Automated Microarray Image Analysis Toolbox
for use with MATLAB®

Amanda M. White
Don Simone Daly
Pacific Northwest National Laboratory
Richland, WA 99352

November 7, 2005

Contents

Introduction

Microarray technology is an increasingly widely-used biological tool for investigating the genome or proteome. However, the utility of this tool is limited by the image analysis software used to summarize the data contained in the microarray images into a form that can be analyzed with traditional statistical methods. The Automated Microarray Image Analysis (AMIA) Toolbox for use with MATLAB (Mathworks, Inc. Natick, MA) is a powerful tool for analyzing microarray images and extracting response profiles. The AMIA Toolbox is designed to analyze large sets of slides of the same design with little user input, and provides extensive diagnostic statistics, images and plots to help the user pinpoint suspect data or analysis results.

Unlike most microarray image analysis tools, the AMIA Toolbox is an open-source tool, giving the analyst the ability to customize the image analysis for a particular application. The toolbox is distributed as code for use with the MATLAB mathematical programming environment (version 6.5 or higher). The toolbox also requires the Statistics Toolbox 4.1 and the Image Processing Toolbox 4.1 for MATLAB. The open-source nature of this tool allows users to understand how each result is calculated, and to modify algorithms or add additional routines to the analysis process.

The AMIA Toolbox analyzes a set of images at once, and can deal with a many different slide designs. The toolbox prompts the user to enter some preliminary information regarding the slide layout and identify the grid of spots on an initial slide, then this grid is automatically fit to each subsequent slide without requiring user intervention. After the initial spot centers are identified, AMIA uses three methods to determine the exact pixels contained in each spot. The first method is the most simple method, but provides a stepping off point for two more sophisticated algorithms. This method assumes that each spot is identically shaped and that the spots are equally spaced. The typical spot size and shape is dynamically determined from the brightest spots on the slide. The second spot identification method assumes that the spots are still identically shaped, but now allows the spot centers to vary within a small region from the perfect grid. This allows for slight variations in slide printing. The final method uses a seeded-region-growing algorithm [1] to determine which pixels are statistically different from the background. This allows the spots to assume any shape within a region around the expected center. This is helpful when the slide printing process has a great deal of variation, as can occur with custom-printed slides. However, if a spot has not reacted, then no pixels are identified and no statistics are calculated for that spot for the seeded-region-growing method.

After the spots are identified, summary and diagnostic statistics are calculated for each spot for each of the three spot identification methods. These include spot intensity statistics, local background statistics, and size and shape information. All of this information is saved in an ASCII file of results for each image. Diagnostic statistics and images are also produced for each image and for the entire collection. These statistics are designed to bring to light two different types of

problems: poor quality images and software analysis problems. If potential problems are found, they are flagged prominently for the user, so that one does not have to dig through all the results for each image to find them. The output for each image is saved in a separate subdirectory.

The results and diagnostics are displayed for the user via an HTML interface. This allows the user to easily browse the diagnostic images and statistics for each microarray image, and those for the entire collection. If processing problems are found, AMIA provides the capability to re-analyze chosen images with the user confirming each step of the analysis process to ensure correct results.

Analysis Process

AMIA is designed to analyze a collection of microarray images that share a common layout, including array size and position of control spots. These images should be stored in a single directory, and can take any common image format (e.g. GIF, JPEG, TIF). The analysis process is started by calling the `analyze_slides` function with the directory name as the only parameter. During the analysis, a subdirectory is created for each image where data and analysis diagnostics for that image are stored.

Prior to performing the image analysis, the user should review the images (or some subset thereof) to get an understanding of the quality of the slides and any problems that may interfere with analysis and to determine the layout of spots on the slide.

The `analyze_slides` function first prompts the user to provide a few parameters for the analysis process. These include the maximum and minimum spot size allowed in pixels and two threshold values all for the seeded region growing algorithm. The seeded region grower cutoff determines the threshold for "growing" the spots and the spot mask cutoff is the threshold for estimating the spot size and shape from the brightest spots in the image. (The second is necessarily higher because it only uses the brightest spots.) The last five parameters require yes (y) or no (n). The diagnostic images that are created can be turned on or off. Interactive mode determines whether the tool will prompt the user for input during the analysis. Fix mode is to be used when the user wishes to confirm each step of the analysis, typically as a second pass after automatic analysis produced an error. If the set of images is of a single slide that was not moved between the images, then the images are registered, or lined up. This could be used if analyzing a series of images of a slide taken over time. Finally, the user can choose whether to have all plots displayed during the analysis. (This can slow down analysis and lead to many windows open if a large collection of images is analyzed.)

The next window asks for the types of images to use; the default is all image types. AMIA next prompts the user to input information about the structure of the slides in the directory (if this information is not already present). The user will need to know how many blocks (or subarrays) of spots are present, how they are arranged, and the location of the positive control spots (or positional markers) within the blocks. In addition, the user is asked to give an initial estimate of spot diameter, in pixels. This information is saved in the file `slide_info.mat` in the directory with the images. If another collection of slides with the same layout is analyzed, this file can be copied to the new directory so that the information need not be entered again.

Once the slide structure has been defined, the function begins to analyze the images, one at a time. For each image there are two parts to the problem: finding the spots and characterizing the spots. AMIA also includes several functions for data visualization and analysis diagnostics.

Find the Spots in the Image

Finding the spots means identifying the pixels belonging to each spot. Since traditional microarray analysis tools have assumed that the spots are laid out in a perfect grid and are identically shaped, we have also implemented this for consistency. However, this application goes further and defines two additional spot identification algorithms. The first allows the identically-shaped spots to float within a small region from the expected spot center, and the second uses a seeded region growing algorithm to find spots that are not perfectly shaped. This is important because while some printing processes are highly accurate, others (e.g. robots) introduce imperfections to the slides.

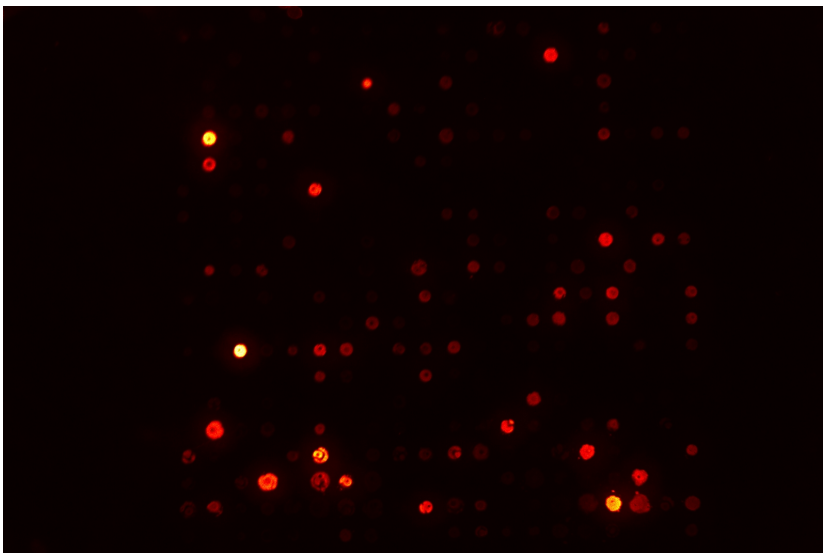

Figure 1: Original microarray slide image

In order to find the spots, the image must first be contrast-enhanced to make it easier to see the features of the slide. Figures 1 and 2 show the effect of contrast enhancing. The original slide is very dark with a few bright spots, however the enhanced slide shows the spots and background more clearly, including uneven texture of the spots and overshine around the brightest spots. The contrast enhancement is done by censoring the extremes of the data. In addition, a log transformation or square root transformation may be performed if necessary. Following this enhancement, estimates of the spot centers are found with the `spot_centers` function. This function first rotates the slides (if desired) then calls the `find_spot_centers2` function to create a naive estimate of the spot centers. We will call this a naive estimate because it assumes that the grid has been layed out on the slide perfectly: every row and column is equally spaced and the between block spacing is consistent.

