

Mobile Device Reference Apps to Monitor and Display Biomedical Information

by

Ethan Mark Grother

B.S., Kansas State University, 2015

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Electrical & Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2017

Approved by:

Major Professor
Steve Warren

Copyright

© 2017 Ethan Grother

Abstract

Smart phones and other mobile technologies can be used to collect and display physiological information from subjects in various environments – clinical or otherwise. This thesis highlights software app reference designs that allow a smart phone to receive, process, and display biomedical data. Two research projects, described below and in the thesis body, guided this development. Android Studio was chosen to develop the phone application, after exploring multiple development options (including a cross-platform development tool), because it reduced the development time and the number of required programming languages.

The first project, supported by the Kansas State University Johnson Cancer Research Center (JCRC), required a mobile device software application that could determine the hemoglobin level of a blood sample based on the most prevalent color in an image acquired by a phone camera, where the image is the result of a chemical reaction between the blood sample and a reagent. To calculate the hemoglobin level, a circular region of interest is identified from within the original image using image processing, and color information from that region of interest is input to a model that provides the hemoglobin level. The algorithm to identify the region of interest is promising but needs additional development to work properly at different image resolutions. The associated model also needs additional work, as described in the text.

The second project, in collaboration with Heartspring, Wichita, KS, required a mobile application to display information from a sensor bed used to gather nighttime physiological data from severely disabled autistic children. In this case, a local data server broadcasts these data over a wireless network. The phone application gathers information about the bed over this wireless network and displays these data in user-friendly manner. This approach works well when sending basic information but experiences challenges when sending images.

Future work for both project applications includes error handling and user interface improvements. For the JCRC application, a better way to account for image resolution changes needs to be developed, in addition to a means to determine whether the region of interest is valid. For the Heartspring application, future work should include improving image transmissions.

Table of Contents

List of Figures	vi
List of Tables	ix
Chapter 1 - Introduction and Background	1
Johnson Cancer Research Project	1
Heartspring Bed Project.....	2
Chapter 2 - Image Processing	4
Finding the Region of Interest	5
Converting the RGB Image to the YIQ Color Space.....	5
Applying a Threshold to the Image.....	7
Image Morphology.....	10
Finding the Circle in the Image	11
Information Needed from the Region of Interest.....	13
Finding Red, Green, and Blue Peaks	13
Calculating Hemoglobin Level	16
Results and Issues	16
Results and Issues: Region of Interest Algorithm and RGB Peak Values.....	16
Results and Issues: Hemoglobin Level Model.....	19
Chapter 3 - Local Data Server	23
Choosing the Data Transmission Method.....	23
Background on Network Sockets and TCP	24
Overall System.....	24
Results and Issues	26
Chapter 4 - Software	28
Software Architecture	28
JCRC Class Diagram	28
Heartspring Class Diagram	30
Software Development Approach.....	33
Interactive Development Environments and Frameworks/Development Kits	33
Visual Studio and .NET Framework.....	34

Android Studio and Development Kits	34
Results and Issues	35
Issues	38
Chapter 5 - Conclusions and Future Work	39
JCRC App Conclusion and Future Work	39
Heartspring App Conclusion and Future Work	40
References	42
Appendix A - Region of Interest Test Images	45
Image 1 Results	46
Image 2 Results	47
Image 3 Results	48
Image 4 Results	49
Image 5 Results	50
Image 6 Results	51
Image 7 Results	52
Image 8 Results	53
Image 9 Results	54
Appendix B - Other Images	55
Appendix C - Test Images Created for the Enhanced Model Tests	57
Appendix D - Sample Images for Model Definitions	60

List of Figures

Figure 1.1. Overview of the JCRC system [1].	2
Figure 1.2. An overview of the bed monitoring system being developed for Heartspring [2].	3
Figure 2.1. Typical images of a slide post-reaction.	4
Figure 2.2. Images illustrating the second component of the YIQ color space.	6
Figure 2.3. 3D image of the hue of the first image with an average value plane.	8
Figure 2.4. 3D image of the hue of the second image with the average plane.	8
Figure 2.5. The original images with a threshold applied.	9
Figure 2.6. Images with both thresholds and morphology applied.	11
Figure 2.7. Images and their corresponding regions of interest.	12
Figure 2.8. R, G, and B histograms for the first example image.	14
Figure 2.9. R, G, and B histograms for the second example image.	15
Figure 2.10. An example image indicating why a minimum radius is needed.	17
Figure 2.11. Color gradient scale created for testing models	19
Figure 2.12. Bar plot – expected level values in comparison to the values calculated from the two Hgb models.	20
Figure 2.13. Each model’s absolute error versus the expected value for each image.	20
Figure 2.14. Comparison of actual Hgb levels compared to computed Hgb levels for the sample images provided.	21
Figure 2.15. Error between the computed Hgb levels and the actual Hgb levels for the sample images.	22
Figure 3.1. High-level depiction of the proposed system.	25
Figure 3.2. Data packet structure for server-to-client transactions.	25
Figure 3.3. Software interface for the test program to simulate the bed system.	26
Figure 4.1. Class diagram for the JCRC project.	30
Figure 4.2. Class diagram for the Heartspring project.	31
Figure 4.3. Screenshots of the two layouts for the JCRC phone application.	36
Figure 4.4. Screenshots of the Heartspring application that display “real time” plots for simulated sleep stage and heart rate data.	37
Figure 4.5. Screenshots of the Heartspring application's IR image viewer and settings layout.	38

Figure A.1. Region of interest results for image number 1.	46
Figure A.2. Region of interest results for image number 2.	47
Figure A.3. Region of interest results for image number 3.	48
Figure A.4. Region of interest results for image number 4.	49
Figure A.5. Region of interest results for image number 5.	50
Figure A.6. Region of interest results for image number 6.	51
Figure A.7. Region of interest results for image number 7.	52
Figure A.8. Region of interest results for image number 8.	53
Figure A.9. Region of interest results for image number 9.	54
Figure B.10. Images for which the region of interest algorithm was unsuccessful.	56
Figure C.11. Color gradient scale created to assign scalar values to RGB triplets.....	57
Figure C.12. Single-value RGB images created to validate the model calculations.	59
Figure D.13. Subject 1 images for an Hgb level of 9.8. First image – from a test tube. Second image – from a microfluidic chip.....	60
Figure D.14. Subject 2 images for an Hgb level of 9.7. First three images – from a test tube. Last two images – from a microfluidic chip.	60
Figure D.15. Subject 3 images for an Hgb level of 9.5. First three images – from a test tube. Last two images – from a microfluidic chip.	61
Figure D.16. Subject 4 images for an Hgb level of 14.5. First image – from a test tube. Second image – from a microfluidic chip.....	61
Figure D.17. Subject 5 images for an Hgb level of 8.3. First two images – from a test tube. Last image – from a microfluidic chip.....	61
Figure D.18. Subject 6 images for an a Hgb level of 10.2. First image – from a test tube. Second image – from a microfluidic chip.....	62
Figure D.19. Subject 7 images for an Hgb level of 6.0. First image – from a test tube. Second image – from a microfluidic chip.....	62
Figure D.20. Subject 8 images for an Hgb level of 10.1. First image – from a test tube. Second image – from a microfluidic chip.....	62
Figure D.21. Subject 9 images for an Hgb level of 14.1. First two images – from a test tube. Last image – from a microfluidic chip.....	62

Figure D.22. Subject 9 images for an Hgb level of 7.5. First three images – from a test tube. Last image – from a microfluidic chip..... 63

Figure D.23. Subject 10 images for an Hgb level of 7.3. First three images – from a test tube. Last image – from a microfluidic chip..... 63

Figure D.24. Subject 11 images for an Hgb level of 9.7. First three images – from a test tube. Last three images – from a microfluidic chip. 63

List of Tables

Table 2.1. Comparison of ImageJ results to MATLAB results.	18
Table 5.1. Suggested packet definition for the system.	41

Chapter 1 - Introduction and Background

Smartphones are being used more and more to collect information and display it in a manner that is useful to the user. For example, phone applications such as Google Maps and Waze collect position information and user input from smartphone users. Then those applications use that information to determine what traffic is like so that they can alter suggested directions to help the user more quickly reach a destination. Even though several phone applications exist for transportation and other day-to-day needs, applications that collect, analyze, and display biomedical information are not as common.

The work summarized in this thesis relates to two phone applications (apps) that assist with point-of-care projects. An overview of these projects is provided in the next two sections. Chapter 2 describes the first project in more detail: a method to collect biomedical information from an image. Chapter 3 then describes the second project: a method to transmit biomedical information from a system to a phone, where it is subsequently displayed. Those chapters are followed by a chapter that discusses the development process, the software architecture for each app, and the tools needed to create the two phone apps. Each of those chapters includes a results-and-issues section that assesses the performance of each app and describes issues that have been found while testing these apps.

Finally, Chapter 5 provides summary conclusions for the projects as well as future work. The future work points to suggested solutions for the issues mentioned in the preceding chapters as well as recommendations for additional app enhancements to pursue. This chapter also offers suggestions that should help to avoid issues in future development efforts.

Johnson Cancer Research Project

The first project was funded by the Johnson Cancer Research Center at Kansas State University [1]. The project's focus was to develop a point-of-care (POC) system to help diagnose and manage conditions such as anemia and ovarian cancer (refer to Figure 1.1). The overall system would consist of a device mounted on the back of a smartphone that would hold a microfluidic slide in the field of view of the phone's camera. The device mounted on the back of the phone would provide its own light, so the camera's flash would not be needed when acquiring images. A blood sample would be mixed with a reagent in the microfluidic slide, which would then be inserted into the device. Once the slide had been inserted, the phone would

take a picture of the slide, and an app would calculate, e.g., a hemoglobin level based off the color of the mixture (in the case of the anemia application). Once the hemoglobin level had been calculated, the app would display that level to the patient as well as upload it to a cloud server so that the patient's doctor would have access to the results.

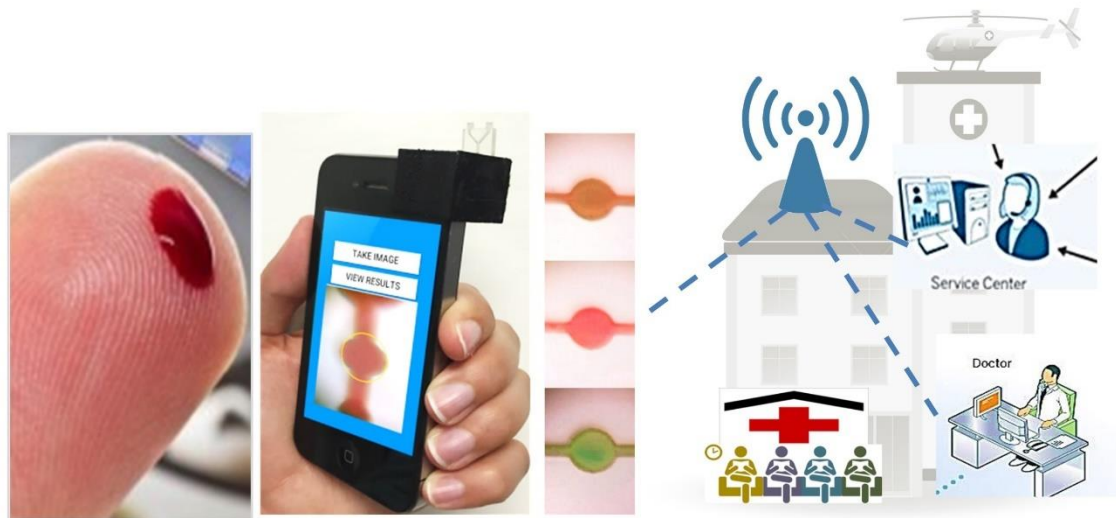


Figure 1.1. Overview of the JCRC system [1].

Heartspring Bed Project

The second project, a collaboration between Kansas State University and Heartspring in Wichita, KS, was funded by the National Science Foundation and focused on the development of a bed system for nighttime monitoring of children with severe disabilities [2, 3]. This bed system would be able to monitor sleep quality based on physiological information and allow the Heartspring staff to manage these children more efficiently. The bed system collects information from multiple sensors located on and around the bed, and then it transfers those data to a computer for analysis. Figure 1.2 depicts the bed monitoring system [4, 5].

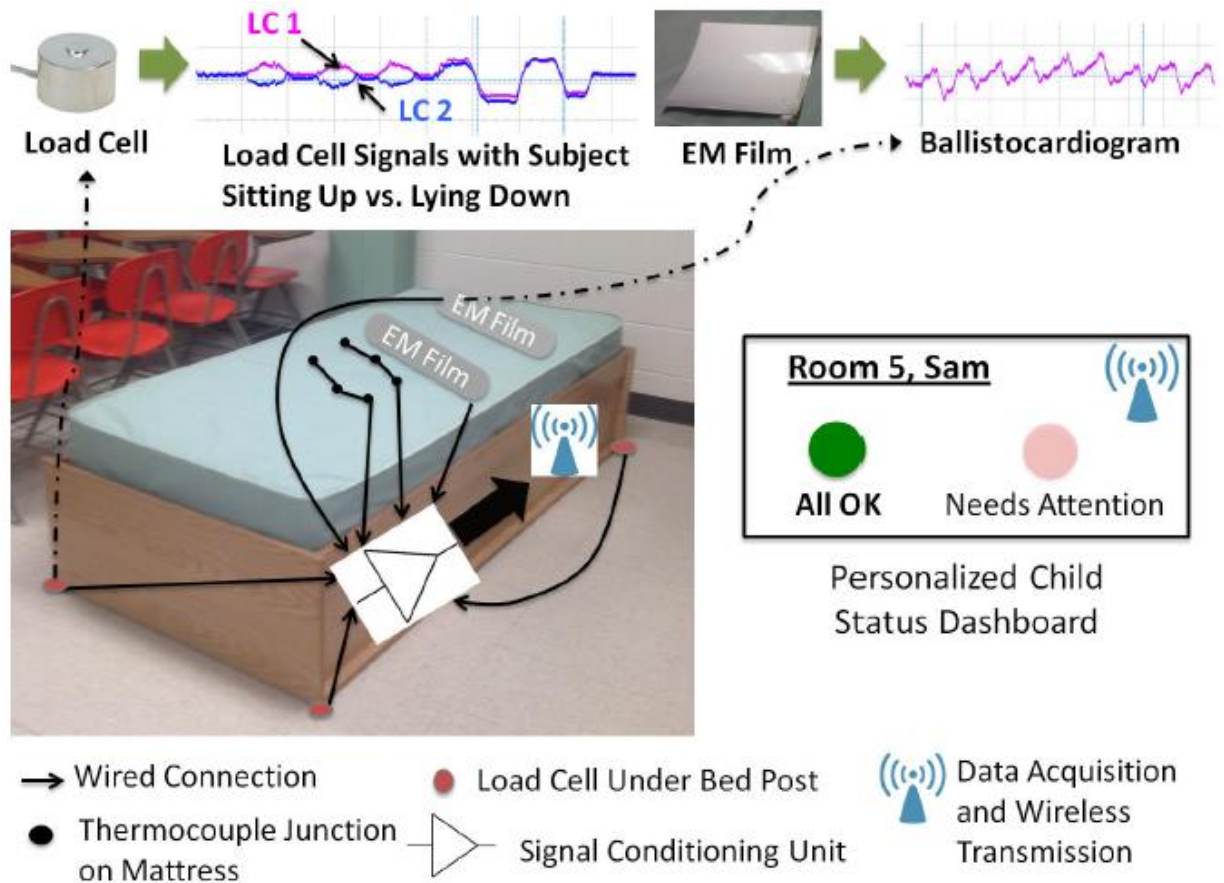


Figure 1.2. An overview of the bed monitoring system being developed for Heartspring [2].

Chapter 2 - Image Processing

The JCRC project noted in Chapter 1 was focused on using a phone application to detect a hemoglobin level based off a color that results from a blood sample mixed with a reagent. To find the hemoglobin (Hgb) level, an image of a slide is taken with the smartphone camera, as depicted in Figure 2.1, and the center circular region of the slide is singled out. This chapter goes on to discuss the process of finding the circular region of interest, how information from that region was used to calculate the Hgb level, and the results and current issues of the algorithm.

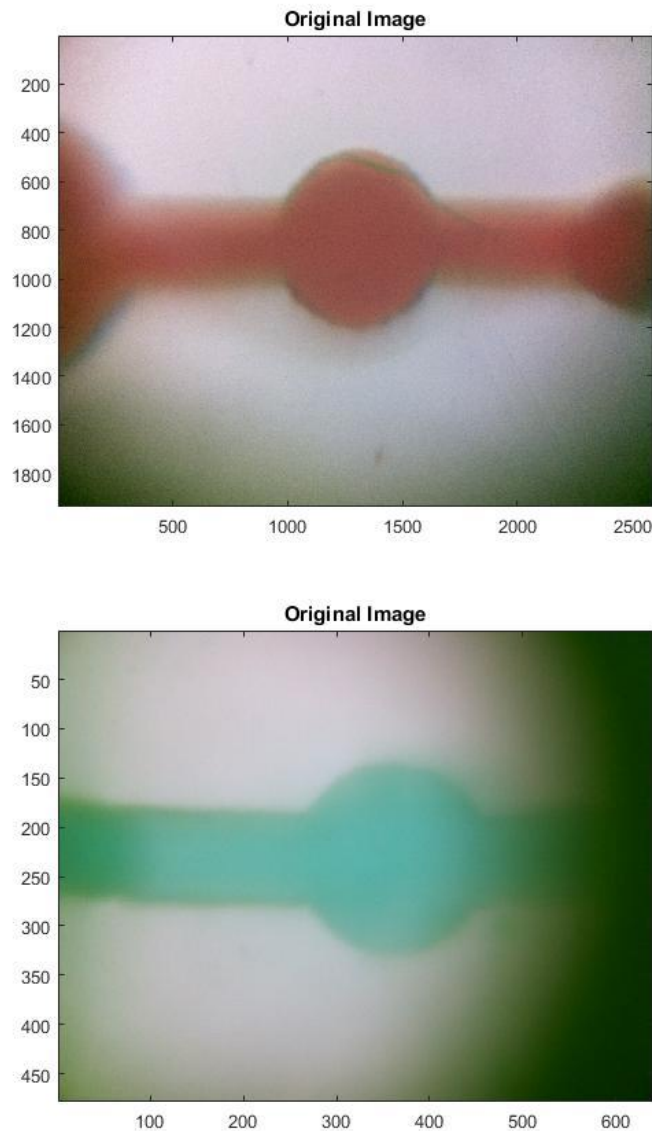


Figure 2.1. Typical images of a slide post-reaction.

