Aro spotFinding Suite v2.5 User Guide

A machine-learning-based automatic MATLAB package to analyze smFISH images.

By Allison Wu and Scott Rifkin, February 2015

1. Installation

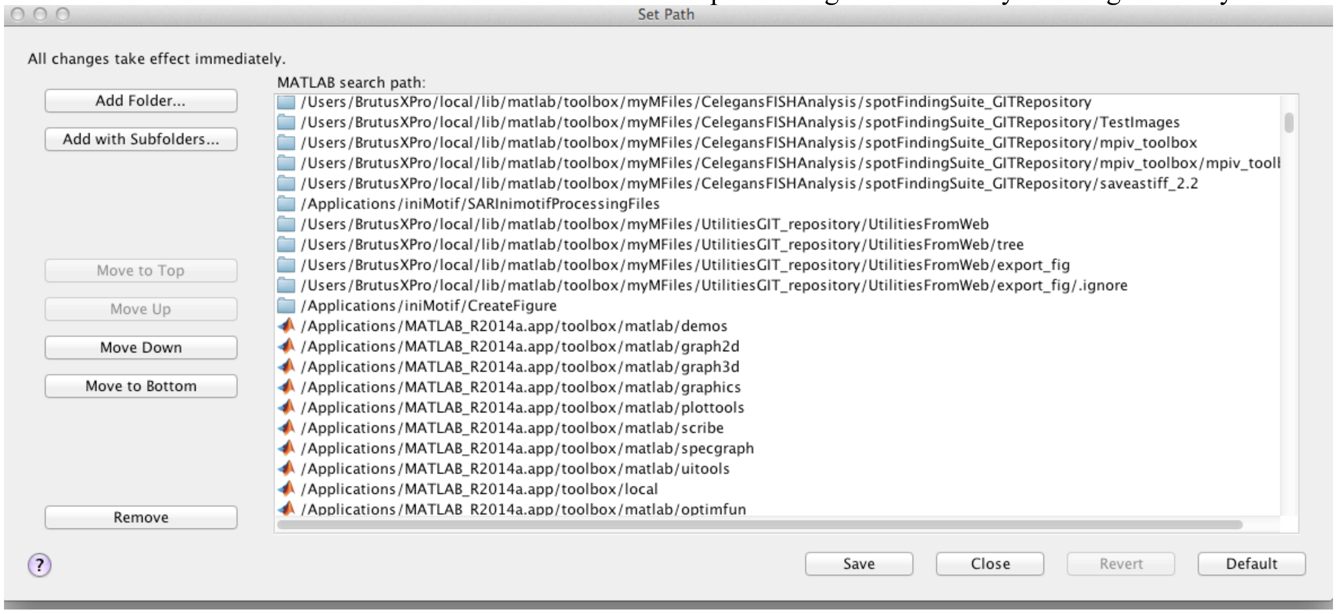
1. Requirements

This software was developed in MATLAB 2014a and has been tested on both Mac and PC. Some functions might not work in earlier versions but the suite should be able to work on either OS platform.

- The user needs basic MATLAB knowledge to utilize the output results.
- TIFF or STK are two currently supported image formats.
- It relies on the MATLAB statistical toolbox
- Third-party functions are included with their licenses in the distribution.
- The example dataset can be downloaded from (~800MB zip file): http://labs.biology.ucsd.edu/rifkin/software/AroSpotFindingSuite/AroSpotFindingSuite_ExampleDataset.zip

2. Installation

You can install from github (https://github.com/evodevosys/AroSpotFindingSuite). Then go to File > Set Path in MATLAB. Press 'Add with Subfolders' to add the directory that saves the spotFinding suite (Fig. 1) and save. One should be able to utilize all the functions in the spotFinding suite from any working directory.



All changes take effect immediately.

MATLAB search path:

```
/Users/BrutusXPro/local/lib/matlab/toolbox/myMFiles/CelegansFISHAnalysis/spotFindingSuite_GITRepository
/Users/BrutusXPro/local/lib/matlab/toolbox/myMFiles/CelegansFISHAnalysis/spotFindingSuite_GITRepository/TestImages
/Users/BrutusXPro/local/lib/matlab/toolbox/myMFiles/CelegansFISHAnalysis/spotFindingSuite_GITRepository/mpiv_toolbox
/Users/BrutusXPro/local/lib/matlab/toolbox/myMFiles/CelegansFISHAnalysis/spotFindingSuite_GITRepository/mpiv_toolbox/mpiv_tooll
/Users/BrutusXPro/local/lib/matlab/toolbox/myMFiles/CelegansFISHAnalysis/spotFindingSuite_GITRepository/saveastiff_2.2
/Applications/iniMotif/SARInimotifProcessingFiles
/Users/BrutusXPro/local/lib/matlab/toolbox/myMFiles/UtilitiesGIT_repository/UtilitiesFromWeb
/Users/BrutusXPro/local/lib/matlab/toolbox/myMFiles/UtilitiesGIT_repository/UtilitiesFromWeb/tree
/Users/BrutusXPro/local/lib/matlab/toolbox/myMFiles/UtilitiesGIT_repository/UtilitiesFromWeb/export_fig
/Users/BrutusXPro/local/lib/matlab/toolbox/myMFiles/UtilitiesGIT_repository/UtilitiesFromWeb/export_fig/.ignore
/Applications/iniMotif/CreateFigure
/Applications/MATLAB_R2014a.app/toolbox/matlab/demos
/Applications/MATLAB_R2014a.app/toolbox/matlab/graph2d
/Applications/MATLAB_R2014a.app/toolbox/matlab/graph3d
/Applications/MATLAB_R2014a.app/toolbox/matlab/graphics
/Applications/MATLAB_R2014a.app/toolbox/matlab/plottools
/Applications/MATLAB_R2014a.app/toolbox/matlab/scribe
/Applications/MATLAB_R2014a.app/toolbox/matlab/specgraph
/Applications/MATLAB_R2014a.app/toolbox/matlab/uitools
/Applications/MATLAB_R2014a.app/toolbox/local
/Applications/MATLAB_R2014a.app/toolbox/matlab/optimfun
```

Fig. 1 Add the Aro spotFinding Suite folder to MATLAB's set path.

- **The following steps that are marked with '*' take more than an hour for a batch of data with ~40 images but since these commands can operate automatically, no hands-on time is needed.**
- **Each function is annotated with detailed explanation. Please use 'help' for further details. e.g. `help createSegImages`**
- **All the files created by the function will show up in the working directory and all the functions will only search under the working directory.**

2. Preview and directory structure

Running Aro spotFinding Suite consists of the following series of functions:

1) createSegmenttrans

2) createSegImages
3) doEvalFISHStacksForAll
4) createSpotTrainingSet [GUI]
5) trainRFClassifier
6) classifySpots/classifySpotsOnDirectory
7) reviewFISHClassification [GUI]
8) spotStatsDataAligning2

Each will be described in more detail below. Each function is also annotated with a detailed explanation. Use MATLAB's help command for further details.