The `create_well_map` function is called to extract information from the user to create a grid of points for a single well if there is a teflon mask on the slide, or for the entire image. The user is prompted to click on each of the positional markers or control spots (and additional spots as desired). A seeded region grower is used to find the extent of each of these spots, then a linear model is fit to the data to find the row, column and block spacing within a well. Finally, a grid of expected spot centers is created for this well (Figure 3). Next the `find_spot_centers2` function calls the `align_grid_to_well` function for each well. This function searches for the positional

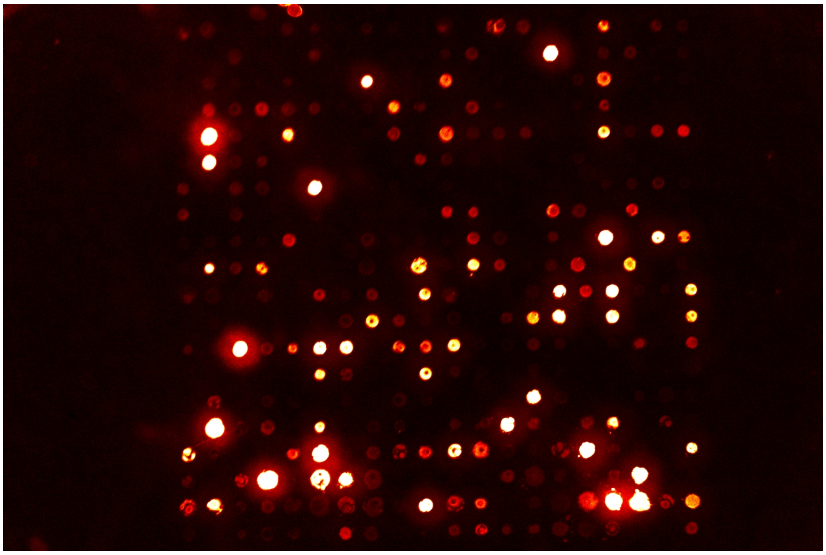

Figure 2: Contrast-enhanced microarray slide image

markers within the well and uses these to fit the previously created grid to the new well. To allow for small differences in spacing between the wells, a linear model is again used to perfect the spacing.

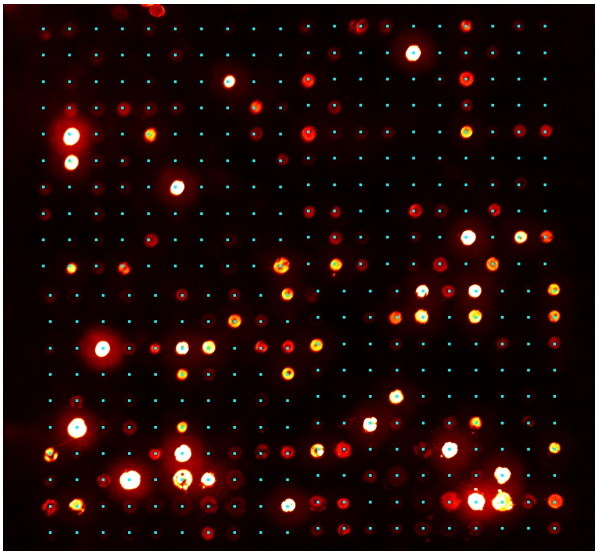

Figure 3: Contrast-enhanced image with naive spot center estimates

The `create_well_map` function also produces a map indicating which pixels belong to which positional marker. This is used by the `create_ideal_spot` function to line up the centers of the positional markers to find the average spot (Figure 4). We use this to create our ideal spot shape and size using the `create_spot_mask` function (Figure 5). The degree of difference required between the spot and the background is controlled by the spot mask cutoff value entered by the user. An increase in this value corresponds to a more strict criteria, and the ideal spot will become smaller. The spot mask is combined with the naive spot center estimates to create the ideal spot

map with the `ideal_spot_map` function, which indicates which pixels belong to each spot.
The second spot-finding algorithm allows the spot centers to vary within a small neighborhood of the expected spot centers, while maintaining identical size and shape for each spot. This spot map is created with the `ideal_shape_float_spot_map`. This function finds the spot center that maximizes the mean spot intensity within the neighborhood.

The final spot-finding algorithm allows the spots to assume almost any shape and size (up to a maximum) using a seeded region growing algorithm (`seeded_region_grower4`) that compares the seed to the corners of the neighborhood to find the pixels that differ sufficiently from the background. The degree of difference between spot and background is determined by the seeded region grower cutoff value entered by the user. The default value for this is 2. (As with the spot mask cutoff, a higher value for the seeded region grower cutoff gives smaller spots.) The `grow_spots` function outputs a map of the slide image where 0 indicates background and $n > 0$ indicates that pixel belongs to spot n.

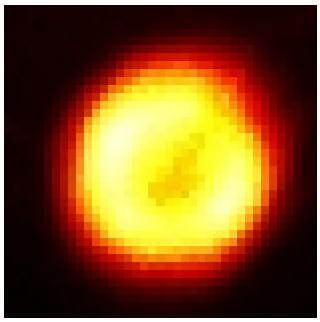

Figure 4: Average spot

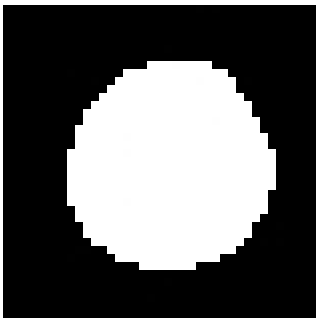

Figure 5: Spot mask

Characterize the Spots

Once the spots have been identified, summary statistics can be calculated on the spot pixels and the surrounding neighborhood to help the analyst determine the degree of reaction of the spots. This is a relatively straightforward process. Note that all statistics are calculated on the original image, not the enhanced image used for identifying spots. The `image_stats` function takes the image, the three spot maps and the spot neighborhood dimensions. For each spot the function takes the section of the spot map and corresponding section of the image determined by the spot center and neighborhood dimensions and calls the `spot_stats` function. This is where the actual statistics are calculated, and if further statistics are desired, they can be easily added to this function. The statistics calculated for each spot on each spot map include mean, median, min, max and standard deviation of the spot, the mean and standard deviation of the background, as well as the mean and standard deviation of each corner of the background individually. For the seeded region grower spots, some shape statistics are also calculated. The statistics are written to a comma-delimited text file with the same name as the image, stored in the directory of data for that image. This file can be easily imported into most data analysis programs because it is an ASCII file.

Analysis Diagnostics

Through every step of the analysis diagnostic images and statistics are created to help the user determine the validity of the results, and if there are problems, to make it clear in what step they arose. An HTML-based interface (results.html in the images directory) is also created during the analysis to organize and display these diagnostics, and to call attention to those which may indicate a problem with data processing. The diagnostic images take up several MB of disk space per microarray image, so if desired they can be turned off by entering 'n' for 'Diagnostics on?' in the Analysis Settings window. If the user wants to have all of these images be displayed on the screen while the analysis is running, enter 'y' for 'Plots on?'. In addition to visual diagnostics, the functions in this toolbox also calculate statistical diagnostics. One such diagnostic is to compare the row and column spacing between the slides. If the spot spacing on one slide is significantly different from the rest, this slide is flagged as containing possible problem. All slides that are identified as potential problems by this diagnostic or any other are prominently flagged in the HTML pages displaying the results. Other diagnostics include looking at background intensity as a function of position in the image, which could possibly indicate uneven lighting during imaging, and a comparison of typical spots between images to identify images that have particularly large or bright (or small or dim) spots overall.

Using the AMIA Toolbox

Setup

To use the AMIA Toolbox for MATLAB, extract the zip file to any location on the hard drive. In MATLAB, choose **Set Path** from the **File** menu. Add the AMIA_Toolbox directory to the path, save, then close. Now the functions in the toolbox can be located by MATLAB.

Example 1

The files for Example 1 can be downloaded from `http://www.pnl.gov/statistics/AMIA/download`. Save the zip file to the hard drive, then extract the files to any directory.

