Quantitative Phase Microscope Scanner

By
Kevin Han
Zelun Luo

Final Report for ECE 445, Senior Design, Spring 2014
TA: Aaron Rosenberg

May 4, 2014
Project No.58

Abstract

We implemented an image stitching program for quantitative phase microscope images, as well as a web-based image viewer and core segmentation program. The image stitching program is able to stitch entire slides consisting of hundreds of gigabytes of data, and the user can upload them to our server and view them remotely. Zooming and panning of images is supported. Finally, biopsy cores in the image can be automatically identified and output into separate images.

Table of Contents

I. Introduction

1.1 Purpose / usefulness of project

Researchers in cellular biology often study the nanoscale structure and functions of cells and microorganisms. Consequently, there is an increasing demand for high-resolution images of specimens. Modern optical microscopy techniques provide high-resolution images, but these images are only limited to a physically small area compared to the size of the sample slide. Therefore, images captured from a microscope must be joined together (stitched) to form a complete picture of the slide. Unfortunately, the stage's mechanical motion is not accurate enough to know the exact position of each image taken, so a stitching algorithm must be used to join them together seamlessly. Current stitching programs do not work on the image sizes needed, when the image of the entire slide is far too large to fit in RAM.

Our main goal for this project is to build a robust slide stitching program with high-resolution output images. We will optimize the algorithm described in [1] to work on large image sizes without consuming excessive computing resources. Also, the output image will be accessible through Internet with viewing, panning, and zooming.

Another goal for our project is to identify and segment biopsy cores in the final image, which are small circular pieces of tissue obtained from core needle biopsy. These cores are arranged in a grid on the slide, but their exact positions are not known beforehand, so to extract each one into a separate image for future processing requires an image segmentation algorithm.

1.2 Project functions

- Perform seamless and globally optimal image stitching on a moderate desktop computer, using less than 32 GB of RAM
- Digital library based on web-based image viewer, so that scientists around the world who don't have the access to the devices can visualize and navigate entire slides and do research based on the images scanned by our setup
- Segment cores in the image automatically and save to separate image files. Core segmentation should have false positive rate of <10% and false negative rate of <10%.

1.3 Blocks / Subprojects

Figure 1 shows the overall block diagram.

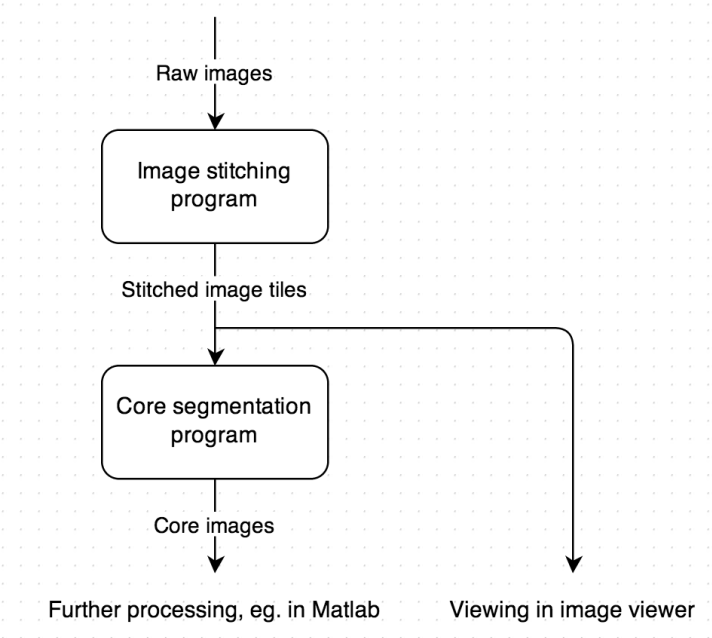

Figure 1. Overall block diagram.

1.3.1 Image stitching

Image stitching uses the algorithm of Prebisch *et al* [1]:

1.3.1.1 Phase correlation

The first step is to compute the translational offsets of all pairs of adjacent images. This is done using phase correlation, a commonly used algorithm for this purpose based on the Fourier transform.

1.3.1.2 Global optimization

After phase correlation outputs the offsets of all image pairs, conflicts between these offsets must be fixed. This is done by minimizing the least squares error between the phase correlation offsets and a set of new image positions.

1.3.1.3 Linear blending/tile generation

Finally, the output images must be generated. The output is in the form of non-overlapping images of a fixed size covering the slide. The positions computed from global optimization are used to generate the tiles, and linear interpolation is used for overlapping images. The output images are then repeatedly downsampled by factors of 2 to generate the "zoomed-out" images for the web-based image viewer.

1.3.2 Web-based image viewer

The output of image stitching can be viewed with CATMAID, an online image viewer developed at the Institute of Neuroinformatics, Switzerland. It allows zooming and panning, similar to Google Maps. We have set it up on a university server so that it can be accessed remotely and so that image datasets can be imported and displayed.

1.3.3 Core segmentation

Core segmentation is done using a heavily downsampled image. Briefly, a binary mask is generated using thresholding, k-means is performed to find the approximate centers of the cores, and then a moving window method is used to locate the cores exactly.

II. Design

2.1 General design alternatives

For the image stitching program, there were two main algorithms available to calculate the transformation between adjacent images: phase correlation and the scale-invariant feature transform (SIFT). Phase correlation can only handle translational offsets between images, but is simpler and requires less processing. SIFT extracts feature points from an image and matches them between images. It can detect more complex transformations between images such as rotation and scaling, but is more complicated and requires more processing steps. Since the microscope error is only translational, SIFT may introduce unwanted image warping, so we decided to use phase correlation.