Finding the Region of Interest

The most difficult part of this project was developing a method to identify the region of interest and how many pixels it included. This work involved converting MATLAB code into the Java programming language for the phone application, discussed in Chapter 4. This avoided the need to employ several built-in MATLAB functions. To find the region of interest, the following steps were followed, as described in the next four sections: convert the image from RGB color space to YIQ color space, apply a threshold to the image based off of one of the YIQ components, apply image morphology to the thresholded image, and then find the image circle.

Converting the RGB Image to the YIQ Color Space

The first step when finding the region of interest is to convert the image from the RGB color space to the YIQ color space. The RGB color space [6] is a way to store information about the image, where each pixel contains three 8-bit values for red, green, and blue. The YIQ color space [7] was developed in 1941 and was used throughout most of the Americas by the NTSC color TV system [8]. It is a three-component means to represent the RGB color space: a greyscale image (represented by “Y” – luma – useful for black and white television), plus two images that add color (“I” – the orange-blue range; “Q” – the purple-green range). The YIQ color space was chosen because of my familiarity with it from an undergraduate project and because the chrominance (IQ information) is helpful when distinguishing the region of interest in this type of slide image. Figure 2.2 contains images of the second component of the YIQ color space with respect to the images in Figure 2.1.

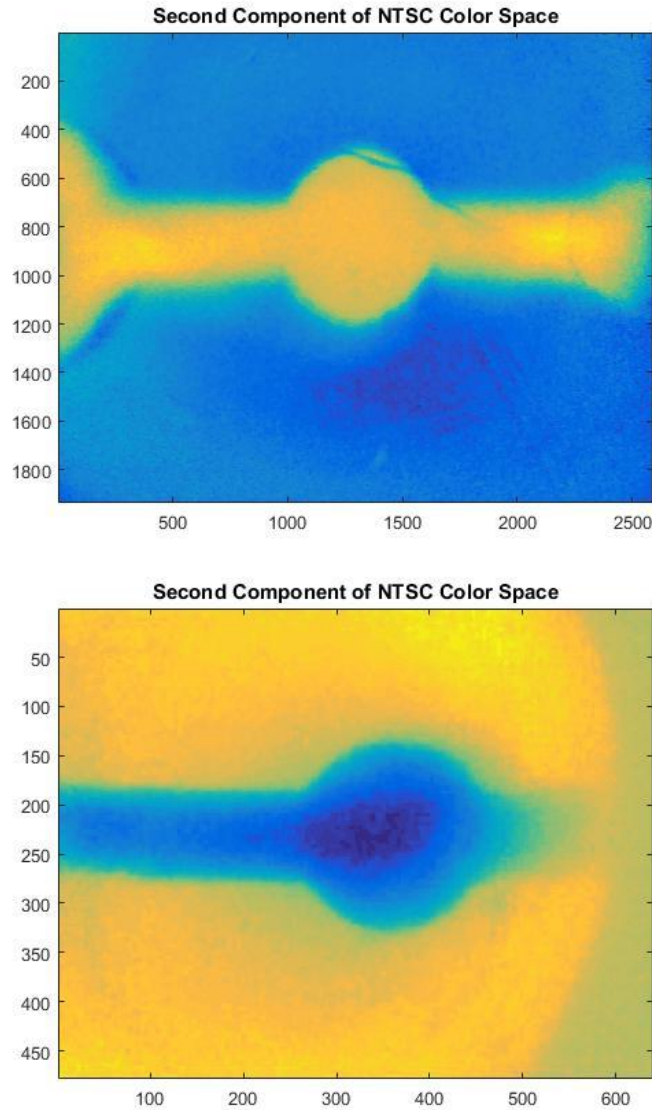


Figure 2.2. Images illustrating the second component of the YIQ color space.

To convert the RGB image to the YIQ format, MATLAB has a built-in function called “rgb2ntsc” that takes the original RGB image map as a parameter and converts it to the YIQ color space. The function returns an image with the three components as described above. The second component, also known as the hue, is used throughout the rest of the process to find the region of interest. As mentioned earlier, the MATLAB code needs to be easy to convert to Java in order to port to a smartphone, and converting from RGB to YIQ is just a matrix multiplication. Equation 1 is the matrix multiplication needed to convert an image from the RGB color space to

the YIQ color space. However, since we are just interested in the hue (I) [9] of the image, the system equation can be simplified to equation 2.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (1)$$

$$I = 0.596 * R - 0.274 * G - 0.322 * B \quad (2)$$

Applying a Threshold to the Image

Once the hue of the image had been found, a threshold that is based on either the minimum or the maximum hue of the image was applied to single out the region of interest. The first images used to test the algorithm were all similar in color, and all hue values of interest were positive numbers, so when the method was first created, only the maximum hue value multiplied by 50% was used to calculate a lower threshold for the hue levels of interest.

However, once a new set of images was received for testing, the method needed to be expanded because all of these images were on the opposite end of the color spectrum in comparison to the original images (i.e., the hue values were predominantly negative). The two images chosen as examples in Figure 2.2 were those that could illustrate why the change in the algorithm was needed. Instead of using the maximum hue value to calculate the threshold, the lower or upper threshold is now based off of the maximum or minimum hue value in the image, respectively, depending on whether the average hue of the image is positive or negative.

Figure 2.3 and Figure 2.4 represent both cases, where the 3D plot of the hue of each of the example images is accompanied by the average hue value plotted as a plane. Figure 2.3 illustrates why the original method of just using the maximum works – the hue values in the region of interest are positive, and the region of interest includes the maximum hue value. However, the image in Figure 2.4 has a region of interest that contains negative hue values, and that region includes the minimum hue value in the image. Also notice that the background plane in each image is close to zero on the hue (z) axis. Therefore, the average hue of the image was utilized as an indicator. If the region of interest was close to the maximum hue value, the average was observed to be positive, and if the region of interest was closer to the minimum hue value, the average was observed to be negative. So, if the average hue was positive, the maximum was

used to calculate a minimum threshold for the hue values. If the average hue was negative, the minimum was used to calculate a maximum threshold for the hue values.

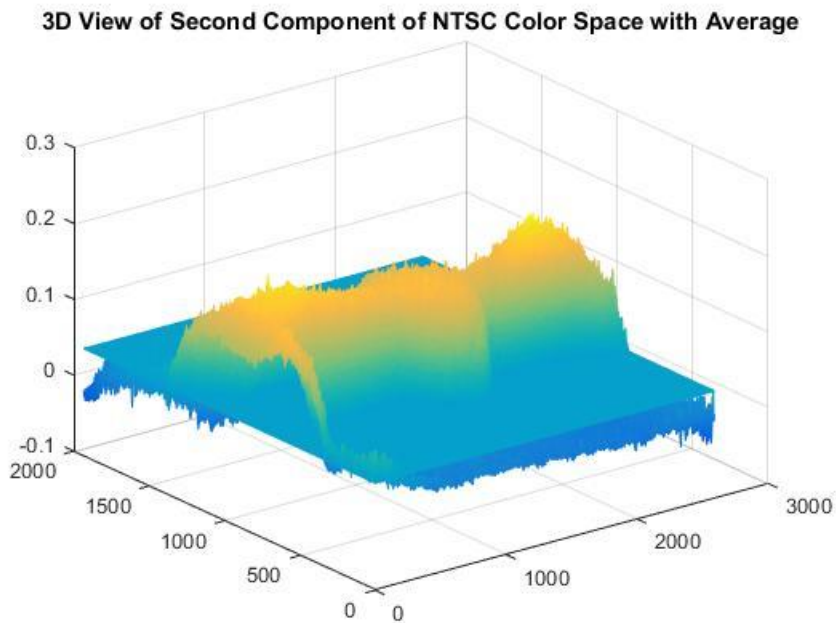


Figure 2.3. 3D image of the hue of the first image with an average value plane.

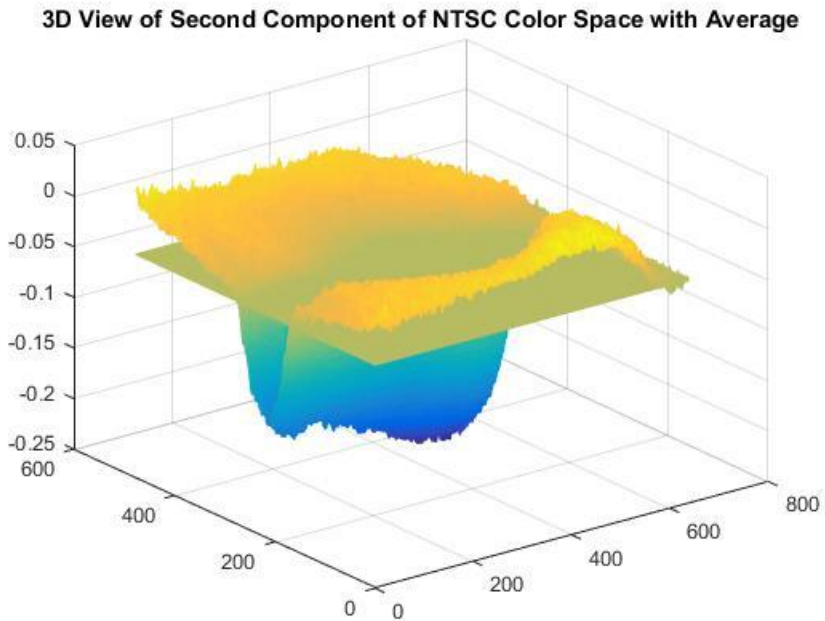


Figure 2.4. 3D image of the hue of the second image with the average plane.

If the maximum hue value is used to generate a threshold, the algorithm steps through each pixel of the hue image, and if a pixel is greater than or equal to the threshold, the pixel is set to one, else it is set to zero. If the minimum hue value is used to calculate the threshold, each pixel is compared to the threshold, and if it is less than or equal to the threshold, that pixel is set to one, else it is set to zero. Figure 2.5 contains the original images with the thresholds applied.

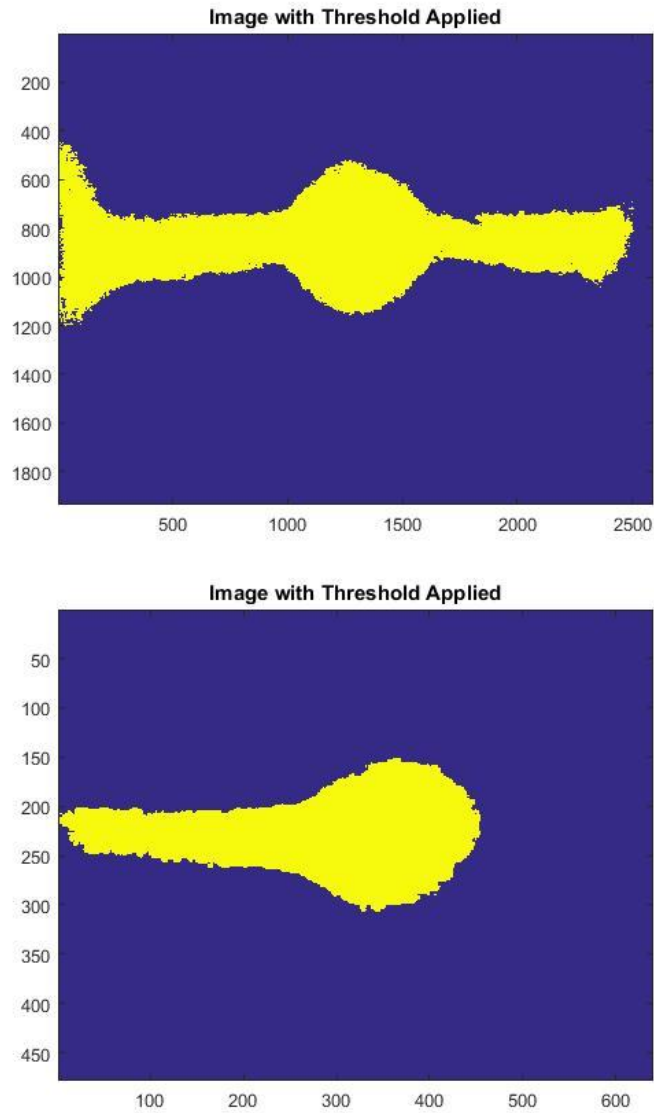


Figure 2.5. The original images with a threshold applied.

Image Morphology

After applying the threshold to the image, there may be some pixels in the region of interest that should have been set to one but were not. So, to ensure that all of these pixels are properly identified, image morphology is applied to the previously thresholded images. Image morphology [9] is used to remove or enhance features by using image dilation [9] and image erosion [9]. In this case, we wish to remove pixels that might not have been set to one, so image dilation is applied, followed by image erosion.

Image dilation is performed by taking a shape (in this case, a disk with a certain radius in pixels) and iterating through each pixel that is zero, using that pixel as the center of the shape, in the image with the threshold applied. If any pixel within the shape is one, the center pixel is set to one. Once the image dilation is completed, image erosion is used to get the region of interest back to its original size. This is done by iterating through each pixel that is one and using that pixel as the center of the shape. If there is any pixel within the shape that is zero, then the center pixel is set to zero. Once this is completed, the region of interest will contain no pixels that are zero, making it easier for the circle-finding algorithm in the next step (see the next section) to find the region of interest. Figure 2.6 contains images with both thresholds and image morphology applied.

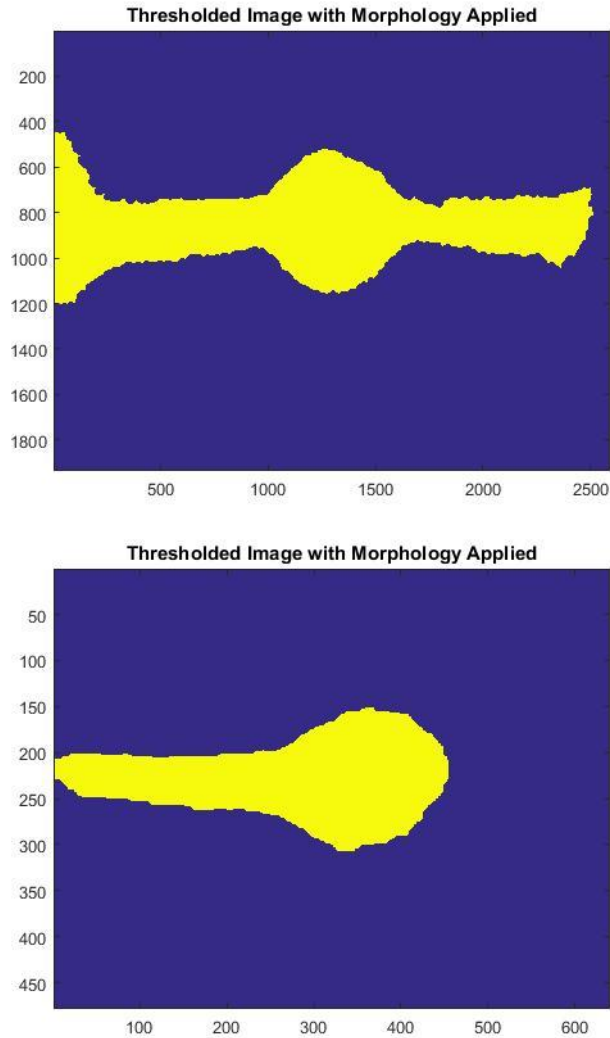


Figure 2.6. Images with both thresholds and morphology applied.

Finding the Circle in the Image

Since the region of interest in each image is circular, an algorithm was developed to enlarge a circle inside of the region of interest until it encounters a pixel that is zero. The algorithm then moves the center of the circle around and keeps enlarging it until the circle cannot be enlarged any more, thus resulting in a circle that maps well to the region of interest.

The algorithm starts with a circle of a certain radius (50 pixels was the original start radius) in the center of the image, which assumes that the region of interest overlaps with the image center. The algorithm then checks to see if every pixel inside the circle is one. If it is, the radius of the circle is increased and the process is repeated. After the radius of the circle cannot be enlarged without encountering a pixel with a value of zero, the center of the circle is moved

towards the edge in each quadrant of the image in the horizontal direction, and the enlarging process is repeated. Once the circle encounters a pixel that is zero, the algorithm moves the circle center back to the center of the image and moves the center of the circle one pixel towards the edge in a vertical direction, and then proceeds with the enlarging process by moving the circle center horizontally. The algorithm ends once it has iterated through each horizontal and vertical position. The result of the algorithm is the coordinates for the center of the circle and the radius for that circle, which are then used in the next step to compute additional image information (see the next section). Figure 2.7 contains the original images that have circles corresponding to the regions of interest overlaid on them.

