Project report for CS279

Amanda Miguel and Handuo Shi

Background

In our lab, we have been developing an imaging acquisition platform named Strain Library Imaging Protocol (SLIP), which speeds up the traditional imaging acquisition protocols by taking advantage of computerized control of our microscopic imaging system via MATLAB.

Bacterial cells growing in a multi-well format (96-, 384-, or 1536-well) are transformed onto an agar plate. Once the user identifies the A1 position of the plate and picks a certain plate type (Figure 1), the program runs automatically through all selected well positions, and takes certain numbers of images around the center of each well. Such highly automated protocol permits screening through a 96-well plate in 10 min (25 images for each well, 100 ms exposure), and generates a rich dataset for subsequent quantification and analysis.

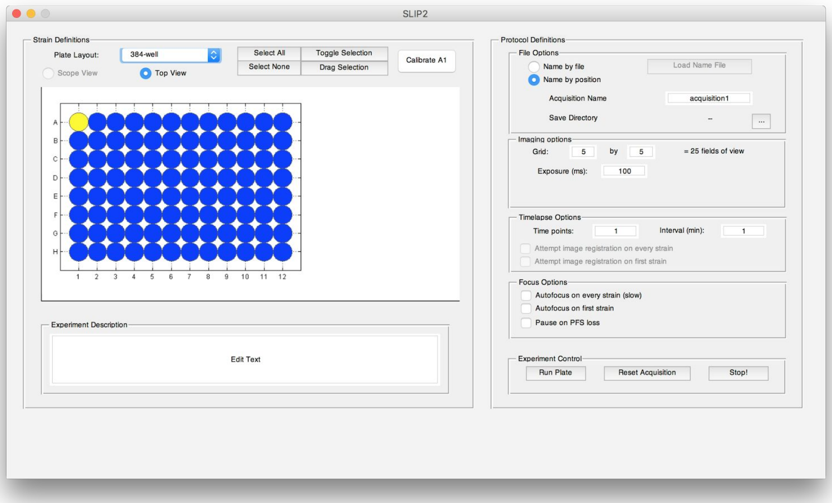


Plate Layout: 384-well
Scope View [✓ Top View]
Name by file [✓ Name by position]
Acquisition Name: acquisition1
Save Directory: --
Grid: 5 by 5 = 25 fields of view
Exposure (ms): 100
Time points: 1 Interval (min): 1
☐ Attempt image registration on every strain
☐ Attempt image registration on first strain
☐ Autofocus on every strain (slow)
☐ Autofocus on first strain
☐ Pause on PFS loss

Figure 1. Interface of SLIP. The left part is an illustration of wells on the plate that are included in imaging acquisition, the right part has options about imaging, such as how many images to take at each well, exposure time, time-lapse options, *etc.*.

Specific Aims

Although SLIP has already been widely used in our lab by several lab members and has helped them to generate useful datasets, issues still remain for improvements. For our project, we focus on two aspects that can improve data acquisition.

Aim 1. On-the-fly cell identification and counting

It is common for SLIP to not get enough cells at certain wells. This is probably due to the intrinsic variability of cell density across different wells, and also has to do with sample preparation, where the cells in a certain well is off the grid and cannot be captured by the program. Taking more images at each well can partially compensate for cell numbers, but then the whole acquisition takes more time and data storage becomes a problem. Actually, for our current setup, we usually take 25 images at each well, and most wells ended up getting more than 500 cells, whereas 200 cells are enough for most of our analysis.

We propose to develop an algorithm that can quickly identify cells on the acquired image and then provide an estimation of number of cells on each image. With such cell counting algorithm, SLIP will perform adaptively image acquisition, stopping taking images on a certain well once it collects enough cells. This way, we will 1) shorten image acquisition time and lower storage burden for dense cultures, and 2) allow SLIP to sample more areas for diluted cultures and obtain enough single-cell data for subsequent analysis.

For this purpose, we will need a fast edge detection algorithm, potentially sacrificing some accuracy. We aim to set a benchmark of 200 ms for each image that processed; this way, we can potentially reduce the images we acquire by 60%, and still finish the whole acquisition within the same time slot (10 min) as before.