From the MATLAB command line, type

```
analyze_slides('Example1DirectoryName')
```

When Analysis Settings window appears enter the values shown in Figure 6 and choose **OK**. Next you will be asked to choose the type of images to analyze. The default is all image types; you can leave the default since there are no other image types in this directory, or choose ".img". (The .img file type is a custom image type. Other custom image types may be incorporated by adding the necessary code to the `load_image` function.)

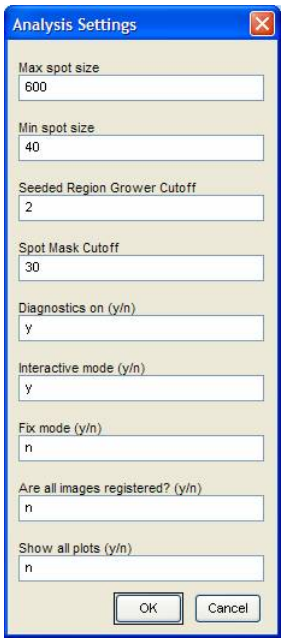

Figure 6: Analysis settings for Example 1

For this example, each array on the slide has been imaged separately, thus all layout questions will be answered with respect to the image, not the entire slide. When asked if the slide has a teflon mask separating wells on the slide, choose "No." You will next be asked to enter information about the slide layout, including the number and arrangement of wells and blocks of spots, and the approximate dimension of the spots. The appropriate values are shown in Figure 7. Since all blocks have the same size but not the same control spots (or positional markers) you will be asked to enter this information for each block individually. Then you will enter the size of the blocks and the positions of

the positive control spots. Figures 8 and 9 show the layout information for Block row 1, column 1. Each block is the same size (10 by 10) and has 2 control spots. All of the control spots are shown in Table 1. Now all of the slide layout information is entered and will be saved in the file `slide_info.mat` in the Example1 directory with the images. Any time you want to analyze a new set of images that has the same layout as a previously analyzed set, you can simply copy the `slide_info.mat` file from the previous set to the new directory and you will not be prompted to enter this information again. (Note that all slide layout information is relative to the image, so if each image contains part of a slide, for example, one array, then answer the questions relative to the image, not the entire slide.)

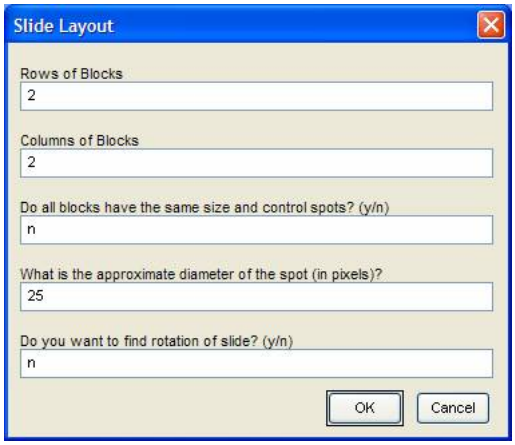

Figure 7: Slide layout parameters for Example 1

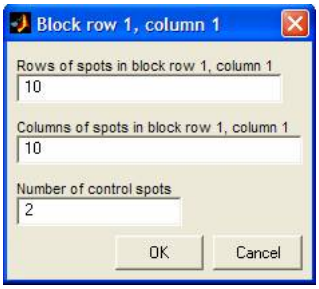

Figure 8: Block size and number of control spots for Example 1

You will next be asked to click on a point in each image to give the algorithms a starting point for placing the grid. As each image appears, use the figure buttons to zoom in, if necessary, then hit enter. When you see the crosshairs mouse, click on a spot on the image. For this example, it is easiest to click on the spot in Row 6 and Column 2 of Block Row 1, Block Column 1. Then in the command window, enter the block row and column and spot row and column (1, 1, 6, 2). This information must only be entered once for each image since it is saved in a file (grid_fit.mat in the directory with the images) so if you need to re-analyze an image it will load this information automatically.

You will be offered the choice of which files to include in the analysis, so if you want to only analyze a subset you can choose the images at this step. For the example, choose all images. Next choose the image on which to create the initial estimate of the spot grid spacing. This step will

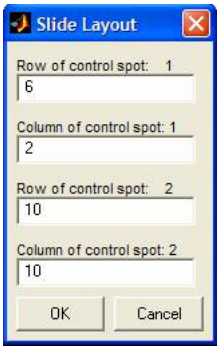

Figure 9: Control spot position for Example 1

Block Row	Block Column	Control Spot Row	Control Spot Column
1	1	6	2
1	1	10	10
1	2	2	5
1	2	9	10
2	1	7	1
2	1	1	10
2	2	9	10
2	2	5	4

Table 1: Control Spot Positions for Example 1

be done on one image only, then this grid will be applied to the other images in the collection, therefore at this step you should choose the highest quality image (best contrast, low background, uniform spots, etc) in the set. For the example, choose any image.

The next step is to estimate the grid spacing on the initial image. For this step you will be asked to click on the control spots, or positional markers, previously entered. You will also be asked if you want to zoom in closer on the blocks, which should not be necessary in this case. Next, you will be offered the opportunity to discard the previously entered control spots and enter new positional markers or to keep the control spots and enter additional spots. For the example, do not take either of these options. However, if positional markers of a set of slides happen to fall in a single row or column on an image, it would be necessary to enter additional spots here. These additional spots do not have to be visible in all images, just the initial image.

Click on each control spot when prompted. The program will use this information to create a grid of expected spot centers, and then this will be displayed for your approval. After the initial grid is approved, the program will not prompt for any more information for this image. The progress will be displayed in the command window as each step in the analysis is completed. When the first image is completed, the subsequent images will be analyzed. When all images are complete, the results can be viewed via an HTML-based interface that enables quick browsing of the diagnostics and analysis performance. To view these diagnostics, open the `results.html` file that is created in the same directory as the images. The quantitative results are located in the .csv files found in the data directory for each image.

Example 2

The files for Example 2 can be downloaded from `http://www.pnl.gov/statistics/AMIA/download`. Save the zip file to the hard drive, then extract the files to any directory.

From the MATLAB command line, type

```
analyze_slides('Example2DirectoryName')
```

When Analysis Settings window appears enter the values shown in Figure 10 and choose **OK**. For

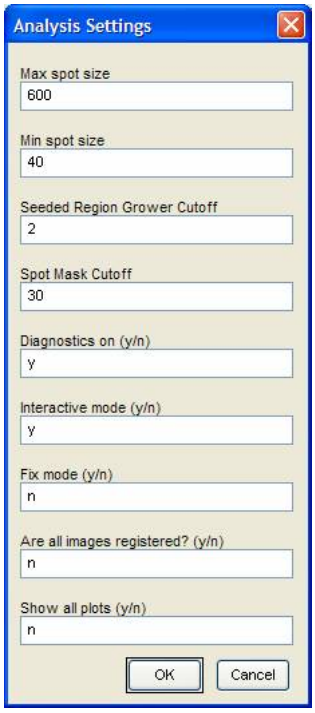

Figure 10: Analysis settings for Example 2

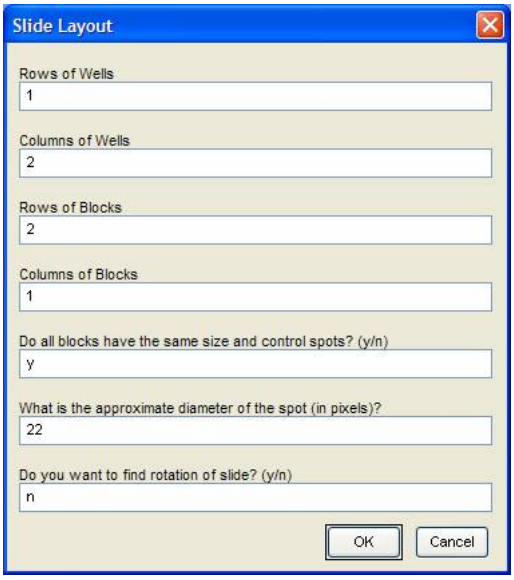

Figure 11: Slide layout parameters for Example 2

image types, either leave the default of all types or choose ".tif". When asked if the slide has a teflon mask separating wells on the slide, choose "Yes." For the slide layout, each block has the same size and control spots, so it is only necessary to enter this information once. Enter the values shown in Figures 11-14 in the appropriate windows.