For image viewing, the web-based image viewer was chosen due to its portability and ease of use (simply log on to a website to view images), as opposed to distributing large image files over FTP or email. Also, the size of the output image exceeds the maximum size of common image formats and the size supported by most of the currently existing image viewer, so the image needs to be splitted into tiles and is viewed through the web-based image viewer.

For core segmentation, there were several alternative approaches. Since the cores are roughly circular, the Hough circle transform can potentially be used to detect circles in the image. Image processing techniques such as Gaussian blur, erode, and dilate can be used to preprocess the dataset and remove noise. These techniques can also be combined to form the overall image processing pipeline. In our case, we used three main techniques: Otsu thresholding to generate a binary mask, k-means clustering to find the approximate core centers, and our own “moving window” method to find the exact core centers. This design appears to work well for our data and satisfies our accuracy requirements.

2.2 Equations / Simulations / General Circuits

As our project was software-only, we did not have any simulations or general circuits, although we have included some equations and a circuit schematic for our special circuit in the Appendix. The equations pertaining to phase correlation, global optimization, and image blending can be found in their respective subsections in section 2.3.1.

2.3 Detailed description of design

2.3.1 Image stitching

First, we give an overview of the input and output formats of image stitching:

- Input data - image data comes in the form of 32-bit floating point TIFF images captured from the camera. Each image is 2544 x 2160 pixels in size, or about 21MB. The microscope also outputs a text file containing what it believes to be the position of each image. This is used to guide our algorithm as explained in 2.3.1.1.
- Output data - the final output is in square TIFF or JPG images of a user-defined size covering the entire dataset. Each image is a tile with the filename corresponding to the position and zoom level: y_x_z.jpg denotes a tile with position (x, y) and zoom level z. The minimum zoom level 0 is

the most zoomed in, and each successive zoom level is zoomed out by a factor of 2. The total number of zoom levels can be configured. This format is the input format for our image viewer.

2.3.1.1 Phase Correlation

The phase correlation method is a well-known image processing method, which was proposed by Kuglin and Hines in 1975 [2]. It is a method of image registration, and uses a fast frequency-domain approach to estimate the relative translational offset between two similar images. While the basic method can be used to register the images which have been shifted (translation) relative to one another, it can be extended to support affine transformation, which includes rotations, translations, reflections, scalings and their combination. In our case, there is only translations involved for neighboring images, so the basic phase correlation is sufficient to solve the problem.

More specifically, this method relies on the Shift Theorem: If we have two images differing only by displacement:

$$f_2(x, y) = f_1(x-d_x, y-d_y),$$
$$F_2(\omega_x, \omega_y) = F_1(\omega_x, \omega_y)e^{-i(\omega_x d_x+\omega_y d_y)}.$$

The two images thus have the same Fourier magnitude, but differ in phase to a degree directly proportional to the displacement. And the cross-product spectrum should provide the phase difference between the two images. Here are the steps to obtain the final result:

1. Given two images g_a and g_b, we first calculate the discrete 2D Fourier transform of both images:

$$G_a = F\{g_a\}, G_b = F\{g_b\}.$$

2. Calculate the cross-power spectrum by taking the complex conjugate of the second result, multiplying the Fourier transforms together elementwise, and normalizing this product elementwise:

$$R = \frac{G_a \circ G_b^*}{|G_a G_b^*|},$$

where ∘ is the Hadamard product (entry-wise product).

3. Obtain the normalized cross-correlation by applying the inverse Fourier transform:

$$r = F^{-1}\{R\}.$$

4. Determine the location of the peak in r:

$$(\Delta x, \Delta y) = \text{argmax}_{(\Delta x, \Delta y)}\{r(\Delta x, \Delta y)\}.$$

If all the steps are done properly, the output should be similar to the peak in Figure 2.

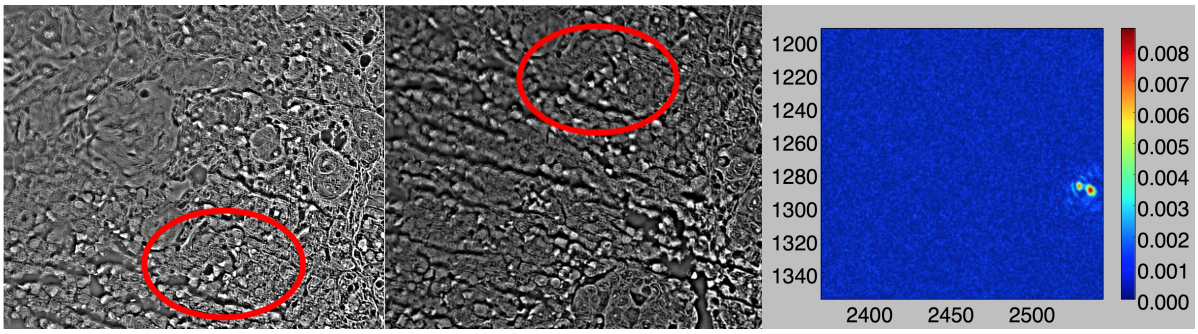

Figure 2. Phase correlation of two vertically adjacent images acquired from microscope. Red circles represent the same region on the slide. Output image is highly zoomed in so peak is visible.

The peak then corresponds to the offset between the two images. Note that since it is essentially computing the 2D circular convolution of the two images, there are four potential shifts corresponding to one peak. This is where the microscope position data comes in - we choose the shift that is closest to the microscope's guess for the offset between the two.

Implementation

We used C++ to implement the algorithm. The libraries we used are OpenCV and FFTW:

- OpenCV (Open Source Computer Vision Library) is a library for real-time computer vision, originally developed by Intel. It has a lot of functionality, including face recognition, image processing, HCI and so on. In our case, we use OpenCV to load and process images for phase correlation, as well as for blending.
- FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, with any input size, and of both real and complex data. In our case, we use FFTW extensively to perform all of our Fourier transforms. FFTW generally has the best performance compared to other libraries, and the difference becomes significant in our project because the Fourier transform is performed thousands of times.

