB values of the displayed image. The RGB values help establish a suitable threshold for positive pixels. *Shift* + B toggles the threshold view for the current image set. The current threshold and color channel are utilized to visualize blobs that pass the threshold. The background is displayed in a dark blue color. Pixels passing the threshold are then grouped and displayed as a unique color. Changing the blob parameters updates the threshold view and shows the currently found cells. Additionally, while in threshold view, performing a middle mouse button click on a blob will also provide its area and circularity in the status bar. This quickly provides feedback on why some blobs are ignored and help with selecting blob finding parameters.

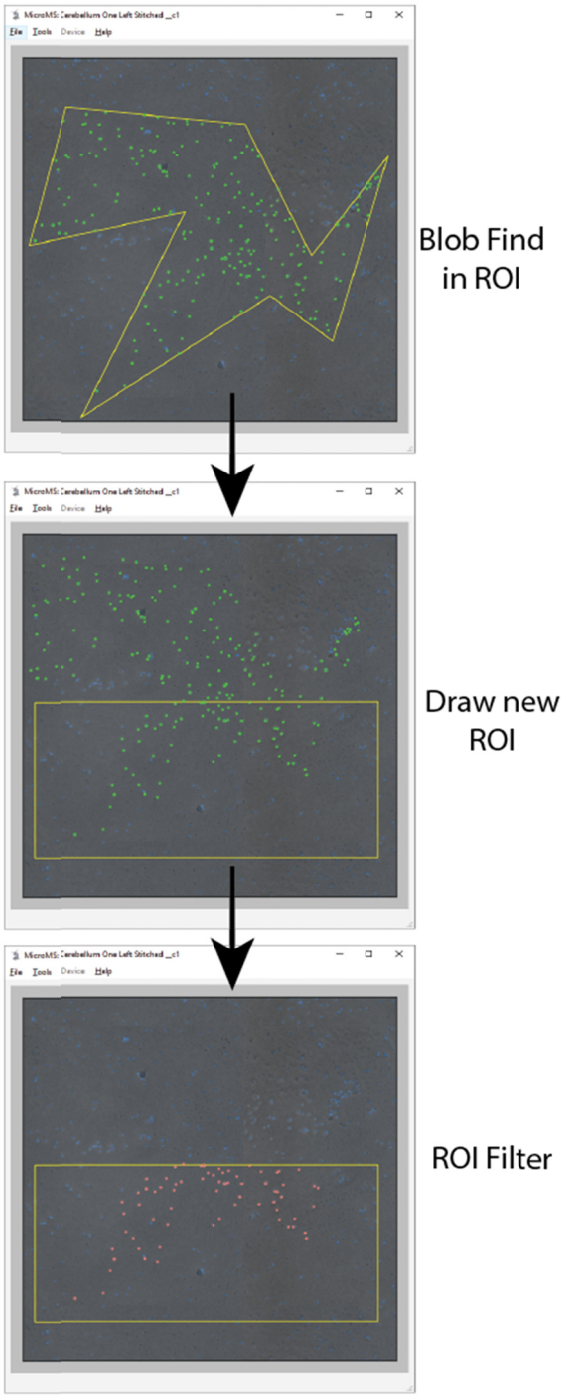


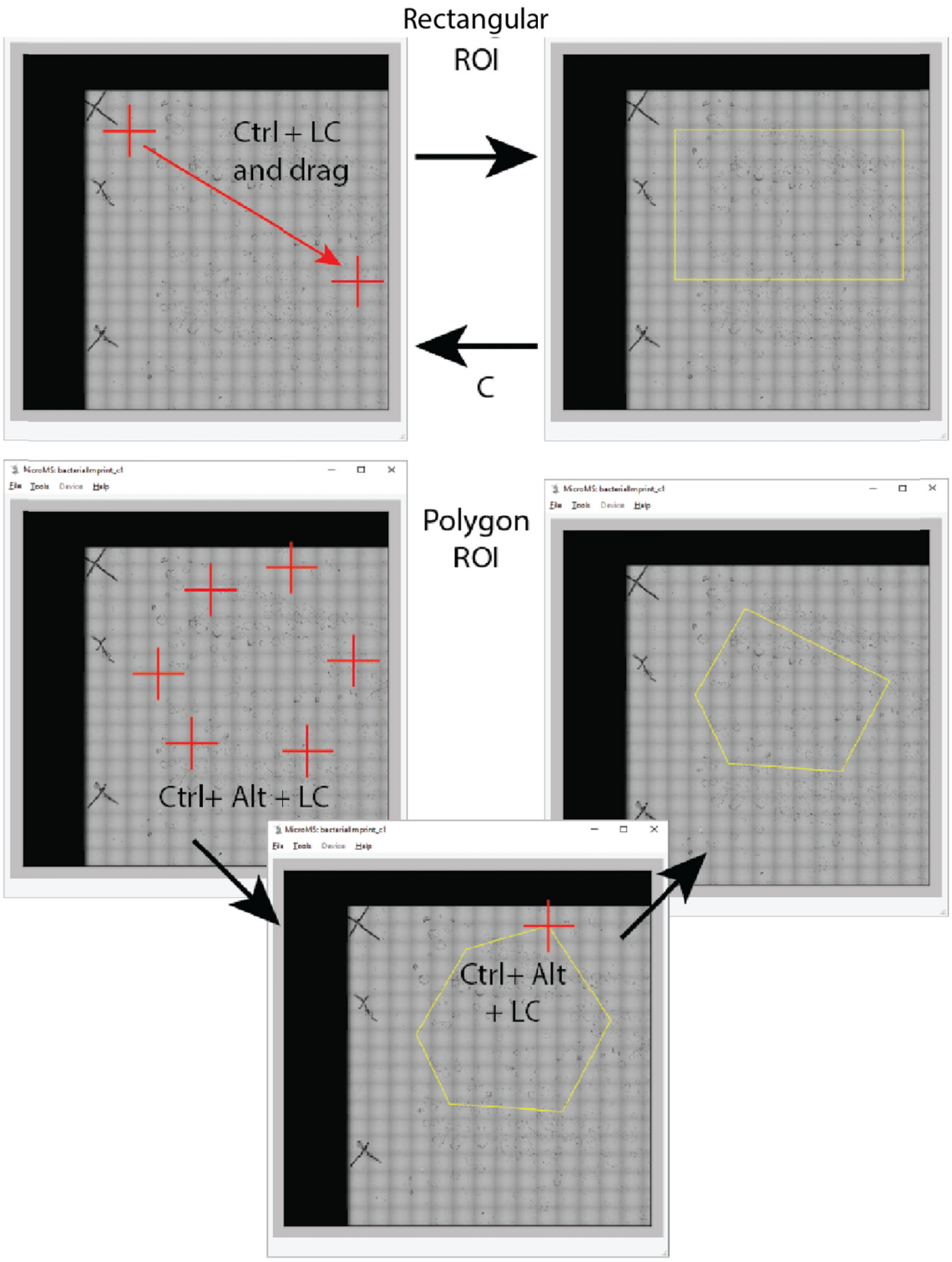
As seen above, at a threshold of 75, the selected blob is not included in found blobs because its circularity is below the set point of 0.6. Increasing the threshold removes the right feature of the blob and its circularity increases to 0.83, hence passing the circularity threshold.

Regions of Interest

Once blob finding parameters are chosen, automatic blob finding is performed by selecting the Blob *Find* option under the *Tools* menu. By default, this will perform blob finding on the entire slide. Note that this operation can take several minutes for large areas. Upon completion, the user is prompted for a base filename which is used to write a <BASE>.txt cell finding file and <BASE>.msreg file in the image file directory. These will be described in further detail below. All found blobs will be stored in the current cell finding list. By default, only a random set of 150 blobs will be drawn in a field of view. This significantly speeds up drawing speed when moving around a slide. This option may be toggled by the *Limit Drawn Blobs* option under the *Tools* menu.

Since multiple sample populations may be present in one image and fiducials should be excluded, it is frequently useful to draw an ROI to restrict blob finding. An ROI may be set up prior to blob finding or ROI filtering can be performed on an existing blob list. Blob finding is restricted to an ROI, so it is faster to perform blob finding when the fiducials and exterior areas are excluded. ROIs are drawn as either rectangles or polygons. To draw a rectangle, hold *Ctrl* and left click and drag between two diagonal vertices of the rectangle. As the mouse moves, the ROI will update to show how the ROI would look if the mouse button was released. To reduce latency, only the slide image is displayed during this process (no blobs are shown). Once drawn, a rectangular ROI will behave identically to a hand-drawn polygon ROI. To interact with ROIs on a vertex level, hold *Ctrl* and *Alt*, and move the mouse around the ROI. Again, the ROI will update as the mouse moves, showing the ROI that would result upon a mouse click. The ROI cannot have overlapping edges and its vertices are removed by clicking on or near them. Alternatively, vertices can be added in order, by holding *Ctrl* and *Shift*. This simplifies drawing complex shapes, but is more difficult to modify. Finally, ROIs are cleared by pressing the C key.



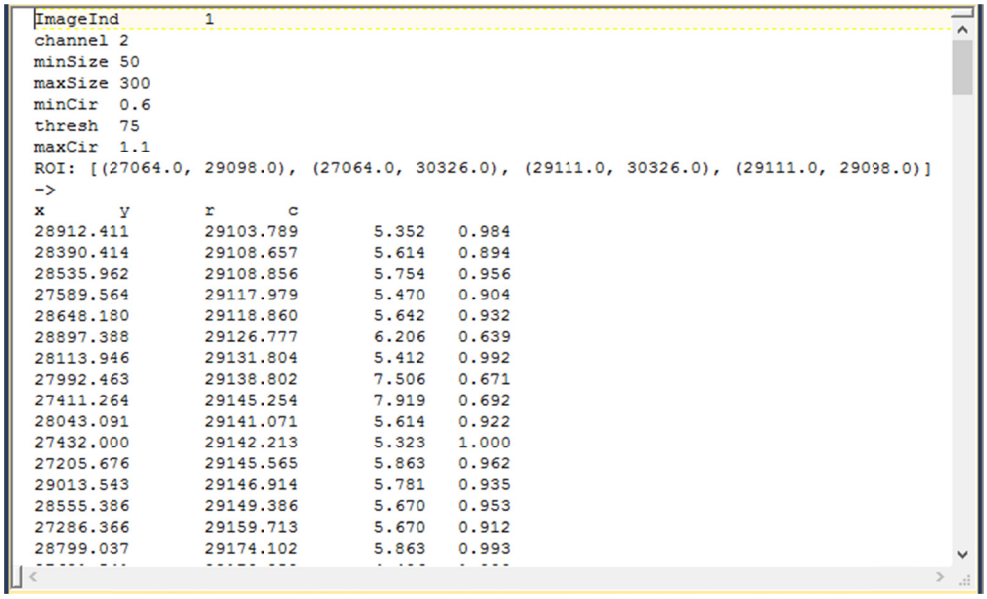


ROI filtering can also be performed at any time after blob finding. Simply draw an ROI for the area to keep blobs, and select either *ROI Filter Retain* or *ROI Filter Remove* under the *Tools* menu. The

former operation removes all blobs outside of the specified ROI and moves the resulting list to the next open blob list. The latter operation performs the opposite filtering, placing all blobs outside the ROI into a new list. The original blob list is retained in its list number in either case.

Saving and loading blob lists

Once blobs have been found automatically or manually placed, their positions and finding parameters can be saved for later examination or for use in other analysis steps. The file is in plain text format and human readable, as shown below:



```
ImageInd        1
channel 2
minSize 50
maxSize 300
minCir  0.6
thresh  75
maxCir  1.1
ROI: [(27064.0, 29098.0), (27064.0, 30326.0), (29111.0, 30326.0), (29111.0, 29098.0)]
->
x       y       r       c
28912.411       29103.789       5.352   0.984
28390.414       29108.657       5.614   0.894
28535.962       29108.856       5.754   0.956
27589.564       29117.979       5.470   0.904
28648.180       29118.860       5.642   0.932
28897.388       29126.777       6.206   0.639
28113.946       29131.804       5.412   0.992
27992.463       29138.802       7.506   0.671
27411.264       29145.254       7.919   0.692
28043.091       29141.071       5.614   0.922
27432.000       29142.213       5.323   1.000
27205.676       29145.565       5.863   0.962
29013.543       29146.914       5.781   0.935
28555.386       29149.386       5.670   0.953
27286.366       29159.713       5.670   0.912
28799.037       29174.102       5.863   0.993
```

The first 7 lines correspond to the blob finding parameters used for automatic cell finding. This will still be present for manually found cells. Also note the 'ImageInd' is zero based (i.e. the first image is 0) and channel is encoded so that [0,1,2] -> [Red, Green, Blue]. Next, the ROI is stored as a list of xy pixel coordinates for the polygon. Since the ROI may change after blob finding, this may not represent the correct ROI used throughout the operation, but rather the most recent ROI. However, after automatic cell finding the current blobs are saved which will include the ROI. Next, the arrow symbol "->" is used to list the "filters" used during the generation of the blob list. Entries include distance filtering and

histogram filtering. These are not used in the software logic but provide a limited record of parameters used for generating a list of blobs. Finally, the x,y coordinate, radius and circularity of each blob are recorded.

There are three options for saving the blob lists: (1) The *Save Current Blobs* option under the *File* menu, (2) *Histogram Divisions*, and (3) *All lists of blobs*. The first option will launch a save file dialog, allowing the user to specify the filename to save the current blob list. This is useful if only the last step of filtering and processing is needed or if only one list was utilized. The second option, histogram divisions, will be covered in more detail later, but this option saves the low and high intensity populations as separate files with encoded filenames. When the user selects a file, it is used as a base file name with additional information added to the end. Note that because no actual filtering was performed, the final divisions are not included in the filters list. Finally, the last option, saving all lists of blobs, provides a quick way to export all blob lists. Again, the user-specified file is utilized as a base file name and each list number is saved as <BASE>_<LIST NUMBER>.txt in the specified directory. The list number is again zero based so that the first list is saved as <BASE>_0.txt.

Previously generated blob list files can be loaded back into microMS. This restores the blob finder parameters from the file, the ROI, the filter list and the collection of blobs. To open a blob file, select the *Load/Found Blobs* under the *File* menu. The contained blobs will be used to populate the currently selected blob list, overwriting any existing information.

Filtering and Stratifying Blobs

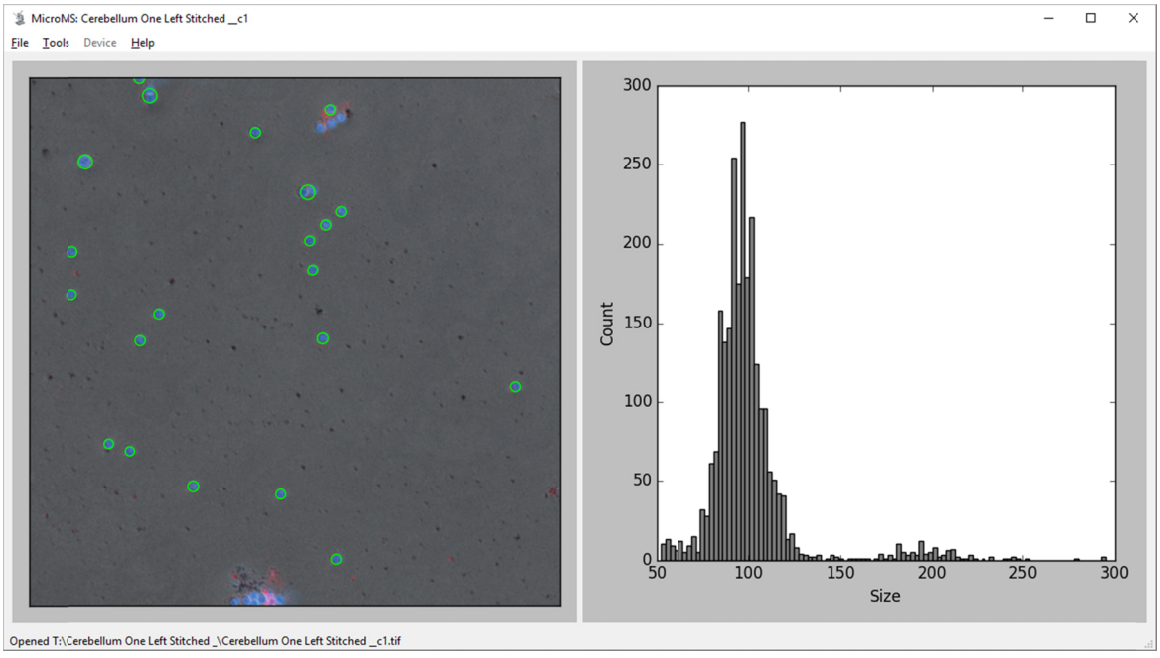
After all blobs have been found or manually selected, it is frequently useful to begin segregating different classes and remove uninteresting blobs. In addition to refining the ROI as mentioned above, microMS provides filtering based on size, circularity, pairwise distance, and fluorescence channels. All of

these metrics can be examined and filtered by interacting with the population level histogram. Additionally, pairwise distance filtering is applied by selecting the *Distance Filter* under the *Tools* menu.

Introduction to the histogram

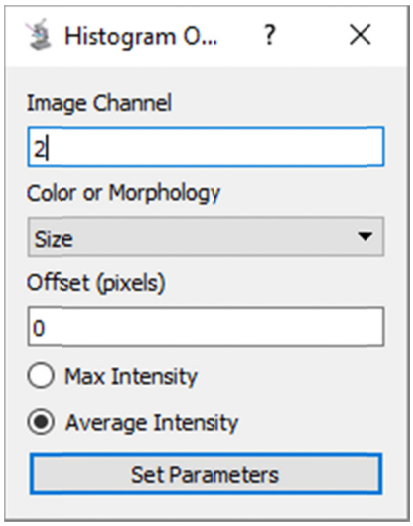
The histogram interface of microMS provides an interactive method for examining population-level statistics of the blob list. Details of each population metric are described in more detail below. This section will discuss how to set, interact with and utilize the histogram.

Once a list of blobs is generated, the histogram is activated by selecting the *Histogram Window* option under the *Tools* menu or *Ctrl* + H. Changing blob lists, opening a new blob list, or performing blob finding will have the histogram be recalculated. Opening a new image, manually adding a blob, or performing blob patterning will close the histogram window. By default, the distribution of sizes of the current blob list will be shown when the histogram is initially opened.