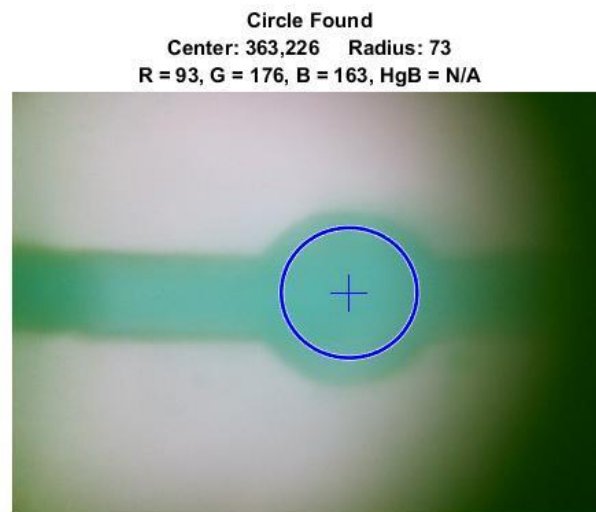
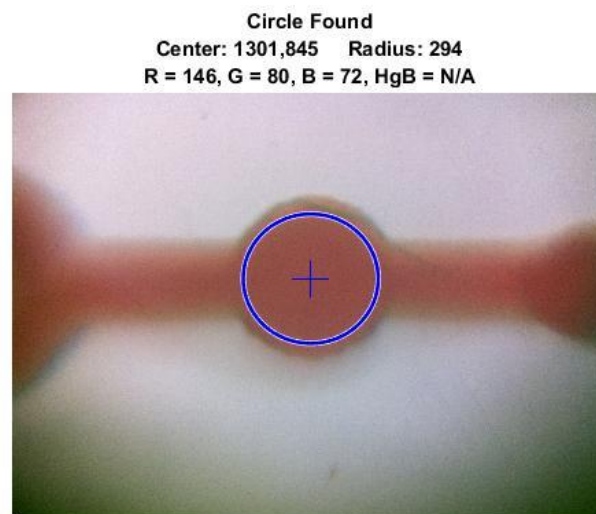


Figure 2.7. Images and their corresponding regions of interest.

Information Needed from the Region of Interest

The main purpose of this project is to calculate a hemoglobin (Hgb) level based on the predominant color in the region of interest. To do that, the red, green, and blue values that occur the most in the region of interest are required; those values will be referred to as the RGB peak values. Once those values are found, they are used with a model to calculate the Hgb level as represented by the image.

Finding Red, Green, and Blue Peaks

The red, green, and blue levels for the image pixels are 8-bit values ranging from 0-255. What is needed for the model is the value that occurs the most for each color. To determine this, a histogram is created for each color, and then each pixel in the region of interest is iterated through while the corresponding bins for those pixel values are incremented. Once the whole region has been searched, the bin with the maximum, or peak, value is then recorded for each color and used in the Hgb level calculation. Figure 2.8 and Figure 2.9 contain the histograms created from the regions of interest for the two example pictures.

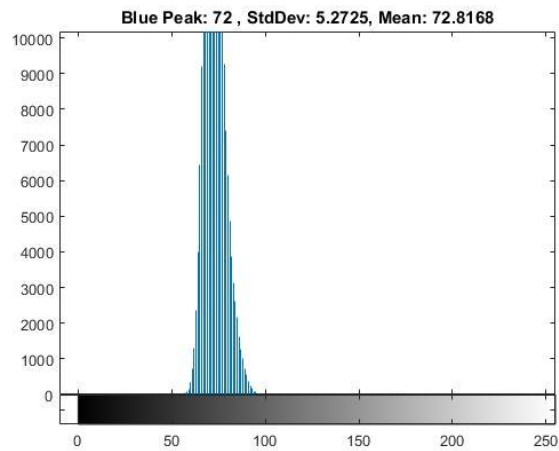
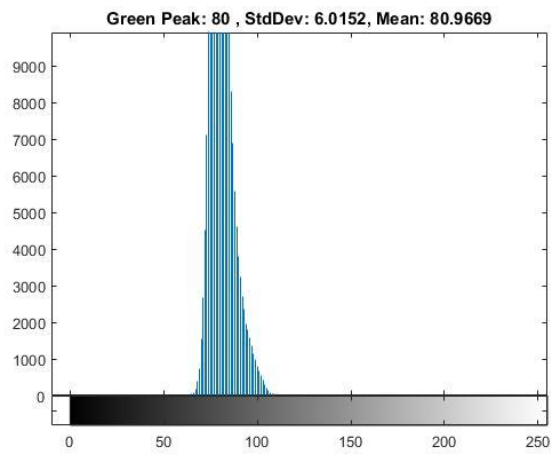
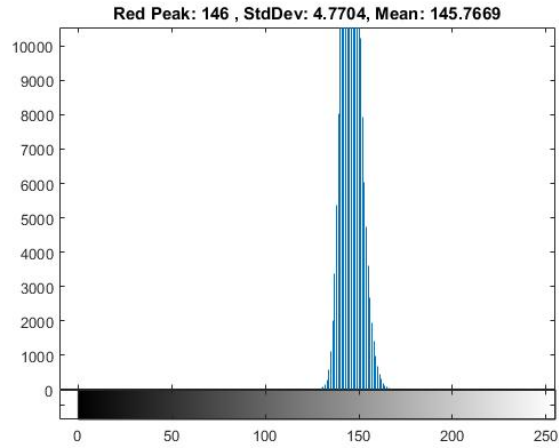


Figure 2.8. R, G, and B histograms for the first example image.

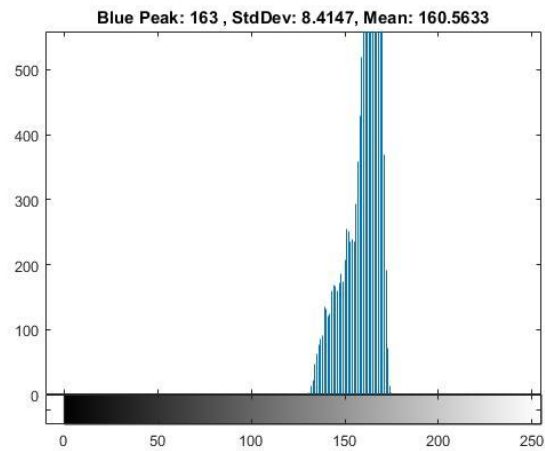
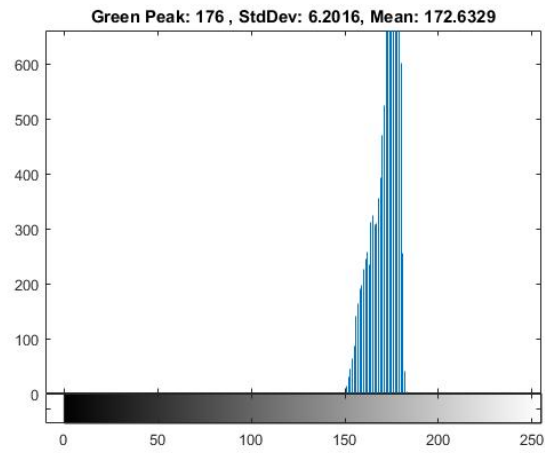
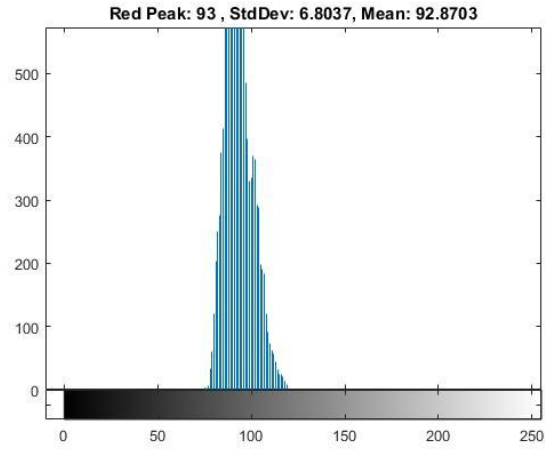


Figure 2.9. R, G, and B histograms for the second example image.

Calculating Hemoglobin Level

To calculate the hemoglobin level based off of the given RGB peak values, an initial model [10] was used that required converting those RGB values to the CIE Lab color space [6] and then using the resultant L , a , and b coefficients in equation 3. After unsuccessful color-matching attempts that will be discussed in the next section, a simplified model that used the R, G, and B peak values was created, as noted in equation 4.

$$Hgb\ Level = w + x * (L) + y * (a) + z * (b) \quad (3)$$

$$Hgb\ Level = w + x * R + y * G + z * B + i * R * G + j * R * B + k * G * B \quad (4)$$

In each of the equations 3 and 4, the coefficients w , x , y , z , i , j , and k are calculated using MATLAB's linear regression function. To calculate these coefficients using MATLAB, the linear regression function needs to be given sample information to compute the coefficients off. Two sets of base values were used. The first set was obtained from a group of images who RGB values aligned with a collection of corresponding Hgb levels. However, after unsuccessful attempts to find fitting coefficients that worked with both model equations, a new set of images was created using Adobe Illustrator, where a created scale was applied to those images to prove that both models could work and that the new model was an improvement over the old model.

Results and Issues

The next two sections discuss the results of the methods described earlier in this chapter. First, the text addresses the region-of-interest algorithm and the RGB peak values computed for the corresponding regions. This will utilize a set of experimental images, where RGB values will be compared to values obtained from earlier studies with those same images. Second, the expanded hemoglobin-level model will be compared to the old model in terms of performance. Issues with the sample images used to create the model will be discussed. Each section will also include currently known issues and potential problems.

Results and Issues: Region of Interest Algorithm and RGB Peak Values

To verify that the MATLAB program was finding each region of interest and was correctly identifying the RGB peak values within that region, each region of interest was plotted on top of its corresponding image within the test set. Figure 2.7 previously displayed two such

example images used throughout this chapter, where each region of interest was correctly determined by the MATLAB algorithm.

The main issue identified for the region-of-interest algorithm is that it does not adjust well to varying image resolutions. For example, two sets of images were originally received. Images one through four were received when the project was initiated, and images five through nine were received a few months later (see Appendix A). The images in the first set have higher resolutions, and the minimum radius assigned for the region-of-interest-finding algorithm was based off of that set of images. However, when the second set of images was received, even though the regions of interest looked like they were found, they happened to be just the regions of interest with minimum radii corresponding to the centers of these images. A properly assigned minimum radius is important because it helps to reduce the amount of time for the algorithm to run. Another reason for having a minimum radius is supported in the example image in Figure 2.10. The image corresponds to a slide that does not contain a sample, yet a region of interest was found, and information was gathered from it anyway. In this image, the algorithm should return a message stating that the Hgb level could not be calculated due to the lack of a valid sample, as indicated by the minimum radius. Image number five in Table 2.1 below is the image shown in Figure 2.10.



Figure 2.10. An example image indicating why a minimum radius is needed.

The final issue with the algorithm is that, even though the algorithm correctly finds a region of interest for a given image, the computation time is longer than desired. The time

required to find a region of interest in a higher-resolution image is about 20 seconds (using a Windows 10 desktop with an Intel quad-core processor running at 3.7 GHz and 8 MB of RAM); the lower resolution images take about 5 seconds. Smart phones offer generally less capable processors. One initial solution could be to simply reduce the resolution of each image. As indicated in Table 2.1, the results may not be affected by this process, so the algorithm still has the potential to be optimized.

RGB peak values determined from each region of interest via MATLAB were compared to values determined using a program called ImageJ [11]. ImageJ is an image viewing and analysis program supported by the National Institutes of Health; it was created to aid projects like the JCRC effort. The sequence for finding the RGB information using ImageJ starts with loading the image into the program. A circle is then drawn around the region-of-interest using an oval brush. Finally, <Control-h> on the keyboard brings up the histograms. Table 2.1 displays RGB peak values determined for nine test images, where results from the MATLAB algorithm are compared to ImageJ analysis results. In this table, images number 1 and 9 correspond to the images depicted in Figure 2.8 and elsewhere in this chapter.

Table 2.1. Comparison of ImageJ results to MATLAB results.

Image #	ImageJ			MATLAB		
	R	G	B	R	G	B
1	145	79	71	146	80	73
2	131	117	82	132	118	82
3	122	78	62	123	80	63
4	134	107	76	135	111	80
5	166	176	176	168	177	177
6	174	153	124	175	154	125
7	56	188	116	57	189	117
8	52	186	114	53	187	115
9	92	175	162	93	176	163

The table indicates a good correlation between the RGB values obtained using the two programs. Most of the values only differ by 1, primarily because MATLAB uses a basis of 1 for its arrays, and the value reported by the MATLAB program just obtains the array index with the

largest value. That is, in most cases the values from the two calculation approaches match up directly. The processing results for each image in Table 2.1 above can be found in Appendix A. A set of images was received later in the process for which the algorithm did not successfully find the regions of interest. These images can be found in Appendix B.

Results and Issues: Hemoglobin Level Model

As noted earlier, the initial and expanded Hgb models were the following:

$$Hgb\ Level = w + x * (L) + y * (a) + z * (b) \tag{3}$$

$$Hgb\ Level = w + x * R + y * G + z * B + i * R * G + j * R * B + k * G * B \tag{4}$$

To test the validity of the expanded model, a scale was created that was a gradient from blue to yellow to red: see Figure 2.11. This scale was assigned values from 0 to 10, and the scalar coefficients for the Hgb models were determined by getting an RGB value for each color at each even number. Once the fitting coefficients had been found, 16 more RGB values (and their corresponding level values) were chosen from random locations on the scale. The 16 images created for these RGB values, as well as the corresponding scale, can be found in Appendix C. A chart of the values computed using the two Hgb models in comparison to the anticipated levels is contained in Figure 2.12. Figure 2.13 contains the error plot for the two models in comparison to the expected values.

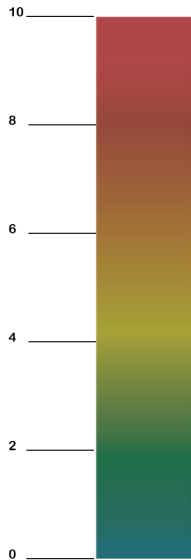


Figure 2.11. Color gradient scale created for testing models

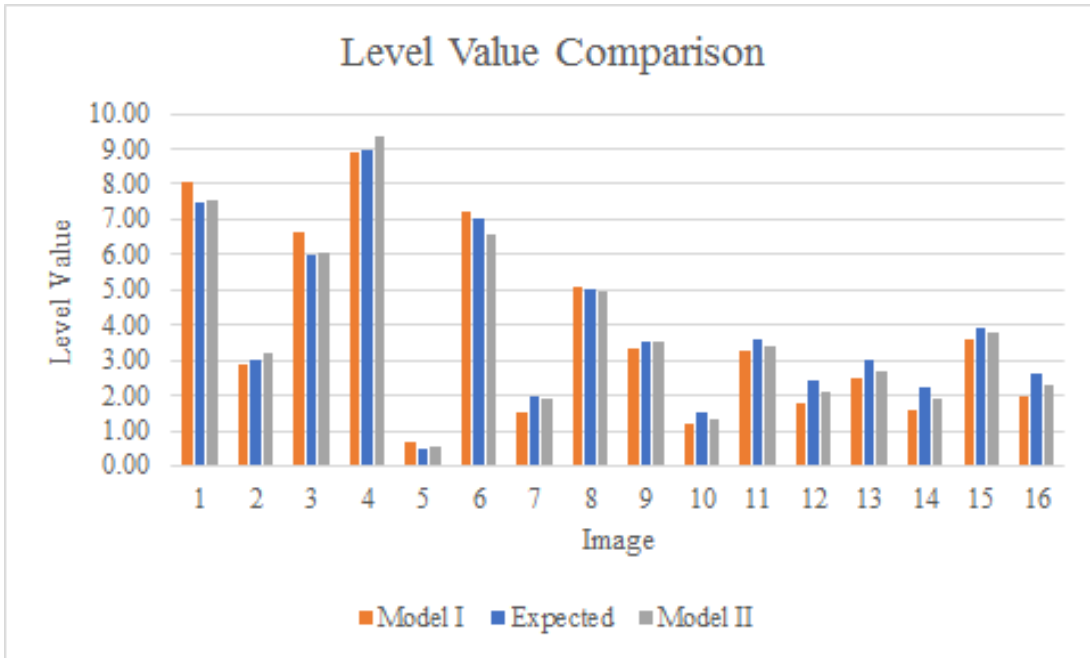


Figure 2.12. Bar plot – expected level values in comparison to the values calculated from the two Hgb models.



Figure 2.13. Each model’s absolute error versus the expected value for each image.

As indicated in Figure 2.13, the expanded model showed an improvement in the error relative to the original model. For the original model, the level error was 0.14 ± 0.11 for this set of images, whereas the error was 0.06 ± 0.05 for the expanded model. However, both models

worked well for computing level values for this test set of images. So, the test set fulfilled its purpose in demonstrating that either model would work and that the expanded model was an improvement. One can then conclude that the original sample images that were provided to determine the model coefficients for calculating the Hgb level were perhaps not consistent (e.g., in terms of acquisition parameters, environment, etc.) with later sample images. These later sample images can be found in Appendix D. Figure 2.14 depicts a comparison between the Hgb levels for these sample images and the computed Hgb levels after the model was fitted to those same images. Figure 2.15 displays the error between the actual and computed values for those images.