The software is capable of handing two types of directory structures: *flat* and *nested*. In the *flat* structure, all the raw image files and all the files generated by the software are contained within a single directory. The suite should be run from this directory. However, the software generates at least three files for each raw image file with some additional ones. If there are a lot of raw image files, this leads to a very large directory. We have noticed that this can slow down the runtime, particularly on a Mac. The default *nested* directory structure looks like (two dyes in this example):

```
MainDirectory >
       ImageData >
              dye1 >
                     dye1_Pos1.tif
                     dye1_Pos2.tif
                     ...
              dye2 >
              dapi >
              trans  >
              thumbs >
       Analysis1 >
              SegStacks >
                     dye1 >
                            dye1_Pos1_SegStacks.mat
                            ...
                     dye2 >
                     dapi >
              SpotStats >
                     dye1 >
                            dye1_Pos1_spotStats.mat
                            ...
                     dye2 >
              TrainingSets >
                     dye1_gene1_RF.mat
                     dye2_gene2_RF.mat
                     trainingSet_dye1_gene1.mat
                     trainingSet_dye2_gene2.mat
              WormGaussianFit >
                     dye1 >
                            dye1_Pos1_wormGaussianFit.mat
                            ...
                     dye2 >
```

The directory structure (*flat* or *nested* as well as the names and locations of the relevant directories) are set in a file called Aro_parameters.m

Modifying Aro_parameters.m

Aro spotFindingSuite has very few parameters that need to be set. These are of three types: (a) Conservative thresholds that mainly increase efficiency (see doEvalFISHStacksForAll below); (b) parameters that control the random forest – how many trees to run, how pick variables, etc.; (c) confidence interval width. These, along with the directory specifications above, can be found in the file *Aro_parametersTEMPLATE.m*. The user should copy this file to the MainDirectory that either contains all the image files (*flat*) or is as described above (*nested*), modify the values as desired, and rename it *Aro_parameters.m.* There are instructions within the file that describe what the parameters are and how to modify them. The default settings can always be recovered from the *Aro_parametersTEMPLATE.m* file.

3. Getting started

Aro is agnostic as to what actual biological specimens are being analyzed, whether cells or embryos or other things. Below we refer to the specimens being analyzed generically as *objects*. Note that *cell* is a specific type of data structure in Matlab and so this is what the word *cell* refers to below. Note that in the source code, objects are often referred to as 'worms' because the software was originally developed using FISH images from worms.

1. Create Masks for Your Images and Get the File Formats Correct

   The masks are logical images that have entries of 1 where the objects are in the image and 0 denotes the space with no objects (Fig. 2). These masks are necessary for reducing the amount of memory needed for analyzing each image and can ensure proper scaling within the objects. However, Aro does not provide a way for the users to create masks for each image because there are currently many segmentation algorithms that can segment different kinds of images efficiently and automatically. The users should find their own ways to create the masks for their images. The Rifkin lab currently has another simple semi-automatic segmentation program for worm images, and we will happily share it with any other interested labs. However, segmentation is beyond the scope of this user guide. Here we will only discuss what one could do after all the segmentation masks are generated.

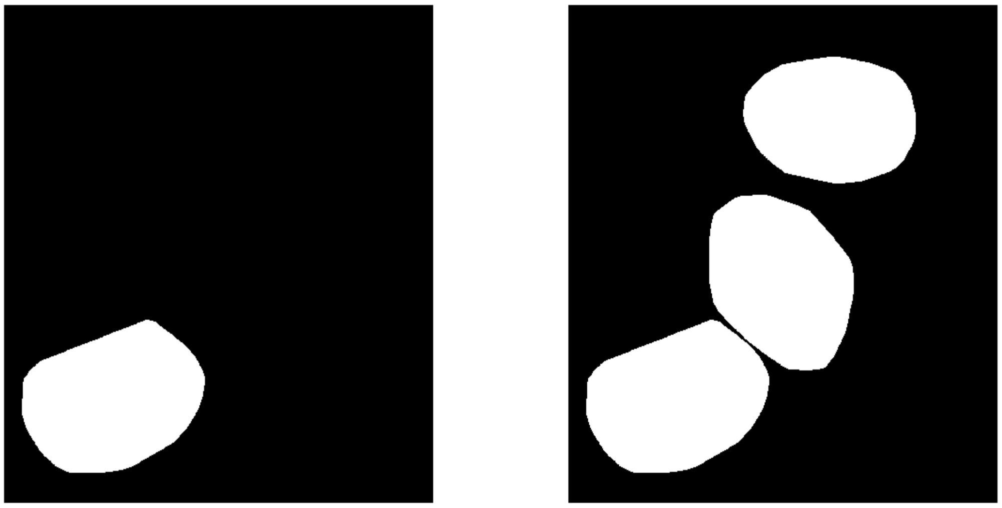


Fig.2 A mask that has one single object (left) and another mask that has multiple objects (right). Both of the masks are 1024 x 1024 pixels.

It is recommended that each image is segmented into no more than 5 masks for all the analyses. Each mask can have different number of objects in it as long as the exposure levels of all the objects in the same mask can be scaled evenly. However, the total spot number estimate that the program will output is for each mask. To get a total spot number estimate per object, one still need masks with single objects but these can be applied after all the analysis is done. This will speed up the analysis and reduce the number of files generated. The example file contains an example of typical worm images that are best dealt with masks with single objects because worm embryos sometimes have different background exposure level in the same image so it would not be appropriate to scale their intensities and analyze them all together.

After creating masks for each image, the user has to make the file formats recognizable for the following steps. Note that the curly braces used below just designate the variable parts of the names. Do not include curly braces in your actual file names:

- Please make sure each of the tif or stk files has a 3-dimensional image stack and that the z-axis order is the same as the real z-axis order.
- Make sure the mask files have the same x-y dimension as the image stack; that is, if the image stack has a size of 1024 x 1024 x 30, then each of the mask images should be 1024 x 1024.
- The entries in the mask file should be class uint8, uint16, uint32 or singles or doubles.
- Please name the image file names as: {dye}_{Position Identifier}.tif with no underscore within the text bounded by the curly braces. (remove the curly braces
  For example:
  - ✓ tmr_Pos1.tif
  - ✓ cy5_001.tif
  - ✗ tmr_Pos_1.tif
  - ✗ Cy5001.tif
  - ✗ TmrPos01.tif
- Make sure the mask files have the following naming pattern so that the suite can pair them with the correct image stacks: Mask_{Position Identifier}_{Mask Number}.tif, e.g. Mask_Pos1_1.tif is the mask for the first mask for the first image.
  - ✓ Mask_Pos1_1.tif
  - ✓ Mask_001_1.tif
  - ✗ Mask001_1.tif
  - ✗ MaskPos1.1.tif
  - ✗ MaskPos1-1.tif

When all the above mentioned criteria are met, one can use the following MATLAB command to create the mask file format needed for the suite.