Aim 2. Feature-based image registration for time-lapse

Another common problem with SLIP is interference of cell lineage tracking do to image drift. This is particular important in timelapse, where it is necessary to track the same single cells for an extended period of time. In our particular setup, image drift is mostly caused by 1) the drying of the agar surface, causing the pad to shrink and the cells to move out of the field of view or 2) physical sliding of the coverslip on top of the pad as the oil-objective objective moves through the oil coated on the surface. Because SLIP is programmed to move to absolute positions on an agar surface without any reference to the cells we are interested in, cells will drift out of the field of view, cutting the length of time they can be observed and forcing us to rely on taking more single cell images.

We propose to develop an algorithm that performs image registration on specific cells in a field of view during the course of the time lapse, resulting in a SLIP that dynamically adjusts the absolute positions to the correct cells. This algorithm will result in 1) maximizing the period of time in which cells are observed and 2) allow us to monitor less fields of view per well, allowing for overall faster acquisition. As part of this algorithm may require retaking of images, a potentially time expensive process depending on the drift, we aim to set a similar benchmark of 200 ms for each image processed.

We aim to compare our algorithm to the current post-imaging registration method we used called StackReg (http://bigwww.epfl.ch/thevenaz/stackreg/) , a plugin in the image processing package FIJI. This software works very well for the majority of images but can give errors in scenarios where the background interferes with the alignment of the desired objects i.e. cells. Our analysis and results examine two single cell alignments which were poorly constructed in StackReg that are improved upon in our algorithm.

Analysis and Results: Aim 1

1. Edge-detection method

(Code used in this section: main_edge_test.m and edge_method.m)

For typical phase images from SLIP (Figure 2, left panel), the cells are completely dark, with a bright ring at the periphery, and the background is gray. With these characteristics, the most straightforward way for cell identification is through edge detection. We first referred to MATLAB documentation on detecting cells (http://www.mathworks.com/help/images/examples/detecting-a-cell-using-image-segmentation.html), and optimized the parameters. The basic steps for this method is,

1. Detect the cell contour by segmenting algorithms that detect fast gradient changes
2. Dilate the image to create continuous contours for each object
3. Fill cell interiors
4. Remove connected objects on cell borders
5. Smooth the cell edges
6. Filter out small non-cell objects based on area threshold

A typical resulting image is shown in Figure 2, right panel. The algorithm only missed one cell (red arrows), without picking up any non-cell objects. As an estimation of cell number, such accuracy is good enough.

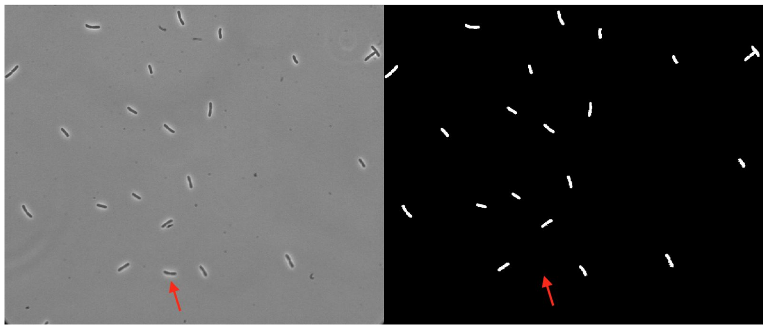


Figure 2. Original phase image (left) from SLIP, and the result after edge detection and segmentation (right). As shown here, with some tuning of parameters, the algorithm can identify most cells, with only one missing (red arrows).

However, if we benchmark the time cost for this algorithm, the key algorithm (excluding reading in images and displaying images), takes about 3.8 seconds, and about 2 seconds are spent on the edge detection. Since the edge detection is a built-in function in MATLAB, it is really difficult to further improve this method.