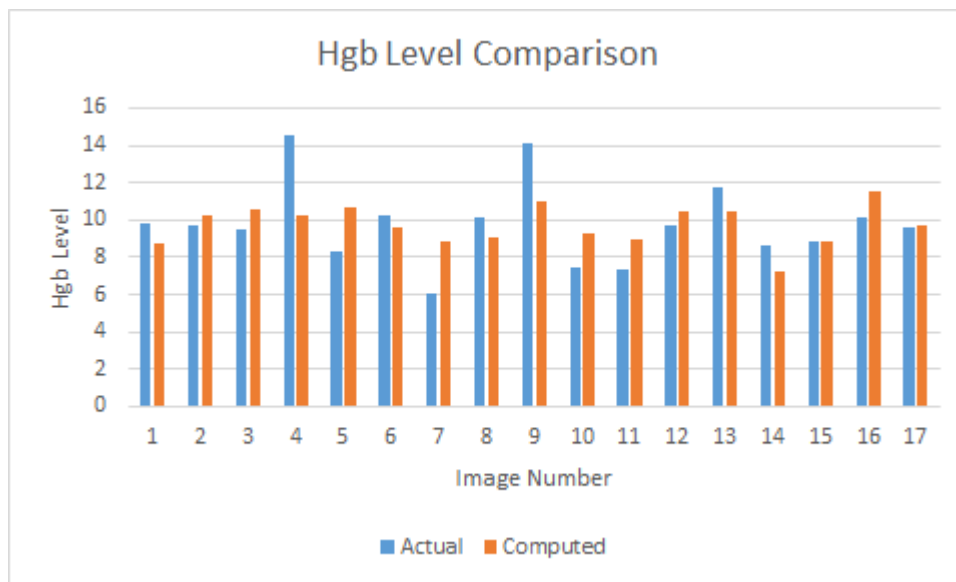


Figure 2.14. Comparison of actual Hgb levels compared to computed Hgb levels for the sample images provided.



Figure 2.15. Error between the computed Hgb levels and the actual Hgb levels for the sample images.

Chapter 3 - Local Data Server

The main objective of the Heartspring project was to display biomedical information from the bed monitoring system on a smartphone. To display these data, the method for getting the information to the phone had to be determined. This chapter discusses the process of choosing the data transmission method, background information related to the process that was chosen, an overall view of the system and how it works, and issues associated with the resultant system design.

Choosing the Data Transmission Method

When considering different protocols to transmit the biomedical information, one requirement was that the user should be able to receive that information from anywhere in the facility. This means that methods that require close proximity, such as Bluetooth, were not considered. Two primary approaches remained for consideration: a cloud-based server and a local server running on the host network.

The first option considered was to upload the biomedical information from the bed system to a cloud-based server so that those data could then be downloaded by any device from any location. Microsoft's SQL [12] server and Google's App Engine [13] were the two cloud-based candidates. It was clear that each type of cloud-based server would take an extended amount of time to implement due to the difficulty of setting it up and developing code to work with it. Security for the cloud server would also cause issues because a subject's medical information would be more difficult to protect. Cloud-server providers also charge fees to users based on the number of uploads and downloads as well as the size of those uploads and downloads, which would end up being expensive in the long run. Another issue with this approach is that, if the server goes down for an unexpected time period, communication between the bed system and the phones will not commence until the cloud server is running again.

The second option that was considered, and used, was to set up a local data server on the host network to which the beds would be connected. This local data server uses a network socket [14], which is a node in a network used to send and receive information. Setting up a local server on a network has several benefits, one of which is security. Since the server would only run on a local network, the only devices that could gain access to the broadcasted information would be devices connected to that local network. Further advantages to this approach are that it does not

cost anything to use, and it is simple to implement. It also provides more flexibility, since it can be developed from the ground up to fit our specific need.

Background on Network Sockets and TCP

Network sockets are created so that any device connected to that socket can send and receive data from other devices connected to that same socket. “Client” and “server” [15] are two terms that are often used when describing network sockets. A server is like the brain of the system – it oversees the monitoring of the socket for new device connections and the relaying of information sent from one device to the other devices connected to the socket. Clients are the devices connected to the socket that send and receive data – essentially any devices connected to the socket that are not the server.

Two main standard Internet protocols are used with network sockets: the transmission control protocol (TCP) [16] and the user datagram protocol (UDP) [17]. TCP was chosen for this project because of its ability to be easily understood. Communication using network sockets and TCP works in a manner similar to I2C [18], where a master device (server) controls the sending and receiving of information to and from other connected devices (clients). To send information to a client from the server, the TCP packet contains an IP address for that client.

To test whether the local data solution would work, two test C# programs were developed using Visual Studio [19]: one for the client and one for the server. These test programs comprised a basic chat system that would allow multiple computers on the network to chat with each other. It was set up so that a client could send a message, and then that message would be received by the server and sent to the other connected clients. Once the basic program was working, an Android phone application was created to allow the phone to connect to the chat system and send its own messages. This simple chat system verified the notion that a local data server would work well for sending/receiving biomedical information to/from the bed system

Overall System

A system was proposed that would use a local data server to facilitate the sharing of data gathered from the bed. As depicted in Figure 3.1, the computer would receive these biomedical data from the bed system and then process and store them. The same program would also run the server that transmits this information to a client device.

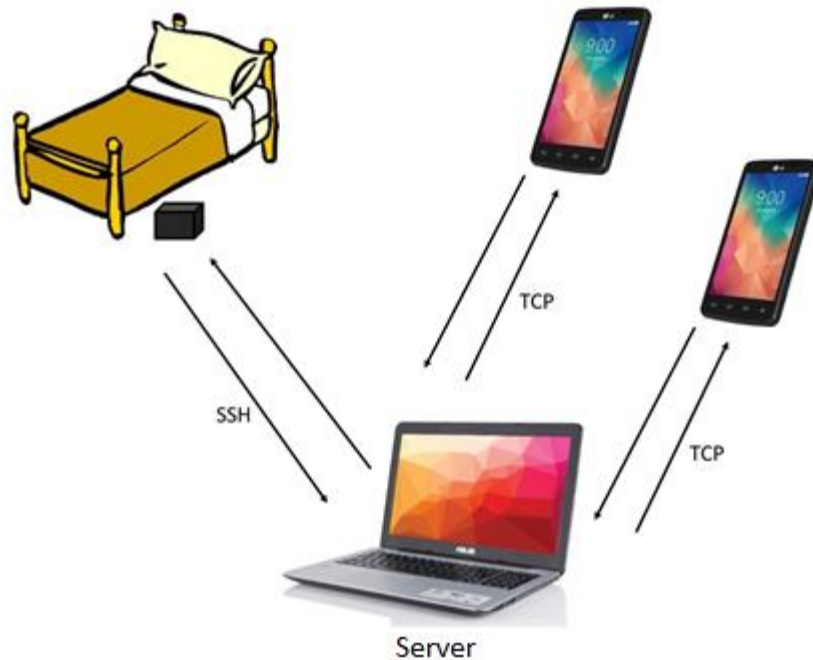


Figure 3.1. High-level depiction of the proposed system.

To send bed-based information to a client, the server program packages the information into the string format laid out in Figure 3.2. The string format is like a global position system (GPS) string in ASCII format. The indicator character is a dollar sign, followed by a subject ID number, the subject’s sleep stage, and the subject’s heart rate. The format is then repeated for each subject for whom the server holds information. Each piece of information is separated by a comma, and the string is terminated by a carriage-return character.

```


 $\{\text{SubjectId}\#,\{\text{SleepStage}\},\{\text{HeartRate}\},\dots\{\text{Repeat for number of subjects}\}\backslash\text{r}$ 


```

Example:

```


 $\$123,3,67,203,5,94\backslash\text{r}$ 


```

Figure 3.2. Data packet structure for server-to-client transactions.

A software program was developed to simulate the program running on the laptop that receives physiological information from the bed, calculates parameters to be displayed, and then sends that information to the connected phones. The program allows the user to create several different subjects and their related information, such as sleep stage and heart rate. Every second, the program uses the sleep stage and heart rate values as inputs, creating a triangle wave with

those inputs being used as the average of the triangle wave. Once a new value represented in the triangle waves has been calculated, that information is sent to any connected clients by the server. The program is also set up to send an image over the local server. Figure 3.3 is an image of the software interface created for this program.

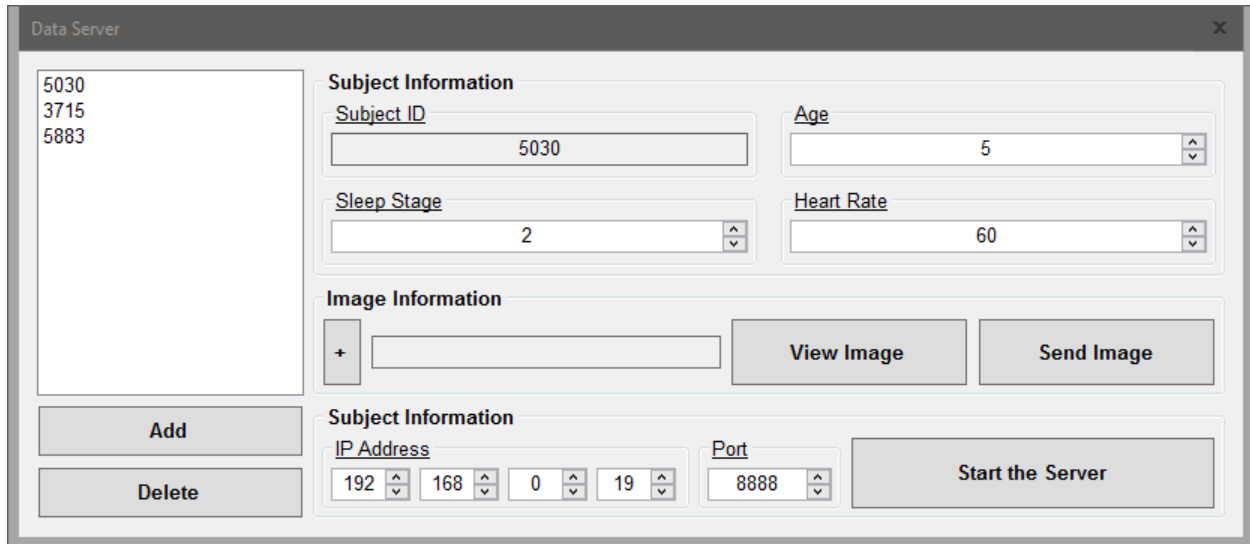


Figure 3.3. Software interface for the test program to simulate the bed system.

The simulation program allows us to develop the phone application without having the bed-to-computer portion of the system in place. If the phone application is completed using the simulation program, then the computer software responsible for the bed system communication and data processing can then use the code from the simulation program to inform the creation of the server, particularly with regard to sending processed data to the connected phones. This should make the overall system easier to debug since the communication to the phone should already be debugged.

Results and Issues

The successful operation of both the chat system and the simulation program for the overall bed monitoring system indicated that the local data server approach was a viable option and would serve the needs of the project. Sending images using the local data server was tested as well and worked to an extent for both the chat system and the simulation program. There were a few issues that were encountered during testing: efficient sending/receiving of images, the server shutting down before clients are disconnected, a lack of data validation, and the battery usage of the phone application.

The first major issue, which continues, is the amount of time required to send a full color image across the data server, coupled with the inability to receive the image correctly on the Android device. Currently, the RGB portions of the image are sent separately and then recombined on the receiving device. The C# program running on a laptop is able to successfully receive, recombine, and display the RGB image. However, when that same code is ported to the Android device, portions of the image are not being correctly received. This could be due, e.g., to a portion of the image that interrupts computation that is still underway for a previously received portion. Low resolution images take significantly less time to transmit than high resolution images, and there is a higher probability that these images will be correctly received.

When testing the simulation program, if the simulation program was closed while the server was running, any client connected to the server would become unresponsive, and the application would then throw an error and close. A similar issue was observed when the client was disconnected from the server and then reconnected, causing the same client to be associated twice with the server. This was solved by storing a record of each client in a dictionary collection, with the key being the IP address of the client. If a new client connected with the same IP address, the old client would then be replaced with the new one. Another observation was that the phone application would only be able to find the local data server if the computer running the server program was connected to the network through an Ethernet port. This was noticed after several unsuccessful tests to connect to the server running on a wireless laptop.

Another potential issue is the lack of data validation when parsing information received from the server. If a transmission error occurs, such as a byte not being received or two bytes being received in the wrong order, problems will occur in terms of system operation and/or data integrity.

Finally, the process of connecting to the server and waiting to receive data to update the display uses a lot of battery power. During testing, battery life dropped by about 20% over an hour, which is not ideal for a program that may need to run throughout the course of a night. The excessive battery usage is due, in part, to the need to always be connected to the server and to update the display every second.

Chapter 4 - Software

As mentioned in Chapter 1, this work addressed two projects, each of which required its own phone application. This chapter will address the architecture for each project, the development process, the interactive development environments (IDEs) and frameworks/development kits employed, and the results and issues for both software projects.

Since software development results in versions of code that build on one another, Github [20] and Git [21] were used as version control tools used for this effort. This choice was driven by tool familiarity from previous classes and internships. Git is the actual version control tool. It allows a user to upload/download code from its online repository. It also allows for the use of different code branches, so that if a user wants to try something new, they can keep a copy of the current working version of the code. Github is just a web application for Git – a way for the user to visualize code operation and view the code they have pushed to the repository. It also allows the user to see changes that were made from previous ‘commits’ (code uploads). Github supports both public and private online code repositories. It also provides students with nine free private repositories, where a private repository is hidden from other viewers. This mechanism was used for the two software projects addressed here.

Software Architecture

The architectures for the two projects are similar in that each one has user interface classes and classes that deal with the collection of data that are to be displayed. The class diagrams in this section were created using NClass [22], a free class diagram designer specifically designed for C# and Java. In these diagrams, an “activity” is a class that represents the primary source code for a user interface displayed on the phone. In the class diagrams, a dotted line represents a dependency between classes, whereas a solid line represents an association between classes. In a dependency, one class relies on another class without creating an instance of that class in its code. In an association, one class creates an instance of another class in its code.

JCRC Class Diagram

Figure 4.1 contains the class diagram (created with NClass) for the JCRC project. This class diagram identifies two different activities that will appear on the phone. The first and

starting activity is the MainActivity class. This class is the user interface that appears when the phone application is first opened. The main screen offers two buttons stacked near the top and an image view to display the image taken by a camera. The upper button opens the phone's built-in camera application to capture an image. The original plan was to display a live feed from the camera (in the image view) and then write the code to focus the camera and capture an image. However, because of the potentially significant amount of development time involved and the fact that the Android system already has a built-in camera application, that application became a better option. The other button is used to open a new view that contains image analysis results and a depiction of the resulting image.

Once the analysis is both complete and successful, the MainActivity class then allows the ResultViewActivity class to be created, which displays the analysis information for the image. The class is only accessed after a button click in the MainActivity class, at which point the analysis information is passed from the MainActivity class to the ResultViewActivity class using the Intent class. The Intent class is built into the Android framework and is used to pass data from one activity to another activity. To add data to the Intent class, you must give it a string associated with those data so that the next activity can receive the data via the same string. This is where the ApplicationConstants class is useful, as it supports the creation of those strings. Each of the activities is dependent on this class.

The AnalysisBundle class, which is only used by the MainActivity class, is the controlling class that does the image analysis. The class receives an image from the MainActivity class. When that occurs, it replaces the old image with a new one and proceeds to attempt to run the image analysis sequence described in Chapter 3. This class depends on three static classes which help to analyze the image: Rgb2YIQ, ImageMorph, and ImageCircleFinder. The Rgb2YIQ class takes RGB color space values and converts them to the YIQ color space. The ImageMorph class uses image morphology to fill in holes in an image, and the ImageCircleFinder class finds the region of interest in the image.