The above histogram shows a sample distribution where cells were found with a size less than 300 pixels (in area). Clearly, there are two populations present, those with areas ~100 pixels and some

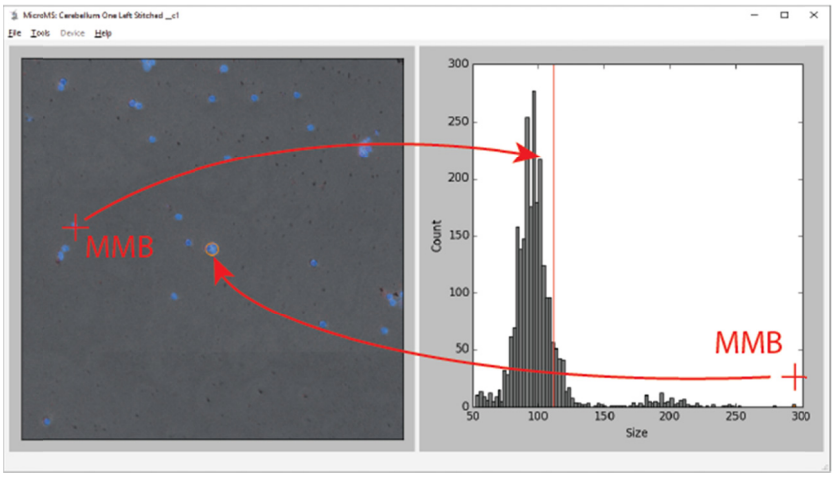
approximately twice as large. Using the mouse scroll wheel zooms the x axis in and out. The histogram settings are accessed in *Histogram Options* under the *Tools* menu. Here the metrics used for generating the histogram and some useful parameters are selected. Under *Color or Morphology* the fluorescence channel, size, circularity or distance may be selected. The remaining options relate solely to fluorescence intensity. Image channel is the image to examine for parsing fluorescence intensities, where 1 refers to the brightfield image. The corresponding color channel must also be specified. For example, if you wanted to examine the red intensity, collected in sample_c3.tif, image channel should be "3" and 'Color' must be red. Offset corresponds to the amount to increase (> 0) or decrease (< 0) the blob radius when examining the blob region. Finally, the intensity to display may correspond to the maximum or mean intensity within the blob region. To improve speed of analysis, the intensity corresponds to the entire circumscribed *square* for each blob. As such, neighbors at the "corners" of each region may skew results. Furthermore, the mean intensity has a dependence on the size of circular objects.



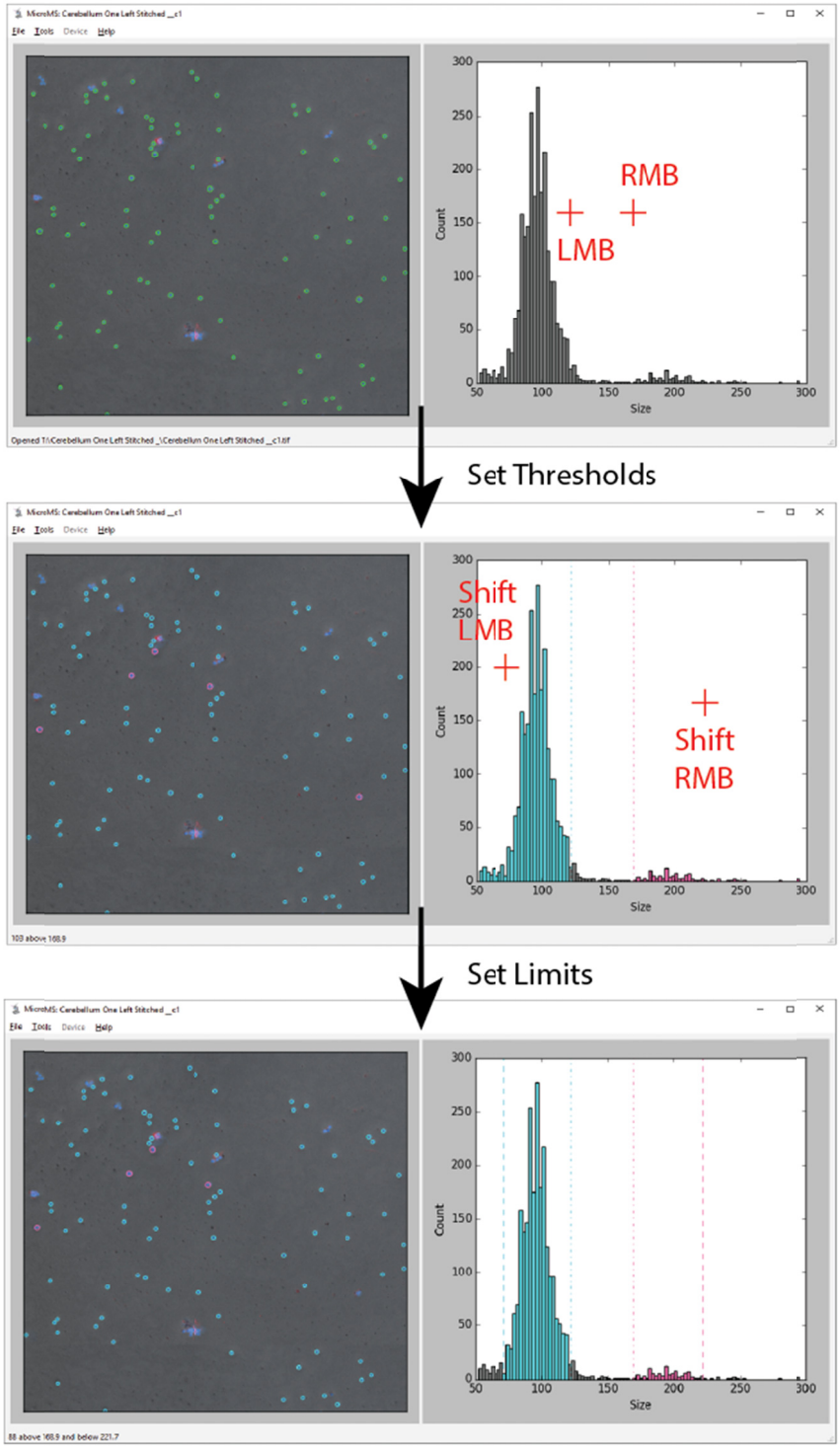
At any time, the values displayed in the histogram and the histrogram image may be saved by selecting *Save/ Histogram Image* or *Save/Histogram Values* option under the *File* menu. The image is a png of the current figure with all markup described below. Values are tab deliniated text files with the name (x,y coordinates) and corresponding metric value for each blob.

The histogram and slide image interact with each other to assist with picking values for filtering the population. A middle mouse button (MMB) click selects a single blob in the image or bar in the histogram. When a blob is MMB-selected, its population metric (e.g., size) is shown on the histogram as

a red, vertical line. This is helpful to assess where in the histogram certain blobs are located. MMB clicking a bar on the histogram shows just the blobs falling in that range of values. The bar and blob are both colored orange. Additionally, the image view is centered on the first blob falling in that range. Showing a single bar is helpful to see what blobs look like which fall in a specific bin in the histogram. To clear this, and any histogram filters, press C.

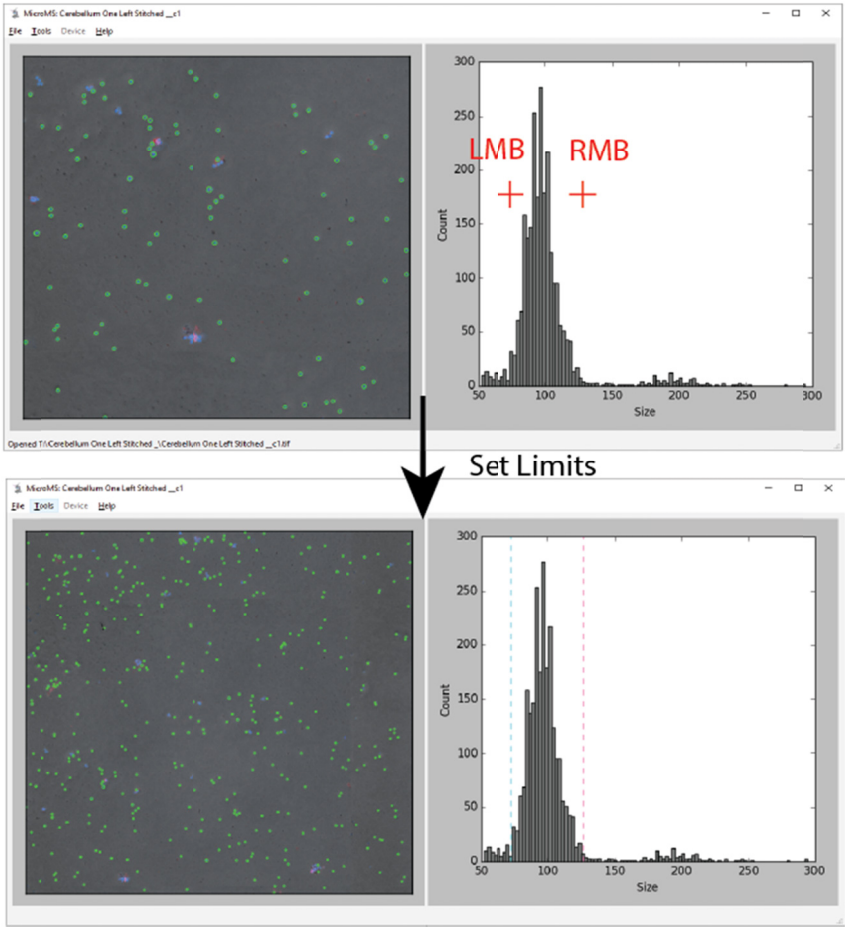


Examining single blobs and bars are useful for determining threshold values, but cannot be used as filters. Two independent filters are provided to partition the blob populations. Nominally they correspond to high and low pass filters, but they can overlap and function similarly. The low pass filter is activated with the left mouse button, high pass with the right mouse button. These set a high or low pass threshold for the histogram and cause the image to redraw, showing the blob locations satisfying the filter. The threshold values are shown as vertical lines and bins of the histogram within range are colored to match the corresponding blobs. The value selected and the number of blobs within range are also displayed on the status bar. High and low *limits* are applied for the low and high pass filters, respectively, by performing a *Shift*+ Left or Right Mouse Click at the desired location.

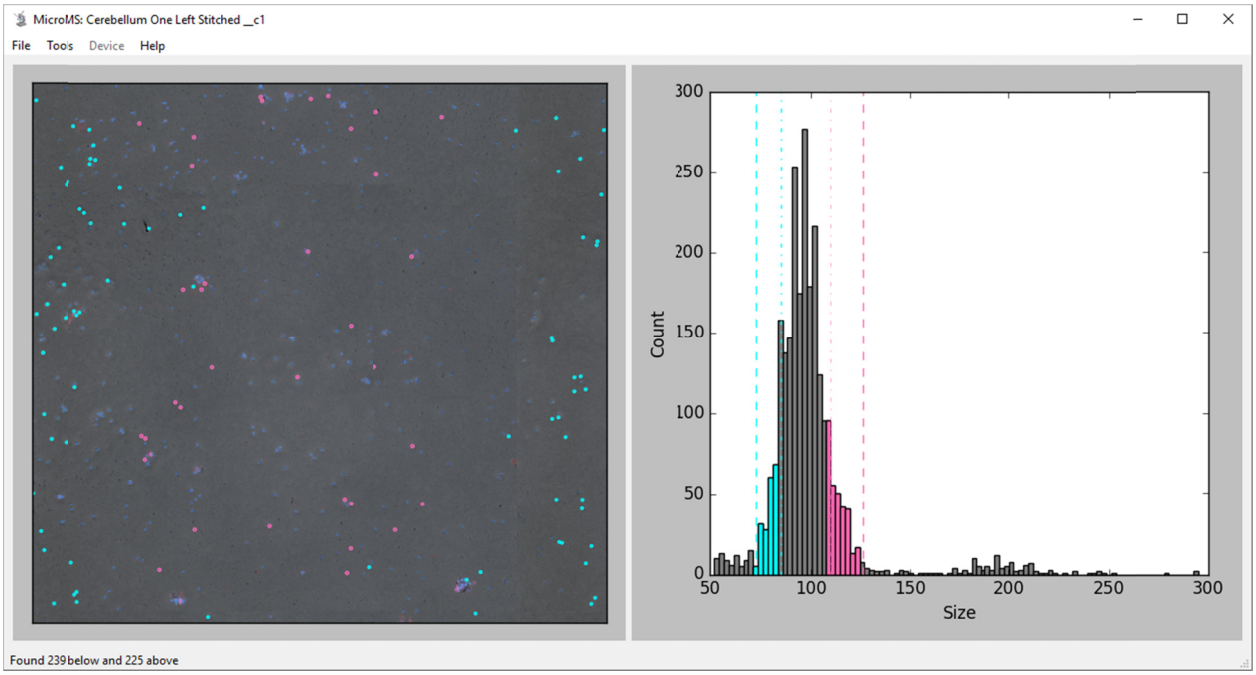


Filters can also be generated automatically, which is useful for examining a set number of extreme members within a population. In the example above, there appear to be two populations. The following steps illustrate how the largest and smallest cells (250 cells for each category) within the

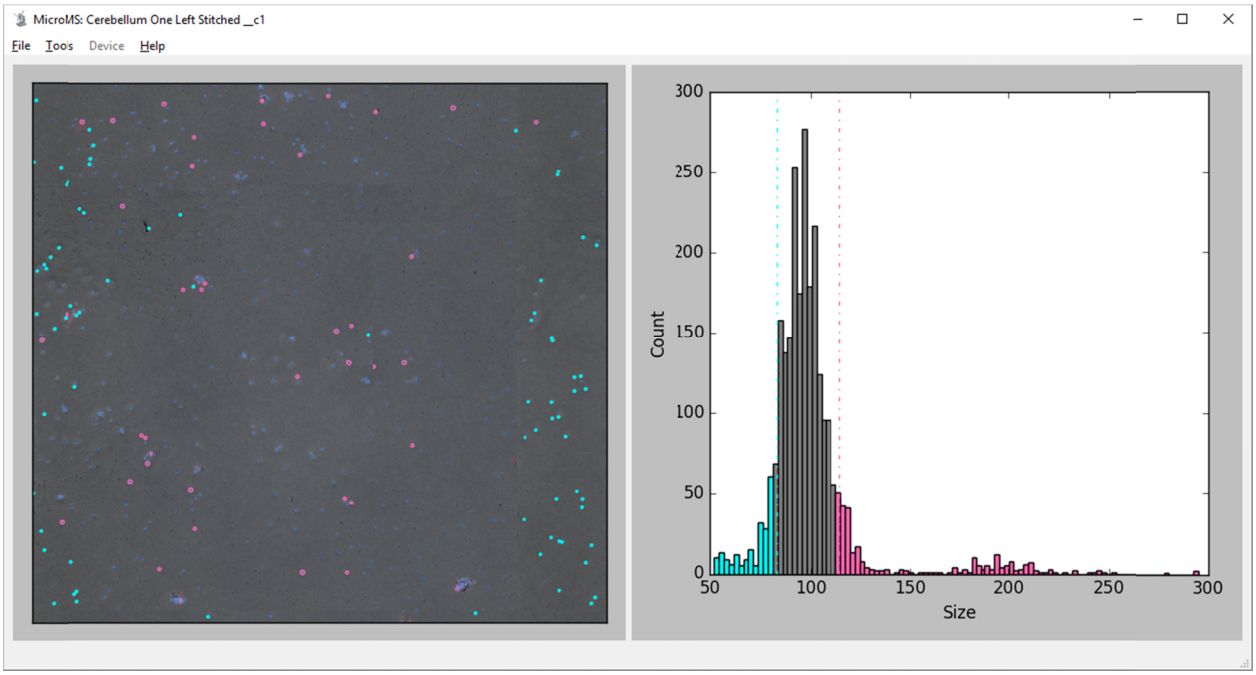
population centered on 100 pixels are selected. First, select high and low *limits* with *Shift* clicking (LMB = left mouse button, RMB = right mouse button).



Note this step does not set any filters, only defines limits. The extremes within these limits (or the entire population, if no limits are set) are found by selecting the *Pick Extremes* option under the *Tools* menu. This launches a popup box requesting the number of blobs to try and find for each filter; in this example, '250' should be entered. Next, microMS attempts to find the histogram divisions that provide approximately the requested number of cells in the high and low range of the population. The actual numbers in each filter are displayed in the status bar.



Because this distribution is fairly symmetric, the locations of the thresholds are approximately the same distance from the limits. However, repeating the request without set limits produces the following:

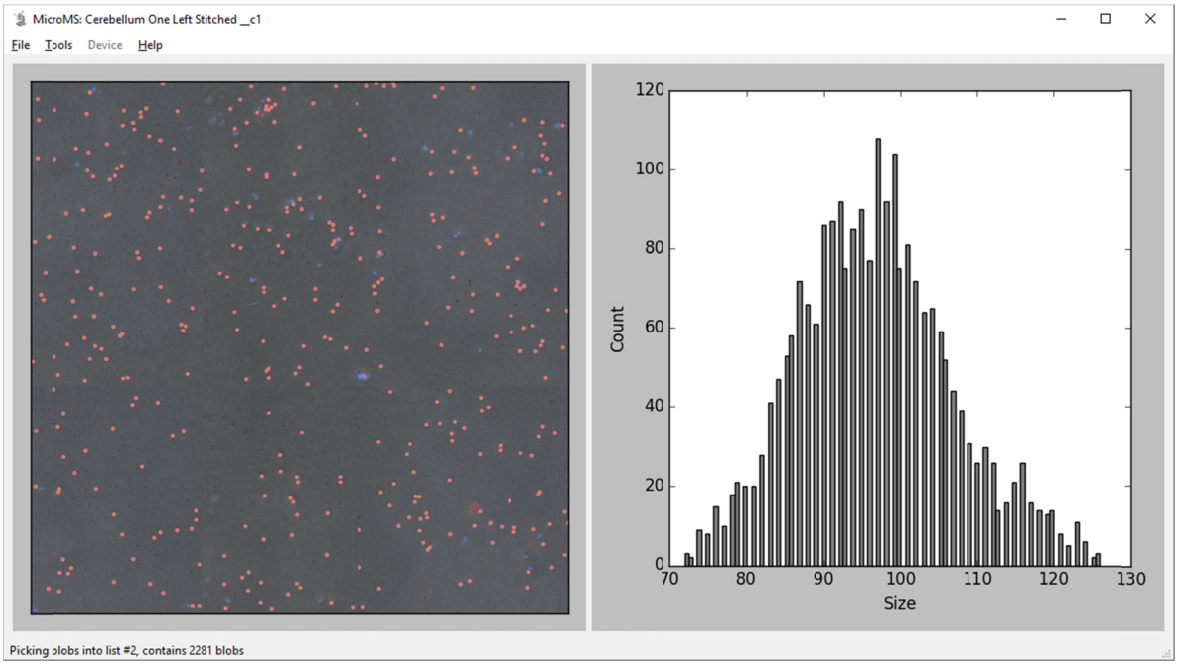


Where the range of values included in the high threshold is much larger to obtain the same number of blobs as the low threshold.