MATLAB Command :
```
>> createSegmenttrans(positionIdentifier)
```
For example:
```
>> createSegmenttrans('Pos1')
```

2. Getting Your Images Ready *

MATLAB Command :
```
>> createSegImages('tif')
```

This command creates a {dye}_{Position Identifier}_segStacks.mat file for each image, e.g. cy5_Pos10_segStacks.mat is the segStacks file for position 10 in the cy5 channel. This mat file contains three cell variables: segStacks, segMasks, and BoundingBoxes. Each element of the cell variable, segStacks, contains a segmented image in a numerical matrix for each individual cells in this image and its counterpart in segMasks contains a logical matrix of the mask for the individual cell. These images are NOT the same size as the full image because to save on memory, the suite only saves the minimal rectangle (in x-y) necessary to contain the object (1s) indicated by the each mask (Fig.1) with an additional boundary 3 pixels wide around the outside (unless this is not possible because the specimen is too close to the edge of the image). From this point on, all the analyses use these segStacks.mat files and not the original image files. Each element of the BoundingBoxes variable contains the BoundingBox (see MATLAB's regionprops command) for the corresponding mask. Note that the segStack for a particular object will also contain pixels outside that object - they are not set to zero. However, in downstream analyses the software only pays attention to pixels inside an object as described by the corresponding segMask.

The program currently calculates statistics on a 7x7 square of pixels and so it is assumed that the spots in your image fit nicely within 7x7 pixels. (See the example image files). If your spots are bigger or smaller, it would be best to rescale the image so that they fit into a 7x7 square. Future modifications may include the ability to work with larger or smaller spots, but this will require finding a way to calculate scale-independent statistics or to programmatically change that statistics to reflect the spot size.

Note: Currently, the suite supports TIFF files and STK files. You can specify the file type as the input to createSegImages. Support for other file formats will be included in a future release. In the meantime, interested users could convert their images to TIFFs using other programs such as imreadBF() on the MATLAB file exchange.

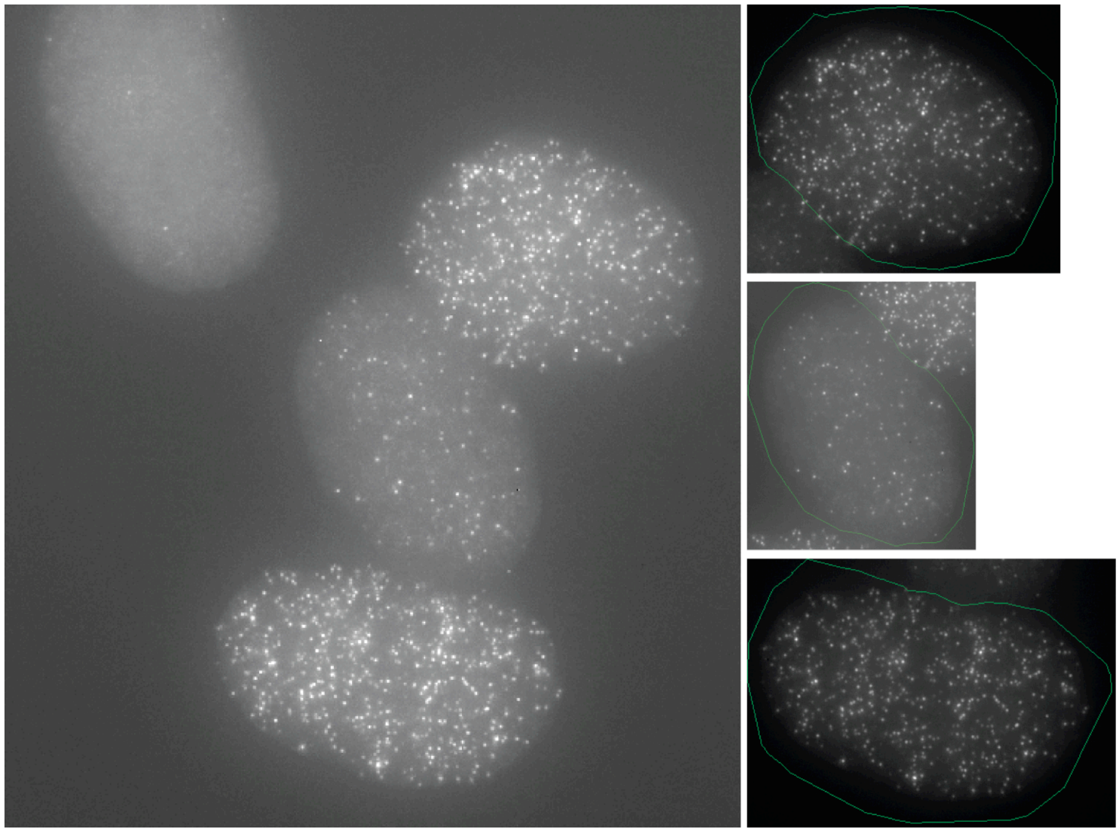


Fig. 3 (Left) A maximum projection cy5 image. (Right) Three segmented individual cell images saved in the segStacks.mat file. The green outline is the border of the segmentation mask describing the object boundary.

3. Find the Candidate Spots in Each Cell *

MATLAB Command :
```
>> doEvalFISHStacksForAll
```

The next step goes through each segStack in the segStacks.mat files, except for DAPI ones, finds the local maxima in 3D, and computes statistics that describe each maximum. These statistics include features that describe the shape features including how well a local maximum fits to a 2D Gaussian distribution, which reflects the fact that each spot is a diffraction-limited spot, etc. To see a full list of the features calculated, please refer to Appendix 1.

If the software were to process every local maximum, it would waste most of its time and memory on analyzing spots that are obviously bad. Instead it pre-filters out spots that are extremely unlikely to be a good spot by ignoring local maxima where one of the features - the scaled coefficient of determination from the fit to a 2D Gaussian - is below the specified threshold. The default setting is a very conservative setting that, based on our empirical explorations, will not exclude any good spots. All the statistics of each spot of each object are saved in the {dye}_{Position Identifier}_wormGaussianFit.mat files. Each file contains a cell array variable called 'worms', the elements of which save the spot information for each object in the image. To access the spot information for a particular object in a particular position, you need to first load in the wormGaussianFit.mat file for the specific image and type 'worms{object number in the cell array}' to view its statistics.