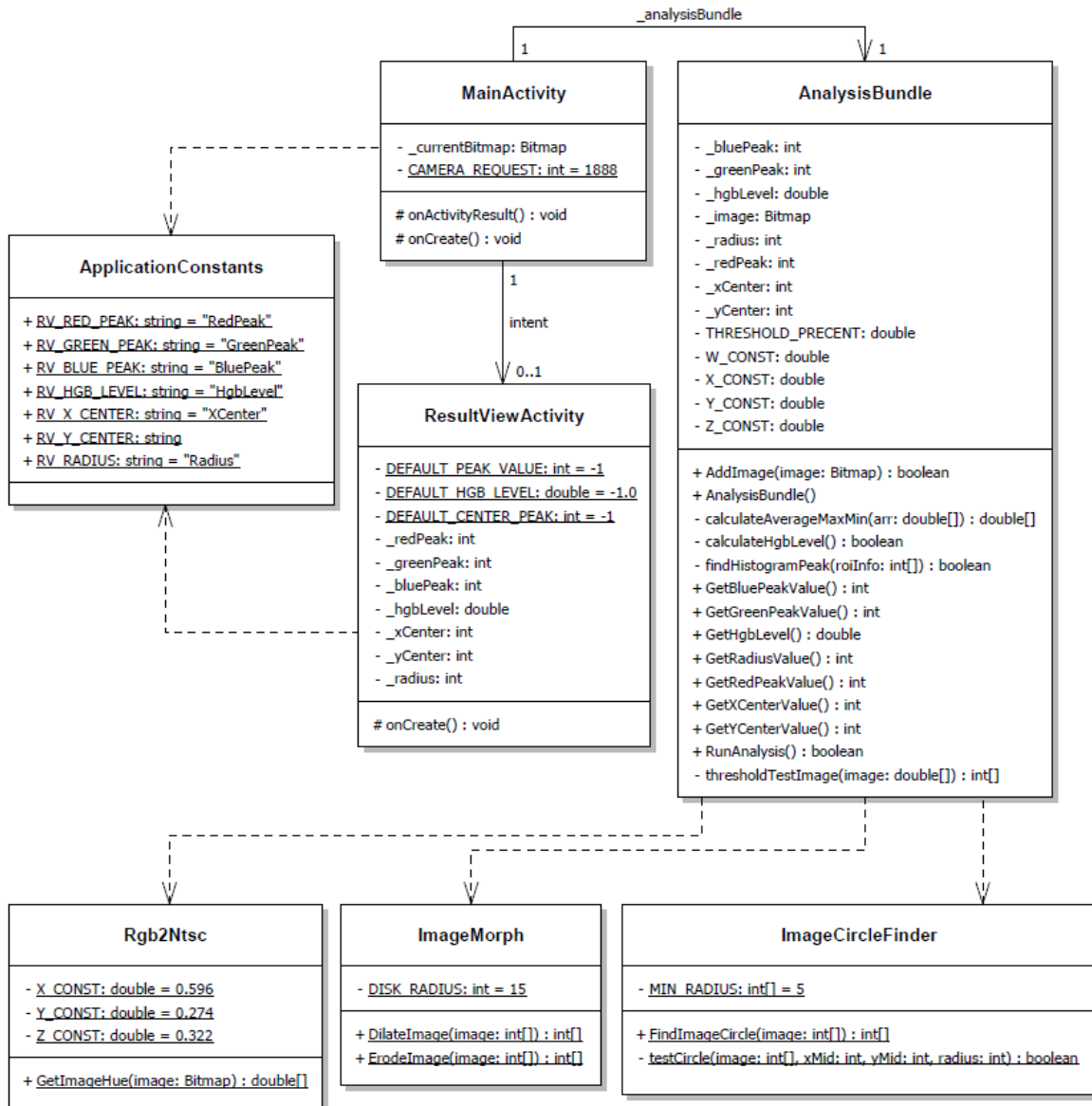


Figure 4.1. Class diagram for the JCRC project.

Heartspring Class Diagram

Figure 4.2 contains the class diagram for the Heartspring project. In the class diagram, three activities (MainActivity, ViewImageActivity, and SettingsActivity) are used to display desired information while minimizing screen congestion. There are also three classes (Client, Subject, and SubjectTracker) that assist these activities and two static classes (ApplicationHelper and ApplicationConstants) that store constants and program settings. There is also one interface (SubjectListener) that allows the class to let the activity know information on the screen can be updated.

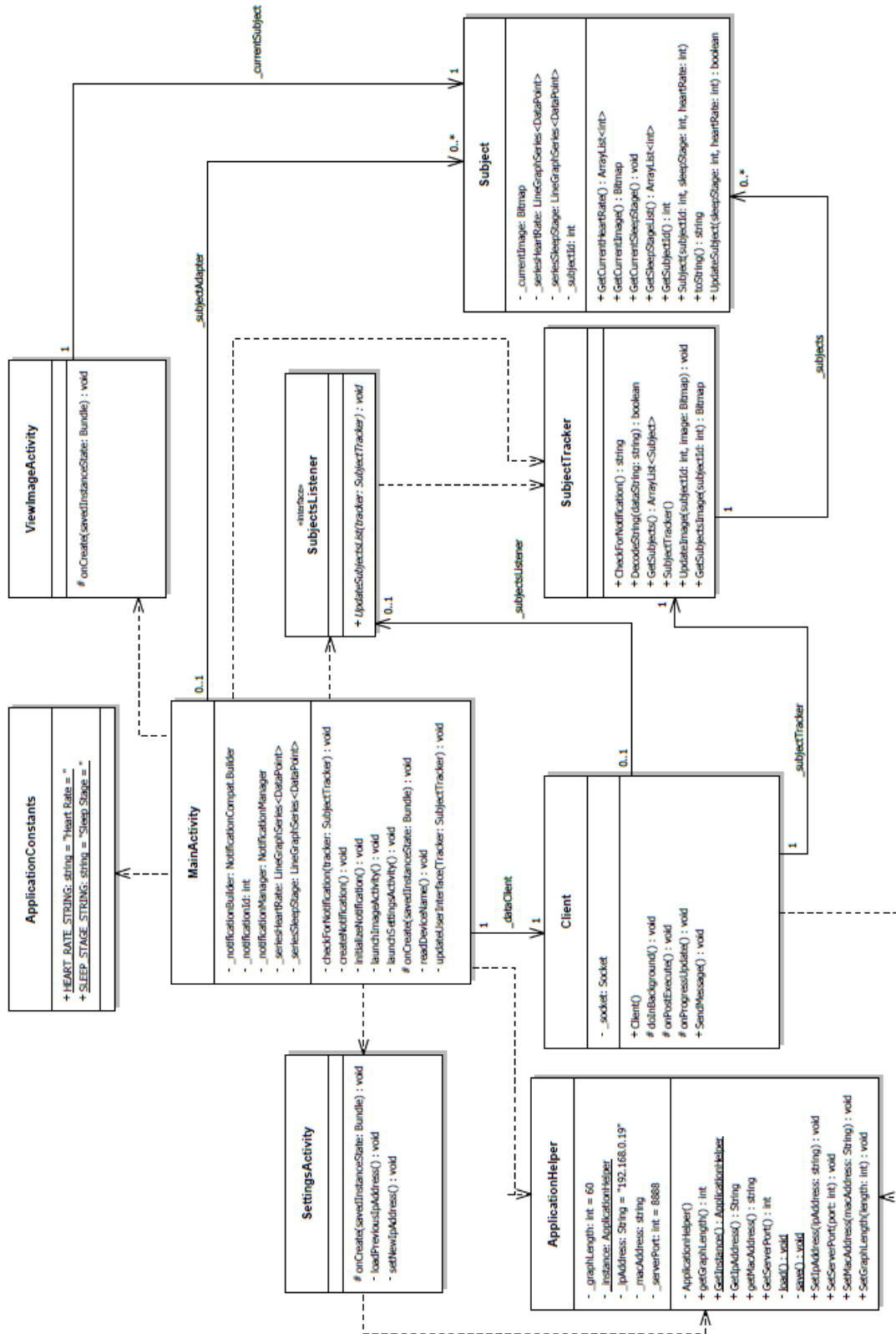


Figure 4.2. Class diagram for the Heartspring project.

The MainActivity class corresponds to the starting user interface when the application is opened. It includes a drop-down list to select the subject whose information is desired. A tab control allows the user to switch between the information categories. Two other buttons exist on the main screen. The first button allows the user to go to a different activity to see the current thermal image of the subject. The second button allows the user to change application settings such as the IP address of the local data server. To use a drop-down list in the Android environment, an activity must contain a collection of data that comprise the list. This collection is called an adapter and is a member of the Subject class. The Subject class is responsible for storing information and images pertaining to specific subjects; it gets updated every time new information is received from the local data server.

This main activity also contains a reference to the Client class. The Client class runs in the background and oversees creating, starting, monitoring, and stopping a connection to the local data server. When new data are received from the data server, these data are combined into a string and then sent to the SubjectTracker class. The SubjectTracker class contains a collection of the Subject class that is updated based on the information decoded from the data string received from the client. The SubjectTracker class is also in charge of notifying MainActivity that new information has been received from the server and that it is time to update the user interface. The notification process is done by using the SubjectsListener interface. A listener interface is needed because the decoding of the data string in the SubjectTracker class is done within a background thread, which prevents us from updating the user interface.

A ViewImageActivity instance is created when the “View Infrared Image” button is clicked in the main activity. Much like the code for the JCRC project, an Intent class is used to pass the current Subject class to the new activity so that we can get the current subject’s latest infrared image to display. Also like the JCRC project, an ApplicationConstants class is used to manage the string that stores subject information.

Finally, the SettingsActivity class is used to display the current program settings and allow the user to change those settings. One setting, for example, is the IP address of the server that broadcasts information. Another future setting might give the user the ability to change how and when they are alerted, e.g., based on a subject’s sleep stage. An ApplicationSettings class allows the program to save these settings so that the user does not have to reenter the settings

every time the application is started. This class is a static class that can be accessed by any of the classes, and it is loaded only when it is needed and is saved every time a setting is changed.

Software Development Approach

One goal, when starting these projects, was to find a way to develop in my strongest language, C#, in order to save time and deliver a working program prior to completion of my degree. Phone application development started with a program called Xamarin [23], a cross-platform development tool created by Microsoft. The intent was to develop the project code in C# and then convert that code to Java due to the similarities between the two languages.

At the beginning of these projects, Xamarin had just been released. Xamarin allows a developer to develop most of their code – code that will run under multiple operating systems (e.g., Android, Apple OS, and Windows) – in the C# language. The remaining user interface code must then be developed in the operating system’s native language. However, after a couple of months, it became clear that this development process would be slow. Even though some of the program features could be developed in C#, many of the useful C# libraries (e.g., Bitmap) were unavailable, meaning those features would have to be created from scratch. Additionally, for these phone applications, much of the code could not be written in C#; classes specific to the native language of the operating system would have to be created. This would have resulted in the need to learn two new languages: Java and Objective-C. A study [23] also found that there were several bugs in the Xamarin framework which would have caused even more development delays.

At that point, the drawbacks of the Xamarin tool outweighed its benefits, so a different development path was chosen. I decided to develop most of the project code in C# within the guise of a set of windows programs and then convert that code to Java. As noted above, the differences between Java and C# libraries [24] are minor. This approach made it easier to create test programs in C# and ensure that the Android code worked correctly. The approach also made it easier to create a simulator program for the Heartspring project as explained in Chapter 3.

Interactive Development Environments and Frameworks/Development Kits

During the process of developing both software projects for Android smartphones (which uses the Java programming language), duplicate projects were created for Windows programs (which use the C# programming language) due to my experience with the C# language and the

similarities between C# and Java. Two different interactive development environments and three frameworks/development kits were used during this process. To create the Windows programs in C#, Visual Studio Community 2015 and the .NET Framework 4.5 were used. To create the Android program, Android Studio, the Java Development Kit (JDK), and the Android Software Development Kit (SDK) were used. The next two sections discuss these IDEs and their corresponding frameworks/development kits.

Visual Studio and .NET Framework

Visual Studio Community 2015 [19], developed and maintained by Microsoft, is mainly used for C# code development. The community edition of Visual Studio is free for students. Visual Studio comes with a built-in debugger and an option to allow the user to debug the program in “Release” or “Debug” mode. Each program was created within the .NET 4.5 framework, which uses the latest version of Microsoft’s collection of libraries for the C# language.

The JCRC project was created using only standard C# libraries. When creating a new C# project in Visual Studio, Microsoft automatically also includes some non-standard libraries. Two such libraries that were useful for the Heartspring project were the System.Net and System.Net.Sockets libraries. These libraries contain many classes that assist developers and decrease development time for projects that deal with the use and creation of Internet protocols. The following classes were used in the Heartspring project: TcpListener, IPAddress, TcpClient, and NetworkStream. The TcpListener class is used to connect to the local data server over an Internet connection and to listen for incoming data packets. This class uses the IPAddress class – a class that stores information about the local data server to which we wish to connect. The TcpListener class is assisted by the TcpClient class, which creates a client for the local data server. A client in the networking application domain is a piece of hardware or software that is used to access a service on a server. The TcpClient is used to get the NetworkStream class, which reads information from the server byte by byte.

Android Studio and Development Kits

Android Studio [25] is developed and maintained by Jet Brains [26], a company whose first integrated development environment (IDE) supported the Java language. Jet Brains now has more than 20 IDEs for different languages. Android Studio, a version of Jet Brains’ Java IDE,

makes it easy to create user interfaces with drag-and-drop features, and it also allows a developer to debug an application by either installing it on a phone connected to the developer's computer or by using an Android emulator that is just a simulated phone program on the computer. During development of these software projects, testing of the phone applications was conducted on two phones (a Samsung Galaxy S6 and S7) instead of using the Android emulator.

When an Android project is created, the developer specifies the lowest version of application programming interface (API) [27] that they want the application to support. An API level of 21 (also known as Lollipop) was chosen for both projects. A basic Android project is created to use both the JDK and the Android SDK. Much like C# and the .NET framework, an Android project is initialized to use a number of standard libraries from both development kits. Two important classes used for the JCRC and Heartspring efforts are not a part of the standard libraries included in Android projects; these are the Socket and InputStream classes. Socket is a combination of C#'s TcpListener and TcpClient class, and InputStream is like C#'s NetworkStream class.

Results and Issues

Most of the results for each application relate to their individual functionality as discussed in Chapters 2 and 3. Each of the phone applications are currently working and operate as specified. Two screenshots for the JCRC phone application are depicted in Figure 4.3, and images of the Heartspring phone application are included in Figure 4.4 and Figure 4.5. The first screenshot related to the JCRC phone application (Figure 4.3, left) depicts an orange bottle cap on a white sheet of paper; the region-of-interest image results are displayed in the second screenshot (Figure 4.3, right). That second screenshot, which lists information related to the region of interest and its peak RGB values, also displays a square sub-image whose color maps to the peak RGB values. The circular region of interest was correctly identified in this example, because the algorithm was able to identify the peak RGB values from the region of interest and produce a color that is a visibly similar to the color of the orange cap. The current model built into the phone app is based upon the test images created that can be found in Appendix C, and the Hgb level calculated for the orange cap was 6.763, which correlates well to the scale found in Appendix C.

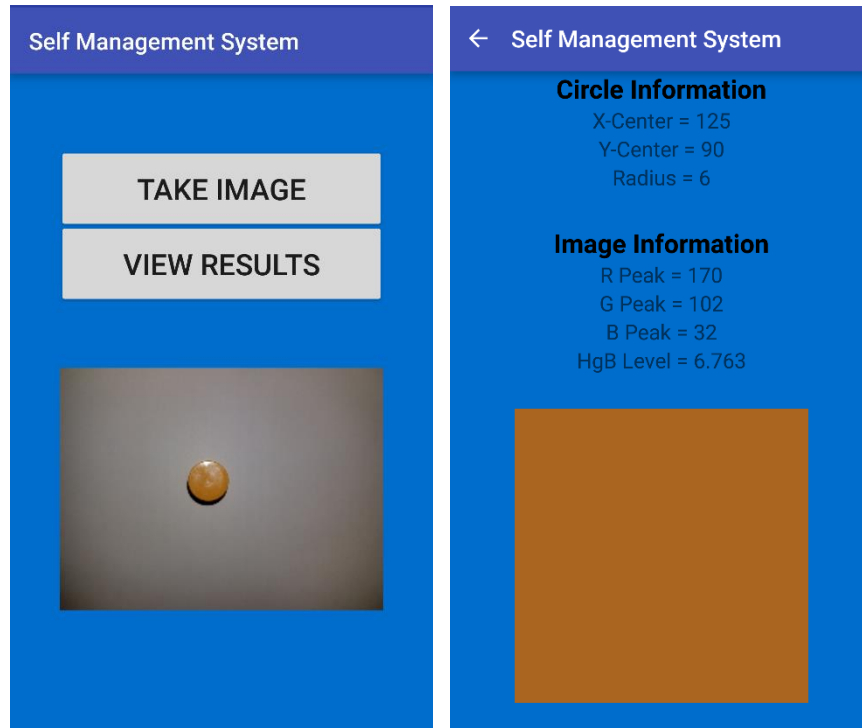


Figure 4.3. Screenshots of the two layouts for the JCRC phone application.

For the Heartspring phone application, the images were gathered by using the test program created to act as the local data server transmitting generated data. The first image in Figure 4.4 displays a sleep stage trend based on these simulated data, whereas the second image displays a heart rate trend. Each tab has a label that displays the current information as well as a graph that illustrates the trend over a certain time period. The graphing software is a third party tool called GraphView [28]. Both images include a third tab labeled “Other.” This tab is not currently used; it was added as a placeholder for other supplemental data. Both graphs update when a new reading is received and/or when the room number is changed. The first image in Figure 4.5 would normally show the infrared image from the thermal camera, but receiving images over the server is not currently working. The second image shows the settings layout screen, which currently only allows the user to change the IP address of the server. This screen will accommodate other settings in the future that allow the user to customize the program in a sensible way.