Using the ranges: saving and filtering regions

With the appropriate filters in place, their output is either saved as separate blob finding files or used to generate additional blob lists for further filtering. To save the current histogram filters, select *Save/Histogram Divisions* under the *File* menu. This prompts the user for a base name to save the resulting text files. For each histogram division, a text file is saved with the name <BASE>_<high or low>_<VALUE>.txt where BASE is supplied by the user, high or low depends on the filter division, and VALUE is the high or low *threshold* value defining the region. To utilize the output for performing measurements, the generated cell lists must be loaded as separate lists to generate an instrument file.

Alternatively, the filters can generate new blob lists for further filtering or generation of separate instrument files. To utilize this feature, select the ranges of interest and select the *Apply Filter* option under the *Tools* menu.



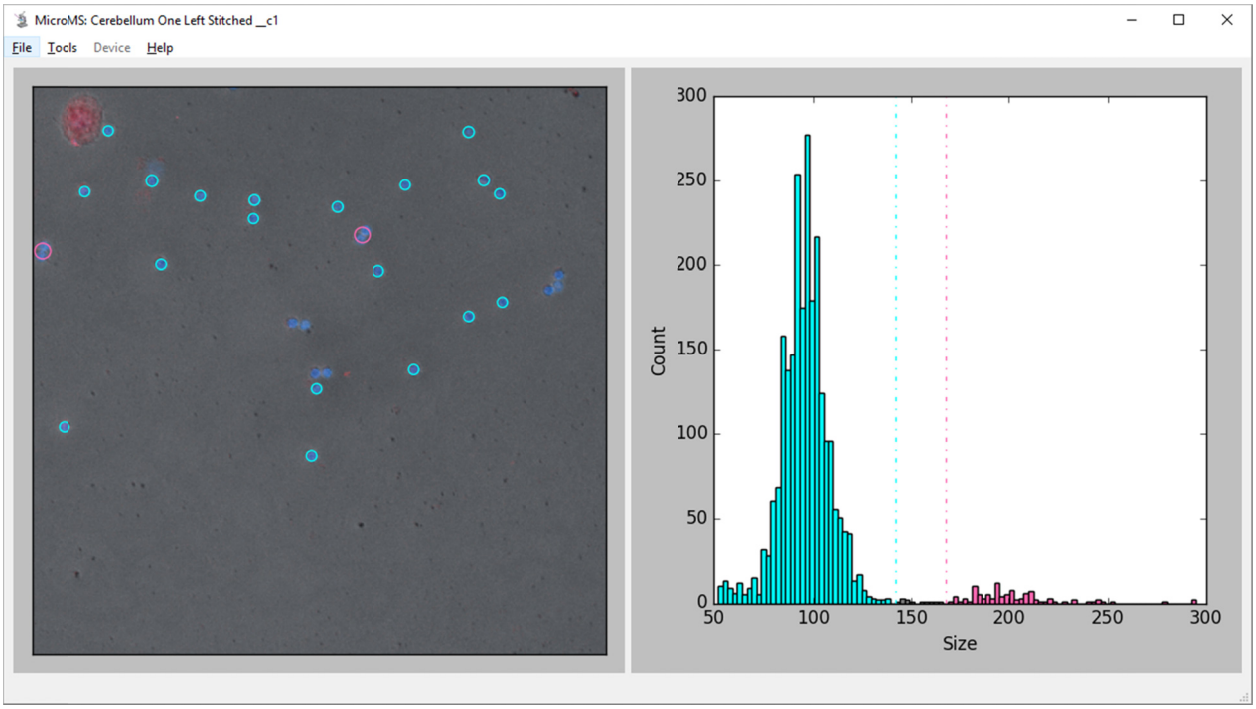
Each filter generates its own blob list and fills an empty list. The new list allows the application of additional filters prior to saving. Note that unlike saving histogram divisions, the filter set used to generate the blob list *is* stored in the blob finding file. For example, the list above looks like:

```
thresh  75
minCir  0.6
maxCir  1.1
maxSize 300
minSize 50
ImageInd        2
channel 2
ROI: []
->71.0<c1[Size]<126.7;mean;offset=0->
x       y       r       c
33669.686       27590.892       5.698   1.000
33173.677       27600.917       5.528   1.000
```

Where the line "`->71.0<c1[Size]<126.7;mean;offset=0->`" indicates image channel 1 (c1) was used to filter using Size between 71.0 and 126.7, with a mean reduction and offset of 0. Additional filter steps will be listed in the order they were performed to show what actions produced the current blob list.

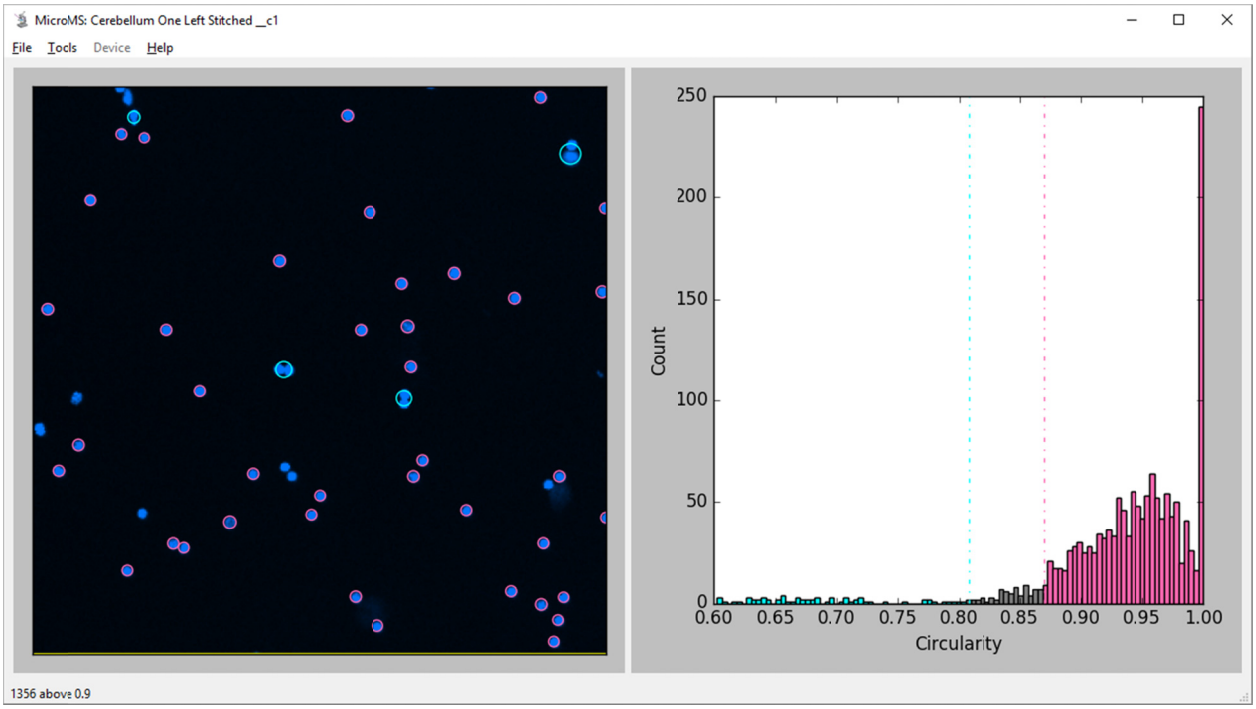
Morphology: Size and circularity

Note that size and circularity effectively provide the same filtering as blob finding. However, it is frequently useful to perform automatic blob finding with less stringent metrics and refine the population later. Following this protocol ensures that distance filtering does not skip neighbors that are clustered together. Using the same population as above:



Notice that dividing the population in half on size allows the identification of cells too close to resolve during cell finding (shown in pink) while individual cells are highlighted in teal. However, the cells near the center of the image are not identified because *they were not located during blob finding*. As such, they will not be considered during distance filtering.

Circularity is also helpful to identify unresolved cells or other debris:



Now the unresolved cells are shown in the low pass filter because they have low circularity. Examining the size and circularity helps ensure data quality by removing unresolved features and other imaging artifacts.

Distance filtering

Proper distance filtering is crucial to ensure acquired spectra belong to a single blob. The exact value will depend on the instrument position accuracy, probe size, and any suspected analyte delocalization during sample preparation (e.g., MALDI matrix application). Generally, the distance filter should be larger than the probe size to prevent contamination during acquisition. Only found blobs will be considered with distance filtering. At any time, a set value can be used for distance filtering in the *Distance Filter* under the *Tools* menu. Upon completion, the blobs passing the filter will be copied to the next open blob list. Alternatively, an appropriate distance filter may be investigated by using the histogram window. When selected, the minimum pairwise distance is calculated. Generally, the distribution should be Poisson. Large populations of blobs very close together indicate a high seeding

density. By selecting high pass filters, microMS will report the number of blobs passing the filter in the status bar and display their locations.

Fluorescence intensity

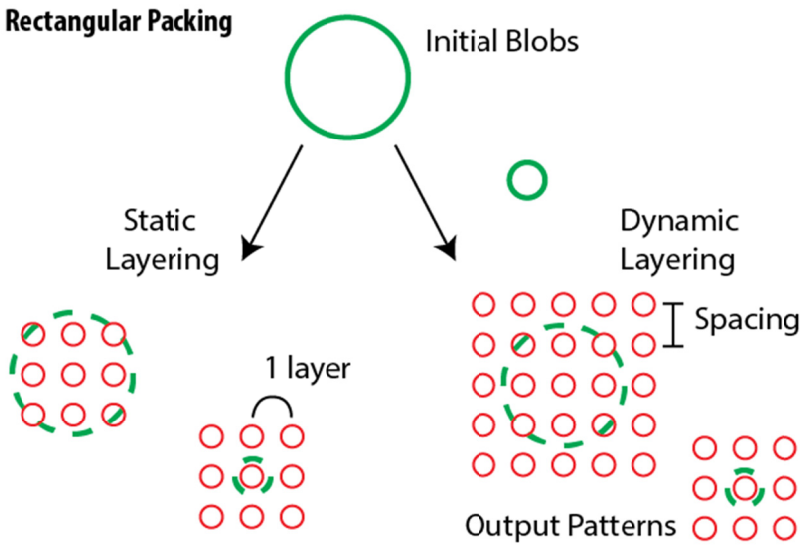
While the previous metrics provided sample quality checks, fluorescence intensity helps generate orthogonal "labeling" of blobs even before mass spectral acquisition. Additionally, examining the same fluorescence channel as used for blob finding helps remove dim blobs from consideration. Generating the intensity histograms will be slower than morphology due to the required image analysis steps.

Blob patterning

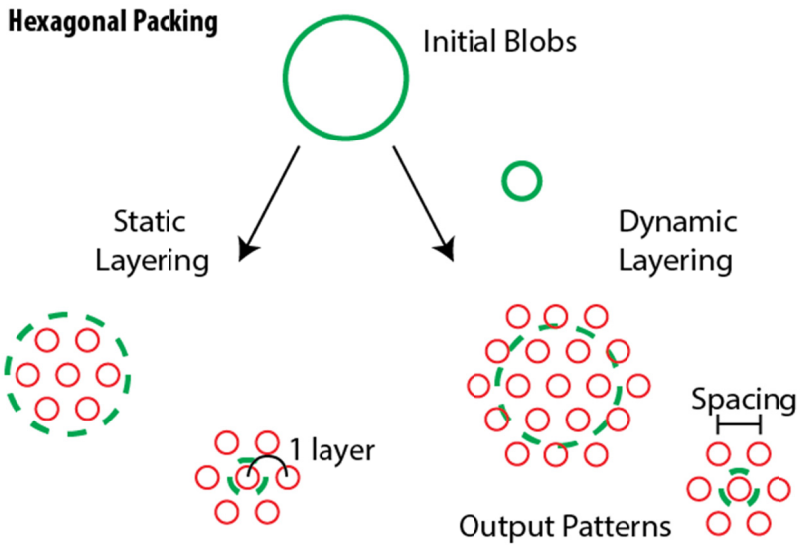
Once blobs have been found and filtered, it is occasionally useful to further pattern target positions, either to acquire an "image" of the area surrounding each blob or to more effectively sample blobs much larger than the probe size. Three different packing patterns are available for performing blob patterning: circular, rectangular, and hexagonally close packed. Each pattern is accessed through the *Tools* menu and prompts the user for additional packing parameters. The resulting patterns are stored in a new blob list which consists of x,y points, a single radius, circularity of 1, and a group number which uniquely ties each pattern to a parent blob. Note that generated patterns may cause overlap of target positions.

Rectangular packing generates even x,y spacing in a grid of target positions, effectively allowing the generation of mass spectral images over each blob. Rectangular packing requires three parameters: spot to spot distance, number of layers, and whether to perform dynamic layering. The spacing dictates the pitch, or pixel size, of the resulting image. The number of layers corresponds to the number of horizontal/vertical pixels from the center to generate. A value of 0 results in a single target per blob, 1 results in 9 targets, 3 in 25 targets, etc. Due to the way the patterns are generated, acquisition proceeds

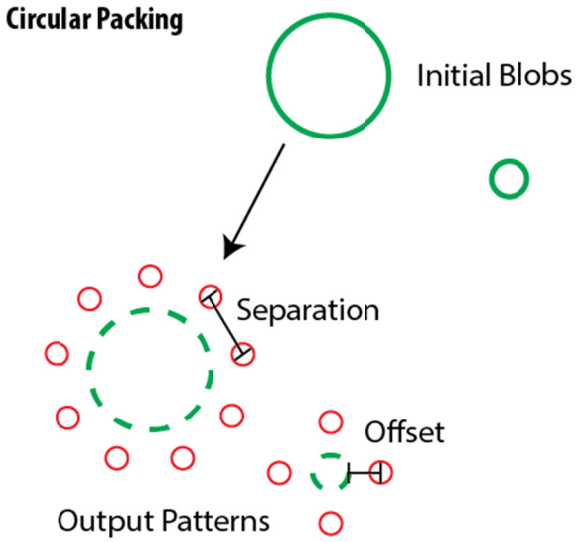
by spiraling out from the center position. With static layering, all resulting patterns will be the same size, regardless of blob shape. If dynamic layering is chosen, the number of layers will be adjusted for each blob to ensure that the entire area is covered in targets. In this case, the input number of layers controls the *extra* layers to include during patterning.



The same set of parameters is required for hexagonal and rectangular packing, except instead of having constant x,y spacing, targets are positioned in a hexagonally close packed arrangement to maximize the number of acquisitions per area. Resulting data can provide mass spectral images, but most analysis software assumes rectangular packing.



Finally, circular packing allows the analysis of the circumference of blobs. This allows an efficient sampling of compounds which migrate from blob locations or are only found around the exterior of blobs. Unlike the previous packing, circular packing requires the minimum separation, maximum number of targets per blob and an offset from the circumference. There is also a minimum number of targets which is set to 4 by default and must be adjusted in the source code.



In circular packing, the number of targets is always dynamically adjusted based on the blob radius plus the user supplied offset; negative values of offset place targets in the blob interior. Very large blobs will have the maximum number of targets placed evenly around their circumference. Smaller blobs will have fewer targets, but will maintain the minimum separation between targets. Very small blobs will have the minimum 4 targets, regardless of the resulting target separation. This strategy provides several replicates per blob, but prevents repetitive acquisition for larger blobs.

Instrument correlation

Once all targets are found, filtered and patterned, the blob information needs to be translated to instrument input. MicroMS provides an interface for performing instrument integration either offline or with direct instrument control. Full instrument control requires significant extensions of the connected Instrument interface and will differ dramatically between instruments. As such, this section will only cover aspects applicable to all instruments.

Point-based similarity registration and fiducials

At its core, microMS utilizes a point-based similarity registration to map a set of fiducials between physical space and an image coordinate system. The target locations in physical space are then inferred from their locations in the image using a linear coordinate transformation. The specific registration accounts for translation, rotation and scaling. Some limited support is available for reflections, but no corrections are made for skewed perspectives.

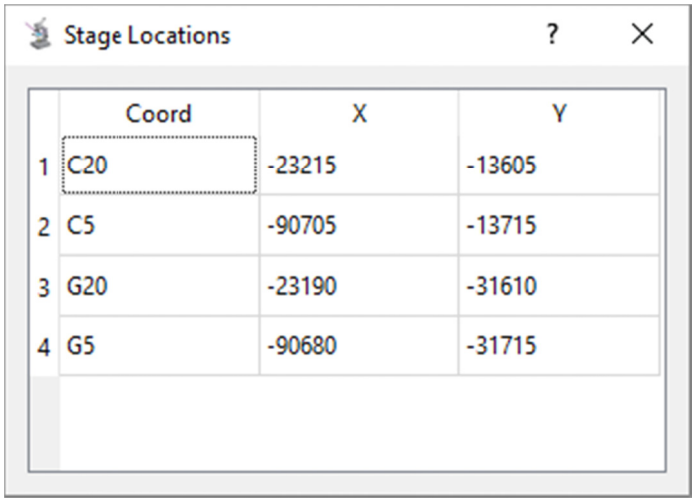
Accurately analyzing target locations depends on the precision of the sample stage, the microprobe size, correct stitching of optical images, and accurate estimations of the fiducial locations. The location of fiducials have drastic effects on accuracy and therefore requires care. We have successfully utilized etched fiducial markers in the shape of an X. Location of the intersection of the two lines can be

performed accurately and is less susceptible to distortion between image systems, particularly after MALDI matrix application. Other options include placement of dyes/paints, selective laser ablation, or beads. Generally, smaller fiducials are located with higher precision, but they must be large enough to locate on the instrument camera system. Ideally, the fiducials would fluoresce in the same wavelength as the blobs, otherwise multiple image channels must accurately overlay.