2. Thresholding method

(Code used in this section: main_edge_test.m and thre_auto.m)

Inspired by Assignment 3 of this class, we thought the thresholding method may be a good way to reduce time, since it circumvents the direct edge detection method. For this, we need to identify a threshold. Based on the phase image in Figure 2, we can tell that if we generate a histogram of pixel values, we will see two peaks: a large peak corresponding to the gray background and a small peak at lower pixel values for the dark cells. Therefore, a natural definition of threshold is the local minima of these two peaks, and we found a function that calculates such local minima:

http://www.mathworks.com/matlabcentral/fileexchange/31570-finding-dominant-peaks-and-valleys-of-an-image-histogram/content/findlocalminima.m

For this method, our work flow is,

1. Identify the local minima of the image histogram
2. Use the local minima as a threshold and binarize the phase image
3. Filter out small non-cell objects based on area threshold

Such method doesn't have free parameters since the threshold is determined by the local minima. And applying it on the same phase image, we were able to identify all cells, along with three non-cell objects which are actually dusts on the camera.

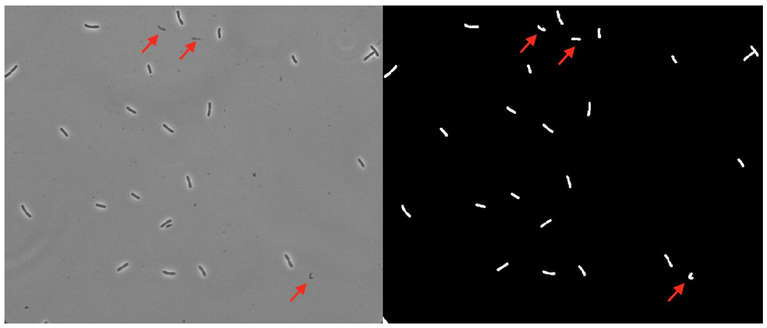


Figure 3. Original phase image (left) from SLIP, and the result after threshold method (right). This algorithm does not have tunable parameters, and it identifies all cells, with three dust points (red arrows).

Similarly, we performed benchmark on this algorithm, and the key algorithm takes about 1.2 seconds, much better than the edge detection method. Within that, 0.47 seconds are spent on identifying the local minima, and about 0.7 seconds are on filtering. Therefore, these two parts have room for improvement.

Improvement 1. Pre-defined threshold

(Code used in this section: main_edge_test.m, thre_manu.m, and threshold_test.m)

Since automatic thresholding based on local minima takes relatively long time, we thought if the threshold can be pre-defined. First, we manually varied the threshold by about 20% and found that the change of threshold within this regime did not quantitatively affect the thresholding result in this image, and all the cells in Figure 3 (right panel) were still identified. Therefore, the threshold does not need to be so stringent. Also, we scanned through a set of images from one SLIP experiment, and found that the local minimas in each image varied by about 10% (this part of images not included in final report due to file size concerns). Therefore, we can safely use one pre-defined threshold for all images and still have good segmentation results.

Improvement 2. MEX functions

(Code used in this section: main_edge_test.m and thre_manu_mex.mexmaci64)

The other time-consuming part of our script was in the filtering part. In this part, we used the MATLAB function *bwareaopen* to remove connecting components that have an area lower than 1,000 pixels, which are generally dusts or other non-cell objects. This method is very essential since our phase images always have noise and simple thresholding picks up a lot of small objects.

Luckily, MATLAB provides a way to speed up some of its functions by calling MEX files, which are dynamically linked subroutine executed in the MATLAB environment. Since the MEX files are precompiled and dynamically linked, they are generally faster compared to the native MATLAB functions. And MATLAB has a built-in application called MATLAB Coder (http://www.mathworks.com/products/matlab-coder/), which generates MEX files based on MATLAB function. Although most of the image processing functions are not supported by MATLAB Coder, it has just incorporated support for *bwareaopen* in the recent R2015b version. We tried to compile a MEX file for the *bwareaopen* function, and its runtime reduces to about 10% of the original one.