You will next be asked to click on a point in each well to give the algorithms a starting point for placing the grid. (If the tool is unable to locate the edges of the wells automatically, it will ask you to click on the edges so it can locate them.) As each image appears, use the figure buttons to zoom in, if necessary, then hit enter. When you see the crosshairs mouse, click on a spot on the image. For this example, it is easiest to click on the spot in Row 1 and Column 1 of the top block. Then in the command window, enter the block row and column and spot row and column.

When offered the choice of which files to include in the analysis, choose both images. Then choose either image in order to create the initial grid estimate. For this step you will be asked to click on the control spots, or positional markers, previously entered. Prior to clicking on the spots, you may

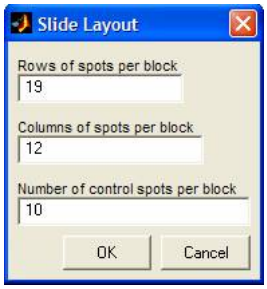

Figure 12: Block size and number of control spots for Example 2

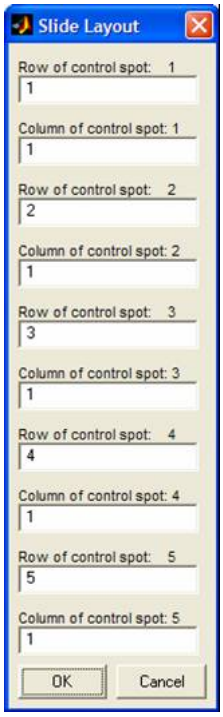

Figure 13: Control spot position for Example 2

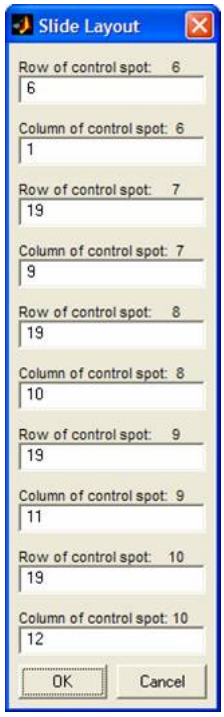

Figure 14: Control spot position for Example 2

be asked to help the tool identify the boundaries of the wells and to choose a single well in which to identify the spots. For the example, choose well 1. You will also be asked if you want to zoom in closer on the blocks, which may be necessary depending on the screen resolution. Do no discard the previously entered control spots or enter any additional points. You will be prompted to click on each control spot in both blocks in the well. If the grid of initial spot estimates is reasonable type 'y' and the program will continue with the analysis.

As the progress of the analysis is displayed in the command window, be aware that each step may take a little time since these example images are very large. When all images are complete, open the `results.html` file that is contained in the same directory as the images. The quantitative results are located in the .csv files found in the data directory for each image.

Additional examples may be found at `http://www.pnl.gov/statistics/AMIA/download`. Each

example contains the layout and parameter values for that set of images.

Appendix: Functions in Automated Microarray Image Analysis Toolbox

align_grid_to_well.m

```
Purpose:    The align_grid_to_well function takes an image of a single well
            and a grid of expected spot centers created from another well
            and finds the expected centers of the spots from these.

Usage:      [spot_centers, spacing, ideal_spot] = align_grid_to_well(well_img, ...
                grid, well_map, control_spots, block_rows, block_cols,...
                rows_of_blocks, cols_of_blocks, init_spacing, curr_grid_fit)

Inputs:     well_img = grayscale image (uint16) of a single well
            grid = the arrangement of the spot centers from a previous well
            well_map = the spot map of the previous well (0 = background,
                n > 0 indicates that pixels is part of spot n)
            control_spots = the relative position of the control spots within
                each block
            spot_diam = the diameter in pixels of the spots
            block_rows = number of rows of spots in each block (matrix)
            block_cols = number of columns of spots in each block (matrix)
            rows_of_blocks = number of rows of blocks per well
            cols_of_blocks = number of columns of blocks per well
            init_spacing = the row, column and block spacing for the grid
            curr_grid_fit = the initial grid fitting info for this well

Outputs:    spot_centers = matrix of spot center locations (not necessarily
                integer values) relative to the dimensions of the
                well (row, column)
            spacing = 6-column matrix giving row spacing, column spacing,
                vertical and horizontal block spacing, and vertical
                and horizontal offsets (row 1) along with 95% confidence
                intervals for each (row 2 and 3) and associated
                R^2 statistics (row 4)
            ideal_spot = a composite image of the brightest spots on this well

Requires:   find_peaks, solve_grid_spacing, centroid
```

amia_inputdlg.m

```
Purpose:    The amia_inputdlg function is a bug fix for Matlab 7.0.4
            which throws an error if the num_lines parameter of the
            inputdlg function is two-dimensional.  This function queries
            Matlab to determine the version being used.  If version
            7.0.4 then num_lines will be one-dimensional only.

Usage:      answer = amia_inputdlg(prompt,dlg_title,num_lines,defAns,Resize)

Inputs:     see Matlab documentation for inputdlg

Outputs:    answer = output form inputdlg function

Requires:   (nothing)
```

analyze_slides.m

```
Purpose:    The analyze_slides function runs the microarray image analysis
            process on a collection of slides.

Usage:      analyze_slides(curr_path)

Inputs:     curr_path = directory containing the images to be analyzed
                (optional, user will be prompted if missing)

Outputs:    none

Requires:   read_img, spot_centers, mat2index_img, create_spot_mask,
            bwmorph (Matlab Image Processing Toolbox), grow_spots,
            ideal_spot_map, image_stats, display_wells_with_spot_map,
            draw_boxes_on_img, make_synth_img
```

ars_sw_stat1.m

```
Purpose:    The ars_sw_stat1 function is a test statistic for
            identifying if a spot is on or off by comparing it to the
            local background.

Usage:      [swStat1, swStat1On] = ars_sw_stat1(ODMat, forePixNum, ...
                bckPixNum, critvalue)

Inputs:     ODMat = a 1x4 vector of Spot Mean, Spot Std Dev, Background Mean,
```

```
                Background Std Dev.
              forePixNum = the number of pixels in spot mean
              bckPixNum = the number of pixels in background mean.
              critvalue = critical value to use as cutoff for z-score

Outputs:      swStat1 = z-score statistic comparing spot to background
              swStat1On = 1 if swStat1 >= critvalue, 0 otherwise

Requires:     (nothing)
```

centroid.m

```
Purpose:      The centroid function takes an array of the form
              (row, column, weight) and calculates the two dimensional
              weighted average of the points.

Usage:        [row, col] = centroid(weight_array, round_ans);

Inputs:       weight_array = a 3-column array with point locations
                  (row, column) and weights (or intensities)
              round_ans = optional string variable that can take on
                  values 'round' or 'noround', if no value specified
                  it is assumed to be 'noround'

Outputs:      row = the weighted average of the rows
              col = the weighted average of the columns

Requires:     (nothing)
```

choose_files.m

```
Purpose:      The choose_files function allows the user to specify
              which files to analyze and to choose which image should
              be used to create the initial grid if it does not
              already exist.

Usage:        files_to_open = choose_files(curr_path, file_types)

Inputs:       curr_path = the directory containing the images
              file_types = list of image file extensions to look at

Outputs:      files_to_open = list of image files to analyze

Requires:     get_files
```

col_ind.m

```
Purpose:    The col_ind function allows the user to access data in
            a matrix using column headers rather than column numbers.
            For example, if we had a matrix called M and a list of
            column header names called colnames we could get the
            column(s) corresponding to 'name' by
            M(:, col_ind('name', colnames))

Usage:      index = col_ind(names, column_header, match_type)

Inputs:     names = 1 or more names of columns
            column_header = list of column names (in order)
            match_type = (optional) 'exact' (default),
                'begin' or 'contain', indicates which type of match

Outputs:    index = the column numbers corresponding to names

Requires:   (nothing)
```