No assumptions are made regarding the instrument besides the basic requirements that fiducial locations must be found and recorded, and the instrument must be directed to arbitrary target locations. Frequently, several intermediate steps are required to accomplish these goals, even in the simplest cases. Each instrument has its own coordinateMapper which defines instrument-specific functions for interacting with microMS. Some instruments have multiple coordinateMappers if different instrument control software is targeted (e.g., the solariX targets positions through autoexecute or flexImaging functions). The current instrument is displayed and changed under the *Instrument* option under the *File* menu.

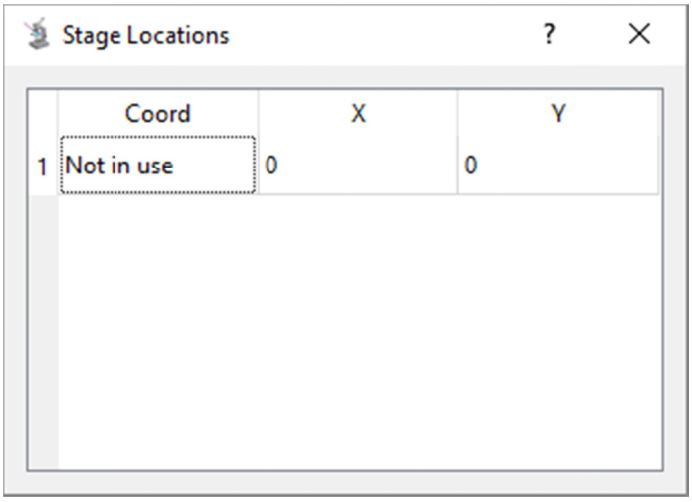
Instrument settings and intermediate coordinates

Occasionally instruments utilize more than one coordinate system for physical locations. For example, Bruker MALDI instruments provide direct output of the 2D linear stage positions. However, the coordinates used to direct motion during automatic acquisition are scaled fractions of the entire sample plate. In this case, microMS must utilize an additional coordinate transformation to map between the intermediate, motor position and the final, fractional distance position. To adjust or calibrate these positions, microMS provides a simple interface displaying set coordinates:



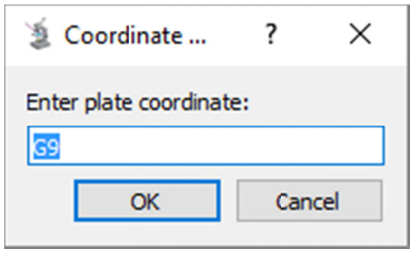
Here C20 refers to a set position in the ultrafleX software, and when the stage is directed to that point, the motor coordinate has the x,y position of -23215, -13605. The coordinate mapper then incorporates this information to generate a new intermediate map between the motor coordinates and the final output. These values rarely change, but adjusting the motor stage or using a different set of teaching points cause them to move and should be calibrated occasionally. Closing the window updates the information of the coordinate mapper and a txt file in the source folder.

Some coordinate mappers do not require this intermediate map and will display that these values are not used:

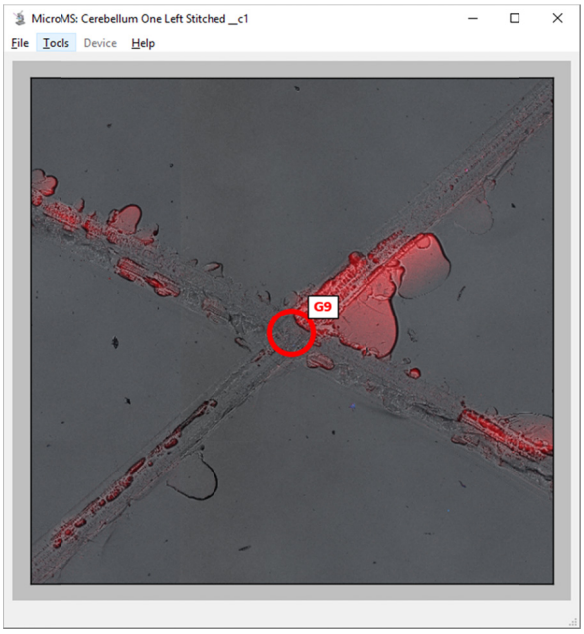


Interacting with fiducial markers

Accurate fiducial training requires precise location of the fiducial on the instrument and the image. Once found in the instrument, the physical position is input into microMS by RMB on the image. This causes a popup window to appear to input the physical coordinate:

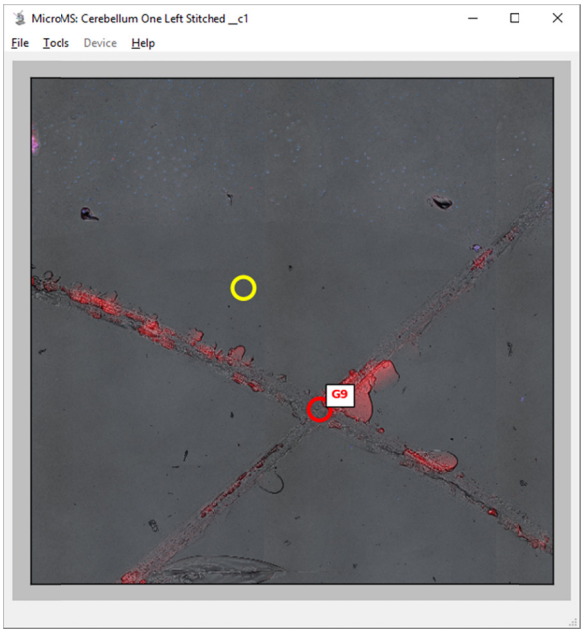


A predicted location is displayed for some instruments (G9 above) but should only be used as a quick way to assess if the registration is drastically incorrect. This will only be available after at least 2 fiducials are trained. Entering the coordinate and clicking "*OK*", or pressing enter, will record the fiducial:

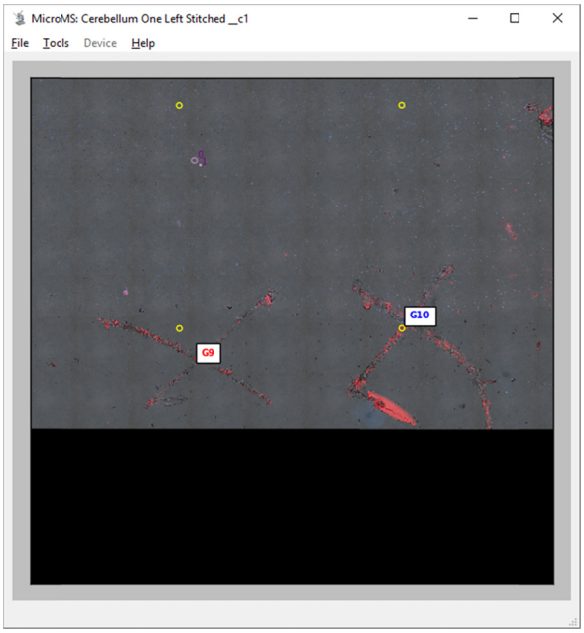


Fiducials are displayed as either red or blue circles with text indicating the closest predicted point. If the input value cannot be parsed correctly, an error message will appear on the status bar and the fiducial will not be recorded. Fiducials are removed by holding *Shift* and RMB.

To help locate fiducials in the instrument, microMS displays predicted plate locations for some instruments. These represent set points on the stage which are encoded positions and easily located. Here the predicted location of G9 is shown in yellow:



The quality of the fiducial set is assessed interactively through predicted points. As new fiducials are supplied, the predicted points are updated. Large movements of predicted locations or deviations between actual positions indicate poor accuracy. microMS also tries to detect fiducial training errors by highlighting the fiducial with the worst fiducial localization error in red:



The fiducial localization error is estimated by using the current registration to predict the physical location of each fiducial based on its pixel position. The fiducial with the largest deviation is then drawn in red. While this is the "worst" fiducial mark, it is not necessarily bad or an unacceptable amount of error. Generally, the worst fiducial should be removed and reselected until it is no longer the worst. This process is repeated until the fiducial continues to be the worst, which indicates the registration can no longer be improved.

Two fiducials are required to utilize predictions, worst fiducials, and save instrument positions. Generally the error decreases by 1/sqrt(number of fiducials), so more fiducials result in higher accuracy. Accuracy also depends on fiducial location relative to the target positions. We have found surrounding the target area with fiducials produces high accuracy while minimizing possible interference of chemical information between fiducial and sample.

Saving and loading registration

The current registration information, including the instrument name and fiducial set, is saved by selecting *Save/Registration* under the *File* menu. This generates an msreg (mass spectral registration) file, which is a human readable text file.

```
ultrafleXtreme
S:0.5486963242166799
R:[[  9.99999772e-01  -6.75350115e-04]
 [  6.75350115e-04   9.99999772e-01]]
T:[[-88909.33559845]
 [-12029.68588949]]
image x image y physical coordinate
5756    3766    -85755.16001992302      -14103.896234978392
13699   3457    -81393.61102458672      -13975.810546419789
23429   2385    -76013.90742220463      -13336.153128124613
31016   2711    -71856.31638169987      -13501.721107700443
37956   3335    -68002.3891659187       -13827.570504536736
46751   3390    -63206.06210860018      -13895.825731133042
46673   11955   -63265.93930551935      -18557.07686469661
47124   21064   -62958.5668228457       -23588.61046751959
45704   36179   -63794.10718030092      -31916.41307565295
37933   35571   -68035.33181945984      -31595.50099322035
19935   35527   -77923.61086599139      -31607.28530550839
13706   36321   -81353.33264724944      -32046.16425675582
4536    34092   -86423.73958607297      -30844.323690462254
4591    28339   -86392.90881788862      -27645.064003520394
5397    20094   -85958.0517145891       -23090.923757333934
5889    11567   -85684.92893680255      -18397.235958513906
30244.0 36959.0 -72681.2626327591       -31684.955378219744
```

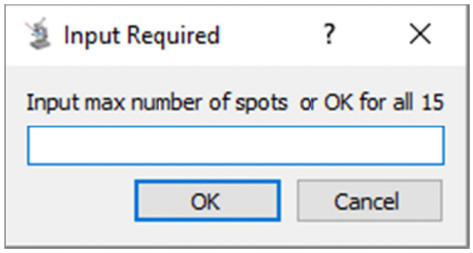
The first line has the instrument name. The next 5 lines contain variables of the linear transformation for mapping pixel positions to the physical coordinates. The remainder of the file records the pixel and physical location of each fiducial mark.

Previous registrations are loaded by selecting *Load/ Registration* under the *File* menu. This populates the fiducial training set and changes the current instrument if needed. Generally, it is not advised to reuse registration if the sample has been removed and inserted into the instrument as small changes in sample positioning manifest as systematic errors, casing a missed acquisition at every target location.

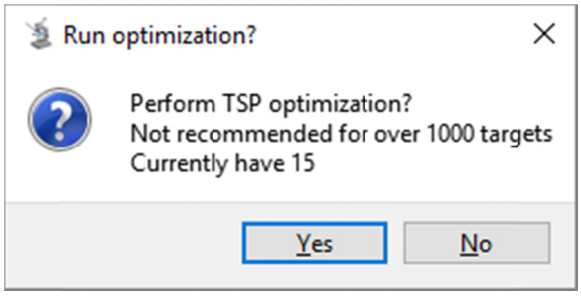
Saving and loading instrument files

With a good set of fiducials trained, the blob positions for the instrument may be generated. Again, there is an opportunity to assess fiducial training by selecting *Save/Fiducial Positions* under the *File* menu. This generates an instrument-specific file of target locations for the fiducials. By using the fiducial file as input for the instrument, the difference between expected and actual fiducial location helps determine if accuracy is sufficient for the given experiment.

To generate target points (blob positions), select *Save/Instrument Positions* under the *File* menu. Only the current blob list will be used for generating target positions. Again, an instrument-specific file type is saved, but there are several more options during export. First, the user is prompted for the number of blob positions should be exported:



This function provides a method to randomly select a subset of the blobs for analysis, especially useful to conserve instrument time. Clicking "*OK*" with a valid number in the text box selects the specified number of blobs for analysis. Leaving the box blank uses all spots and clicking "*Cancel*" stops the export operation. Next, the user is asked about path optimization:



microMS uses a simplified version of traveling salesperson path (TSP) optimization which optimizes the travel path, but bounds the optimization run time to at most 3 minutes, so the resulting path will not be fully optimal. The optimization determines the order to visit each target to minimize the total distance traveled. However, the computation requires calculating each pairwise distance between targets, effectively consuming RAM on the order of the number of points squared. Clicking "*No*" orders the points from top to bottom, left to right, and generally causes the stage to move about twice as far as the optimized path.

Sample positions can also be loaded from instrument files. Targets will retain their x,y position and group number, but will lose their size and circularity measurements. As such, if these values are important they should be loaded from a found cell file.

Advanced topics

The above should be sufficient for most users. However, the real power of microMS comes from the design choice to make instruments an abstract base class, greatly simplifying the work required to support new and diverse instruments. microMS also offers more advanced operations including direct instrument control.

Customizing GUI Settings

Several settings for the GUI are set in the file GUICanvases/GUIConstants.py. This file is fully commented with brief descriptions. The top section defines several colors for blob lists, predicted points, and fiducials. Of note, `MULTI_BLOB` contains the colors for all blob lists, in order from List 1 to 10. The `ROI_DIST` is the minimum distance between two vertices before the current vertex is removed. Lowering the value will allow drawing more complex shapes, but make deleting points more difficult. Next, some constant values are provided for the default blob radius and default fiducial radius.

The default blob and fiducial values were chosen for a particular application which may not be suitable for all purposes. Setting the `DEFAULT_RADIUS` to the probe size simplifies detection of targets too close together. Fiducial radius should be approximately the size of a given fiducial mark to help assess if the fiducial was placed in the correct position in the image. Next, the `DRAW_LIMIT` and `TSP_LIMIT` define limits on the maximum number of blobs for computationally expensive operations. More powerful computers can increase these values as needed, or if higher performance is required they may be decreased. `DRAW_LIMIT` defines the maximum number of blobs to draw from each list. This is overridden in the menu bar option. `TSP_LIMIT` defines how long a blob list to consider for TSP optimization be default. Again, this can be bypassed in the GUI when saving instrument positions, but this acts as a simple guard to consuming too much memory.

The next section lists colors utilized in drawing the population level histogram. These are all aesthetic changes and do not affect function of the histogram. The constants for blob shapes allows further customization of the default blob size, for either manually drawn targets or automatically generated patterns of targets.

Next, several files and directories are defined for performing standard debugging loads. These assist in opening a "standard" image data set for testing new features or replicating bugs. Once all files are defined, the debug data set is opened by pressing *Ctrl* + D. This is only operational when microMS first opens and if all files exist on the current machine.

Supporting new instruments

The goal is that a user with moderate python experience will be able to support new instruments in the future while maintaining the image analysis functionality of microMS. Hopefully, this section will act as a template for generating offline instrument coordinate mappers with arbitrary systems. The details are not important for a general reader, but should help guide more advanced users to support their own

instruments. This section will demonstrate how to support two new instruments, a hypothetical Generic XYsampler and equally-absurd Bruker flexArmstrong.

microMS coordinate mapper organization and requirements

The main GUI of microMS interacts with coordinate mappers through the supportedCoordSystems.py module. This initializes new instances of each supported mapper and generates their names for display. Each member in supportedMappers must inherit from the abstract base class defined in coordinateMapper.py. This module defines all necessary functions to fully leverage microMS and implements some basic functions. CoordinateMappers may also have an instance of connectedInstrument.py (another abstract base class) to interface with instruments. However, implementing direct control will not be covered here as it is specific for each instrument.

Due to the shared characteristics between Bruker MALDI mass spectrometers, another abstract base class is included with microMS, brukerMapper.py. This contains implementations for many of the functions specified in coordinateMapper and defines a smaller set of functions required for off-line analysis with Bruker instruments.

Implementing coordinateMapper for the Generic XYsampler

coordinateMapper.py contains the required methods and information on how to implement them. To reemphasize, inheriting classes must:

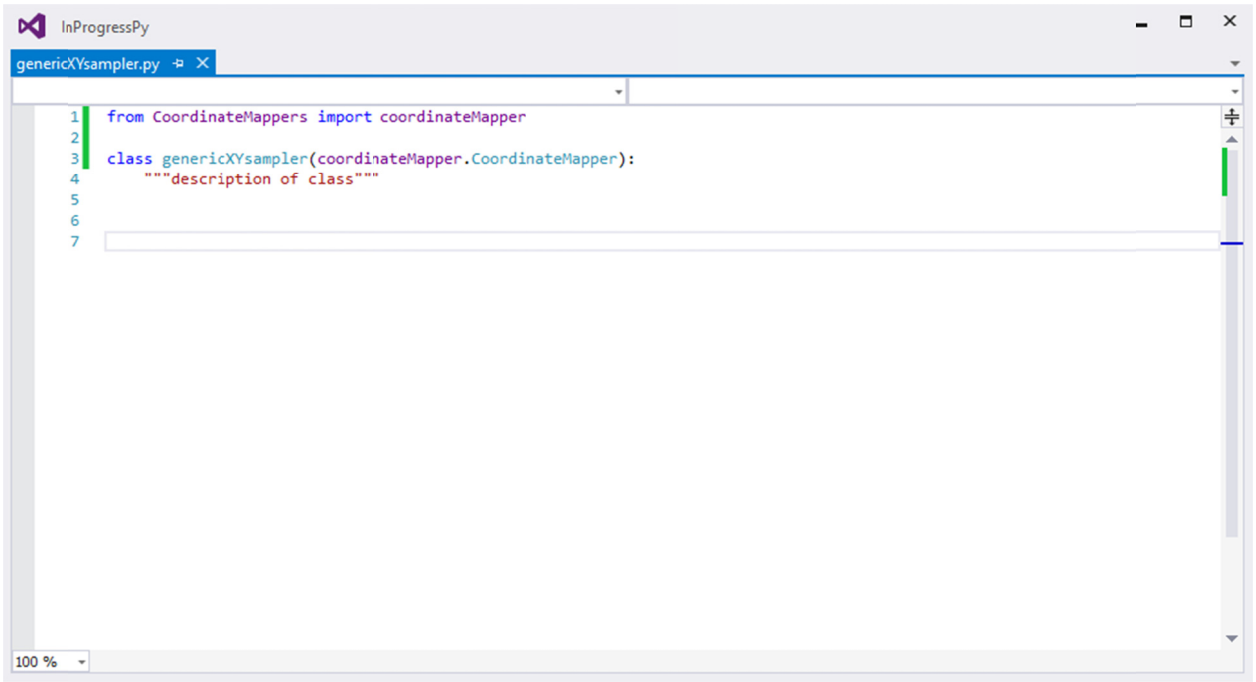
- Define self.instrumentExtension and self.instrumentName
- Implement isValidEntry, extractPoint, predictName, predictLabel, predictedPoints, loadInstrumentFile, saveInstrumentFile, getIntermediateMap and setIntermeidateMap.
- Add an import and initialize an instance in supportedCoordSystems.