2.3.1.2 Global Optimization

After phase correlation yields a set of translation vectors, one for each pair of adjacent images, there will be some conflicts between them, as shown in Figure 3.

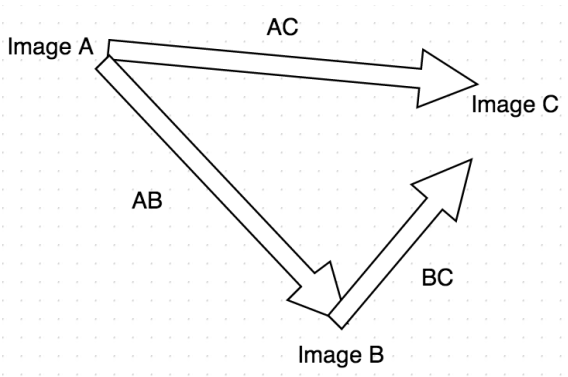

Figure 3. Conflicts between translation vectors. Images A, B, and C are all adjacent. Vectors **AB**, **BC**, and **AC** are those computed from phase correlation.

Ideally, **AB** + **BC** = **AC**. However, this is not the case - we cannot say for sure where image B or C belongs relative to A. Therefore, there must be some method to compute a set of globally consistent image positions. This is done using least squares minimization.

First, we pick the top-left image as the fixed image, F. Then, for every other image I, the goal is to compute $\mathbf{t}_{FI}$, the vector from F to I, in order to minimize the sum of squared distances between all the old translation vectors (computed from phase correlation) and those formed from the new ones $\mathbf{t}_{FI}$. The error term is:

$$E = \sum_{\text{adjacent images I,I'}} \left| t_{FI} - t_{FI'} - t_{I'I} \right|^2,$$

where $\mathbf{t}_{I'I}$ is computed from phase correlation**.** Setting the gradient of *E* with respect to each coordinate of each vector $\mathbf{t}_{FI}$ equal to zero yields a set of linear equations, which can be solved to get the final image positions $\mathbf{t}_{FI}$.

Implementation

We used the library TMINRES (https://code.google.com/p/tminres/), a sparse linear solver, to perform global optimization. It is designed for sparse symmetric matrices so it fits our problem well. It is also written in C++ so we can directly integrate it with our main program.

2.3.1.3 Linear Blending/Tile Generation

Once the translations have been computed, the final task is to actually generate the output image. Since it is impractical to generate a single 80 gigapixel image, which is difficult to load and process all at once, we split the output into non-overlapping tiles of configurable size. In this way, subregions can be loaded and viewed without loading the entire output.

For each tile, the first step is find all the images within that tile. This is done by constructing a mapping from tile coordinates to a set of image grid coordinates - for each image, add it to the sets corresponding to the tiles that contain it.

Then, for each tile, we apply a linear weighting function to each image contained within, and add them. For example, blending two images together:

$$I_{out}(x,y) = \frac{I_1(x,y)W(x,y) + I_2(x - x_0, y - y_0)W(x - x_0, y - y_0)}{W(x,y) + W(x - x_0, y - y_0)},$$

where I_{out} is the output image and image I_2 is offset by (x_0, y_0) relative to I_1. The linear weighting function is a pyramid shape. It is designed to taper to zero at the edges so that blending is smooth:

$$W(x,y) = \begin{cases} 1 - \left|\frac{2y}{h}\right|, |y| > |x| \\ 1 - \left|\frac{2x}{w}\right|, |y| < |x| \end{cases},$$

where x runs from $-w/2$ to $w/2$, and y runs from $-h/2$ to $h/2$. Pseudocode for the algorithm:

```
for each tile
      initialize tile image to all 0
      initialize weight image of same size to all 0
      for each image contained in tile
            multiply image by weight function
            add it to tile image at correct position
            add weight function to weight image at correct position
      divide tile image by weight image entry-wise
      save tile image
```

Thus, at each pixel the "numerator" of the output image I_{out} is calculated first, then dividing by the "denominator" is done all at once at the end.

Each tile is saved at zoom level 0 (format described above in output data). Then another program reads images in groups of 4, composes them together and downscale them by 2 in each dimension, to form zoom level 1. This is repeated until the desired outermost zoom level is reached.

2.3.2 Web-based Image Viewer

The image viewer software we are using is CATMAID, the Collaborative Annotation Toolkit for Massive Amounts of Image Data (http://catmaid.org/index.html). It is designed for viewing and annotating large image datasets like ours. It expects the input images to be in a specific format, described in part A above. It is still being actively developed and contains some bugs. Our work was to get it working on the lab server. We followed the instructions on the CATMAID website (under "Basic Installation Instructions"), but had to change some settings and debug the setup since we encountered many issues, such as missing libraries and server errors.

In the end our setup consisted of an nginx frontend forwarding requests to gevent (a Python server), with Django (a Python web framework) as the backend. nginx was chosen for its high performance and

low memory footprint, and gevent was needed to provide a WSGI interface for Django, the framework CATMAID uses. A screenshot of CATMAID is shown in Figure 4.