collect_results.m

```
Purpose:    The collect_results function collects the data from each file
            of the AMIA analysis and creates a single spot statistic
            using the stat_summarize_fcn and outputs results to
            output_file.

Usage:      collect_results(data_path, output_file, stat_summarize_fcn)

Inputs:     data_path = the path containing the image files
            output_file = file to store results
            stat_summarize_fcn = (optional) handle of the function that
                calculates the spot estimate from the variety of
                statistics AMIA calculates.  Default values is
                'default_stat_summarizer'.

Outputs:    get_column_names

Requires:   (nothing)
```

config_diagnostic_statistics.m

```
Purpose:    The config_diagnostic_statistics function sets up the
            variables related to diagnostic stats.
```

```
Usage:      [diagnostic_statistics, spacing, diag_names, ...
            error_ind, diag_ind] = ...
                config_diagnostic_statistics(curr_path, curr_file)

Inputs:     curr_path = the directory containing the images
            curr_file = the name of the current image file

Outputs:    diagnostic_statistics = array of structures that contain
                information about the analysis of each image
            spacing = array of structures containing information
                about the grid spacing of each image
            diag_names = array of file names indicating which
                element of diagnostic_statistics goes with
                which file
            error_ind = vector indicating errors/problems
            (0=OK, 1=error in processing, 2=spacing problem)
            diag_ind = index of current file in error_ind and diag_names

Requires:   (nothing)
```

conv2D.m

```
Purpose:    The conv2D function converts the arguments m1 and m2 to double
            before applying conv2. (Necessary for MATLAB 7.0)

Usage:      result = conv2D(m1, m2, type)

Inputs:     m1 = matrix
            m2 = matrix
            type = type of convolution (see conv2 for options)

Outputs:    result = result of 2 dimensional convolution

Requires:   (nothing)
```

create_diagnostic_images.m

```
Purpose:    The create_diagnostic_images function creates diagnostic
            images that may help pinpoint problems in the image processing.

Usage:      create_diagnostic_images(original_img, grid, stats, columns, ...
                i_spot_map, float_spot_map, srg_spot_map, spot_diam, output_path)

Inputs:     original_img = the original slide image
            grid = the grid of expected spot centers
```

```
            stats = the matrix of statistics
            columns = a vector of column names
            i_spot_map = the 'ideal' spot map
            float_spot_map = the 'float' spot map
            srg_spot_map = the 'srg' spot map
            spot_diam = the expected spot diameter
            spot_mask = the spot mask
            neighborhood_height = vertical radius of the
                spot neighborhood
            neighborhood_width = horizontal radius of the
                spot neighborhood
            output_path = the directory in which to save the images
            plots_on = 0/1 indicator, 1=show plots

Outputs:    (nothing)

Requires:   draw_boxes_on_image, estimate_quantiles, outline_spots_image,
            make_synth_image
```

create_experiment_diagnostics_html.m

```
Purpose:    The create_experiment_diagnostics_html function creates an HTML
            page to display the experiment diagnostics.

Usage:      create_experiment_diagnostics_html(root_path, file_list, flags)

Inputs:     root_path = path containing the images to be analyzed
            file_list = list of image files analyzed
            flags = output of create_experimetn_diagnostics function

Outputs:    none

Requires:   (nothing)
```

create_grid_from_spacing.m

```
Purpose:    The create_grid_from_spacing function takes the spacing
            structure and creates a grid of expected spot centers.
            This version assumes that each block in a row or column
            may not be aligned--must be used with the version of
            solve_grid_spacing that assumes the same.

Usage:      grid = create_grid_from_spacing(spacing)

Inputs:     spacing = structure giving spot spacing information, the
```

```
                output of solve_grid_spacing function.

Outputs:    grid = matrix giving expected pixel row and column, with
                spot row and column and block row and column.

Requires:   (nothing)
```

create_html.m

```
Purpose:    The create_html function creates the html results page for each
            image analyzed.

Usage:      create_html(output_path, filename, stats)

Inputs:     output_path = the path containing the output for this image
            filename = the name of the image file
            stats = the structure containing the diagnostic statistics

Outputs:    (none)

Requires:   (nothing)
```

create_ideal_spot.m

```
Purpose:    The create_ideal_spot function creates an image of the ideal
            spot from the control spots found with the seeded region
            growing algorithm.

Usage:      ideal_spot = create_ideal_spot(control_spot_map, ...
                well_img, spot_diam)

Inputs:     control_spot_map = matrix the same size as well_img where
                0 indicates background and integer n>0 indicates
                that pixel belongs to control spot n, OR
                a list giving pixel row, pixel column and spot
                number for every pixel in a control spot
            well_img = grayscale image (uint16) of a single well
            spot_diam = the estimated diameter of the spots
            input_type = 'matrix' or 'list', indicates what form
                the control spot map is in (default is 'matrix')

Outputs:    ideal_spot = a matrix giving the average intensity of the
                control spots

Requires:   find_peaks
```

create_main_html.m

```
Purpose:    The create_main_html function creates the main results html page
            called "results.html" in the image directory, with links to
            the results for each image, and experiment diagnostics.

Usage:      create_main_html(output_path, file_list, error_list, statistics)

Inputs:     output_path = the path containing the output for this image
            file_list = list of images analyzed
            error_list = list of error indicators
            statistics = the array of structures containing the diagnostic
                statistics to be passed to create_html

Outputs:    (none)

Requires:   experiment_diagnostics, create_experiment_diagnostics_html,
            create_html, find_struct, create_help_html
```

create_spot_mask.m

```
Purpose:    The create_spot_mask function takes the matrix of an "ideal"
            spot (created in spot_centers) and creates a 0/1 matrix
            indicating which pixels are in the spot and which are background.
            If zcrit is so high that there are no pixels that meet this
            criteria, then zcrit is decremented by 1 until there is at least
            one pixels that is chosen.

Usage:      spot_mask = create_spot_mask(ideal_spot, zcrit)

Inputs:     ideal_spot = the matrix of an ideal spot created in
                spot_centers (create_ideal_spot function)
            zcrit = the critical value cutoff for comparing to
                the background (z-score cutoff)
            interactive = prompt user to approve spot mask? (0/1 flag)

Outputs:    spot_mask = a matrix the same size as ideal_spot where
                1 indicates spot and 0 indicates background

Requires:   bwmorph (Matlab Image Processing Toolbox)
```

create_well_map.m

```
Purpose:    The create_well_map function creates a grid of expected spot centers
            for a single well and creates a spot map for the control spots.

Usage:      [spot_centers, control_spot_map, well] = ...
                create_well_map(original_img, well_row_bd, well_col_bd, ...
                control_spots, spot_diam, block_rows, block_cols, ...
                rows_of_blocks, cols_of_blocks)

Inputs:     original_img = grayscale image (uint16) that has been rotated
                so the array is parallel to the image
            well_row_bd = 2-column matrix where the first column indicates
                pixel row position and the second column is a -1/1
                indicator where -1 is beginning of well and 1 is end
            well_col_bd = same as well_row_bd for pixel columns
            control_spots = the relative positions of the control spots
                in each block
            spot_diam = the user-provided estimated spot diameter in pixels
            block_rows = the rows of spots in each block (matrix)
            block_cols = the columns of spots in each block (matrix)
            rows_of_blocks = the number of rows of blocks per well
            cols_of_blocks = the number of columns of blocks per well
            block_has_controls = 0/1 matrix indicating whether a block has
                control spots

Outputs:    spot_centers = matrix of spot center locations (not necessarily
                integer values) with relative position within the slide
                (pixel row, pixel column, relative row, relative column,
                block row, block column)
            control_spot_map = matrix the same size as a single well where
                0 indicates background and integer n>0 indicates
                that pixel belongs to control spot n
            well = the number of the well chosen to create the grid
            spacing = 6-column matrix giving row spacing, column spacing,
                vertical and horizontal block spacing, and vertical
                and horizontal offsets (row 1) along with 95% confidence
                intervals for each (row 2 and 3) and associated
                R^2 statistics (row 4)

Requires:   grow_spots, centroid, solve_grid_spacing
```