Note that any methods not overridden will cause an error immediately upon running microMS.

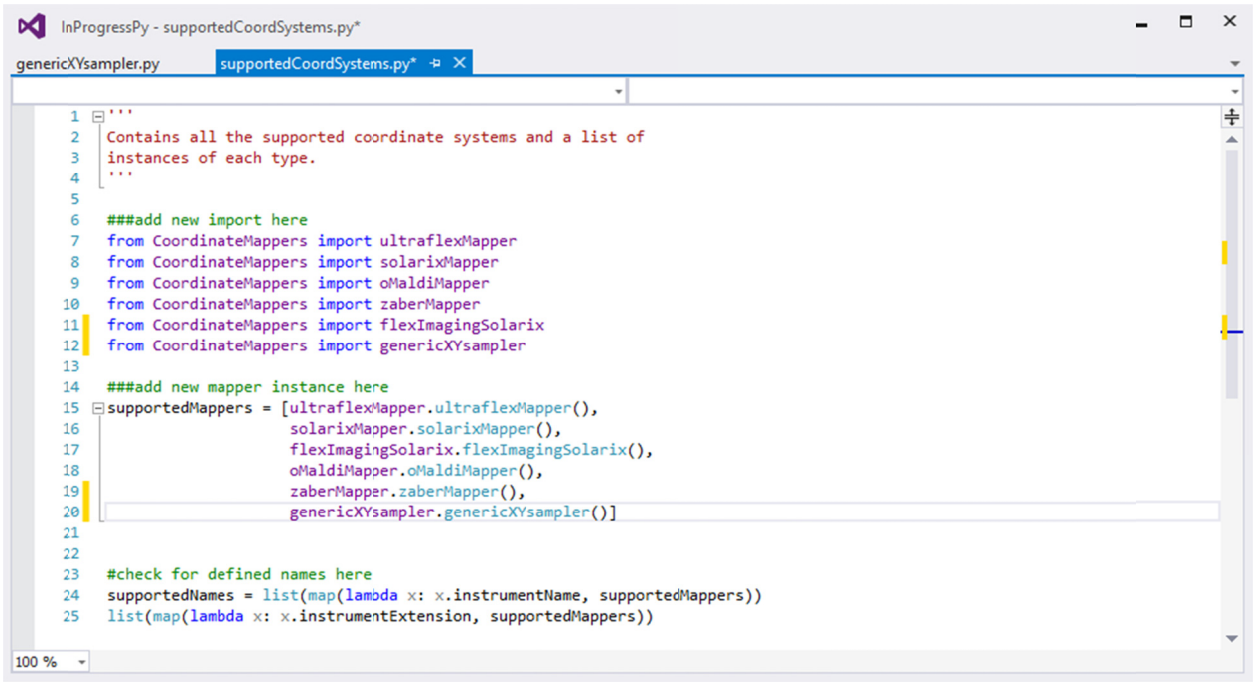
The Generic XYsampler is a fictional, new mass analyzer with some interesting requirements:

- Motor positions are read from the instrument, and are of the form <Xcoordinate>_<Ycoordinate> as floating point numbers. When moving down in the image, the Y coordinate increases.
- There are 25 set points in a 5x5 grid at motor positions 0..100..400 labeled A-Y from left to right, top to bottom.
- Instrument files are in .csv format, with the first column the sample name, the next two x and y coordinates in “GenericCoordinates”.
- GenericCoordinates are an offset of +100 in both directions, but could change on leap years.

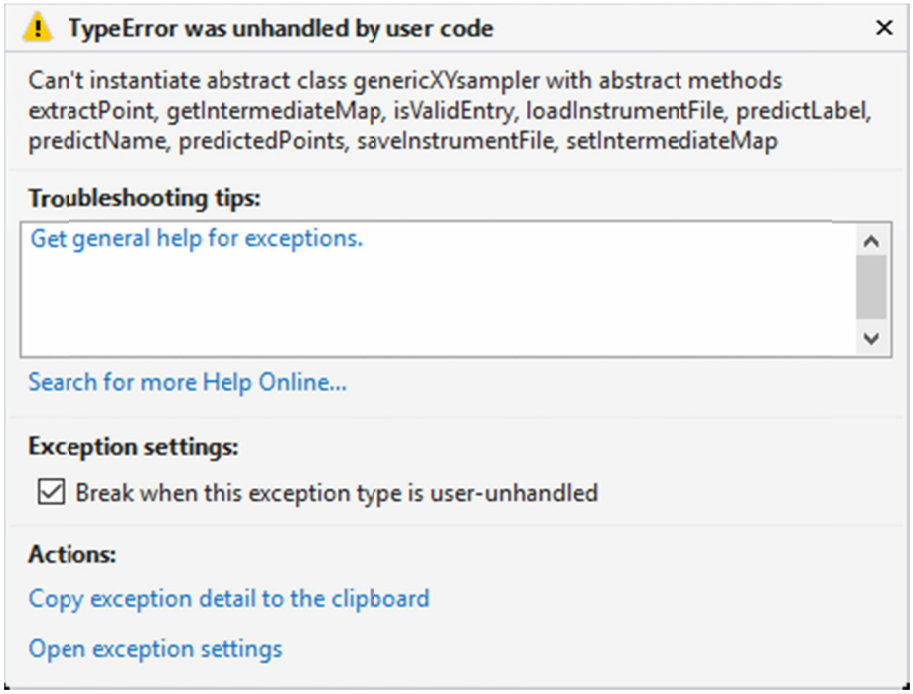
Start by making a new python class in the coordinateMappers package called genericXYsampler.py that inherits from coordinateMapper.



And add the genericXYsampler to the supportedCoordSystems



Running microMS causes the following error:



Next, fill in genericXYsampler with some unimplemented methods for each of the abstract methods:

InProgressPy - genericXYsampler.py

genericXYsampler.py | supportedCoordSystems.py

genericXYsampler | loadInstrumentFile

```python
from CoordinateMappers import coordinateMapper

class genericXYsampler(coordinateMapper.CoordinateMapper):
    """description of class"""

    def __init__(self):
        super().__init__()

    def isValidEntry(self, inStr):
        return super().isValidEntry(inStr)

    def extractPoint(self, inStr):
        return super().extractPoint(inStr)

    def predictName(self, pixelPoint):
        return super().predictName(pixelPoint)

    def predictLabel(self, physPoint):
        return super().predictLabel(physPoint)

    def predictedPoints(self):
        return super().predictedPoints()

    def saveInstrumentFile(self, filename, blobs):
        return super().saveInstrumentFile(filename, blobs)

    def loadInstrumentFile(self, filename):
        return super().loadInstrumentFile(filename)

    def getIntermediateMap(self):
        return super().getIntermediateMap()

    def setIntermediateMap(self, points):
        return super().setIntermediateMap(points)
```

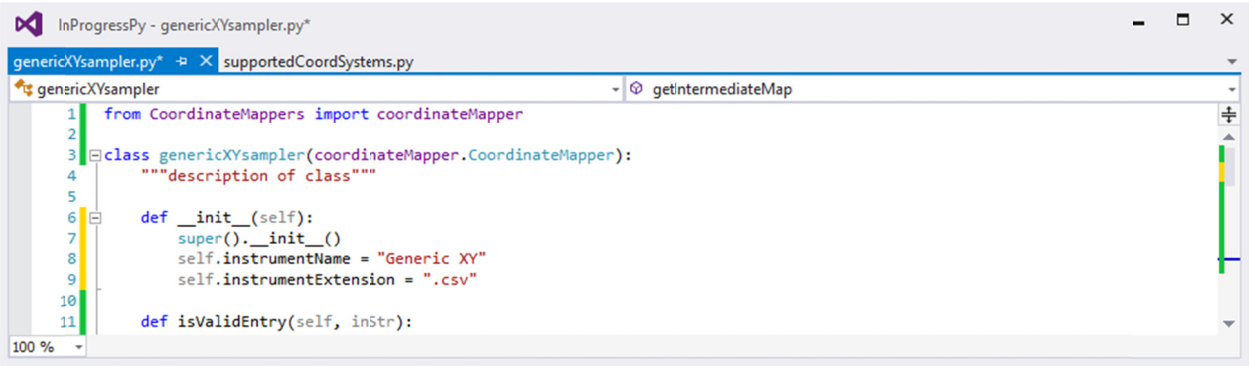
100 %

Running now generates the following error:

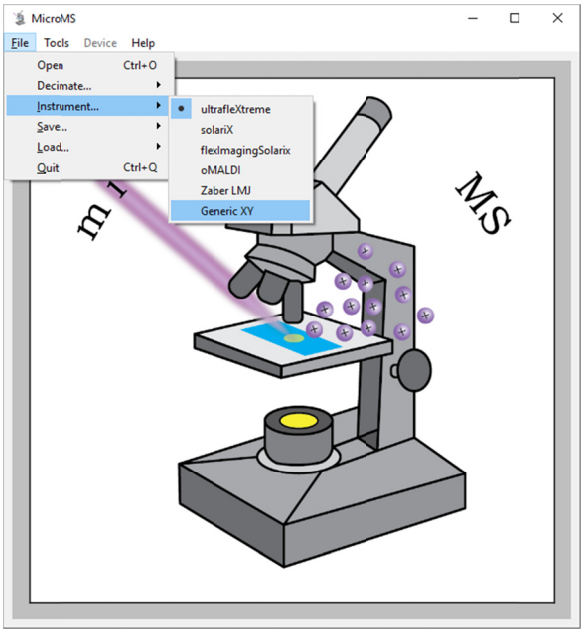
C:\Program Files\Anaconda3\python.exe

```
Traceback (most recent call last):
  File "D:\Sweedler Lab\GitRepositories\microMS\microMS.py", line 5, in <module>
    from GUICanvases.microMSQTWindow import MicroMSQTWindow
  File "D:\Sweedler Lab\GitRepositories\microMS\GUICanvases\microMSQTWindow.py",
line 5, in <module>
    from CoordinateMappers import supportedCoordSystems
  File "D:\Sweedler Lab\GitRepositories\microMS\CoordinateMappers\supportedCoord
Systems.py", line 24, in <module>
    supportedNames = list(map(lambda x: x.instrumentName, supportedMappers))
  File "D:\Sweedler Lab\GitRepositories\microMS\CoordinateMappers\supportedCoord
Systems.py", line 24, in <lambda>
    supportedNames = list(map(lambda x: x.instrumentName, supportedMappers))
AttributeError: 'genericXYsampler' object has no attribute 'instrumentName'
Press any key to continue . . .
```

Because instrumentName hasn't been defined. Define that variable name as well as instrumentExtension in the __init__ method:



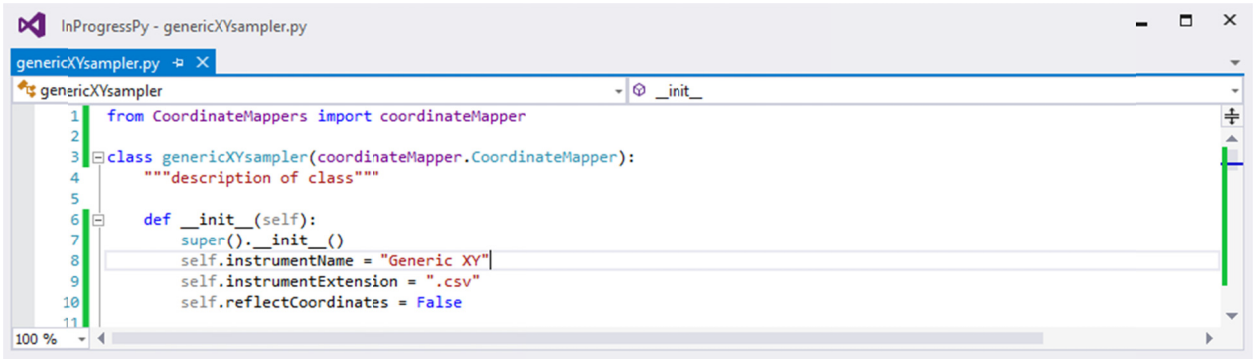
The instrumentName is used to display the instrument in the menubar and is saved for the registration file. As such it should be fairly short and unique among supported instruments. Adding these variables results in a running, but completely nonfunctional version of the Generic XY sampler:



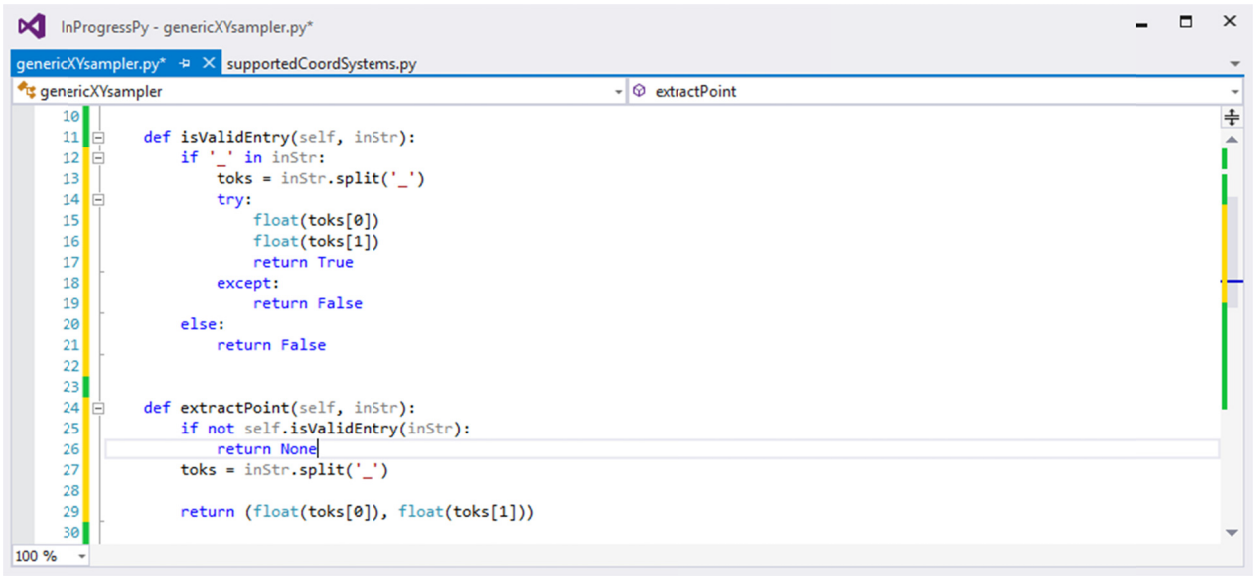
IMPORTANT AND CONFUSING! The last parameter to set in __init__ is reflectCoordinates. In computer graphics (microMS is no exception), the top left of an image is defined as (0,0). The x coordinate increases moving right, the y coordinate increases moving down. Because microMS utilizes a

similarity registration, it will not handle reflections properly by default. The reflectCoordinates Boolean controls if the instrument coordinate system needs to be inverted to register properly. Setting this incorrectly will produce inaccurate positioning! For the instrument, if the x axis increases and the y axis increases moving down and to the right (relative to the image/sample) reflectCoordinates should be false. If both decrease it should also be false. If only one increases, it should be true. Another way to assess this is generate a scatter plot of the x and y coordinates of a triangle in both coordinate spaces. If the positions orient properly with just a rotation, reflectCoordinates is false.