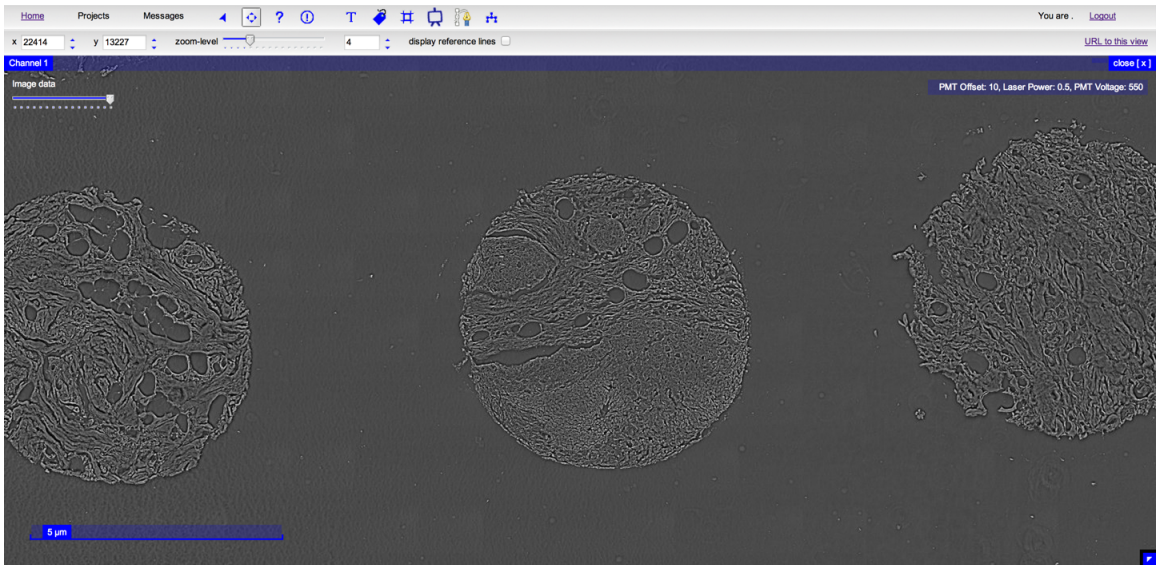

Figure 4. Screenshot of CATMAID viewing our output data.

2.3.3 Core Segmentation

Another goal for our project is to identify and segment biopsy cores in the final image, which are small circular pieces of tissue obtained from core needle biopsy. These cores are arranged in a grid on the slide, but their exact positions are not known beforehand, so to extract each one into a separate image for future processing requires an image segmentation algorithm.

Figure 5 shows the result of our image stitching algorithm with 2480 images and 2544 * 2160 pixels each. Each high-intensity region corresponds to a small piece of issue, or, core, which is used for research in histology. The steps of core segmentation are as follows:

1. Generate a binary mask so that the features of images will be easier to analyze. Thresholding is one of the simplest and most effective way to generate the binary image from the grayscale image we obtained from image stitching. Instead of setting the parameter of thresholding manually, we use Otsu's method [3] to automatically perform clustering-based image thresholding. The output of thresholding is shown in Figure 6. Note that error pixels can be caused by dirt, dust, and debris on the microscope slide.

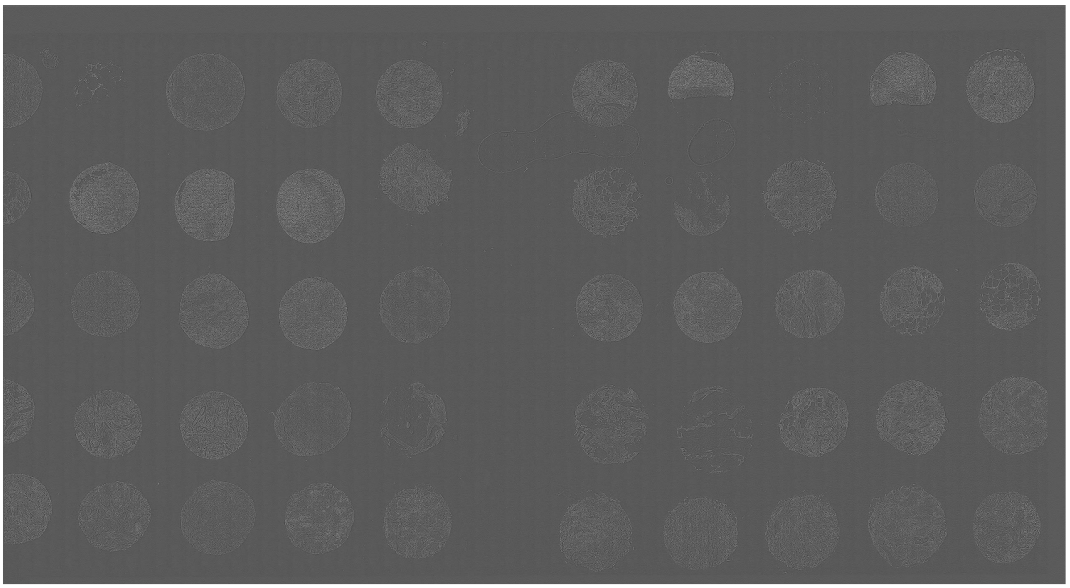

Figure 5. Output of image stitching.