Example: To access the 2nd object in position 3 in the cy5 channel...

```
>> load cy5_Pos3_wormGaussianFit.mat
>> worms{2}
ans =

                 version: 'v2.5'
            segStackFile: 'cy5_Pos3_SegStacks.mat'
          numberOfPlanes: 35
              cutoffStat: 'scd'
    cutoffStatisticValue: 0.7
        cutoffPercentile: 70
           bleachFactors: [35x1 double]
             regMaxSpots: [68246x5 double]
         spotDataVectors: [1x1 struct]
               goodWorm: 1
         functionVersion: {3x1 cell}

>> worms{2}.spotDataVectors

           locationStack: [758x3 double]
                rawValue: [758x1 double]
           filteredValue: [758x1 double]
                spotRank: [758x1 double]
                 dataMat: [758x7x7 double]
               intensity: [758x1 double]
            rawIntensity: [758x1 double]
             totalHeight: [758x1 double]
                                       …
       cumSumPrctile30RP: [758x1 double]
         cumSumPrctile90: [758x1 double]
         cumSumPrctile70: [758x1 double]
         cumSumPrctile50: [758x1 double]
         cumSumPrctile30: [758x1 double]
```

Note: In this object, there are 68246 regional maxima found but only 758 spots are left to be considered after using the cut-off value of 0.7 for the scd variable. By typing worm{2}.spotDataVectors, you can see a list of statistics or features calculated for the 758 spots.

Note also that at this point the worms{x} struct contains information about every candidate spot in an object that will be considered in the rest of the pipeline.

Finally, note that this step can take a while depending on how many images there are to process and how many candidate spots there are in each image. With an Intel Xeon, 2.26 GHz chip, each candidate spot took around 0.05 seconds to process.

3. Analyze the Spots Using the Random Forest Algorithm

1. Create a Training Set

After statistics for all the candidate spots are calculated by `doEvalFISHStacksForAll`, the user needs to prepare a training set to train the classifier. To create a good training set, here are some important points to follow:

- ✓ Because each channel and each batch of data may differ in quality and in the spot characteristics, we suggest that users create one training set for each channel in each batch of data independently so that the training set reflects the spots in each batch.
- ✓ The suite currently does not support using training sets from other batches of data that are not in the same directory. Using training sets from other batch of data will introduce errors in the subsequent functions such as reviewFISHCalssification(). This feature will be implemented in a future release.
- ✓ A good training set should contain should contain clearly good spots, clearly bad spots and some ambiguous spots for which the user may have to make some difficult decisions. As with all supervised learning approaches, the algorithm is only as good as the quality of the training set. We find that as long as there are sufficient numbers of good and bad spots in the training set, it is not necessary to have them strictly balanced. A few hundred spots of each works well.
- ✓ We suggest that the user first examines the max projection images of the particular channel and picks out 2-3 images for training so that the training spots will not come entirely from the same image and so it is assured that there will be a good representation of good spots and bad spots.
- ✓ It usually takes 3-4 rounds of training to get a robust classifier. In other words, the user trains an initial set, sees how it performs, either makes corrections and adds these corrections into the training set using the review GUI or adds more spots using the training GUI, retrains the classifier, and continues until the classifier does an acceptable job. It is better to increase the number of training spots at each round instead of starting with a huge training set since the training time needed is dependent on the number of spots. A training set of 300-400 spots will be a good start.

MATLAB Command :

```
>> createSpotTrainingSet('{dye}_{PositionNumber}','{Probe_name}')
```

Example: to pick out training spots from position 6 in cy5 channel for *C.elegans elt-2* probe, you can use ...

```
>> createSpotTrainingSet('cy5_Pos6','Cel_elt2')
% Note: the probe name (2nd input) is entirely up to the users to decide.  The 1st input should be in the same {dye}_{position} format as described above.
```

Before the GUI opens, the suite will search to see whether there exists any pre-established training set for this probe. If it finds a training set previously established, it will ask the user if he/she wants to overwrite the old training set or simply add new training spots to the training set.

When the GUI is started, a window called *identifySpots* appears. The user should see a 16 x 16 pixel zoomed-in window on the left and the original-sized image on the right. The 'Max. Merged Image' on the lower-right corner is a maximum projection image of the neighboring slices, 2 slices above and 2 slices below and the current slice in the original-sized image.

This GUI allows the users to examine the candidate spots that are ordered by the spot rank, which uses one of the features as a crude quality score. The users can go down the spot rank and annotate each spot as good (Choose *Accept*) or bad spot (Choose *Reject*), or they can pick out some good spots with high spot rank and use the 'Spot Rank' slider to jump to spots with low spot rank to add some bad spots to the training set. The users should keep in mind that this step is only meant to pick out a subset of examples of bad or good spots to train the training set. There will be an opportunity to add to this training set later. If the specimens in your batch of data only have a few spots, this could also be an efficient way to go through and manually classify them, but this will be an unusual circumstance.

In the panel on the right, the green rectangle specifies the area that is currently in the 16 x 16 panel. In the 16 x 16 window, candidate spots that are in the current frame are marked as blue. If the candidate spot is already in the training set, it will be marked as red. If there are multiple spots in the current frame, the user can click directly on the spot in the 16 x 16 zoom panel to reject the spot. If the user clicks *Accept* when there are multiple spots in the current frame, all the spots in this frame will be added to the training set as good spots.