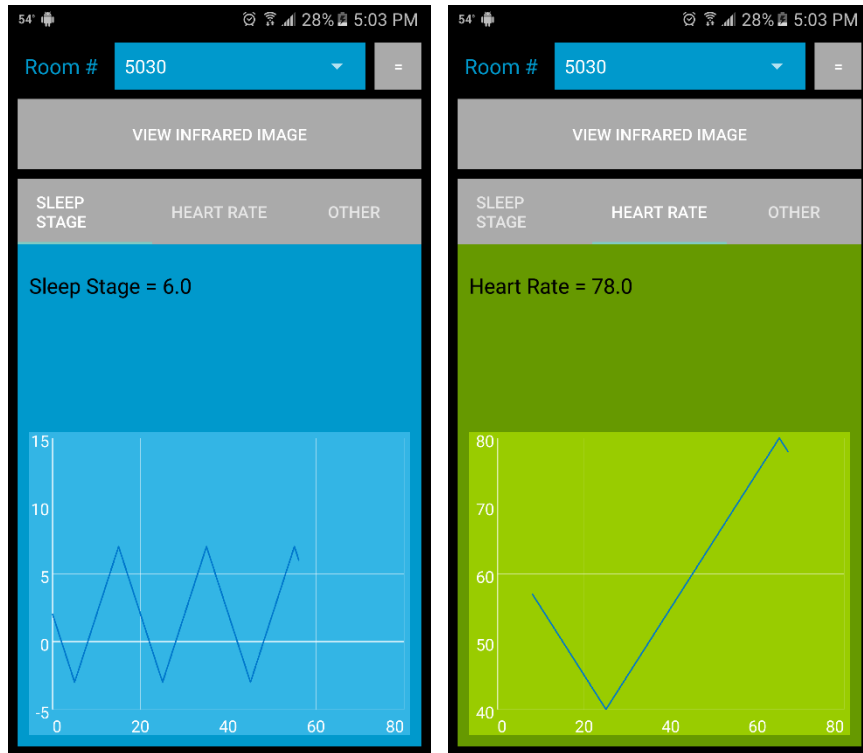


Figure 4.4. Screenshots of the Heartspring application that display “real time” plots for simulated sleep stage and heart rate data.

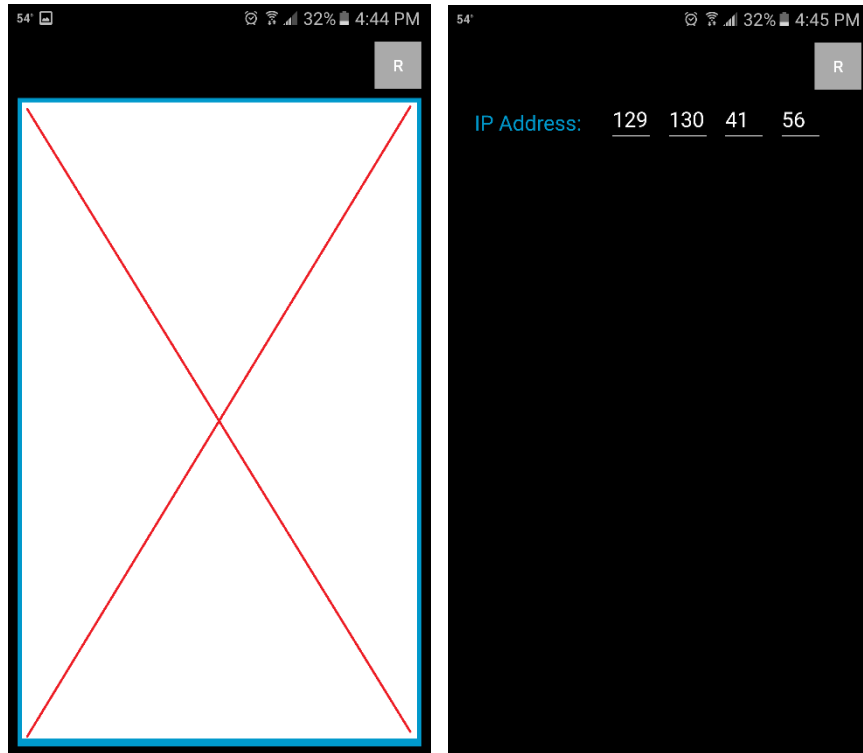


Figure 4.5. Screenshots of the Heartspring application's IR image viewer and settings layout.

Issues

The main issue for both phone applications is error handling. Due to the speed at which the applications were developed and the focus on producing an application that works for each project, error checking and program robustness were not well addressed. For example, the JCRC phone application does not have a means to determine whether the image is valid. For the Heartspring phone application, due to the constant graph updates and the need to listen to the local data server for information, the phone battery is quickly drained by the application. Another issue with the Heartspring phone application is that, when the user clicks on the settings button, the application is supposed to stop listening to the data server, but this causes an error and the application crashes. Once that issue is fixed, another issue dealing with the inability to save the settings will also need to be fixed.

Chapter 5 - Conclusions and Future Work

Both phone applications are developed to a point where they provide basic functionality. The next two sections address the current state of each phone application and the future work that needs to be done, including suggested improvements. In both cases, the user interface did not receive much attention.

JCRC App Conclusion and Future Work

The JCRC phone application can take an image and determine the circular region of interest. It then displays the resulting information on another screen for the user to view. The region-of-interest-finding algorithm works best when the region of interest is in the midst of a white background. From the results in Chapter 2, the RGB peak values calculated with the algorithm appear accurate. Further, the two models can both calculate Hgb level provided the base data to determine their coefficients are reliable. Future work for this project includes the acquisition of reliable base data for the Hgb-level models, improvements to the region of interest finding algorithm, implementation of a calibration procedure to account for different types of phones/cameras, and a means to upload the Hgb information to a cloud server.

As discussed in Chapter 2, since the experimental image data demonstrated irregularities, the models to calculate Hgb levels were not fully tested. Both models do appear to provide accurate results provided the sample images are consistent with the model color scale. Since images will be taken with a given phone using a device mounted on the back, it would be best to acquire sample images for testing the model with the full system so that the model can be based off of images that the system will likely see. This could also help to ensure that there are no other outside factors affecting the Hgb assessments provided by the models.

The algorithm to find the region of interest has a few issues. One issue relates to decreases in resolution that prohibit successful identification of the region of interest. The image resolution and the minimum radius also affect the run time of the algorithm. To address this, a linear adjustment could likely be applied to the minimum radius. The run time of the algorithm does need general optimization as well, specifically the portion of the algorithm where a circle is expanded to its maximum radius. At present, the algorithm searches the entire image, but once the circle is optimally expanded, the algorithm should stop.

As of 2014, over 19,000 different types of Android devices existed [29], all of which could have different forms and cameras. This means that a model optimized for use with images collected with one phone might not work on another phone. To fix this potential issue, a calibration procedure could be designed that allows the user to insert a set of slides into the device, where a picture is taken of each one. Each slide will be a different color and represent a different Hgb level. Once all of these slides have been inserted into the device and their respective images have been captured, the app can adjust the constants of the model so that the calculated Hgb levels match those of the slides for that specific camera and lensing setup.

The final suggestion for future work on the JCRC project is to determine a way to upload the image and its associated information to a cloud-based server so that clinicians can access these data. This information would need to be encrypted so that patient-identifiable data are not accessible to individuals who do not have permission to access those data.

Heartspring App Conclusion and Future Work

The Heartspring phone application is currently working with simulated values as described in Chapter 3. The phone application notifies the user when a subject's sleep stage is above a certain threshold. The application also allows the user to select the subject of interest. Once a subject is chosen, the graphs and labels for sleep quality and heart rate are updated. The rest of this section includes suggestions for future work and related implementation notes. These ideas for future work generally relate to sending/receiving infrared images, adding the ability to save settings for the phone application, changing the packet protocol, and improving the battery life for host phone.

The first issue deals with sending images from the server to the phone. At present, the server and phone applications are set up to send and receive color images, respectively, by handling each RGB portion of the image separately. Since the infrared image only needs to be a grayscale image in most contexts, this process could be optimized to send only a grayscale image. That change, coupled with a decrease in resolution for these infrared images, should make a significant difference. The simulation program is set up to select a photo and send it when a button is clicked, so the primary update would involve the protocol to send the image.

The ability to save settings also does not work. One attempted method works in C# but not in Java. After some research, it appears that Java has a built-in class called

SharedPreferences that can be used as a framework to save primitive data sets. The ability to save settings is critical to the phone application because it will avoid the need to reenter information every time the application is used.

Another suggestion for future work is to create a byte array to hold subject information instead of a string that is then converted to a byte array. Table 5.1 is a suggested packet definition. In this scheme, a value of 255 would be reserved for signaling the start of a subject packet; all of the other values would be from 0-254. If values greater than 254 are required, bytes can be added so that there is a most significant byte and a least significant byte. A way to determine whether the packet data received are intact (i.e., complete and in order) would also be helpful. One way to approach this would be a check sum added to the end of each packet. This would help ensure that the parsed data are valid.

Table 5.1. Suggested packet definition for the system.

Byte #	0	1	2	3	Repeat 0-3 for each subject
Data Type	0xFF	Subject ID(0-254)	Sleep Stage(0-254)	Heart Rate (0-254)	
0xFF(255) Reserved for signaling the start of a packet					

One final item relates to means to improve the battery usage of the phone application. The phone application has a significant impact on battery life. E.g., when the application runs for an hour, the battery is drained by about 20%. This is because the phone application is constantly connected to the server and updating the graph. The simulation program sends information at a one second rate, which probably won't be the actual rate of the system. A solution to this could not be always connected to the server, the phone application could connect to the server at a certain rate, send a command to get the updated information, receive the updated information and then disconnect from the server. Once it has done that it could then update the display. Decreasing the rate at which the display is updated will also help with improving the battery life when the phone application is on.

References

- [1] M. He, S. Warren, and M. Campbell, "Self-Management of Ovarian Cancer Using Smartphone-Based, 3D Chip Diagnostic Strategy," ed. Kansas State University: Johnson Cancer Research Center, 2015.
- [2] S. Warren, P. Prakash, B. Natarajan, D. Thompson, and W. Piersel, "GARDE: Research to Quantify the Health and Development of Children with Disabilities Around the Clock," ed. Kansas State University: National Science Foundation, 2015.
- [3] S. Warren, P. Prakash, B. Natarajan, D. Thompson, and S. Evans, "KSU Student Chapter of the IEEE EMBS as a Focal Point for Senior Design Projects to Aid Children with Disabilities," ed. Kansas State University: National Science Foundation, 2011.
- [4] P. Prakash *et al.*, "Sensors and Instrumentation for Unobtrusive Sleep Quality Assessment in Autistic Children," presented at the 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Sheraton Chicago Hotel and Towers, Chicago, IL, USA, 2014.
- [5] G. Singleton, S. Warren, and W. Piersel, "Clinical Overview of the Need for Technologies for Around-the-Clock Monitoring of the Health Status of Severely Disabled Autistic Children," presented at the 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Sheraton Chicago Hotel and Towers, Chicago, IL, USA, 2014.
- [6] D. Pascale, "A Review of RGB Color Spaces," Montreal, Canada 10/06/2003 2002-2003.
- [7] M. Tkalcic and J. F. Tasic, "Colour spaces: perceptual, historical and applicational background," presented at the The Region 8 EUROCON 2003. Computer as a Tool, 2003.
- [8] K. Jack, "A Handbook for the Digital Engineer," in *Video Demystified: A Handbook for the Digital Engineer* 5 ed. (Video Demystified, Oxford: Newnes, 2007, pp. 257-266.
- [9] W. K. Pratt, *Digital Image Processing (PIKS Inside)*. John Wiley & Sons, Inc., 2001.
- [10] E. A. Tyburski *et al.*, "Disposable platform provides visual and color-based point-of-care anemia self-testing," *The Journal of Clinical Investigation*, Clinical Medicine vol. 124, no. 10, p. 4393, 2017.
- [11] M. Haidekker, *Image Analysis and Visualization Software*, 1 ed. Wiley-IEEE Press, 2011.

- [12] P. A. Bernstein *et al.*, "Adapting microsoft SQL server for cloud computing," presented at the 2011 IEEE 27th International Conference on Data Engineering (ICDE), 11-16 April 2011, 2011.
- [13] I. Shabani, A. Kovaci, and A. Dika, "Possibilities Offered by Google App Engine for Developing Distributed Applications Using Datastore," presented at the 2014 Sixth International Conference on Computational Intelligence, Communicaiton Systems and Networks (CICSyN), 2014.
- [14] M. Cai, "The Design Method of Network Chat System Based on Socket and Cloud Computing," presented at the 2012 Conference on Computer Science & Service System (CSSS), 2012.
- [15] R. Comerford, "Software Engineering," *IEEE Spectrum*, vol. 32, no. 1, pp. 62-65, 1995.
- [16] S. Seth and M. A. Venkatesulu, *TCP/IP Architecture, Design and Implementation in Linux*. Wiley-IEEE Press, 2008.
- [17] G. Fairhurst. (2008). *The User Datagram Protocol(UDP)*. Available: <http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html>
- [18] S. Panich, "A mobile robot with a inter-integrated circuit system," presented at the 10th International Conference on Contorl, Automation, Robotics and Vision, 2008. ICARCV 2008, December 2008, 2008.
- [19] Microsoft. (2017). *Visual Studio Community*. Available: <https://www.visualstudio.com/vs/community/>
- [20] J. C. Carver, J. Cabot, R. Capilla, and H. Muccini, "GitHub, Technical Debt, Code Formatting, and More," *IEEE Software*, vol. 34, no. 2, pp. 105-107, 2017.
- [21] S. Just, K. Herzig, J. Czerwonka, and B. Murphy, "Switching to Git: The Good, the Bad, and the Ugly," presented at the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 23-27 Oct. 2016, 2016.
- [22] sourceforge. (2017). *NClass*. Available: <http://nclass.sourceforge.net/>
- [23] N. Boushehrinejadmoradi, V. Ganapathy, S. Nagarakatte, and L. Iftode, "Testing Cross-Platform Mobile App Development Frameworks (T)," presented at the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015.
- [24] P. Sestoft and H. I. Hansen, "Main Differences Between C# and Java," in *C# Precisely*1 ed.: The MIT Press, 2004, pp. 180-181.

- [25] Android. (2017). *Android Studio*. Available: <https://developer.android.com/studio/index.html>
- [26] JetBrains. (2000-2017). *Create Anything*. Available: <https://www.jetbrains.com/>
- [27] Android. (2017). *<uses-sdk>*. Available: <https://developer.android.com/guide/topics/manifest/uses-sdk-element.html>
- [28] J. Gehring. (2017). *GridView*. Available: <http://www.android-graphview.org/>
- [29] S. Schenck. (2014). *Just how many different Androids are out there?* Available: <http://pocketnow.com/2014/08/21/number-of-android-models>

Appendix A - Region of Interest Test Images

The images in this appendix illustrate the results of each step in the region-of-interest-finding algorithm. In each figure, the upper left image is the original image used to test the algorithm. The upper right image is the hue of the image, which is acquired by converting the RGB values of the original image to the YIQ color space. The middle left image is the 3D view of the image hue, along with a flat plane that is the average hue. The middle right image is the resultant image after a thresholding operation was applied based off of the average hue of the image. The lower left image is the result of image morphology being applied to the thresholded image. Finally, the lower right image is the original image with the region of interest superimposed on it, where the calculation results from the region of interest algorithm are included in the image title. Each image number in this appendix correlates to an image number found in Table 2.1.

Image 1 Results

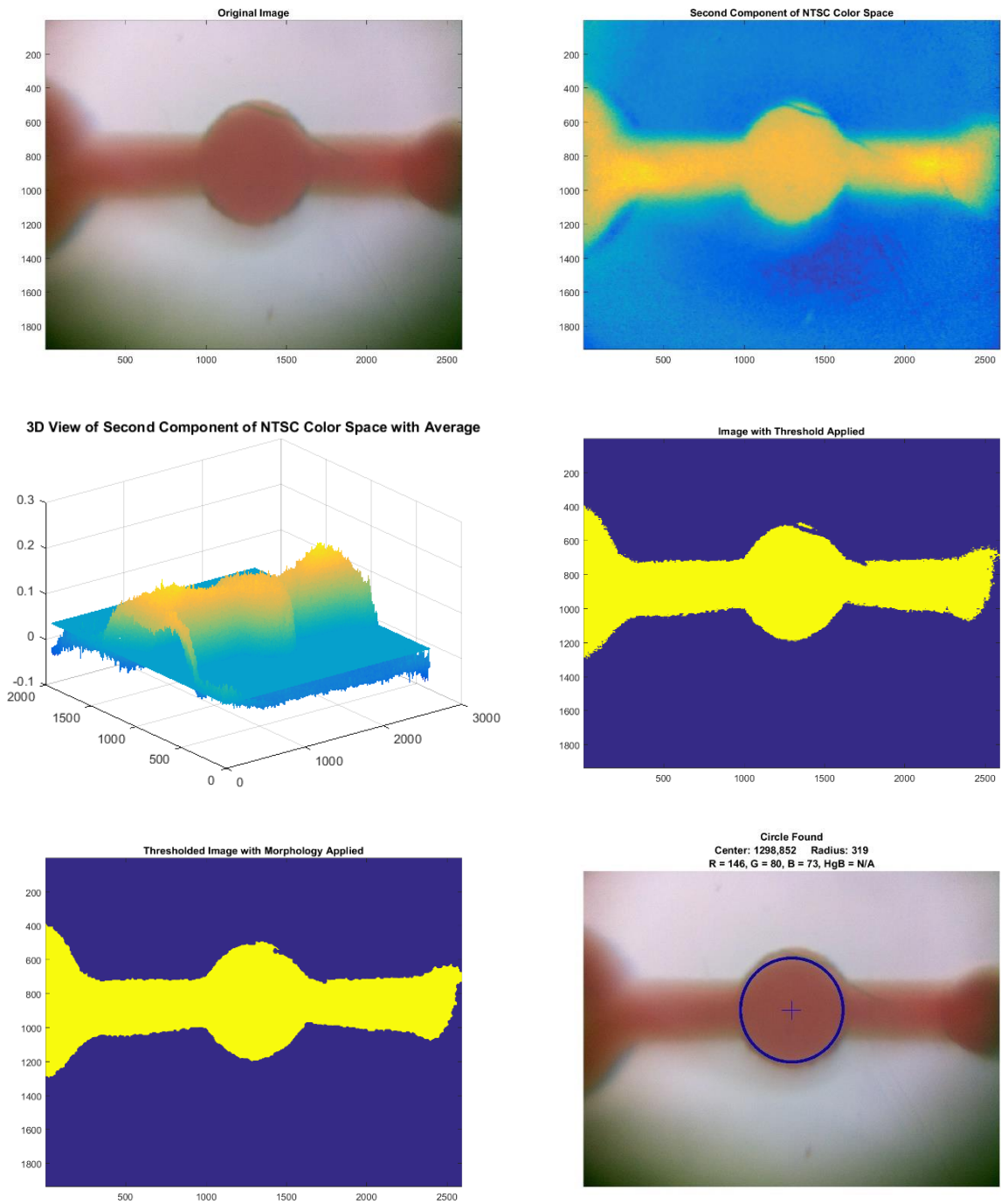


Figure A.1. Region of interest results for image number 1.