3. Final method with optimizations

(Code used in this section: main_edge_test.m and thre_count_mex.mexmaci64)

With the improvements mentioned above, we finally adopted the workflow below:

1. Manually define a threshold for a series of images
2. Use the pre-defined threshold to binarize the phase image
3. Filter out small non-cell objects based on area threshold
4. Steps 2 and 3 are built into MEX functions to speed up the runtime
5. Calculate the total area of identified cell objects and estimate cell count

For the phase image in Figure 2 and 3, our final result gives 23 cells, and the true cells in the image is 22. This is a fairly good estimation. And the final benchmark for the key algorithm is about 0.1 s, satisfying our original goal of 200 ms.

Analysis and Results: Aim 2

Our feature-based image registration, like the previous aim, also depends on identifying cells. In this case, we must identify the cells in a field of view and pinpoint the location of these cells as as a way to transform the next image. In addition to the previous MathWorks documentation, we also reviewed the following tutorials on rotating images (http://www.mathworks.com/help/images/examples/find-image-rotation-and-scale.html) and on image segmentation (http://www.mathworks.com/matlabcentral/fileexchange/25157-image-segmentation-tutorial).

1. The algorithm

(Code used in this section: image_registration.m, test_image_registration.m)

The basic outline of the algorithm is as follows:

1. Read in the reference image and the next image.
2. For each image:

   a. Perform a simple filter to identify cells from the background. Because our cells are phase dark, we can filter pixels with intensities that are less than a certain threshold. This results in a black and white segmented image. The threshold can be either hard set or estimated based on local minima of the pixel histogram as described above. For our test, we set a threshold of 4000 pxl intensity.
   b. Label separate segmented cells using matlab function *bwlabel*()
   c. Find the centroid of these cells by calculating measurements such as area, centroid and other properties using matlab function *regionprops*() of the labeled objects. These centroid points will be used to perform the translation.
3. Populate a list of centroids for all objects in frame. Filter out non-cells based on area threshold so that number of objects identified match between reference and next image.
4. Use the function *fitgeotrans()* to translate the cells in the next image to the reference image. Translate the next image using the results of fitgeotrans
5. Return as variables how much the image was translate and rotated. These parameters would be used to adjust the microscope.

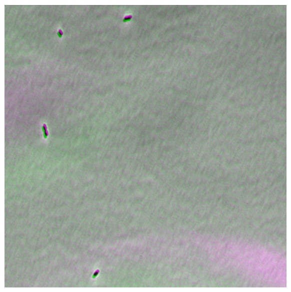
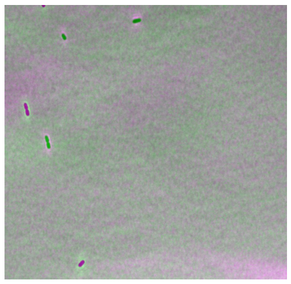
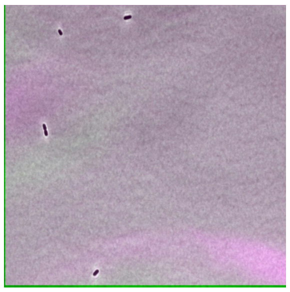
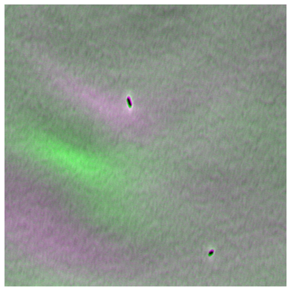
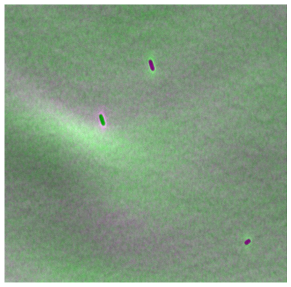
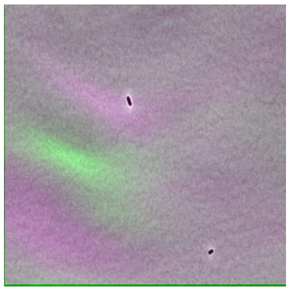
To implement this algorithm in SLIP, we would call the function during image acquisition. If the the translation variables are significant over the course of the time lapse, we would tell the microscope to translate the objective in the calculated X- and Y- directions and overwrite the last image. In this case, 'significant drift' will depend on the length of the time lapse and may need to be dynamically adjusted to fit the specified timeframe.