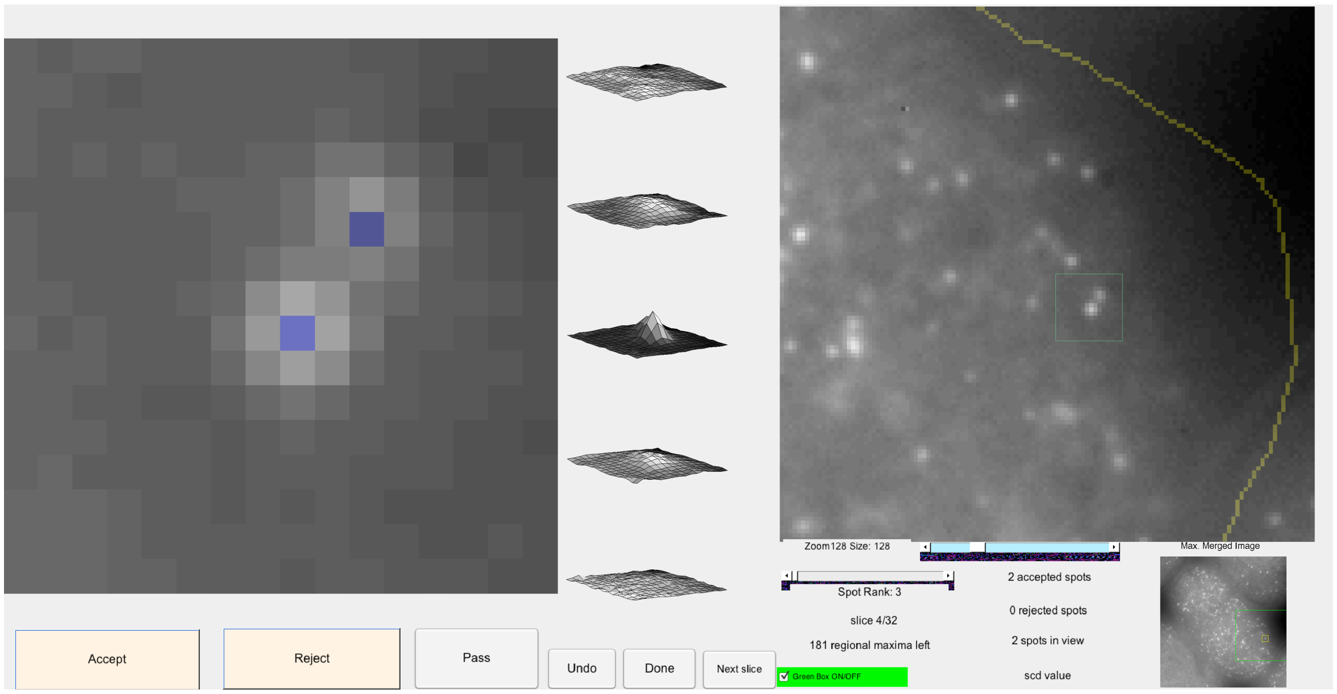


Fig. 4 The createSpotTrainingSet GUI is used to pick out spots for training set.

When the user presses the *Done* button, the GUI will pop up a window asking if it should continue on to the next specimen. If the user selects “Yes”, then the program closes the GUI and goes on to the next object under that position identifier. Do not be alarmed when the spot counts reset to 0. The program concatenates the good and bad spots from each object into a comprehensive curated list later on. When all the objects for a position have been seen, the program will finish making the training set.

After the user has finished building the training set from a certain position and saves the training set, the user should see a new mat file called 'trainingSet_{dye}_{ProbeName}.mat', e.g. trainingSet_tmr_Cel_end1.mat. This is the file that saves all the statistics of each spot in a structure variable called, 'trainingSet'. Later on, the user will use this file for training the classifier and the training results will also be saved in this file.

2. Train the Classifier : [Estimated time: 5-30 mins for 1000 training spots, depending on processing power]

MATLAB Command :
```
>> load trainingSet_{dye}_{ProbeName}.mat
>> trainingSet=trainRFClassifier(trainingSet);
```

Example: to train the training set for *C.elegans end-1* tmr probe...
```
>> load trainingSet_tmr_Cel_end1.mat
>> trainingSet=trainRFClassifier(trainingSet);
```

This function will first determine which features have the least effect on the classification and will leave those out for further training. This is the part that takes the bulk of the time. You will see the variables that are left out in the command window but you can always go back and check the list of variables that are left out after the training, which is saved in 'trainingSet.RF.VarLeftOut'. The second part of the function is to find the best number of variables sampled to construct the decision trees. Both of these parts will take a few minutes but these will ensure the robustness of the classifier.

When the training step is finished, you should see a new field called 'RF' in the trainingSet variable. This field saves all the statistics derived from training the random forest. In addition, you should see a new file added to the directory. The is the {dye}_{ProbeName}_RF.mat file that saves all the trees in the variable 'Trees.' In addition it saves a variable 'BagIndices' which is a cell array where each cell has the indices of the training set spot used in the corresponding tree.

To interpret the training results, one can take a look into the RF field of the trainingSet variable:

Example:

```
>> trainingSet.RF

ans = 
              Version: 'New method of estimating spot numbers, Apr. 2013'
               nTrees: 1000
                FBoot: 1
           VarLeftOut: {14x1 cell}
            statsUsed: {41x1 cell}
      VarImpThreshold: 0.21967
               VarImp: [1x55 double]
       dataMatrixUsed: [903x41 double]
         mTryOOBError: [32x2 double]
         NVarToSample: 6
        ProbEstimates: [903x1 double]
        spotTreeProbs: [903x1000 double]
           RFfileName: 'tmr_Cel_end1_RF.mat'
            ErrorRate: 0.016611
          SpotNumTrue: 560
      SpotNumEstimate: 563
        intervalWidth: 75
         SpotNumRange: [543 610]
  SpotNumDistribution: [1x1000]
               Margin: [903x1 double]
             FileName: 'trainingSet_tmr_Cel_end1.mat'
            ResponseY: [903x1 logical]
```

<Interpretation> In this training set, there are 903 training spots and 41 of the features, or statistics, are used. The 'dataMatrix' is an n-by-m numerical matrix that saves all the statistics for each spot, where n equals to the spot number and m is the number of statistics used. The field, 'dataMatrixUsed', saves the actual dataMatrix that is used for training the classifier. In the field of 'VarLeftOut', out can see the list of variables that have 'variable importance' in the lowest 25% percentile. The variable importance of a certain variable is defined by the change of error rate when the certain variable is permuted. The 'ProbEstimates' field has the average probability estimates among trees for each spot while the 'spotTreeProbs' saves the probability estimates derived from each individual tree for each spot. The training set error rate is 0.016611. The estimated total spot number is 563, which is close to the true spot number, 560. The 'spotNumRange' is the error range with an interval width of 75, which shows that in this set of spots, the estimate would fall between 543 and 610 75% of the time if the process were repeated.

One important thing to note is that in rare circumstances SpotNumEstimate may not be within the SpotNumRange. This is because SpotNumEstimate is calculated by thresholding a spot call probability at 50% while SpotNumRange uses and preserves probabilities directly. If there are substantially more ambiguous spots than non-spots (ambiguous being a probabilities far from 0 or 1) or vice versa, then this mismatch of the statistics could happen. Under most circumstances, however, this will not occur.

3. Classify the Spots with a specified training set

To apply the classifier to a specified image, you needs to first load in the wormGaussianFit.mat file which saves all the spot information of each object in the image. Meanwhile, you also need to load in the specific training set you would like to use to classify the spots.