Image 2 Results

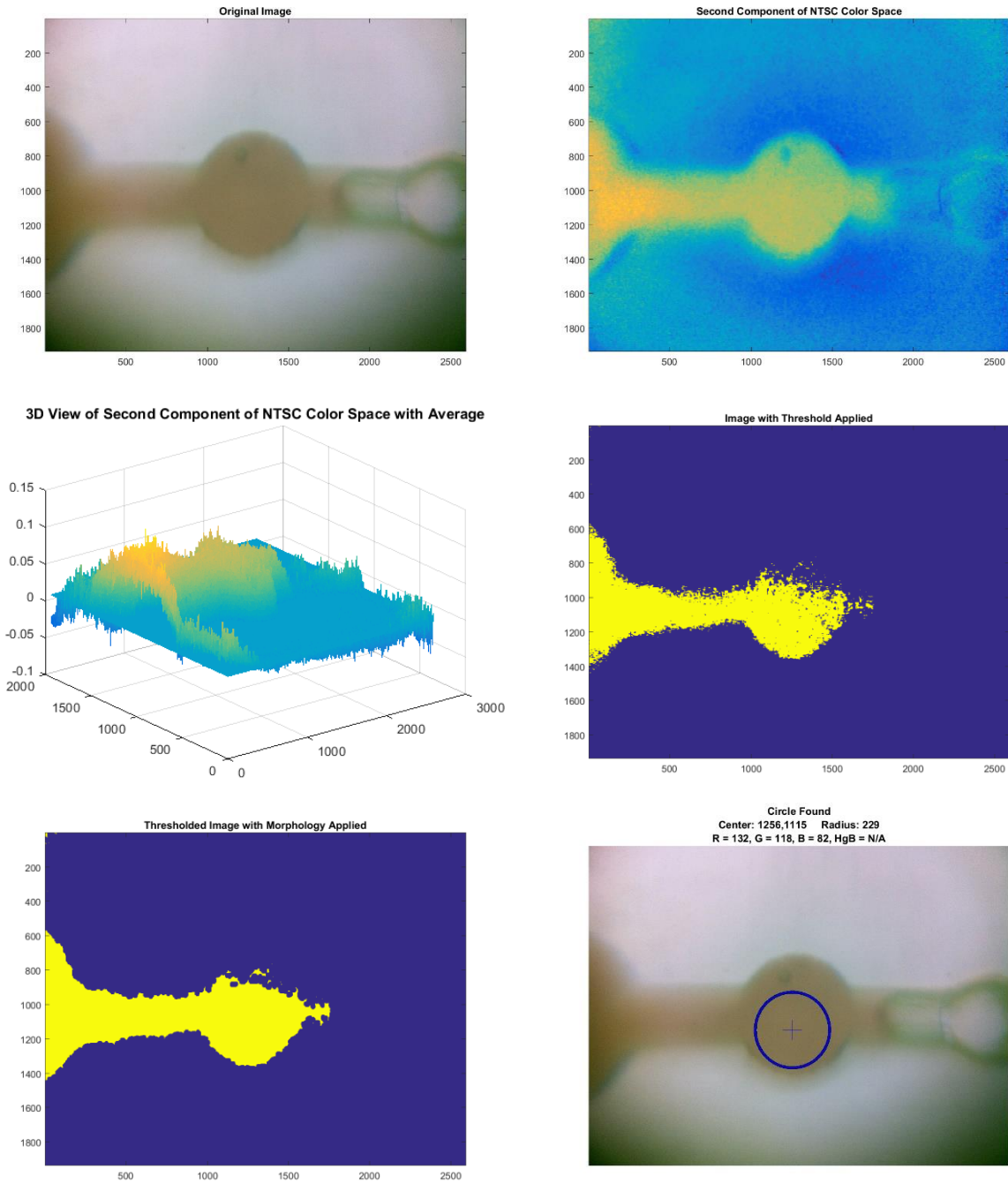


Figure A.2. Region of interest results for image number 2.

Image 3 Results

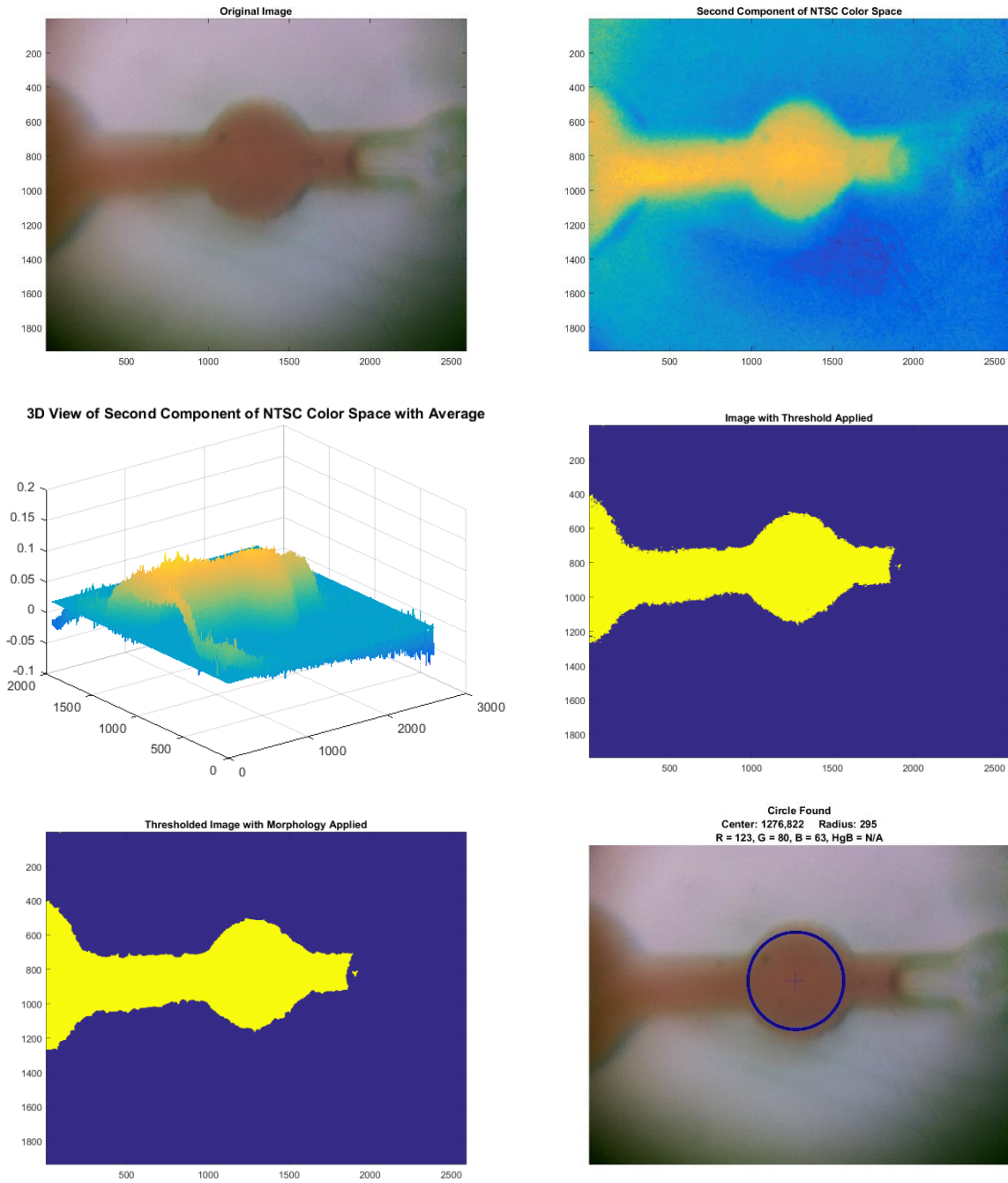


Figure A.3. Region of interest results for image number 3.

Image 4 Results

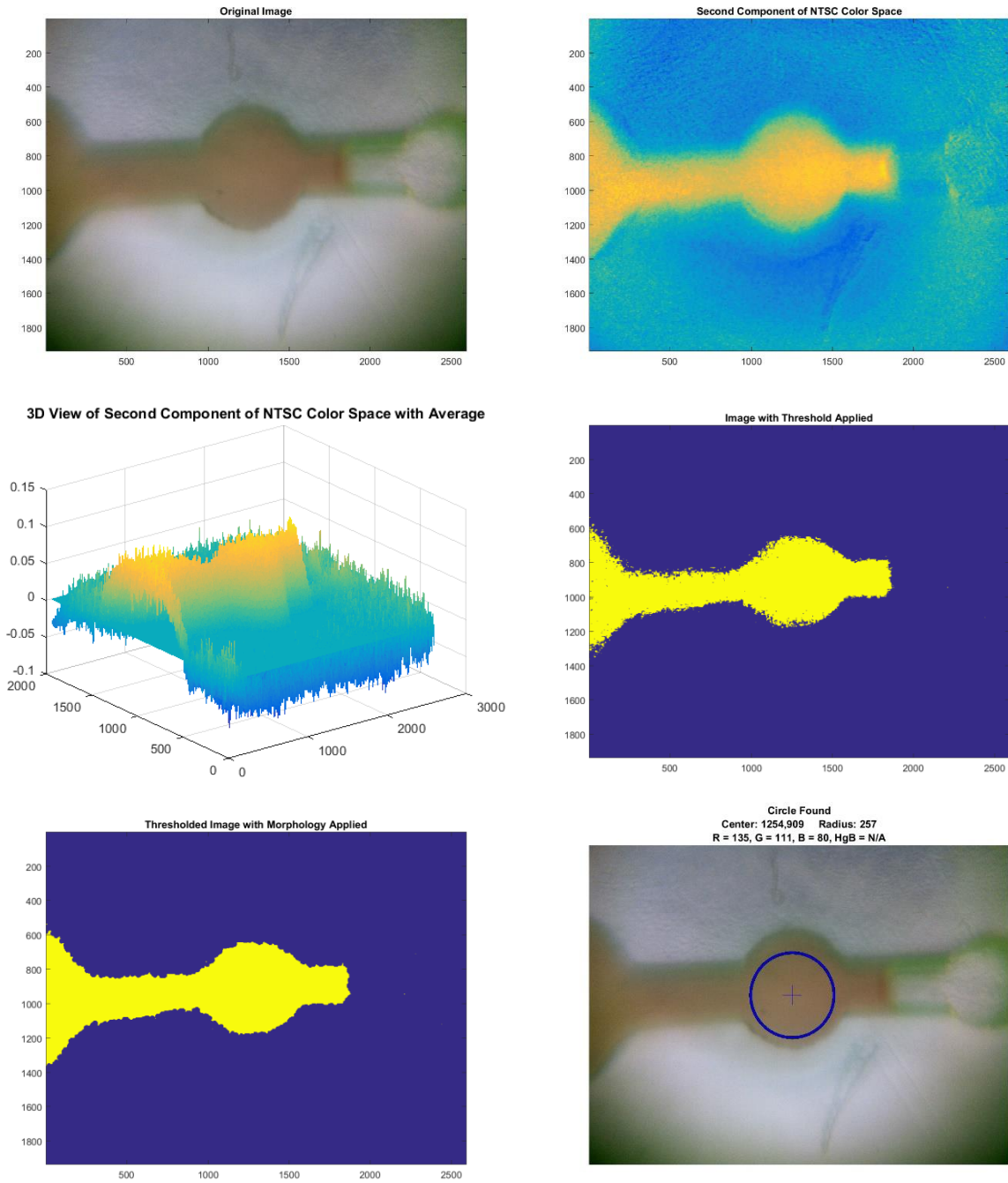


Figure A.4. Region of interest results for image number 4.

Image 5 Results

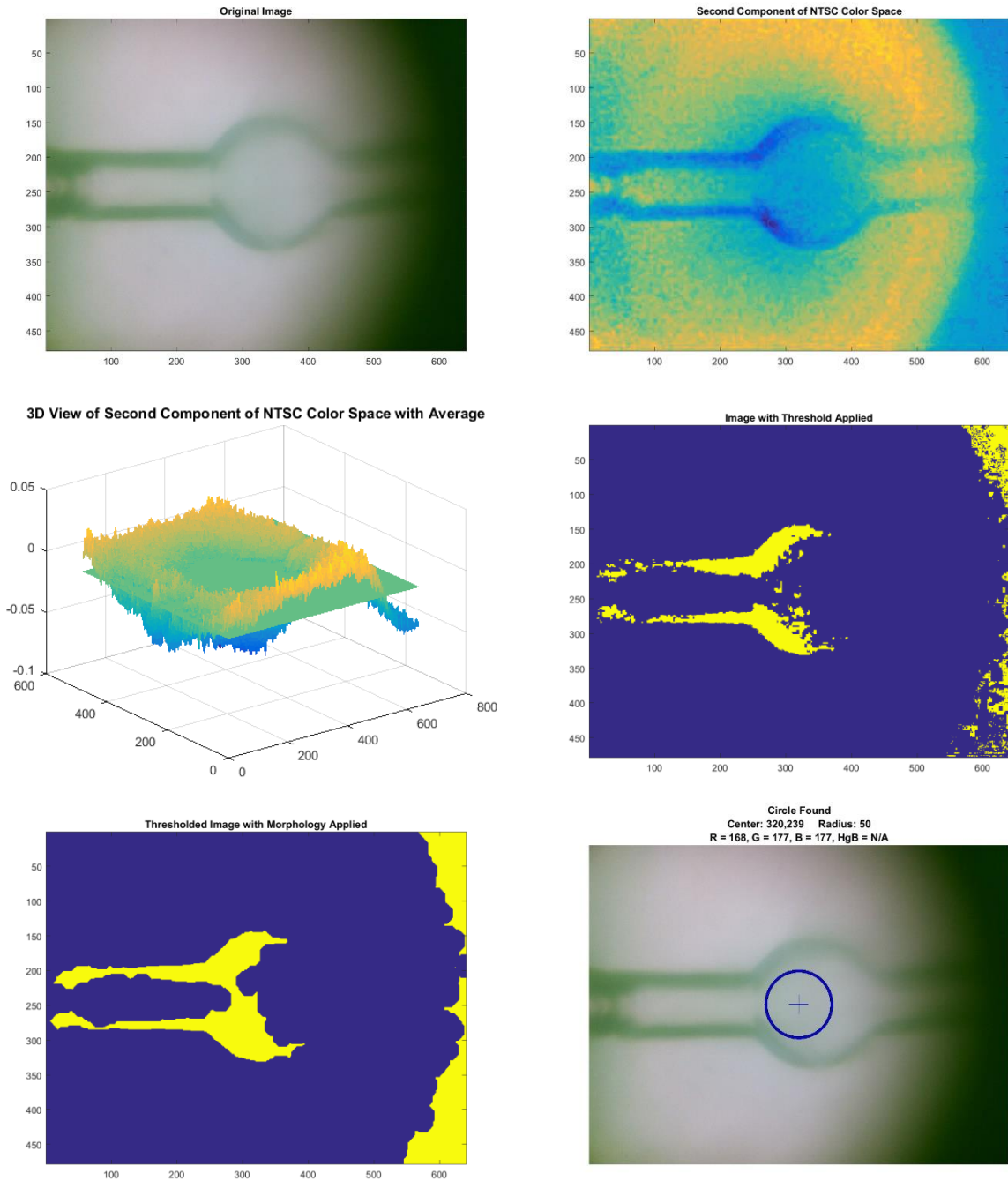


Figure A.5. Region of interest results for image number 5.