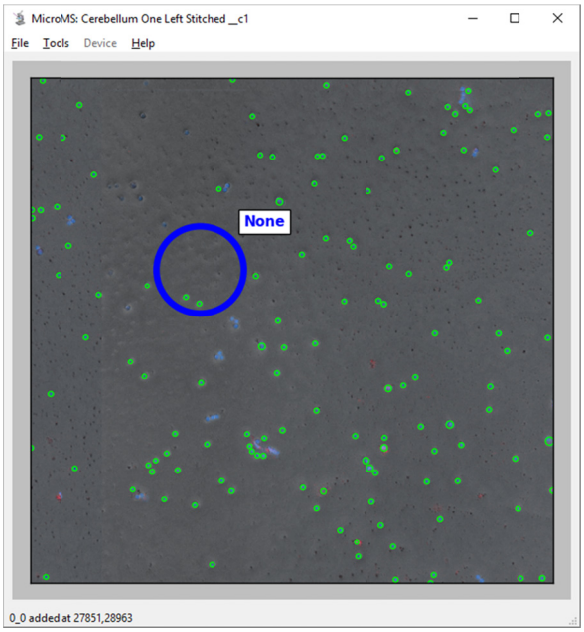
In this case, moving down the image causes the Y coordinate to increase, so:



Next, implement the remaining required methods. First, for handling motor positions, isValidEntry and extractPoint need to be implemented. IsValidEntry checks if the supplied string is a valid motor coordinate and returns true if it is. extractPoint should take a string, validate it with isValidEntry and return a new tuple with x,y coordinates as numerics:



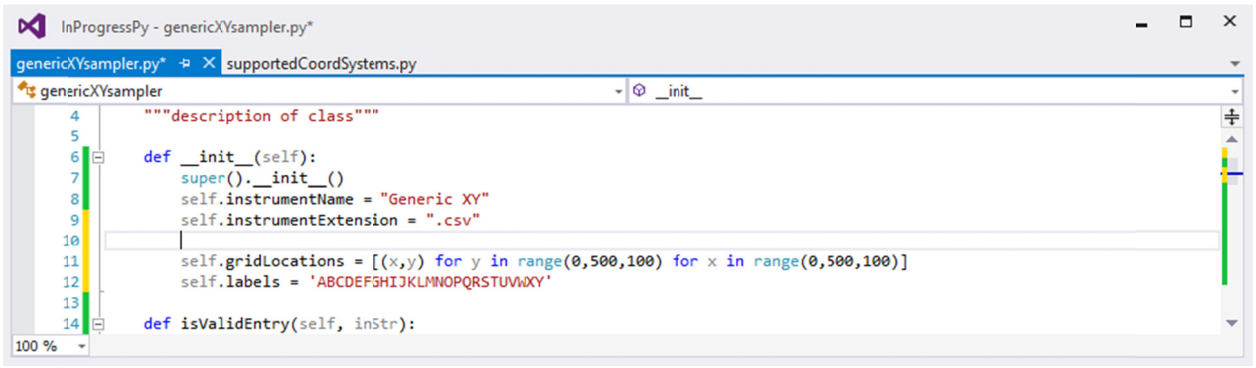
Now new points may be validated and added to the coordinate mapper:



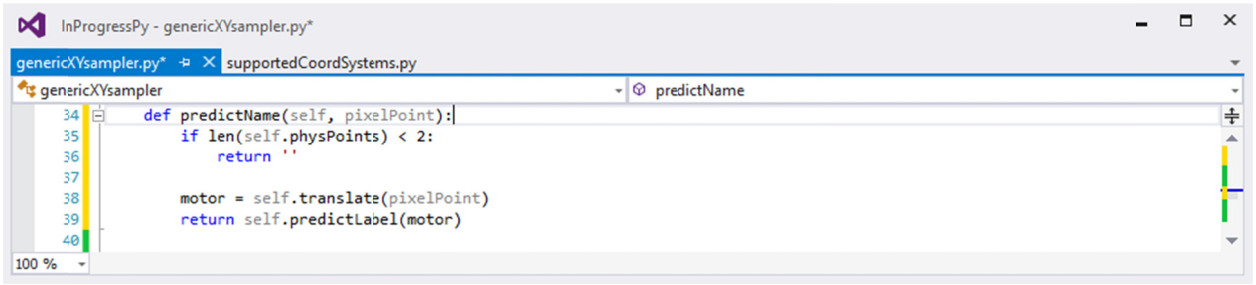
The next methods handle predictions. PredictName is used to fill in the popup text box when a user adds a fiducial mark. Here, it should return the closest label in A-Y. PredictLabel is shown next to the fiducial mark (currently None). Again, this should be the closest label in A-Y. Finally, predictPoints is a set of pixel positions to show when toggled with P. These are the grid positions of 0..100..400 in a 5x5

array mentioned above for Generic XY. While PredictName and PredictLabel appear to perform the same function here, there are important differences. PredictName uses a *pixel* position while PredictLabel takes a *physical* position. When the user first adds a fiducial, the pixel position is known, so PredictName is used for the text. After the fiducial is entered, the physical position is known and PredictLabel is called. There are also cases when the return values should differ. The solarixMapper returns the system clipboard during PredictName, but the closest MTP point in PredictLabel.

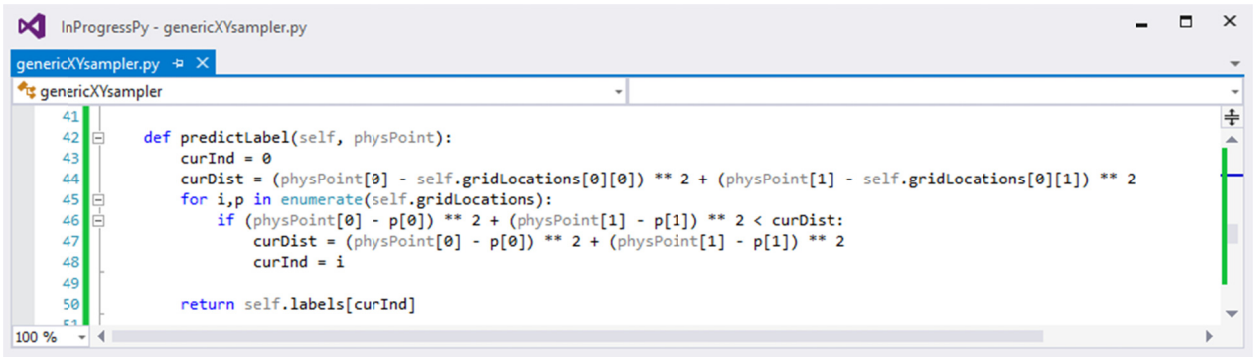
First, generate the grid of locations and a list of labels in the same order (in this case a string).



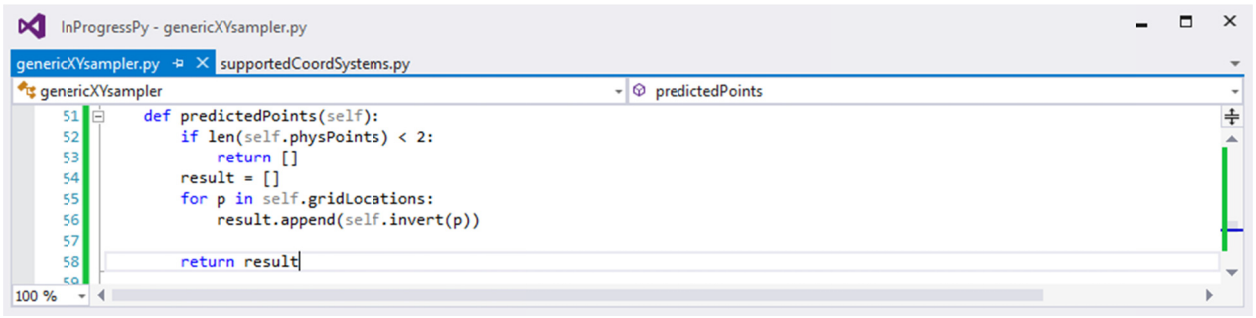
PredictName will take a pixelPosition and try to predict the label by translating the pixel to a physical location:



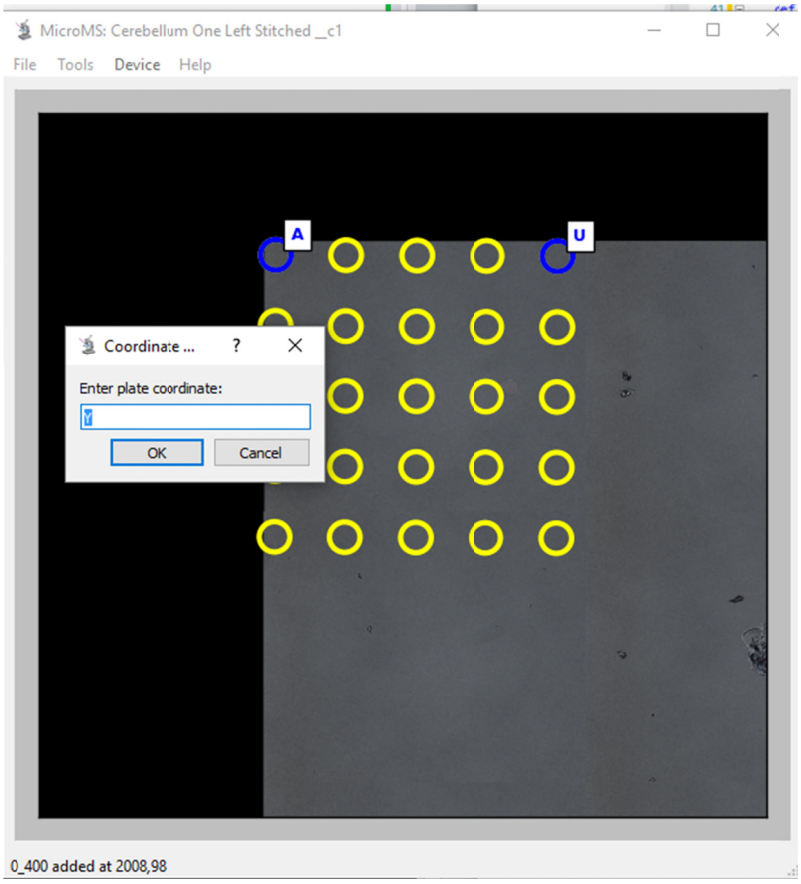
Note that self.translate is a method for converting pixel coordinates to motor coordinates and is implemented in the base class. predictLabel is implemented by iteratively searching for the closest motor position:



PredictedPoints is self.gridLocations when inverted to pixel positions (another base method):



Executing microMS now shows predicted points in yellow, labeled fiducials, and a prediction of fiducial input (bottom right in this case):

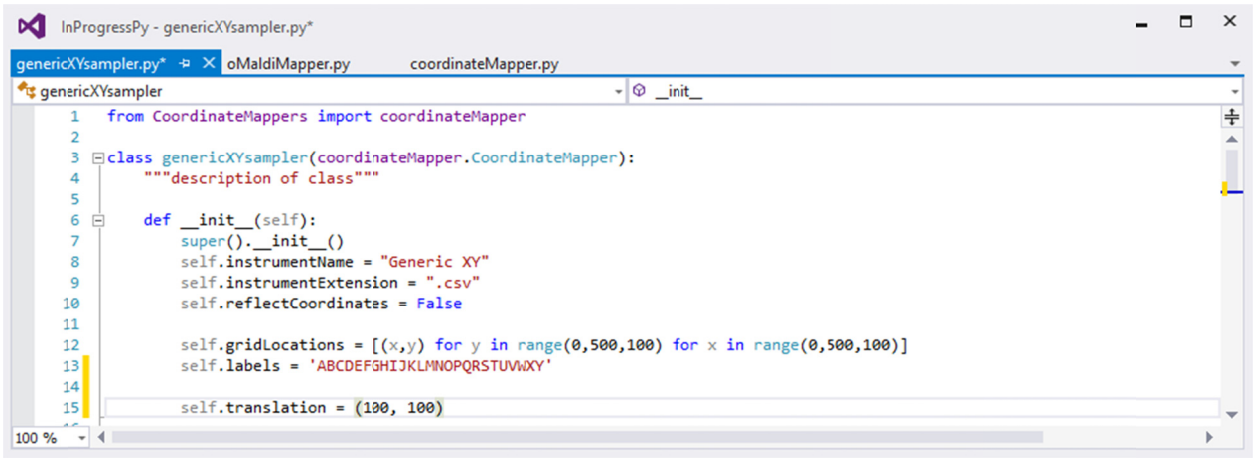


Before moving to the intermediate mapper, note that entering 'Y' is not a valid entry as it isn't in the form of <Xcoordinate>_<Ycoordinate>. To correct this, isValidEntry and extractPoint would need to change to accept letters and map a letter to its motor coordinate. brukerMapper uses a similar method for predicting locations.

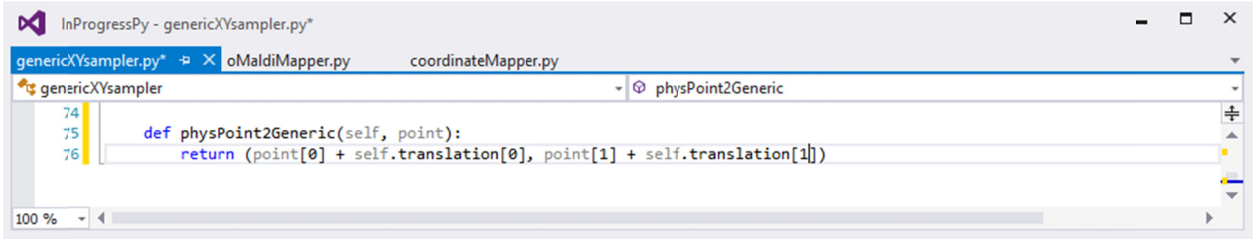
Intermediate maps are a way to move between a coordinate system readily accessible to the user (e.g., motor coordinates) and a system used by the instrument (e.g., GenericCoordinates). microMS passes intermediate training points as lists of tuples, with the first element being a set point (A-Y, here) and the next two being an X and Y coordinate. The list populates the table in the *Grid Settings* in the *Tools* menu. In most applications, the coordinates would need to be stored as a separate file to save intermediate maps between executions. Here, the values will initialize as a static map and updates will last only during execution.

In coordinateMapper, the intermediate map could immediately convert physical coordinates to the second coordinate system and store them in self.physPoints. This would simplify saving instrument positions, but if the intermediate map changes, the current registration will be invalid as it is based on a previous intermediate map. Also, since the user typically cannot access the intermediate coordinate system it is difficult to examine the registration file later to assess its accuracy. For these reasons, microMS utilizes the motor coordinates for physical points and converts them only on saving instrument files.

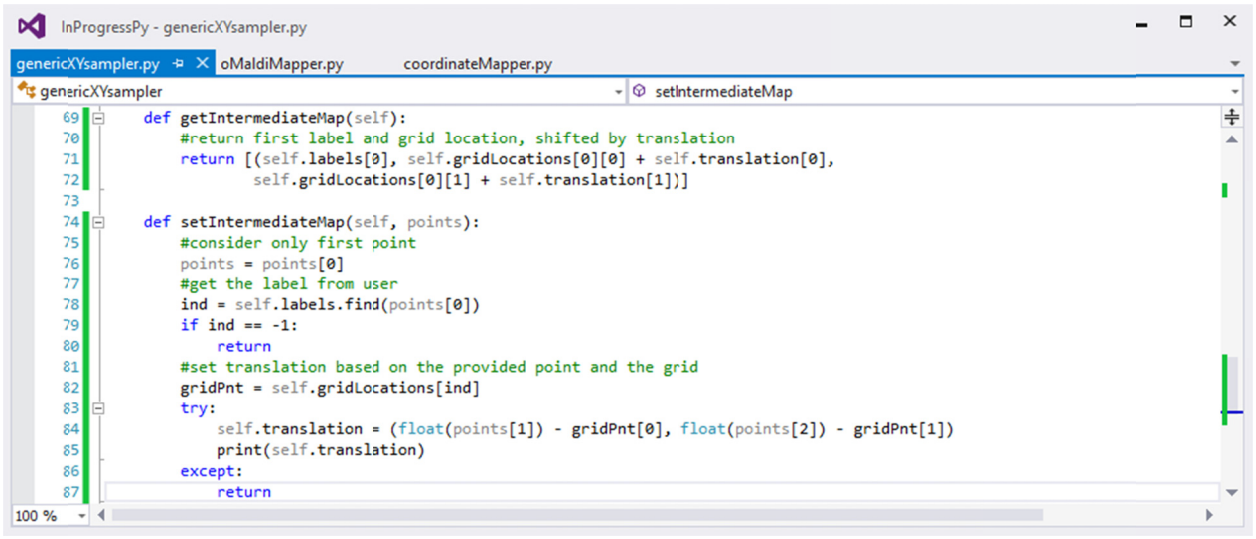
To further simplify genericXYsampler, we will assume only translation is possible and is exactly specified with one point. In practice a point based registration is required, see oMaldiMapper or brukerMapper for examples. A new variable is needed to store the translation as well as a method to map from motor to GenericCoordinates:



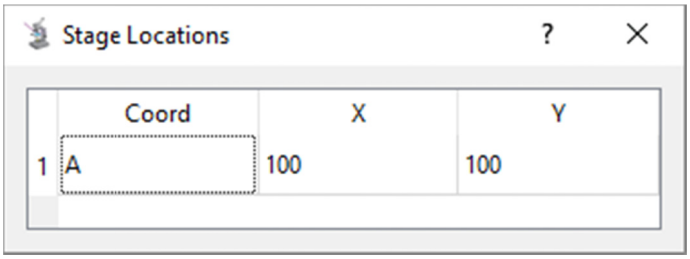
Where (100,100) is from the specification.