MATLAB Command :

```
>> load trainingSet_{dye}_{ProbeName}.mat
>> load {dye}_{PositionNumber}_wormGaussianFit.mat
>> classifySpots(worms, trainingSet)
```

Example: To classify spots in the tmr image of position 6 with *C.elegnas end-1* tmr probe training set....

```
>> load trainingSet_tmr_Cel_end1.mat
>> load tmr_Pos6_wormGaussianFit.mat
>> classifySpots(worms, trainingSet)
```

You can also classify all the spots in the experiment all together with a specified training set. This function is basically a wrapper function for classifySpots. All inputs are optional. The first input 'toOverWrite' is a logical input that specifies whether the user would like to overwrite all current spot results in the directory (default false). If the use does not specify which channel this training set applies to, the program will try to do them all.

MATLAB Command :

```
>> load trainingSet_{dye}_{ProbeName}.mat
>> classifySpotsOnDirectory(toOverWrite*,trainingSet*,dye*)
```

Example: To classify tmr spots in the whole directory with the *C. elegnas* tmr probe training set....

```
>> load trainingSet_tmr_Cel_end1.mat
>> classifySpotsOnDirectory(1,trainingSet,'tmr')
```

When spots in a certain image are classified, one should see a new file with the corresponding name of '{dye}_{PositionNumber}_spotStats.mat', which has a cell variable, spotStats, that has the spot analysis results for each object in the image in each entry.

Example: To examine the spot results in the 1st cell of image 6 in tmr channel....

```
>> load tmr_Pos6_spotStats.mat
>> spotStats{1}

ans =

           dataMatrix: [1099x41 double]
        spotTreeProbs: [1099x1000 double]
        ProbEstimates: [1099x1 double]
       classification: [1099x3 double]
        intervalWidth: 75
      SpotNumEstimate: 496
         SpotNumRange: [444 536]
  SpotNumDistribution: [1x1000 double]
      trainingSetName: 'trainingSet_tmr_Cel_end1.mat'
```

```
            locAndClass: [1099x4 double]
```

<Interpretation> There are 1099 candidate spots in this cell. The total spot number estimate is 496, with a 75% error range from 444 to 536. The 'locAndClass' field saves the relative spot location in this subimage in the first three column and the final classification of each spot in the last column.
Important note: It is possible (although very unlikely) for the SpotNumEstimate to fall outside the SpotNumRange. This is because the SpotNumEstimate is based on a thresholding of the calibrated probability. p>50% means it is a spot. The interval estimate is based on simulating a Poisson binomial process and takes the actual values of the calibrated probabilities into account. Imagine a case where all the calibrated probabilities below 50% were 0, and a sizable fraction of the ones above 50% were 51%. In this case, every simulation would have fewer spots classified as spots than SpotNumEstimate claims because none of the non-spots would switch (they all have probability 0 of being a spot), but all the ones with 51% have a 49% chance of being counted as non-spots. The mismatch simply results from two different ways of counting spots. The first (thresholding on 50%) is often used in random forests and is a natural way to think about it. The second (using probabilities) allows us to make interval estimates. In practice, this mismatch is unlikely to be a problem.

4. Review the Spot Classification Results (and Retrain).

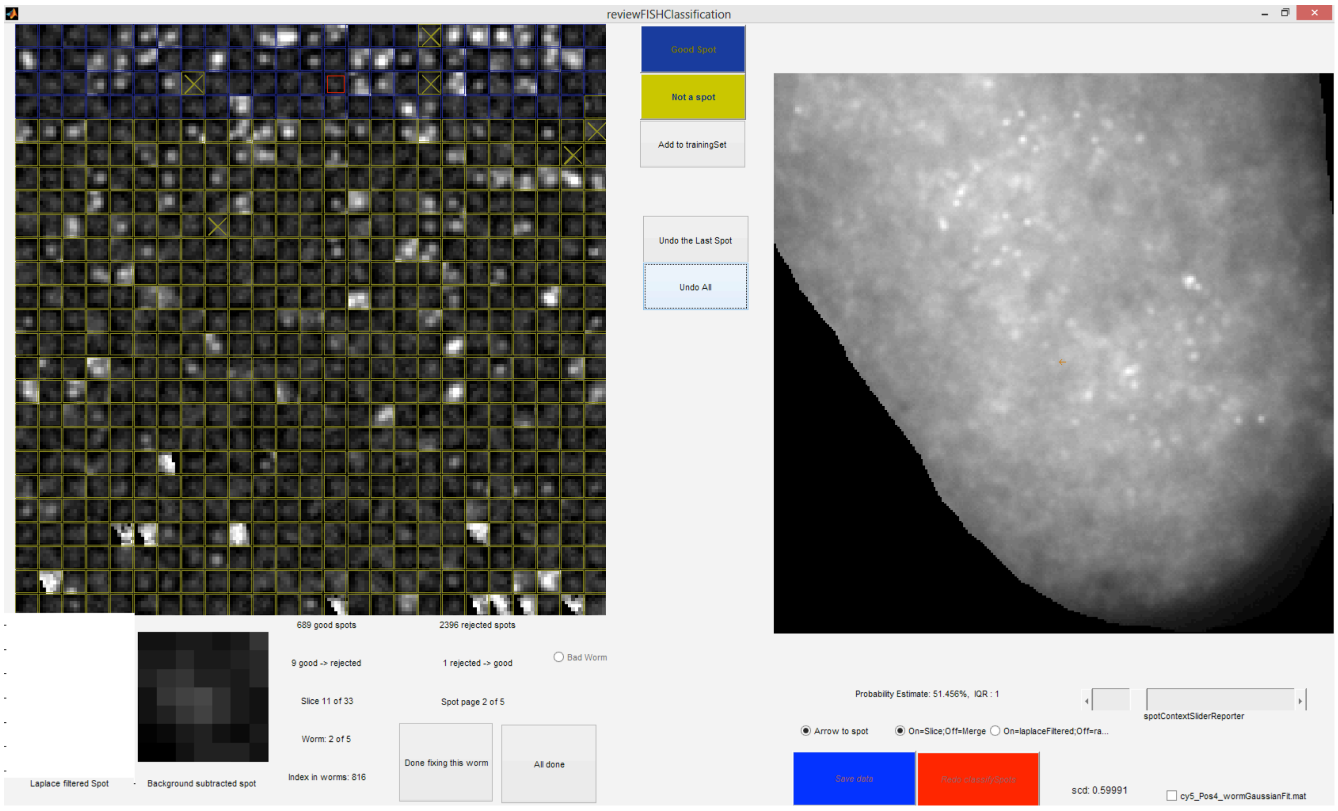
This step is an important step for optimizing the training set. One can use this 'reviewFISHClassification' function to review spot results in some of images, curate the annotation, add some more spots into the training set and retrain the training set. It is common that the first result would not look very good (Fig. 5-1), which might due to some misclassified spots or simply not enough spots to allow the classifier to make good judgment. Usually, after 2-4 times of retraining, one should see a significant improvement of classification accuracy (Fig. 5-2).