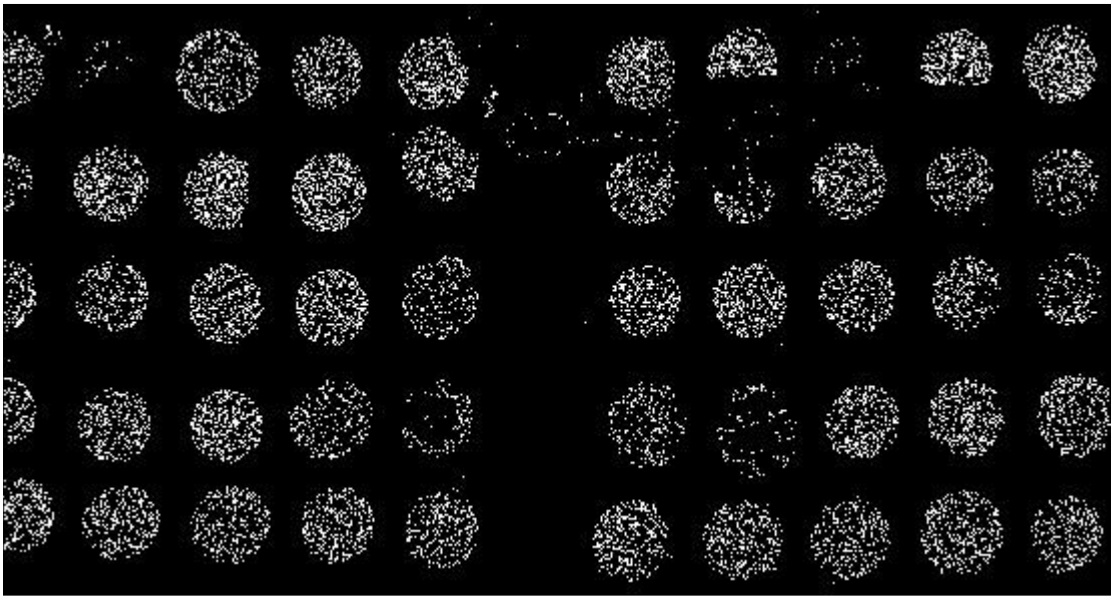

Figure 6. Binary image after applying Otsu's method of thresholding.

2. The next step is to find the approximate centers of the clusters using k-means clustering. If there is no noise in the image, we can set the number of clusters in k-means to the number of cores. However, since there is some noise, we use twice the number of cores as the number of clusters to find, so that cores will not be missed.

3. Calculate the centers of windows recursively. For each cluster center (x, y) found by k-means, we place a window centered at (x, y). The window size is set beforehand by the user. For each window, we find the center of mass (x', y') of all the pixels within the window, and set the new window to center at

this point. Each center is recalculated recursively until it is totally converged. If implemented correctly, some centers will have much higher frequency of occurrence than others. We select the center for each output tile by choosing the k most frequent cores, where k is the number of cores in the image that we specify.

4. Using the center positions, generate the output images from the full-resolution tiles of image stitching.

2.4 Flow diagrams

Figures 7 and 8 show the flow diagrams of image stitching and core segmentation, respectively.

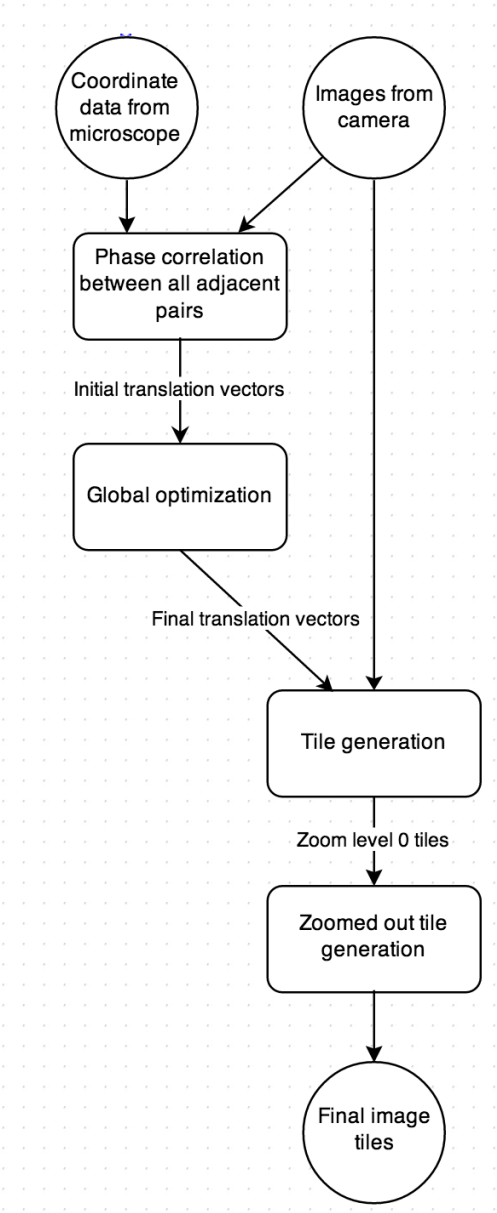

Figure 7. Flowchart of image stitching module.