default_stat_summarizer.m

```
Purpose:    The default_stat_summarizer function takes a matrix of
            statistics from AMIA for a single image, and summarizes
```

```
            the results for each spot to the necessary information.
            Used by collect_results.m

Usage:      (summary, cnames) = default_stat_summarizer(data, colnames)

Inputs:     data = numeric data matrix of statistics calculated by AMIA
                for a single image (each row corresponds to one spot)
            colnames = column names

Outputs:    summary = numeric matrix of summary statistics, one row for
                each spot
            cnames = column names for summary matrix

Requires:   col_ind
```

diagnostic_stats.m

```
Purpose:    diagnostic_stats function calculates some diagnostics using the
            spot and background statistics, and the ideal spot.

Usage:      s = diagnostic_stats(stats, colnames, ideal_spot, output_path)

Inputs:     stats = the matrix of statistics calculated in the image_stats
                function
            colnames = a vector of column names for the stats matrix\
            ideal_spot = the ideal spot for this image
            output_path = the path containing the output for this image
            best_stats = (optional) 'float', 'srg' or 'ideal'
                default is 'float'

Outputs:    s = a structure containing diagnostics including:
                variation in background
                correlation between spot and background intensity

Requires:   col_ind
```

display_image_enhanced.m

```
Purpose:    The display_image_enhanced function displays an image at the largest size
            possible on the screen while preserving aspect ratio, and contrast-
            enhances the image by compressing the upper and lower ends of the data range.

Usage:      fig_handle = display_image_enhanced(img, quantiles)

Inputs:     img = the image
```

```
            quantiles = vector of length 2 giving upper and lower quantile for
                data compression (by default, this is [0.01, 0.99])

Outputs:    fig_handle = handle of the figure created

Requires:   ideal_axes, mat2index_img, estimate_quantiles
```

display_well_boundaries.m

```
Purpose:    The display_well_boundaries function creates a figure
            of the slide image with the well boundaries overlaid.

Usage:      h = display_well_boundaries(img, well_row_bd, well_col_bd)

Inputs:     img = the slide image
            well_row_bd = the row boundaries from find_wells function
            well_col_bd = the column boundaries from the find_wells function

Outputs:    h = figure handle

Requires:   display_image_enhanced
```

dsd_range.m

```
Purpose:    The dsd_range function finds the range of a matrix

Usage:      [tmp_range,tmp_min,tmp_max] = dsd_range(tmp_mat)

Inputs:     tmp_mat = matrix

Outputs:    tmp_range = difference between tmp_min and tmp_max
            tmp_min = minimum of tmp_mat
            tmp_max = maximum of tmp_mat

Requires:   (nothing)
```

estimate_quantiles.m

```
Purpose:    The estimate_quantiles function estimates the
            quantiles of a matrix.

Usage:      [tmp_quant] = estimate_quantiles(tmp_mat,tmp_pct)

Inputs:     tmp_mat = matrix
```

```
            tmp_pct = (optional) vector of percent values

Outputs:    tmp_quant = vector of estimated percentiles of tmp_mat
                corresponding to tmp_pct

Requires:   (nothing)
```

experiment_diagnostics.m

```
Purpose:    The experiment_diagnostics function generates statistical
            diagnostics comparing the results of each image to determine if
            any results are unusual.

Usage:      flags = experiment_diagnostics(diagnostic_statistics)

Inputs:     diagnostic_statistics = array of structures generated during the
                analysis process

Outputs:    flags = array of strings containing messages that will be printed
                in the experiment_diagnostics.html file

Requires:   (nothing)
```

findPeakCenters.m

```
Purpose:    The findPeakCenters function finds the centers of the peaks
            identified by the peak_ids vector

Usage:      centers = findPeakCenters(i_vec, peak_ids)

Inputs:     i_vec = vector of the mean intensities in each
                pixel row or column
            peak_ids = integer vector where 0 indicates not
                part of a peak and n>0 indicates that row is
                part of peak n

Outputs:    centers = the position of the center of each peak
                (column vector)

Requires:   (nothing)
```

find_peaks.m

```
Purpose:    The find_peaks function finds the centers of the peaks
```

```
            in a vector.

Usage:      centers = find_peaks(dist_vec, peak_width, smooth_width)

Inputs:     dist_vec = a double-valued vector
            peak_width = width for ordfilt (optional - default is 13)
            smooth_width = (optional - default is 7)

Outputs:    centers = the weighted centers of the peaks in the
                vector (not necessarily integer)

Requires:   (nothing)
```

find_spot_centers2.m

```
Purpose:    The find_spot_centers2 function takes a grayscale image that
            represents an array of spots and identifies the expected centers
            of the spots.  Also saves information about the row and column
            spacing in a file called 'diagnostics.mat' in the input_path

Usage:      [spot_centers, ideal_spot, well_row_bd, well_col_bd, spacing] = ...
                find_spot_centers2(original_img, input_path, output_path)

Inputs:     original_img = grayscale image (uint16) that has been
                rotated so the array is parallel to
                the image
            input_path = the path where the slide layout file is stored
                (this file contains information about the structure
                of the slide, including blocks, control spots, etc)
            output_path = the path in which to store the images of the
                wells overlaid with the grid

Outputs:    spot_centers = matrix of spot center locations (not necessarily
                integer values) (row, column)
            ideal_spot = average spot intensities of the control spots
                (to be used to create expected spot size and shape)
            well_row_bd = 2-column matrix where the first column indicates
                pixel row position and the second column is a -1/1
                indicator where -1 is beginning of well and 1 is end
            well_col_bd = same as well_row_bd for pixel columns
            spacing = a 6-column vector giving the row, column, and block
                spacing along with the position (row, col) of the block
                within the slide

Requires:   find_wells, create_well_map, create_ideal_spot,
            align_grid_to_well
```

find_struct.m

```
Purpose:    The find_struct function provides matching capability for arrays
            of structures similar to find() for arrays of double.

Usage:      [ind1, ind2] = find_struct(struct_array, fieldname, value, negation)
            ind1 = find_struct(struct_array, fieldname, value, negation)

Inputs:     struct_array = array of structures
            fieldname = the name of the field to match on
            value = value to find in the field corresponding to fieldname
            negation = T/F, find structs where fieldname ~= value?

Outputs:    ind1, ind2 = indices of structures in struct_array that match
                the criteria

Requires:   (nothing)
```

find_wells.m

```
Purpose:    The find_wells function finds the boundaries of the wells on
            a microarray slide.

Usage:      [well_row_bd, well_col_bd] = find_wells(tmp_mat, interactive)

Inputs:     tmp_mat = the image of the slide
            interactive = 0/1 indicator of whether to prompt user if there
                are problems finding the well boundaries

Outputs:    well_row_bd = 2-column matrix where the first column indicates
                pixel row position and the second column is a -1/1
                indicator where -1 is beginning of well and 1 is end
            well_col_bd = same as well_row_bd for pixel columns

Requires:   (nothing)
```

get_analysis_settings.m

```
Purpose:    The get_analysis_settings function creates prompts the user
            to input overall settings for the analysis, including
            whether to show plots, and max and min spot size.

Usage:      [diag_on, int_mode, fix_mode, plots_on, z_crit_srg, ...
```

```
            z_crit_spot_mask, max_spot_size, min_spot_size] ...
          = get_analysis_settings

Inputs:   (none)

Outputs:  diag_on = 0/1 indicator of whether to create diagnostics
          int_mode = 0/1 indicator for interactive mode
          fix_mode = 0/1 indicator for fix mode
          plots_on = 0/1 indicator for display plots
          z_crit_srg = critical value cutoff for seeded region grower
          z_crit_spot_mask = critical value cutoff for creating spot
              mask
          max_spot_size = the maximum allowable spot size (for SRG)
          min_spot_size = the minimum allowable spot size (for SRG)
            images_registered = 0/1 indicator:    are the images registered
          (lined up)

Requires: (nothing)
```

get_bright_spots.m

```
Purpose:  The get_bright_spots function finds the brightest spots in
          the image that have the approximate given diameter.

Usage:    spot_mask = get_bright_spots(well_img, spot_diam, cutoff)

Inputs:   well_img = the image
          spot_diam = approximate spot diameter
          cutoff = percentile cutoff for identifying bright pixels

Outputs:  spot_mask = matrix the same size as well_img where 0 means pixel
              is in the background, and integer n>0 means pixel is
              part of bright spot n

Requires: estimate_quantiles, pad_matrix
```