To review spot classification for a particular image...

MATLAB Command :
```
>> reviewFISHClassification({dye}_{PositionNumber})
```

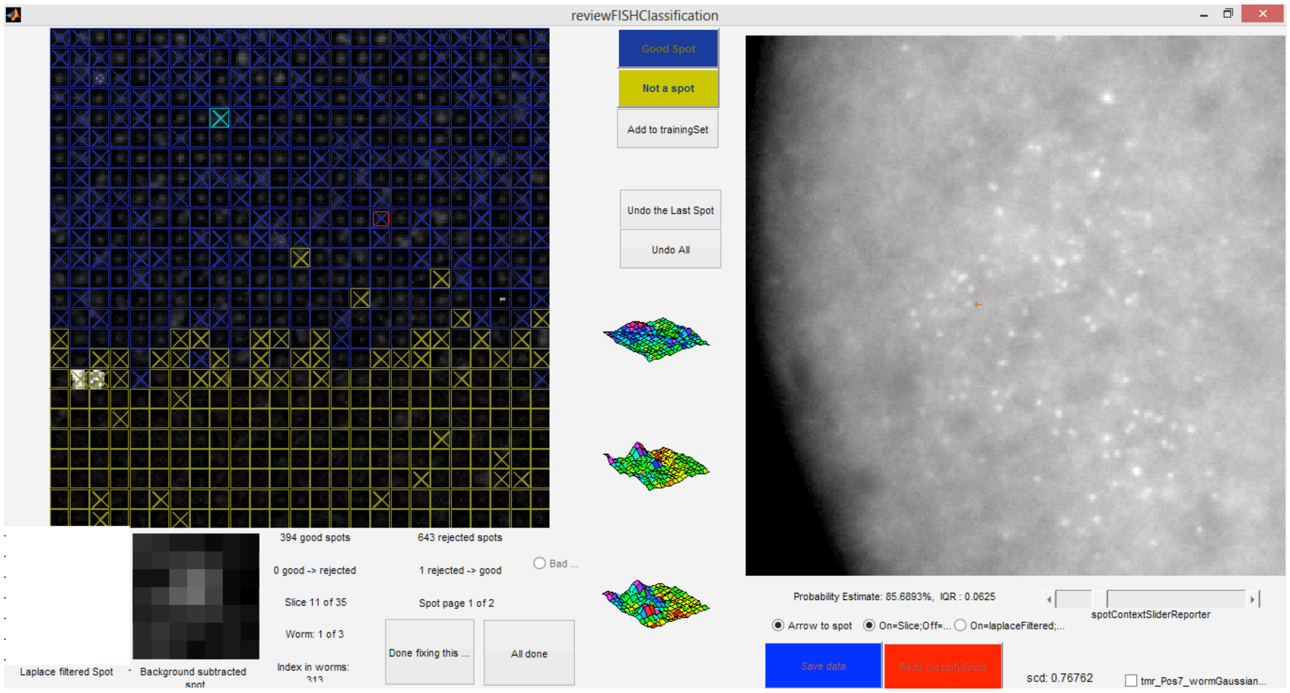
Example: To review spot classification in first image in tmr channel...
```
>> reviewFISHClassification('tmr_Pos1')
```

The GUI starts up with the spot classification panel on the left. The candidate spots are ordered by the probability of being a good spot. The blue spots are classified as good spots while the yellow spots are classified as bad spots. The spot that is marked with red rectangle is the spot that is currently being curated. The user should see where the spot is in the cell, pointed by a small red arrow, in the panel on the right. The spots that have an X in their rectangles are spots that are manually curated and currently in the training set while the spots with slashes on them are manually curated but are not in the training set. These slashed spots may include some imaging anomalies, that are neither typical bad spots nor good spots so they might not be appropriate to be added into the training set. The buttons *Good Spot* and *Not a spot* let the user correct the classification of a particular spot. To add these corrections to the training set as they are made, be sure the toggle button *Add corr. to train set* is on. The button *Add to trainingSet* will add whatever spot is currently in focus to the training set.



689 good spots
2396 rejected spots
9 good -> rejected
1 rejected -> good
Slice 11 of 33
Spot page 2 of 5
Worm: 2 of 5
Index in worms: 816
Probability Estimate: 51.456%, IQR : 1
[✓ Arrow to spot] [✓ On=Slice;Off=Merge] On=laplaceFiltered;Off=ra...
scd: 0.59991
cy5_Pos4_wormGaussianFit.mat

Fig. 5-1 The right panel shows spot classification results from a classifier that has only about 100 training spots. There are apparently too many false positives and false negatives in this classification result.



394 good spots
643 rejected spots
0 good -> rejected
1 rejected -> good
Slice 11 of 35
Spot page 1 of 2
Worm: 1 of 3
Index in worms: 313
Probability Estimate: 85.6893%, IQR : 0.0625
[✓ Arrow to spot] [✓ On=Slice;Off=...] On=laplaceFiltered;...
scd: 0.76762
tmr_Pos7_wormGaussian...

Fig. 5-2 shows spot results of the same embryo using a well-trained classifier which has about 1000 training spots.

After repeating step 3-4 several times on a few images, one should find the classifier's accuracy no longer improves. Then, one can classify all the spots in every image by using 'classifySpotsOnDirectory'. There is a red button on the GUI called *Redo classifySpots*. Pressing this will rerun the training set with the addition of the manually corrected spots and will display the new classification. If the user does not want to add spots from a different position, this is a more straightforward alternative to going back to step 3. When the user clicks *All done*, the program will retrain the classifier once more with the addition of all the manual corrections. Note that the software will use previously selected variables when redoing the random forest by pressing the red button, so this is a relatively quick step. Choosing the variables again would take much longer.

5. Summarize and Interpret the Results

MATLAB Command :
```
>> spotStatsDataAligning2(fileSuffix)
```

Example:
```
>> spotStatsDataAligning('20130615')
% This command will create a file called, wormData_20130615.mat in the
Analysis directory which saves all the total spot number statistics.
```

After the user classifies all the spots, this command can be used to extract total spot number statistics from each position.