2. Implementation & Challenges

To test out our algorithm, we ran our protocol on two pre existing time lapse data sets. In the real setup the algorithm would be providing outputs that adjust an objective on a microscope dynamically during the course of the time lapse. On preexisting data, we expect the algorithm to perform a simple feature image registration based on our selected objects. Therefore, to maximise efficiency, we must compare and show improvement upon pre-existing image registration software, such as the stackreg plugin for FIJI. For our analysis, the two data sets we chose are examples where stackreg FIJI plugin did a worse alignment than the original image. In particular, we observe that these images were cases where changes in the background of the image seem to dominate the alignment, further supporting the idea that a feature-based image registration would be better for our needs.

We encountered two challenges while designing the image registration algorithm that were dependent on the sensitivity of the threshold filter. In the two test examples, both images were taken at the same time and so the threshold value we used (4000 pxl) worked for both cases. This threshold value will infact vary depending on how exposed the image is during the imaging process, which may change the range of the histogram of pixel intensities for the image. Implementing the same local minima threshold described in Aim 1 could automatically determine the best threshold parameter. Another issue is that the bwlabel() function would sometimes create objects that had huge areas and were clearly not cells. We solved this by implementing a minimum area for the cells we align to.

3. The Results

Original Image	**StackReg Alignment**	**Feature-based Image Registration**	Figure 4. Color overlay of two datasets showing the first (red) and last (green) registered timelapse images. Black indicates agreement between images. Original image (far left) from timelapse, results of the stackreg plugin (center), and the result after our image registration method (far right). Our results improve upon pre-existing registration software.
			
			

After we performed our algorithm, we observed that our image registration algorithm demonstrates a better alignment of target cells than both the unaligned original images and the StackReg adjusted images (Figure 4). Separate animated GIF files will be provided in the supplementary for easier comparison.

The takeaway from these results is that approaching image registration using a simple threshold segmentation works well for what we are specifically trying to do, and that more complicated approaches would have a significant cost in terms of computational time, which is something we want to minimize.

Running benchmark on the image_registration.m algorithm, we observed that for a set of images the time elapsed was around 0.1s between consecutive frames, without the need for mex functions. This satisfied our initial goal of 200 ms, though implementation of MEX functions will further improve use of this algorithm, particular in scenarios where the fields of view become more populated with cells.

Next Steps

On-the-fly cell identification and counting:

We have successfully developed a threshold-based cell counting estimation algorithm. We plan to incorporate it into the SLIP software to truly realize the self-adaptive image acquisition, permitting it to acquire more images at wells with lower densities, and reduce images for denser wells.

Sometimes, due to sample preparation, certain wells will deviate the pre-defined grids and thus SLIP will miss these wells. Now, with our on-the-fly cell identification algorithm, we can actually scan through a larger sampling area using a Levy-flight style searching, and find the regions where the cells actually locate. Therefore, we can further improve the yield of our SLIP protocol.

Feature-based Image Registration:

We successfully developed an image registration algorithm that improves upon current image registration package software with our specific examples. We plan to further develop our algorithm with the following two ways:

First, there are things we can do to improve the algorithm. Our current test set included fields of view with low cell count and low cell crowding. It will be important to implement the algorithm for fields of view with more crowded cells. One possible solution to this is that instead of performing the alignment to every cell, we can instead align to a randomly selected subset of cells.

The second thing that must be done is that after we have thoroughly tested the algorithm on a variety of data sets, we plan to incorporate this algorithm into the current SLIP software so that we can move the microscope based on the outputs of the algorithm.

Contributions

Both authors discussed and proposed the aims for this project. Handuo Shi designed and implemented Aim 1, and Amanda Miguel designed and implemented Aim 2. Both authors wrote the report.