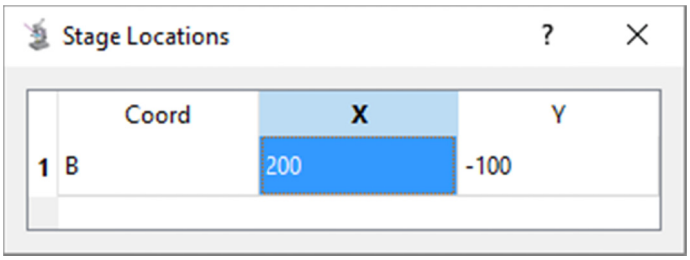
Now the get and set intermediateMap implementations:



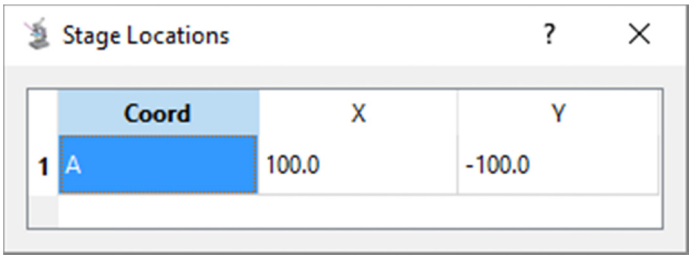
'Get' populates the single table row from the first point in label/grid. Note the return type is a list of triples. Set utilizes the first point to recalculate translation, with minimal error checking on the user input. Upon running microMS, selecting the generic instrument, and opening the instrument settings:



Each entry can be modified, for example:

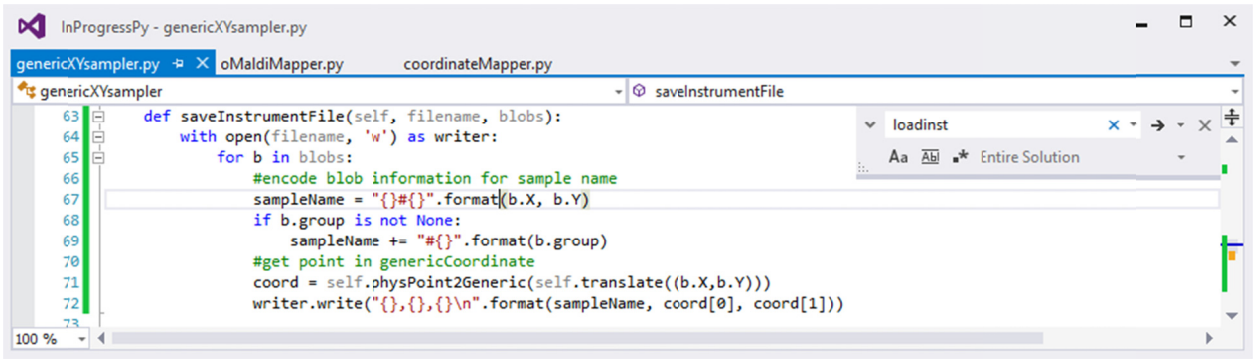


Which should set the translation to (100, -100) once the window is closed. The result is verified by the print statement. Opening the instrument settings again shows the new value of coordinate A:

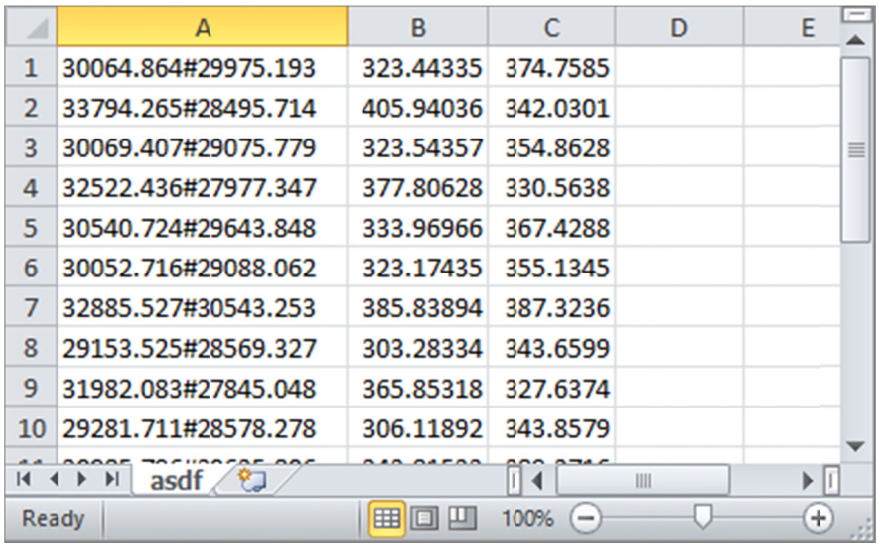


Finally, save and load instrumentFile need implementations. There are few requirements from microMS on saving an instrument file, allowing arbitrary format specific for each instrument. Loading an instrument file requires the method to return a new list of blobs (not a blobList) with the original x,y pixel position and group (if relevant). MicroMS makes no attempt to record blob radius and circularity in instrument files as instrument files specify target *positions* instead of actual blobs. However, the pixel positions and groups are required to correlate those targets back to an image and pattern. Where possible, microMS encodes the coordinates as sample names. For instruments where the instrument file cannot have a sample name, it may be useful to save the x, y, group data as a separate metadata file (see oMaldiMapper).

For genericXYsampler, the blob information will be encoded in the sample name as <X>#<Y> or <X>#<Y>#<Group> as appropriate. The X and Y coordinates in GenericCoordinates require the blob *pixel* positions to be transformed by self.translate to motor coordinates, and then shifted by self.translation, before writing the contents. Here is an implementation of saveInstrumentFile:



Running the code generates a csv like this:



	A	B	C	D	E
1	30064.864#29975.193	323.44335	374.7585		
2	33794.265#28495.714	405.94036	342.0301		
3	30069.407#29075.779	323.54357	354.8628		
4	32522.436#27977.347	377.80628	330.5638		
5	30540.724#29643.848	333.96966	367.4288		
6	30052.716#29088.062	323.17435	355.1345		
7	32885.527#30543.253	385.83894	387.3236		
8	29153.525#28569.327	303.28334	343.6599		
9	31982.083#27845.048	365.85318	327.6374		
10	29281.711#28578.278	306.11892	343.8579		

The sample name formatting is ugly as it displays the entire floating point number, but this is adjustable in the format call.

Loading the instrument file requires parsing just the sample name to populate a new blob list (don't forget to import blob from ImageUtilities):

Now genericXYsampler is completely functional!

Implementing brukerMapper for the Bruker flexArmstrong

To combat the rising popularity of the genericXYsampler, Bruker has introduced its equally fictional flexArmstrong mass analyzer. As mentioned earlier, brukerMapper is another abstract base class within microMS that handles many common functions of Bruker instruments, including handling xeo files, fractional distances, and an intermediate map between motor coordinates and xeo positions. The slideIladapter xeo geometry file defines a set of coordinates for a microscope slides which are referenced by brukerMapper. The brukerMapper interface greatly simplifies the addition of a new Bruker instrument, assuming the xeo files are identical. The solarixMapper and ultraflexMapper are both implementations of brukerMapper and can be used as further examples. ultraflexMapper is a simple implementation that uses motor coordinates separated by a space and saves a single XEO file for each blob list. solarixMapper is slightly more complicated as it populates fiducial locations from the clipboard, automatically splits lists into xeo files of 400 points (a software maximum), and generates xlsx files for starting autoexecute in the instrument control software. flexImagingSolarix is another mapper that inherits from solarixMapper, but defines a different set of instrumentFiles for use in flexImaging.

Classes inheriting from brukerMapper must:

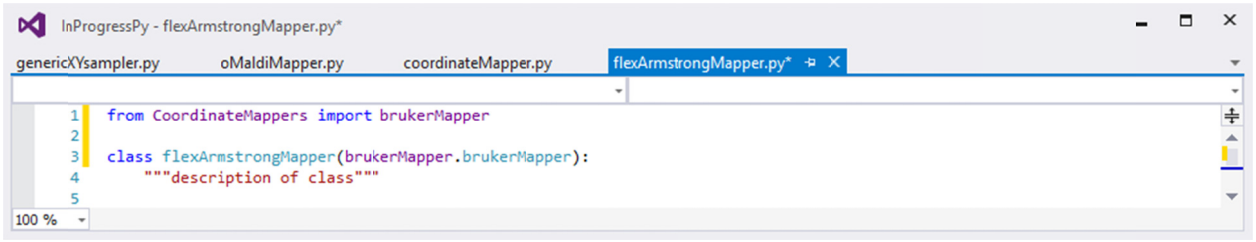
- Define motorCoordFilename to store intermediate maps between runs of microMS, an instrumentExtension, instrumentName and reflectCoordinates
- Implement isValidMotorCoord, extractMotorPoint, loadInstrumentFile and saveInstrumentFile.
- Add an import and initialize an instance in supportedCoordSystems.

The flexArmstrong is a new mass analyzer with control software and autoexecute functions similar to the ultraflex:

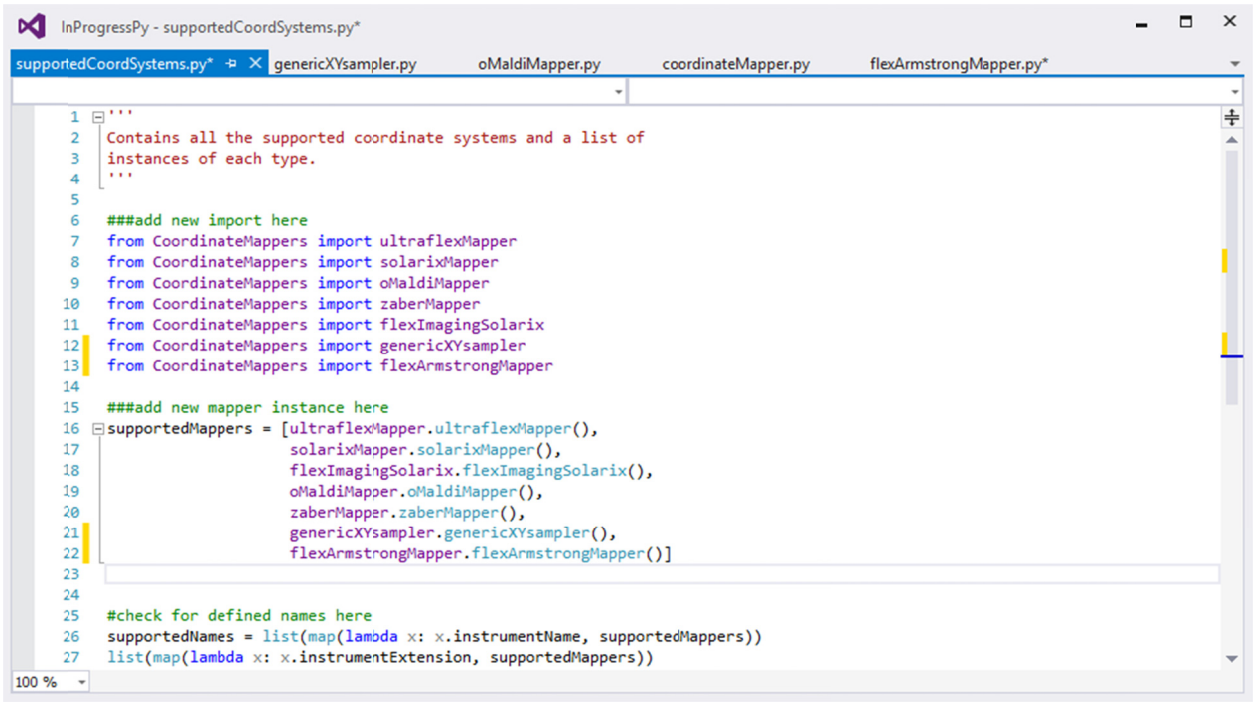
- It handles arbitrarily long xeo files and requires no additional files for performing automatic acquisition.
- Motor coordinates are input as <X>$<Y>.
- Motor coordinates are equal to the fractional distances multiplied by 1000. The Y coordinate *decreases* as the stages moves down the slide.
- The slideIladaptor is supported and unchanged from previous version.

Note that the last feature is the only requirement for inheriting from brukerMapper.

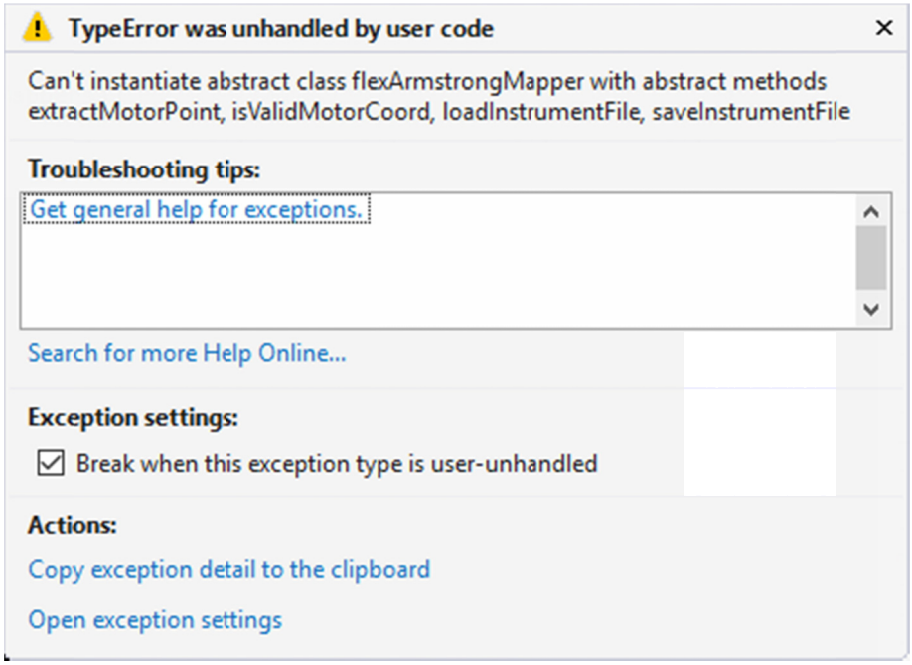
Similar to the genericXYsampler, create a new class in coordinate mappers called flexArmstrongMapper which inherits from brukerMapper:



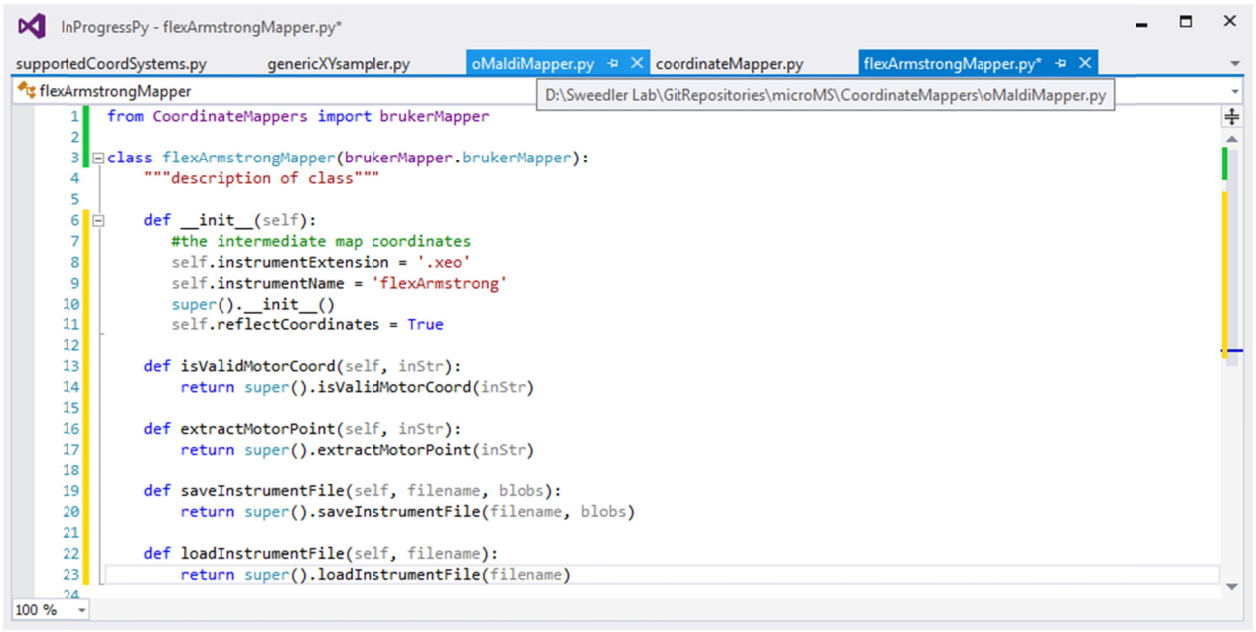
And add the mapper to supportedCoordSystems



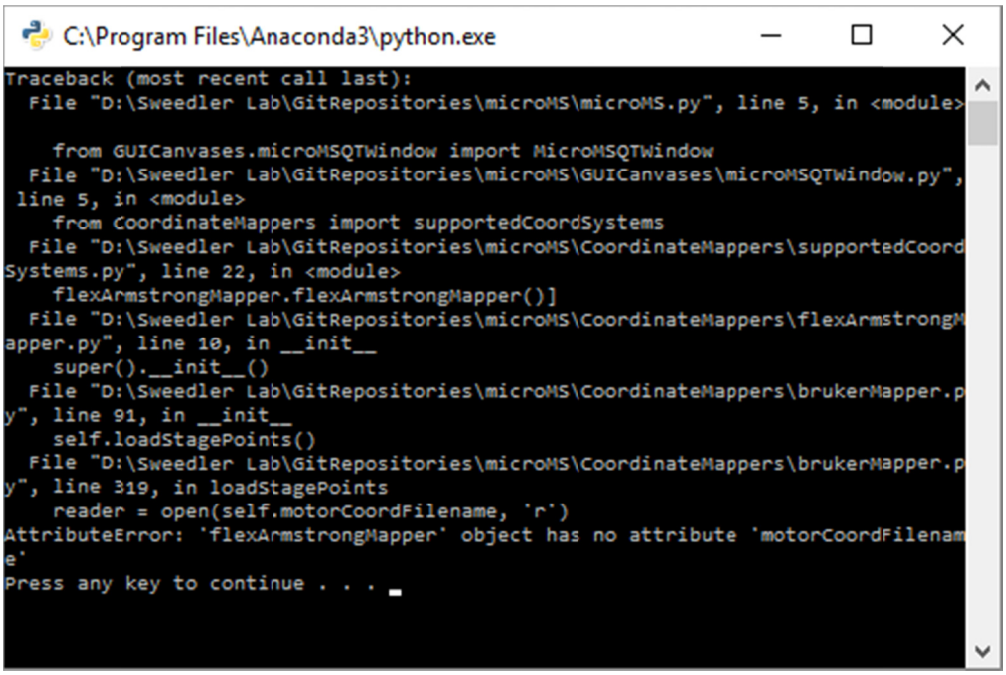
Running again generates errors for missing methods and missing variables:



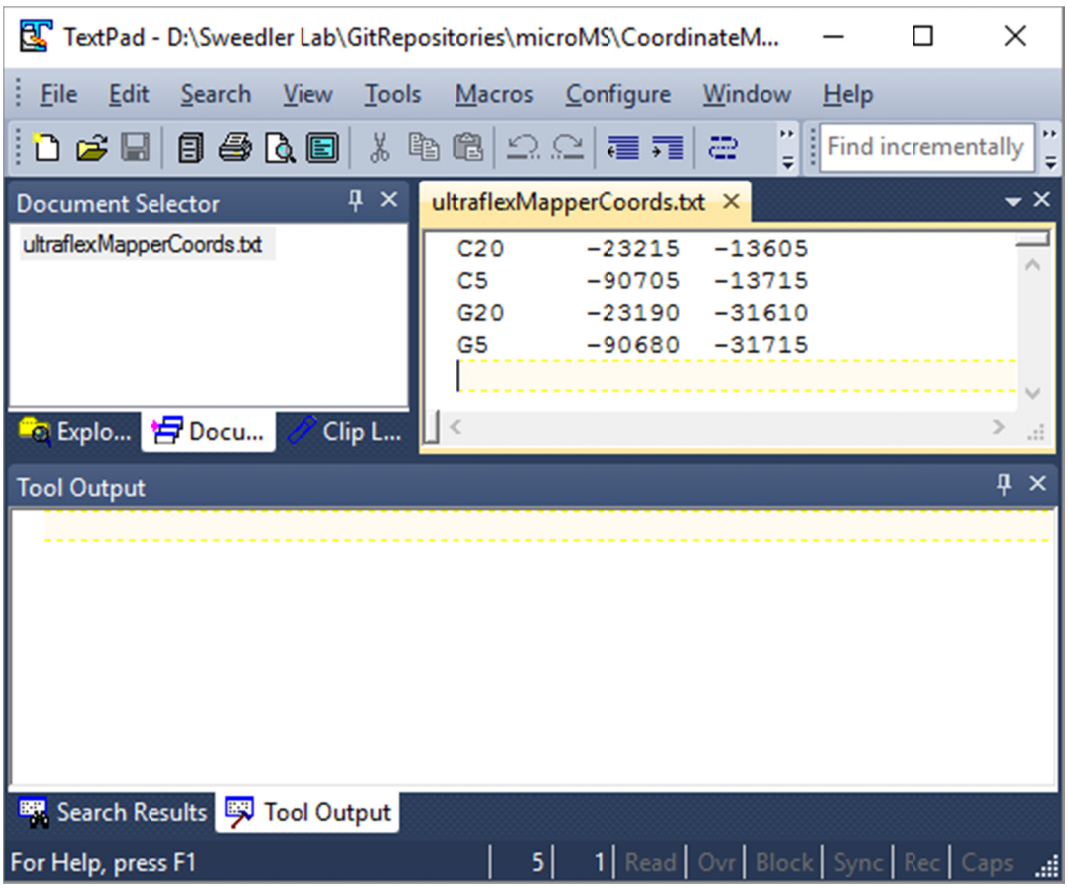
Adding in method stubs and instrumentExtension, name and reflectCoordinates:



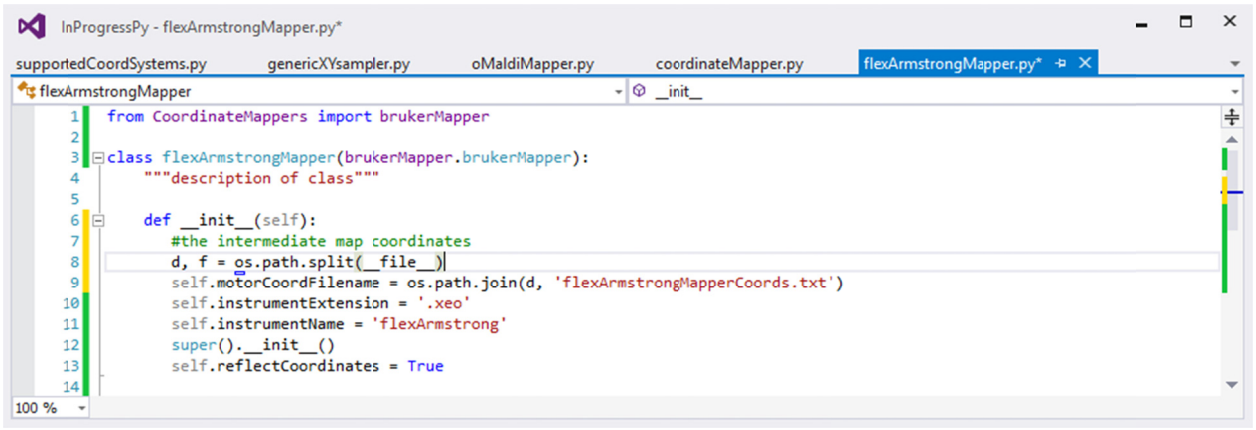
Where reflectCorodinates is True since the Y motor coordinate *decreases* as the stage moves down the image (and pixel positions increase). Running again produces a new error:

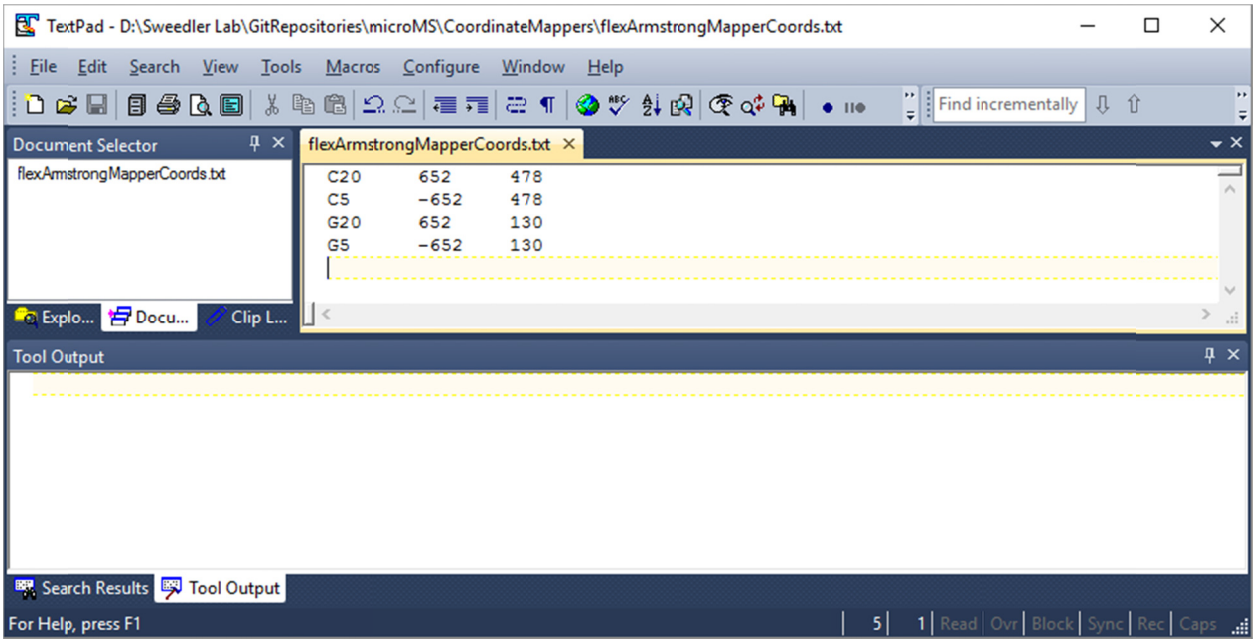


Which complains about the lack of a motorCoordFilename. The filename can be anywhere on the operating system, though for simplicity microMS uses the directory containing the coordinateMappers. The actual file should be tab delimited text, similar to the following for ultraflex:

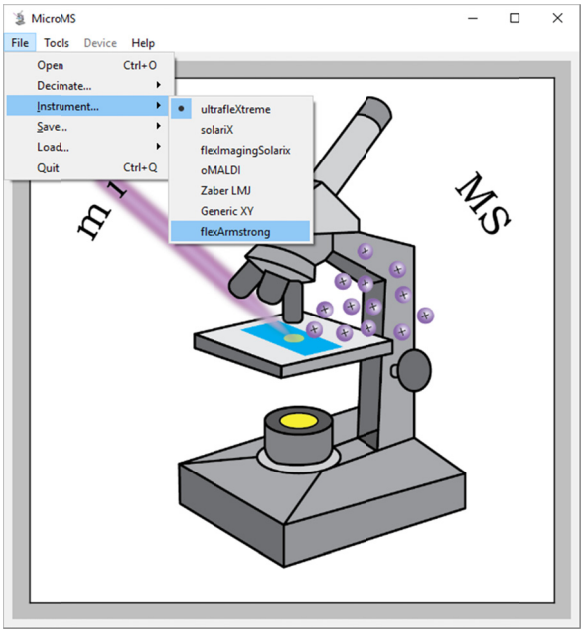


Where the first column corresponds to an xeo position on the slideIladapter and the next two columns are the x and y coordinates. brukerMapper takes care of reading this file, presenting it to the user and applying changes to set the intermediate mapper. Write xeo also handles performing a similarity registration on the intermediate mapper to generate fractional distances. What is needed is a unique filename and corresponding text file with the initial coordinates:

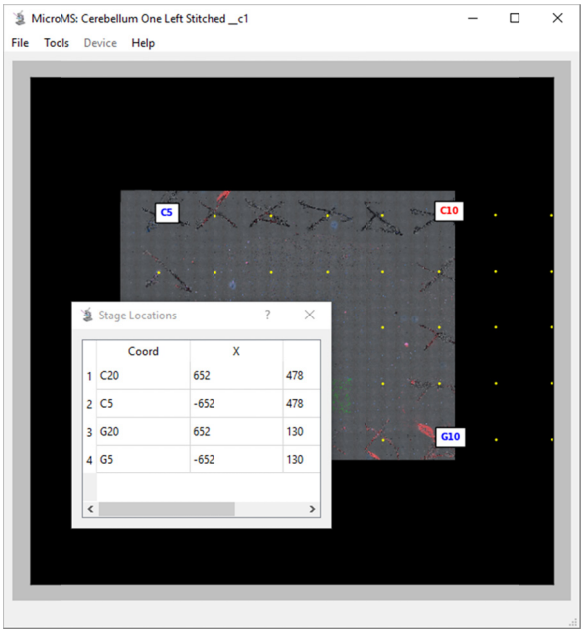




Note that motorCoordFilename must be defined prior to calling super().__init__(), which will try to read the file. The values in the Coords file were chosen based on the third specification and are the fractional distances multiplied by 1000. Now microMS is running, though the mapper is nonfunctional:

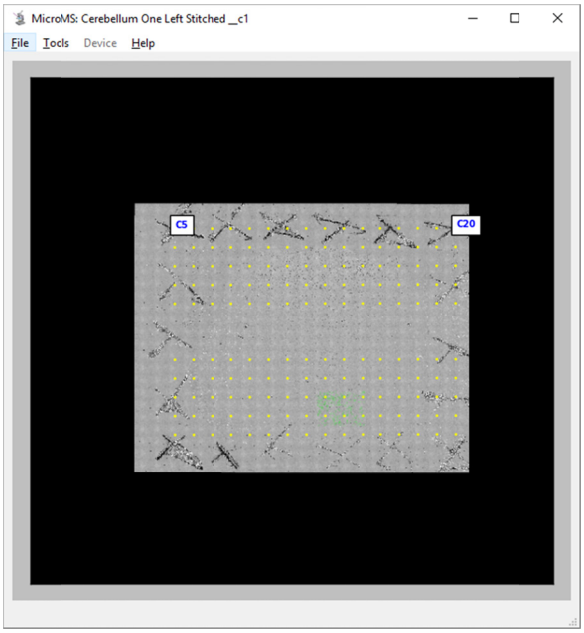


But it is closer to functional than you might think based on the amount of work for the genericXYsampler. Already the flexArmstrongMapper handles named MTP coordinates (like C5), generate predicted names, labels and points, and get and set intermediate maps!

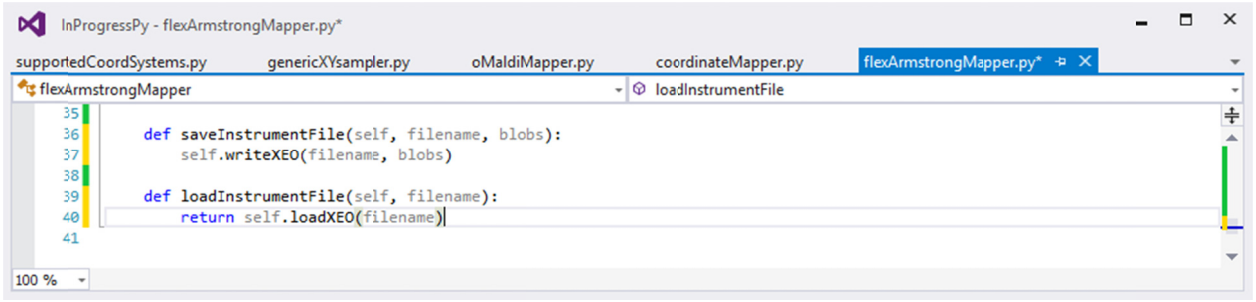


What it *cannot* do yet is utilize motor coordinates as fiducial locations or save/load instrument files. Motor coordinate methods are fairly simple and are almost copied directly from the genericXYsampler for extracting entries:

Since brukerMapper accepts named positions (C5) in addition to motor coordinates, extractPoint must check for a valid MTP or motor coordinate and call the appropriate extraction method. Classes inheriting from brukerMapper only have to handle motor coordinates as the MTP cases are identical. Now the flexArmstrongMapper properly handles motor coordinates:



And finally loading and saving instrument files. As nothing special is required of the xeo files, the base methods of brukerMapper are suitable to load and write XEO files:



And that's it! When testing initial accuracy, it is important to save the Fiducial Positions and check that their locations are accurate. Large deviations indicate an issue with some aspect of the mapper code. While it is impossible to predict future tweaks in new instruments, this should act as a starting guide to getting things up and running.

Direct instrument control

This section describes the organization of code and operation of microMS when used for direct instrument control. As the only supported instrument is a lab-built xyz stage prototype, it is highly unlikely to be useful to general readers, but can assist with attempts to integrate another instrument.

Code Organization

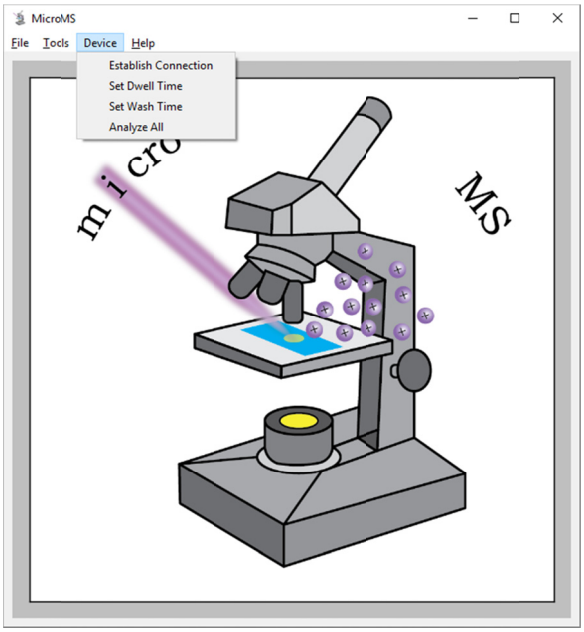
Instrument control with microMS is performed through a coordinateMapper with an instance of a connectedInstrument. ConnectedInstrument is another abstract base class that defines a set of required methods for interacting with connected instruments, such as moving, getting a position, and collecting from positions. The implemented instrument is a Zaber xyz stage used for liquid extraction. microMS interacts with the zaberMapper implementation of the coordinateMapper. This is a fairly simple coordinateMapper as there is no intermediate map or predicted labels. The novel aspects are a connectedInstrument, discussed more below, and directly reading the stage position for predicting the name of a fiducial. The predicted name is then directly read as a fiducial location. Predicted points are also generated from the current stage position.

The connectedInstrument of zaberMapper is a zaber3axis object, which inherits from connectedInstrument and zaberInterface. The zaberInterface is another abstract base class which has a number of wrapper methods and a dictionary of commands to simplify communication with Zaber linear stages. zaber3axis implements communication with the stage including device renumbering, stage initialization, movement and collection. The main GUI interacts with the connectedInstrument of a coordinateMapper, when the instrument is not None and instrument.connected is true.

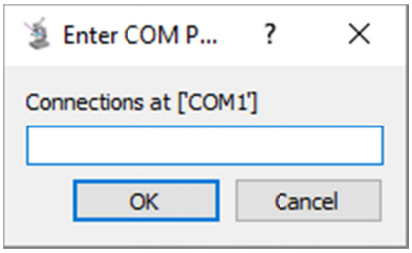
Direct instrument operation

To begin direct instrument control, first select an instrument which has a non-None value for connectedInstrument, currently ZaberLMJ is the only such mapper. With a valid mapper, the *Device* tab

in the menu bar becomes enabled, providing additional options for interacting with connected instruments:



Selecting *Establish Connection* causes the following window to popup:



With a list of valid connections. This is only functional with a Windows computer, but adjusting serial communication would enable it with other operating systems. Entering a valid COM port and clicking OK causes microMS to attempt communication at the port and calls homeAll(). All calls to zaber3axis are blocking so the GUI will appear to freeze while the stage is actively moving.

The stage is moved with the following hotkeys:

- i, j, k, l moves the stage up, left, down, right a small amount, respectively.
- *Shift* and i, j, k, l moves the position 100 times farther than the small step.
- *Shift* and *Ctrl* with i, j, k, l moves the stage 10 times farther than a small step.
- + moves the probe up, or sets a focus
- - moves the probe down. *Shift* and *Ctrl* function similar to i, j, k, l. Additionally, *Shift*+*Ctrl*+*Alt* causes the probe to take a giant step, equal to 1000 times a small step.

Once a fiducial is located on the stage and image, its position is trained by pressing the right mouse button on the image location. The current stage position is read as a predicted point which is directly used in training. As before, *Shift* + RMB removes the closest fiducial and the worst fiducial is shown in red.

With at least 2 fiducials, the stage is moved to a position on the image by pressing *Alt*+LMB. Pressing P to toggle predicted points will cause the current stage position to be read during GUI redraws and displays the probe location as a yellow circle. Note that this function causes lag during stage interactions as each movement triggers a redraw and additional stage communication. Alternatively, pressing *Ctrl*+F will display the stage x,y and z position in motor coordinates in the statusBar.

To perform a collection or measurements, the probe position must be set. With the probe in the correct position, press *Shift* + V to set the position. This also causes the probe to retract. After the position is set, the probe is moved in and out of acquisition position by pressing V. To collect at an arbitrary location, press X. This moves the probe into position and collects for the amount set in *Set Dwell Time* in the *Device* menu. If the wash time (in *Set Wash Time*) is not 0, the probe will then move into its final position for the specified amount of time. After washing, all stages are homed. Setting Wash Time to -1 causes the probe to stay in its final position until the user homes the stages. If wash

time is 0, the probe will simply retract, staying in the same x,y location. Pressing H causes all stages to home, *Shift* + H moves the probe to the final position and stays there.

The final available function is to collect all, in *Analyze All*. This causes the stage to first home, then visit each target blob location for the dwell time. After visiting all blobs, the stage will either home, move to final position, or move to final position for "wash time" and then home if wash time is 0, -1 or a positive value respectively.