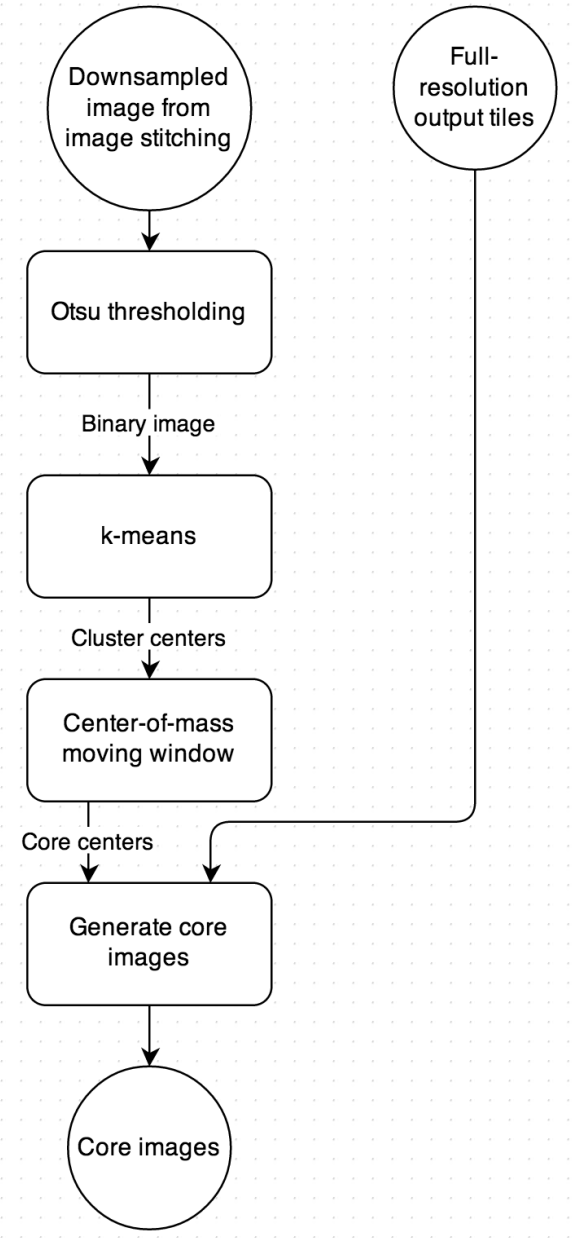

Figure 8. Flowchart for core segmentation.

III. Requirements and Verification

3.1 Requirements

For image stitching, one of the requirements is that the algorithm can run with 32 GB of RAM or less, since that is the amount of RAM on the lab computer. Since the quality of output images is subjective and depends on the characteristics of the input dataset, it is difficult to formulate quantitative requirements for the image quality. So, for quantitative verification, we use individual test cases for phase correlation, global optimization, and tile generation to verify that each part of the image stitching algorithm is implemented correctly. The test cases are as follows:

For phase correlation, we split up a single microscope image into partially overlapping images, then ran phase correlation and checked if the peak position matched the actual offset.

For global optimization, we ran the algorithm on an example 2x2 image grid that can be solved by hand, and checked with computed results. If the optimized offsets matched the hand computed ones within 0.1 pixels, the test was passed.

For tile generation, we split up a single microscope image, used the hard-coded offset to regenerate the original image, and checked if all pixels match within 2 pixel intensity values (out of 255).

For the web-based image viewer, the requirements are to be able to access images remotely and be able to zoom and pan around. Also, we require that images can be imported onto the server to be viewed remotely.

For core segmentation, our mentor emphasized that complete accuracy was not needed, since it is not too labor-intensive to manually exclude incorrect cores found. So, we decided to require a false positive rate of <10%, and false negative rate of <10%. False positive is defined as a non-core identified as a core, and false negative is when a core is not identified.

3.2 Testing procedure

3.2.1 Image stitching

We tested image stitching on four real datasets:

- 324 images; 6.7 GB
- 2,380 images; 49 GB
- 20,453 images; 419 GB (full slide, biopsy cells)
- 8,300 images; 234 GB (full slide, less overlap, breast cells)

To confirm that we implemented the image stitching algorithm correctly, we simply ran the individual test cases.

3.2.2. Web-based image viewer

We can simply access the image viewer online and open a dataset to view, zoom, and pan around.

For image importing, we access the importer on the admin page of the server, and enter the directory of the dataset to import as well as image size and project name. Then we can go back to the main page and verify it was imported successfully.

3.2.3. Core Segmentation

To verify that we satisfy our accuracy requirements (false positive rate of <10% and false negative rate of <10%), we simply count the occurrence of false positive and false negative segmentations in the output.

3.3 Quantitative results

All image stitching test cases were passed successfully. Phase correlation returned the peak corresponding to exactly the correct offset, global optimization produced offsets that were well within 0.1 pixels (0.000023, 0.000013, and 0.000023 pixels respectively), and no pixels in the generated tile differed by more than 2. The algorithm was able to run with less than 32 GB of RAM in all cases.

Core segmentation satisfied our accuracy requirements, with a false positive rate of 2% and false negative rate of 0% on dataset two, and false positive rate of 1.19% and false negative rate of 1.49% on dataset three. Segmentation results are shown in Figures 9, 10, and 11.

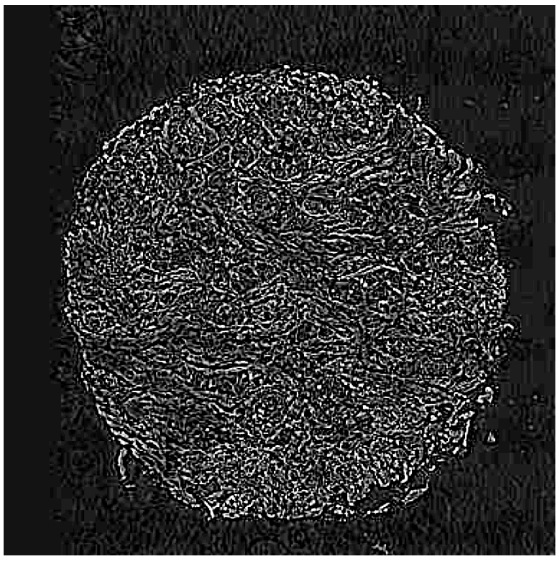

Figure 9. Single core output from core segmentation. Contrast has been enhanced for easier viewing.