Two files should be generated after using this command. One is the wormData_{fileSuffix}.mat file and the other is a figure called ErrorPercentagePlot_{fileSuffix}.mat. The wormData MAT file has a wormData structure variable that saves the total spot number statistics extracted from all the images:

For example:
```
>> load wormData_20130524
>> wormData

wormData =

                dye: {'cy','tmr'}
                 cy: [201x4 double]
                tmr: [201x4 double]
           spotInfo: [201x3 double]
          meanRange: [80.022 79.721]
    errorPercentage: [201x2 double]

>> wormData.spotInfo(20,:)

ans =

     20     4     7

>> wormData.cy(20,:)

ans =

     20    673    23    17
```

<Interpretation> There are 201 objects in this whole batch. In the 'spotInfo' field, the 3 columns are 'object index in the whole batch', 'position number', 'object index in the position'. In this case, the 20th object in the whole batch is 7th object in the position 4 image. The *cy* field has four columns that save the 'object index in the whole batch', the spot number for dye *cy*, the lower error bar, and the upper error bar. Therefore, in this object, the upper bound of the total cy spot number is 673+17=690 while the lower bound equals to 673-23=650. The

'meanRange' field saves the average error range of each channel to give the user a sense of how wide the error range is. The 'errorPercentage' is calculated by ((U+L)/2)/total spot number. This is further visualized in the errorPercentage plot. Both the 'meanRange' and the 'errorPercentage' are meant to give the users a sense of how well the classifier does and whether it improves over several times of training.

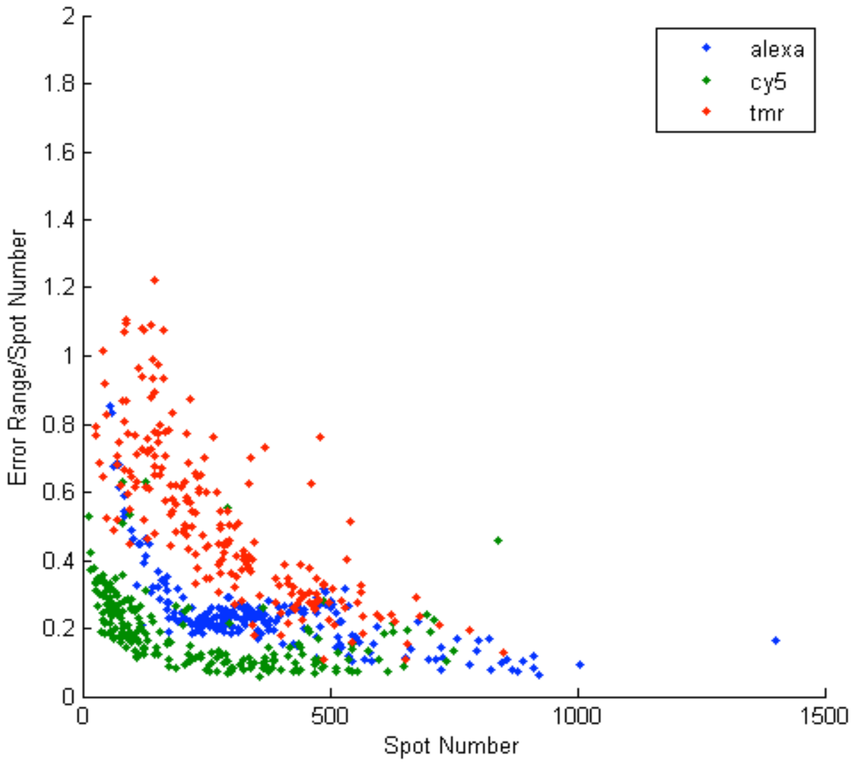


Fig. 6-1 A error percentage plot using spot results derived with an ill-trained training set. Note that error range is large in objects with different total spot numbers.

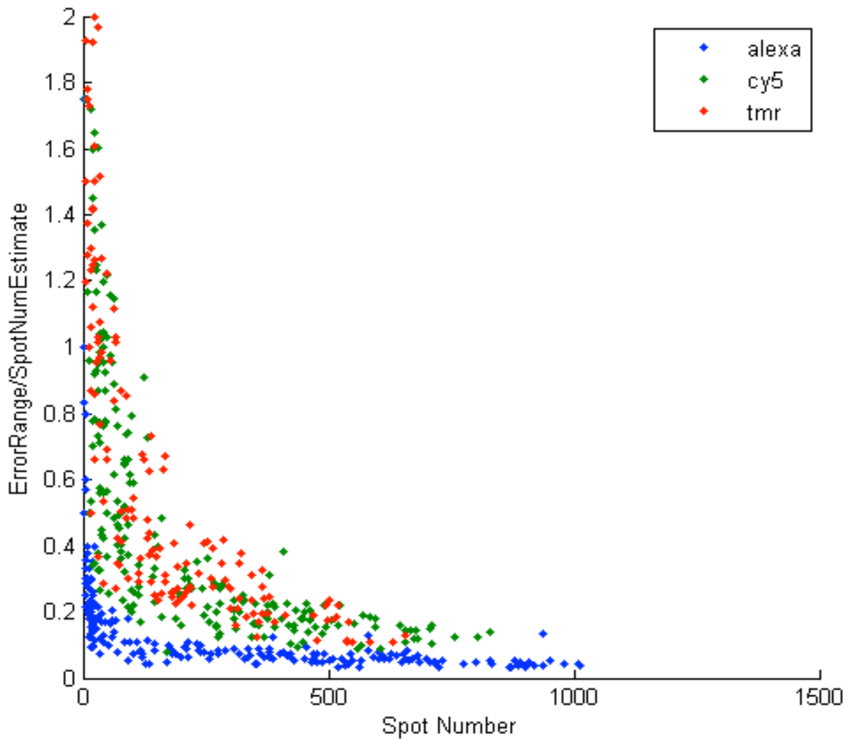


Fig. 6-2 A error percentage plot using spot results derived with an well-trained training set in the same data set. One should notice how the error percentage is reduced.

6. Adding new statistics

The software comes with a set of pre-established statistics/features to use for the classification. It is possible for the user to define his or her own. This entails modifying a few of the *.m files.

calculateFISHStatistics.m has a "Statistics Function Collection" which has subfunctions that calculate the statistics, usually based on a 7x7 square of pixels surrounding a local maximum in the variable *dataMat*. An example statistic function is:

```
function statValues = percentiles(dataMat)
      %calculate percentile-fractions (like qq plot)
      pctiles=10:10:90;
      percentiles=prctile(dataMat(:)/max(dataMat(:)),pctiles);
      for ppi=pctiles
          statValues.(['prctile_' num2str(ppi)])=percentiles(ppi/10);
      end;
    end;
```

The function returns a structure called 'statValues' where each field is a named statistic with a single number numerical value.

calculateFISHStatistics() returns a structure called gaussfit with a substructure called statValues, and the statistics are stored in this substructure. Adding the statistics to gaussfit looks like:

```
stats=percentiles(dataMat);
statFields=fieldnames(stats);
```

```
for fi=1:size(statFields,1)
    gaussfit.statValues.(statFields{fi})=stats.(statFields{fi});
end;
```

The final step is to add the name of the statistic to the cell array *statsToUse* in the *Aro_parameters.m* file. This name is not the name of the function but the name (statFields{fi}).