get_column_names.m

```
Purpose:  The get_column_names function extracts column names
          from a file.

Usage:    colnames = get_column_names(filename, delim)

Inputs:   filename = name of file
          delim = (optional) delimiter (default is ',')
```

```
            skip = (optional) lines to skip before column names
                (default is 0)

Outputs:    column names

Requires:   (nothing)
```

get_files.m

```
Purpose:    The get_files function creates a list of all the files
            in the given directory.  If extensions are provided it
            returns a list of files of those types.

Usage:      file_list = get_files(path, extensions)

Inputs:     path = a string giving the path of the directory
            extensions = (optional) a vector of extensions
                (case-independent)

Outputs:    files_list = a list of files in this directory
```

get_grid_initial_fit.m

```
Purpose:    The get_grid_initial_fit function prompts the user to
            input an initial point for fitting the grid of spot centers
            for each image in the directory. Creates the file
            'grid_fit.mat' for use by analyze_slides.

Usage:      tf = get_grid_initial_fit(curr_path, img_type)

Inputs:     curr_path = the path containing the images and the slide
                layout file.  Alternately, can be a file name.
            img_type = (optional) vector of image types to look at
            (e.g. img_type = strvcat('.gif', '.tif', '.jpg'))

Outputs:    (nothing)

Requires:   (nothing)
```

get_slide_info_diff_blocks.m

```
Purpose:    The get_slide_info function prompts the user to input
            information about the layout of the slides to be analyzed.
```

```
Usage:      tf = get_slide_info(path)

Inputs:     path = full path of the directory where the slides are
                located (this is where the output file will be saved)
            output_file = (optional) name of output file, default is
                'slide_info.mat'

Outputs:    tf = 0/1 indicator where 1 = successful completion

Requires:   (nothing)
```

get_spot_positions.m

```
Purpose:    The get_spot_positions function prompts the user to enter
            positional information about spots of interest.

Usage:      spot_positions = get_spot_positions(num_spots, label)

Inputs:     num_spots = number of spots
            label = label to use for instruction (e.g. 'spot', or
                'control spot') (default value is 'spot')
            block_prompt = (optional) prompt for block row and column?
                (0/1 indicator)
            default values = (optional) cell array of default values for
                block row, block column, spot row and spot column

Outputs:    spot_positions = 4-column matrix holding spot row,
                spot column, block row and block column, in that order

Requires:   (nothing)
```

grow_spots.m

```
Purpose:    The grow_spots function uses a seeded region growing algorithm
            to find the actual shape and size of each spot on the slide.

Usage:      [spot_map, new_seeds] = grow_spots(img, seeds, nbhood_rad, ...
                spot_diam, min_size, max_size)

Inputs:     img = the image
            seeds = 3-column matrix giving row, column and spot number
                (a spot may have more than one seed pixel, but
                they must be neighboring pixels)
            nbhood_rad = 2-vector giving radius (in rows and columns)
                of the neighborhood to use for growing the spots
```

```
            spot_diam = the expected (approximate) spot diameter
            min_size = minimum number of pixels per spot
            max_size = maximum number of pixels per spot

Outputs:    spot_map = a matrix the same size as img where 0 indicates
                background and n>0 indicates that pixel belongs to
                spot n
            new_seeds = the new seeds that are estimated from the seeds
                that are given and the neighborhood radius

Requires:   seeded_region_grower4, find_peaks
```

ideal_axes.m

```
Purpose:    The ideal_axes function creates a figure window and axes that
            are as large as possible on your screen or largest possible
            scaled by given scaling factor.

Usage:      fig_handle = ideal_axes(size, visible, scaling)

Inputs:     size = 2-element vector giving rows and columns
            visible = 'on' or 'off' (optional)
            scaling = number between 0 and 1 (optional)

Outputs:    fig_handle = the handle of the figure window created

Requires:   (nothing)
```

ideal_shape_float_spot_map.m

```
Purpose:    The ideal_shape_float_spot_map function creates the spot map where
            each spot has an identical shape, but the spots are allowed to
            float within a small neighborhood from their expected location.

Usage:      spot_coords = ideal_shape_float_spot_map(original_img, grid, ...
                spot_mask, nbhood_rad)

Inputs:     original_img = microarray image
            grid = grid of expected spot centers, where column 1 is row
            spot_mask = 0/1 matrix giving the spot size and shape
            nbhood_rad = length 2 vector giving the neighborhood radius
                in rows and columns for allowing spots to float

Outputs:    spot_coords = matrix the same size as original_img where 0 = background
                and n>0 indicates that pixel belongs to spot n
```

```
Requires:   centroid, ideal_spot_map
```

ideal_spot_map.m

```
Purpose:    The ideal_spot_map function creates a map of which pixels
            belong to which spot using the ideal spot size and shape.

Usage:      spot_map = ideal_spot_map(grid, spot_mask, image_rows, image_cols)

Inputs:     grid = coordinates (row, column) of spot centers
            spot_mask = a 0/1 matrix giving the size and shape of the
            spot (0=background, 1=spot)
            image_rows = number of rows in original image
            image_cols = number of columns in original image

Outputs:    spot_map = a matrix of size image_rows by image_cols where
                0 indicates background and n>0 indicates that
                pixel belongs to spot n

Requires:   (nothing)
```

image_stats.m

```
Purpose:    The image_stats function collects statistics for
            every spot on the image (both ideal and floating
            spots)

Usage:      [stats, names] = image_stats(img, map1, map2, map3, nbhood_rad, grid)

Inputs:     img = the image
            map1 = first (ideal) spot map (uint16)
            map2 = second (float) spot map (uint16)
            map3 = third (srg) spot map (uint16)
            nbhood_rad = 2-element vector giving the radius
                of the neighborhood to use in rows and columns
            grid = grid of estimated spot centers as (row, column)
                (used to reduce search area)
            srg_seeds = seeds used in seeded_region_grower

Outputs:    stats = a matrix of statistics for each spot
            names = vector of names for each column

Requires:   spot_stats
```

initialize.m

```
Purpose:    The initialize function creates output files, makes sure
            the slide_info.mat and grid_fit.mat files exist and
            gets the image files to analyze.

Usage:      [files_to_open, summary_file_fid, all_results_fid, ...
            experiment_diagnostic_dir] = initialize(curr_path)

Inputs:     curr_path = the path containing the image files

Outputs:    files_to_open = list of image files to analyze
            summary_file_fid = the file id for the summary file
            experiment_diagnostic_dir = if DIAG_ON==1 then this gives
                the directory to store the experiment level diagnostics

Requires:   choose_files, get_slide_info_diff_blocks, get_grid_initial_fit
```

instruction.m

```
Purpose:    The instruction function displays an instruction that may be
            linked to a figure.  The message may be displayed as either
            a message box, positioned so it does not overlap the figure
            or as a title to the figure, if the figure is very large.

Usage:      h = instruction(text, fig_handle, force_msgbox)

Inputs:     text = instruction text
            fig_handle = (optional) figure handle for positioning instruction
            force_msgbox = (optional) 1=force separate message window, do
                not display as title

Outputs:    h = figure handle if separate message window is created,
                otherwise -1

Requires:   (nothing)
```

load_image.m

```
Purpose:    The load_image function reads the image file and if
            it is an RGB function it turns it into a 2D matrix
            (assuming that all channels are identical).

Usage:      original_img = load_image(filename)
```

```
Inputs:     filename = the name of the image file

Outputs:    original_image = the 2D image

Requires:   read_img
```

maha_dist.m

```
Purpose:    The maha_dist function calculates the Mahalanobis distances for
            an array of data where each row is an observation.

Usage:      score = maha_dist(data)

Inputs:     data = array of numeric data

Outputs:    score = vector of scores, 1 score for each row of data

Requires:   (nothing)
```