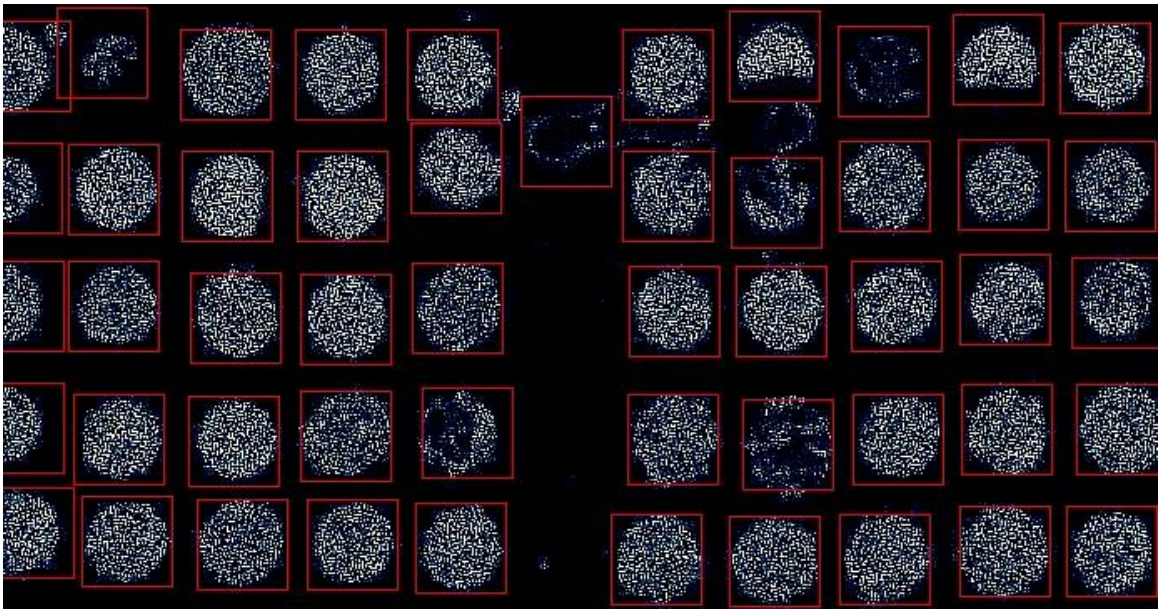

Figure 10. Cores identified in dataset two. Contrast has been enhanced for easier viewing.

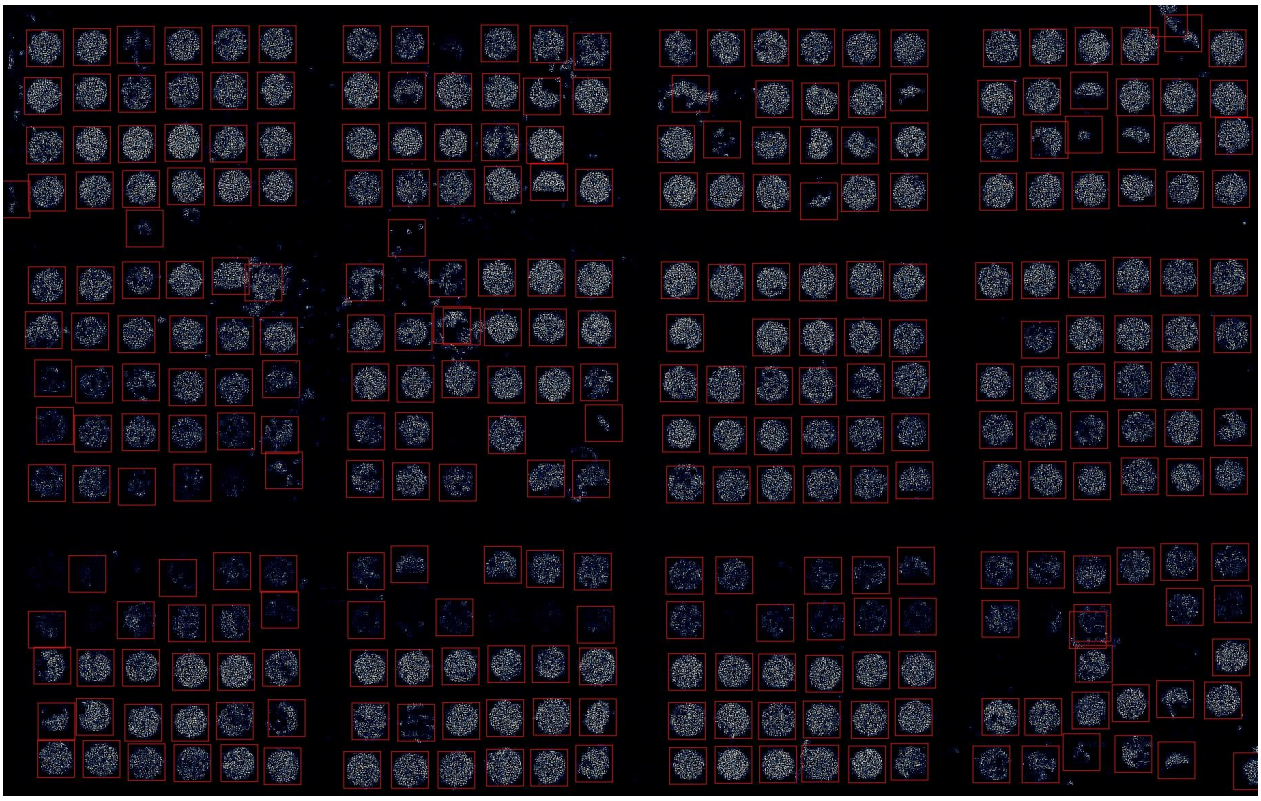

Figure 11. Cores identified in dataset three. Contrast has been enhanced for easier viewing.

3.4 Discussion of results

The web-based image viewer was able to import datasets, as well as view, zoom, and pan remotely. Core segmentation also passed our requirements.

Output image quality was adequate for the first three datasets, but the last one showed some

registration errors, where the offset between some images was visibly incorrect, as seen in Figure 12. This was due to the decreased overlap between input images (30% horizontal overlap and 8% vertical overlap, versus 46% horizontal and 36% vertical overlap for the other datasets), which shows that a sufficiently large overlap is necessary for adequate image quality. We believe this is because phase correlation can be inaccurate for adjacent images with mostly empty space without many features to correlate, especially when there is little overlap between them. This introduces error that propagates into the non-empty regions when global optimization is used. In the future perhaps a weighted least squares approach would provide better results for low-overlap input, where the weight would depend on the peak intensity from phase correlation.

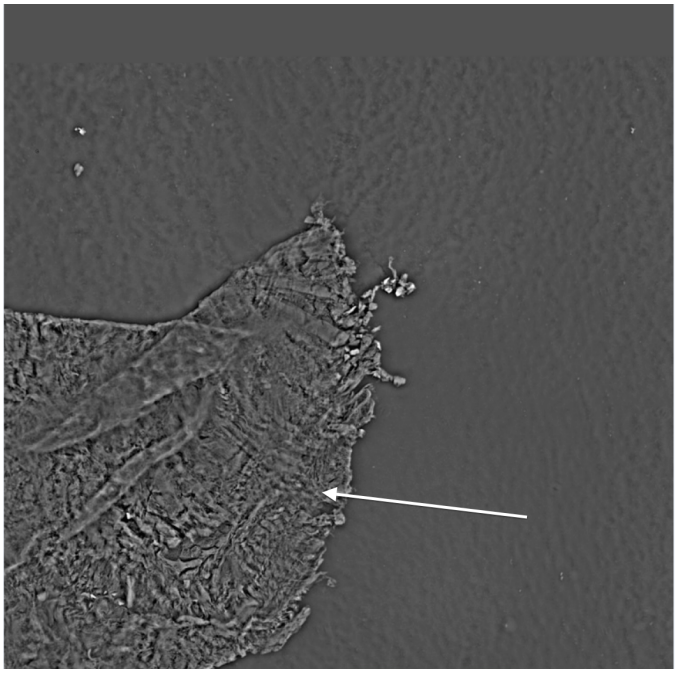

Figure 12. An output image from dataset 4. Registration error is visible as a "doubling" of the features, indicated by the arrow.

IV. Costs

Table 1 shows the cost of parts, Table 2 shows the labor costs, and Table 3 shows the total cost.

4.1 List of parts and equipment needed

Table 1. Cost of parts.

Item	Unit cost	Quantity	Cost
Computer CPU: Intel Core i7 RAM: 32 GB	$2,000	1	$2,000

4.2 Ideal salary (hourly rate) x actual hours spent x 2.5

Table 2. Labor costs.

Name	**Dream salary (Hourly rate)**	**Hours per week**	**Number of weeks**	**Total**
Zelun Luo	$40.00	15	15	$22,500
Kevin Han	$40.00	15	15	$22,500
				$45,000

4.3 Grand total

Table 3. Total cost.

Section	**Total**
Labor	$45,000
Parts	$2,000
Total	$47,000

V. Conclusion

5.1 Accomplishments

We have accomplished all of the goals of our project. We have built image stitching software that successfully stitched real image datasets from the microscope, set up a web-based image viewer that is able to load, view, zoom, and pan images, and developed accurate core segmentation software.

5.2 Uncertainties

We are not sure what the best way is to deal with low-overlap input datasets. We can simply require the input to have sufficient overlap, but that increases scanning time and the size of the input data. In the future we can try different approaches such as weighted least squares, or even performing image stitching during the scanning phase to identify where the regions of interest are to scan, and thus avoiding empty space entirely.

5.3 Ethical considerations

The purpose of this project is to develop slide scanning and processing software, which greatly facilitates people who do research in microbiology. With the functions that make contributions to cutting-edge research, our design is consistent with code of the IEEE Code of Ethics:

1. To accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;
Although our system is built for biological use, we never produce anything with biohazard, and we will dispose waste which has potential danger properly.

3. To be honest and realistic in stating claims or estimates based on available data;
Since we are pursuing a high performance in our slide scanner, the data we get is significant for the evaluation of our project. We will be honest and never falsify any data acquired from our project. Due to the limitation of hardware devices, all margin of errors will be indicated specifically with confidence interval.

4. To reject bribery in all its forms;
There is no motive of bribery in our project and we promise not to commit bribery in all forms.

5. To improve the understanding of technology; its appropriate application, and potential consequences;
This project enhances our understanding in computer engineering and the cutting-edge area of its application dramatically. We hope to facilitate other people's research with our setup and receive feedbacks in order to optimize it.

7. To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;
We are mentored by Professor Carney, Professor Singer, and our TA Aaron Rosenberg. We also meet bi-weekly with our research mentor Mikhail Kandel and Professor Popescu in order to have a review of our

progress. We welcome all forms of advice and criticism, since they help us further improve our project and skills in solving research problems.
Our project is based on the results of many previous researcher in this areas and we will cite all the related research papers properly. We also appreciate the help of all our research mentors.

9. To avoid injuring others, their property, reputation, or employment by false or malicious action;
We do not deal with any hardware devices with high-voltage and we will dispose everything with biohazard properly.

5.4 Future work / Alternatives

- Further optimize the code in order to improve the speed
- Integrate our post-processing part with the slide scanning part that was finished by other people in our research lab and build the whole slide scanner
- Look into other methods such as weighted least squares to improve performance on low-overlap datasets
- Get ready for the publication

VI. References

[1] S. Preibisch, S. Saalfeld, and P. Tomancak. "Globally optimal stitching of tiled 3D microscopic image acquisitions." *Bioinformatics.* [Online]. Vol. 25(11), pp. 1463-1465. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2682522 [Feb 28, 2014].

[2] Phase correlation. (n.d). [Online]. Available: https://en.wikipedia.org/wiki/Phase_correlation

[3] N. Otsu. "A threshold selection method from gray-level histograms". *IEEE Trans. Sys., Man., Cyber*. Vol. 9(1): pp. 62–66. doi:10.1109/TSMC.1979.4310076.