make_synth_image.m

```
Purpose:    The make_synth_image function creates a "synthetic" image
            (matrix) of the slide using the same spot size and shape at
            every position, and coloring the spots according to the given
            vector of values.

Usage:      [synth_image] = make_synth_image(spot_rowcol, spot_value, ...
                spot_mask, plots_on)

Inputs:     spot_rowcol = the centers of the spots on the image (row, col)
            spot_value = a vector giving a value to be used to color
                each of the spots
            spot_mask = a 0/1 matrix giving the spot size and shape
            plots_on = 0/1 indicator, where 1 = display images

Outputs:    synth_image = a double-valued matrix that can be made into
                an image

Requires:   mat2index_img
```

mat2index_img.m

```
Purpose:    The mat2index_img function takes a matrix and creates an
            indexed image (either uint8 or uint16).
```

```
Usage:      [tmp_index_img] = mat2index_img(tmp_mat,tmp_min,tmp_max,u_8_16)
            [tmp_index_img] = mat2index_img(tmp_mat)

Inputs:     tmp_mat = matrix (double, uint8, uint16) representing the
                image
            tmp_min = minimum value for scaling (optional)
            tmp_max = maximum value for scaling (optional)
            u_8_16 = indicator for uint8 or uint16, can take values 8
                or 16 (optional)

Outputs:    tmp_index_img = an indexed image of type uint8 or uint16

Requires:   (nothing)
```

outline_spots_image.m

```
Purpose:    The outline_spots_image function creates an image where the
            spots are outlined.

Usage:      new_img = outline_spots_image(img, grid, spot_mask)

Inputs:     img = the slide image, values scaled from 0 to 255
            grid = coordinates (row, column) of spot centers
            spot_mask = a 0/1 matrix giving the size and shape of the
            spot (0=background, 1=spot)

Outputs:    new_img = the slide img with spots outlined in black

Requires:   bwmorph (Matlab Image Processing Toolbox)
```

pad.m

```
Purpose:    The pad function pads a vector with the same value
            as the end of the vector

Usage:      padded_vec = pad(vec, first, last)

Inputs:     vec = the vector to be padded
            first = # to add to beginning of vector
            last = # to add to end of vector

Outputs:    padded_vec = the padded vector

Requires:   (nothing)
```

pad_matrix.m

```
Purpose:    The pad_matrix function pads a matrix with the same
            value as the outer edge of the matrix.

Usage:      padded_mat = pad_matrix(matrix, rows, cols)

Inputs:     vec = the vector to be padded
            rows = # of rows to add to matrix
            cols = # to add to end of vector

Outputs:    padded_mat = the padded matrix

Requires:   (nothing)
```

plot_points_img.m

```
Purpose:    The plot_points_img function plots a set of points on an image
            by condensing the image values into the range 0-254 and setting
            points pixels to 255, thus making it possible to create and
            save an image overlaid with points without displaying on
            the screen.

Usage:      img = plot_points_img(original_img, grid, spot_rad, enhance)

Inputs:     original_img = the image
            grid = grid of points to plot, where column 1 is pixel row and
                column 2 is pixel column
            spot_rad = radius of spots to plot
            enhance = 'enhance' or 'noenhance', whether or not to contrast-
                enhance the image

Outputs:    img = image with points (to plot and display points, use a colormap
                    similar to this:    my_colormap = [hot(255); 0, 1, 1])

Requires:   estimate_quantiles, mat2index_img
```

read_img.m

```
Purpose:    The read_img function imports image files of type .img

Usage:      raw_img = read_img(filename)
```

```
Inputs:     filename = name of image file

Outputs:    raw_img = matrix representing the image

Requires:   (nothing)
```

seeded_region_grower4.m

```
Purpose:    The seeded_region_grower4 function takes a segment of an
            image and a seed and determines which pixels are similar
            enough to the seed that they are part of the same class.

Usage:      [Current_Region] = seeded_region_grower3(subimage, seedrc, ...
                z_critval,Nmax, plots_on)

Inputs:     subimage = segment of an image
            seedrc = two-element vector consisting of the row and column
                of the starting seed
            z_critval = z-score cutoff for admitting new pixels into
                the spot
            Nmin = min number of pixels in the spot
            Nmax = max number of pixels in the spot
            plots_on = a 0/1 indicator of whether to display plots

Outputs:    Current_Region = a 2-column matrix giving row, column
                coordinates of each pixel in the spot

Requires:   imdilate (Matlab Image Processing Toolbox)
```

solve_grid_spacing.m

```
Purpose:    The solve_grid_spacing function takes the information about the
            control spots and finds the row and column spacing.  This
            version assumes that blocks in a row or column may be
            offset from each other, but the spot spacing within a block
            is the same for all blocks.  Must be used with the version of
            create_grid_from_spacing that also assumes this.

Usage:      spacing = solve_grid_spacing(spot_positions, init_spacing)

Inputs:     spot_positions = matrix giving pixel row and column of
                the spots, along with spot row and column and
                block row and column
            init_spacing =  spacing structure from the original
                well that was created with user input (optional).
```

```
Outputs:    spacing = structure giving the spot spacing, confidence
                intervals and R^2 values of the linear model.

Requires:   (nothing)
```

spot_centers.m

```
Purpose:    The spot_centers function takes a grayscale image that
            represents an array of spots and identifies the
            positions of the spots as well as creating an image
            of the "ideal" spot from the control spots.

Usage:      [grid, ideal_spot, well_row_bd, well_col_bd, spacing] = ...
                spot_centers(original_img, do_rotate, input_path, ...
                output_path, plots_on)

Inputs:     original_img = grayscale image (uint16) that has been
                rotated so the array is parallel to
                the image
            do_rotate = 0/1 indicator, 1=rotate before finding spots
            input_path = the path where the slide layout file is stored
                (this file contains information about the structure
                of the slide, including blocks, control spots, etc)
            output_path = the path in which to store the images of the
                wells overlaid with the grid
            plots_on = 0/1 indicator, 1=show all plots

Outputs:    grid = matrix of spot center locations (not necessarily
                integer values) (row, column)
            ideal_spot = average spot intensities of the control spots
                (to be used to create expected spot size and shape)
            spacing = 6-column matrix giving row spacing, column spacing,
                vertical and horizontal block spacing, and vertical
                and horizontal offsets (row 1) along with 95% confidence
                intervals for each (row 2 and 3) and associated
                R^2 statistics (row 4)

Requires:   rotation, imrotate (Matlab Image Processing Toolbox),
            find_wells, create_well_map, create_ideal_spot,
            align_grid_to_well
```

spot_shape_stats.m

```
Purpose:    The spot_shape_stats function generates statistics about
```

```
            the size and shape of the spots in the given spot map.

Usage:      stats = spot_shape_stats(spot_map, num_spots)

Inputs:     spot_map = the spot map (see ideal_spot_map.m for
                description of a spot map)
            num_spots = number of spots

Outputs:    stats = matrix of statistics, where row i gives these
                    statistics for spot i:  (area, vertical diameter,
            horizontal diameter, eccentricity (vert/horiz diameter),
            and expected area based on diameters)

Requires:   (nothing)
```

spot_stats.m

```
Purpose:    The spot_stats function calculates statistics on a spot and
            its surrounding neighborhood.

Usage:      [stats, names] = spot_stats(img, spot_mask)

Inputs:     img = segment of image (may be uint8, uint16 or double)
            spot_mask = a 0/1 matrix indicating which pixels belong to
                the spot. Must be same size as img.

Outputs:    stats = a row vector of statistics
            names = vector of names of each statistic

Requires:   ars_sw_stat1
```

strip.m

```
Purpose:    The strip function removes elements from
            the ends of a vector

Usage:      stripped_vec = strip(vec, front, back)

Inputs:     vec = the vector
            front = # to remove from beginning
            back = # to remove from end

Outputs:    stripped_vec = the stripped vector

Requires:   (nothing)
```

write_stats_to_file.m

```
Purpose:    The write_stats_to_file function writes the statistics out
            to one or more files.

Usage:      write_stats_to_file(stats, columns, ...
                individual_results_filename, all_results_fid)

Inputs:     stats = the matrix of statistics
            columns = a vector of column names
            filenames = vector of filenames
            permissions = vector of write permissions
                (either 'w' or 'a')

Outputs:    (nothing)

Requires:   (nothing)
```