Image 6 Results

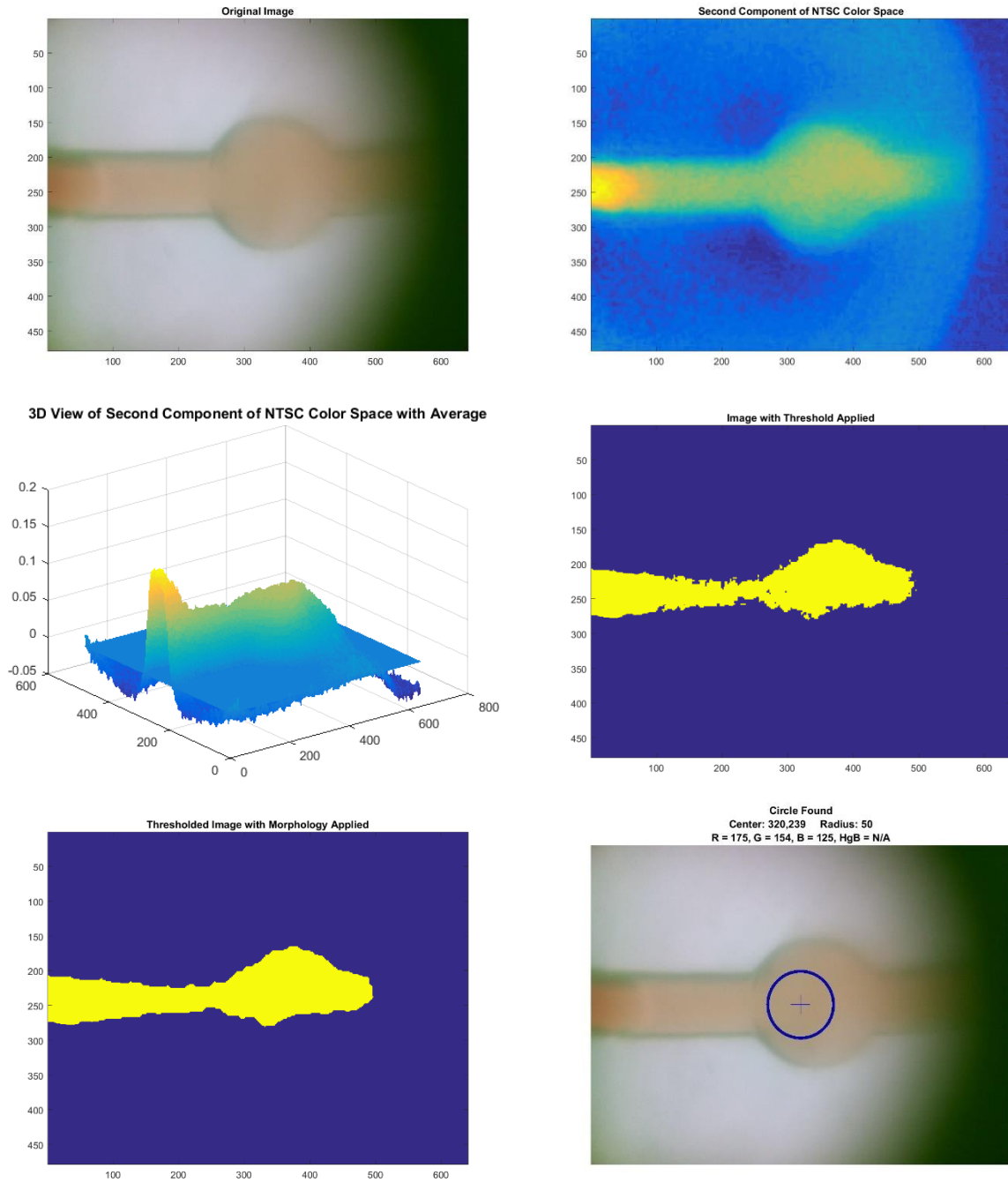


Figure A.6. Region of interest results for image number 6.

Image 7 Results

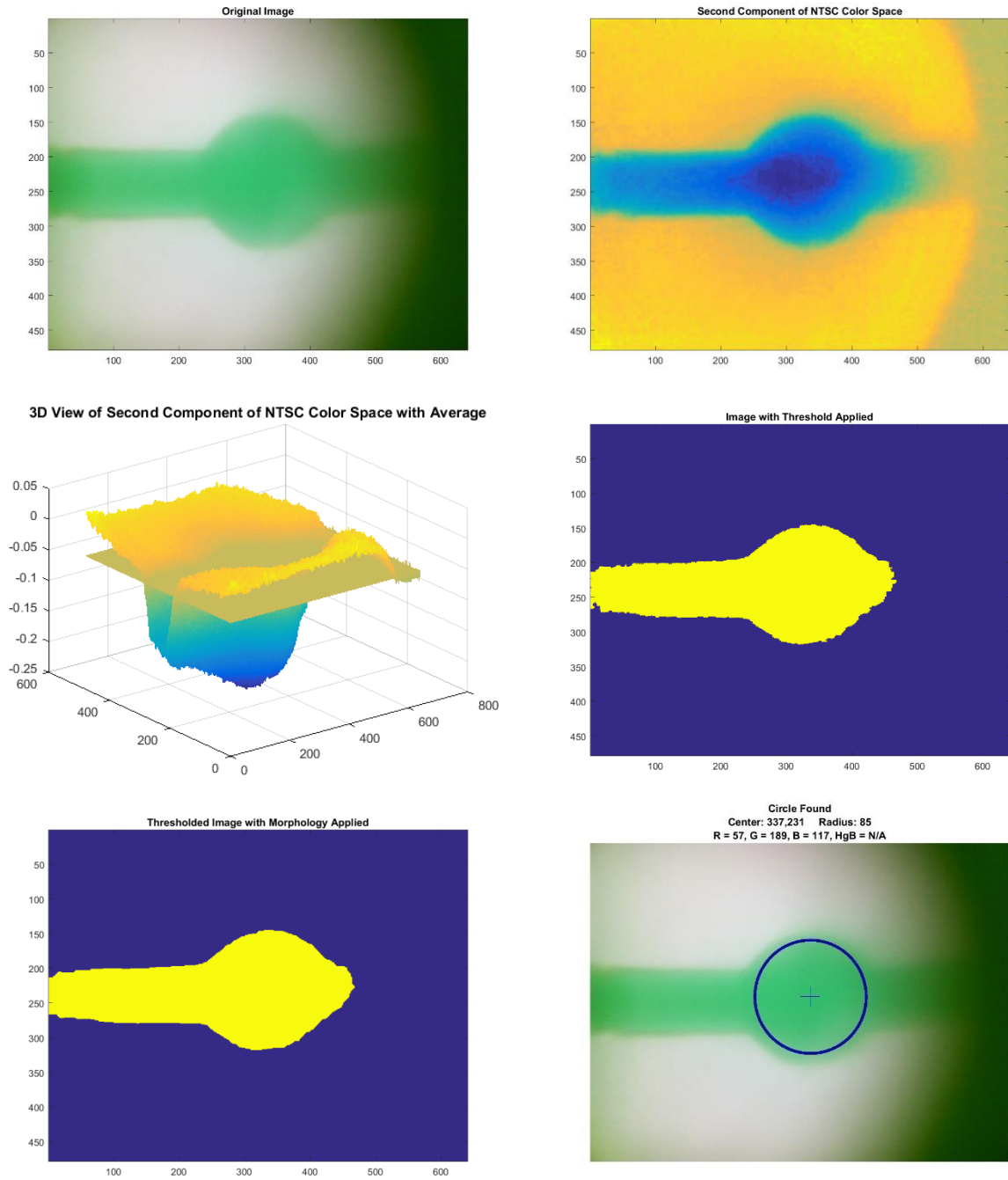


Figure A.7. Region of interest results for image number 7.

Image 8 Results

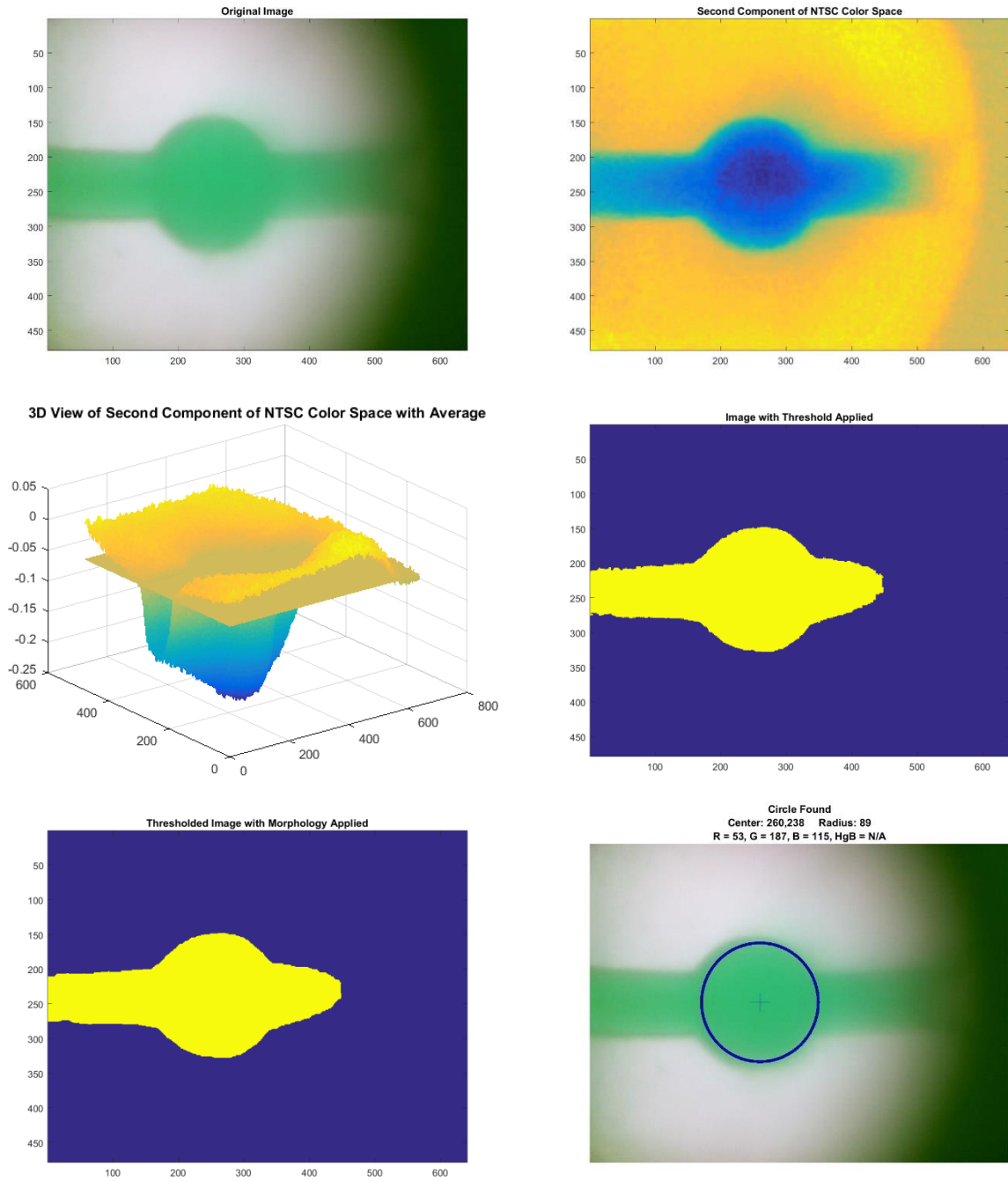


Figure A.8. Region of interest results for image number 8.

Image 9 Results

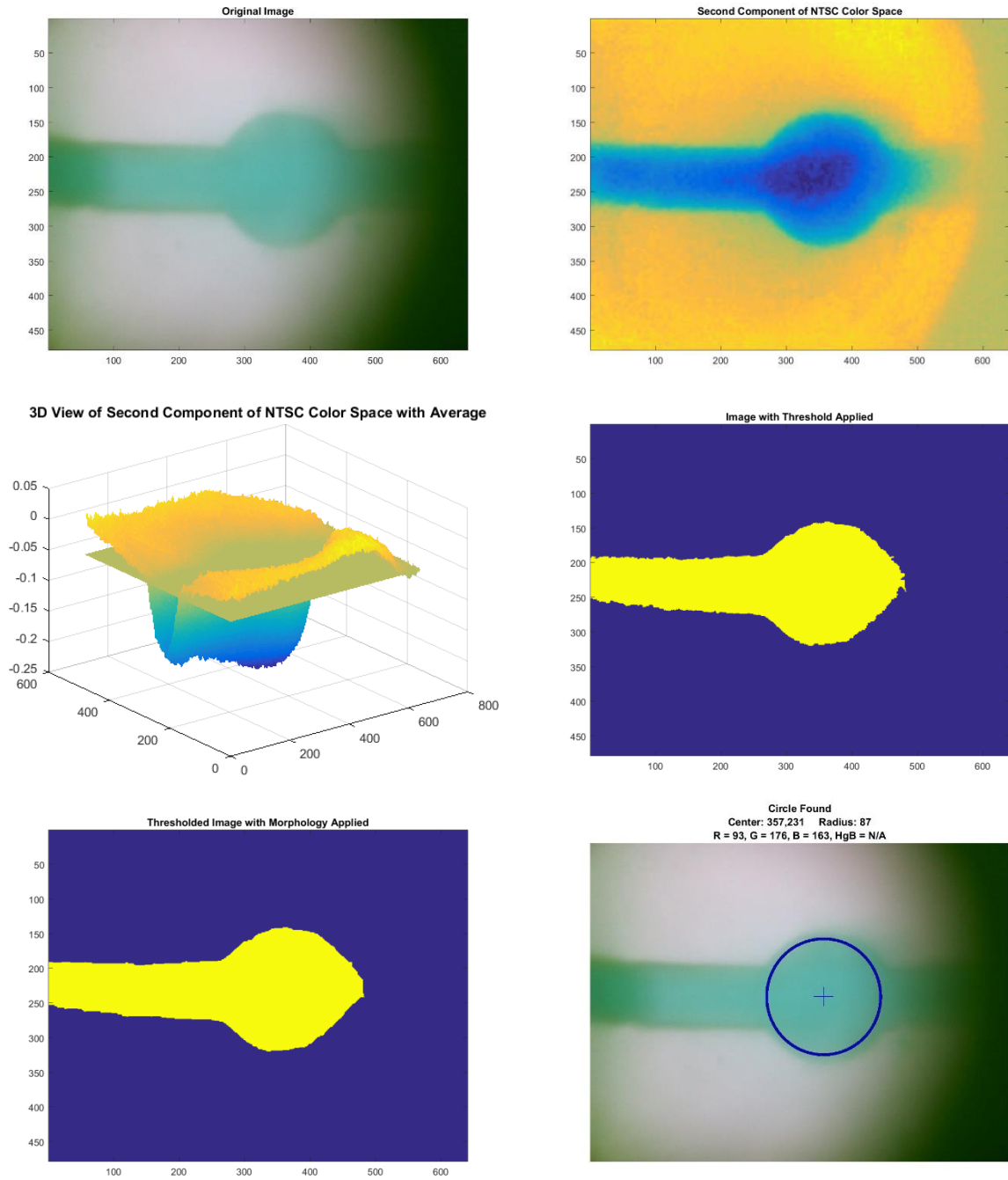
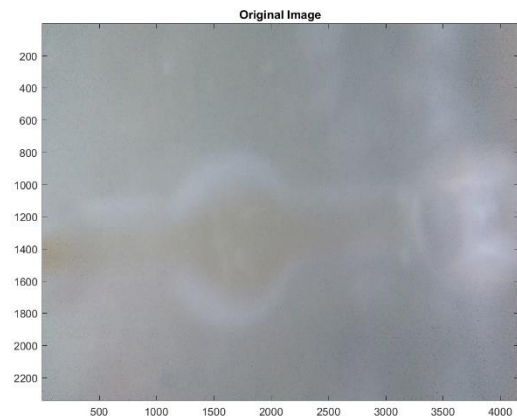
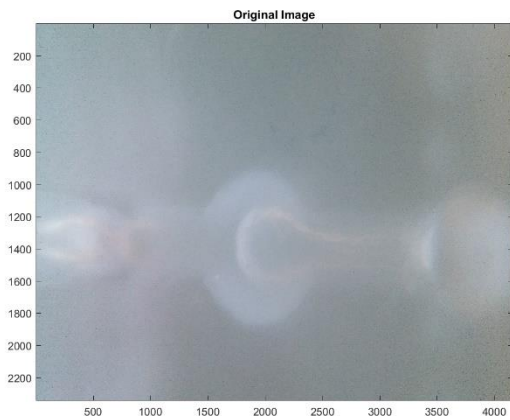
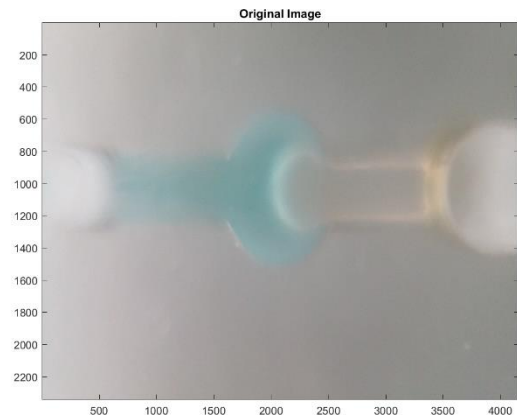
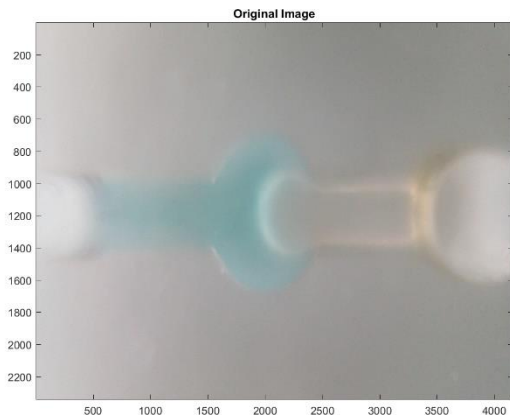
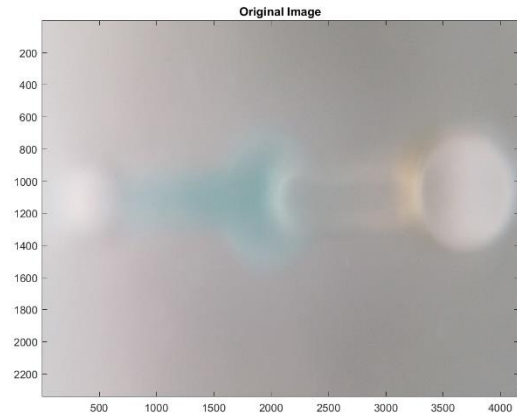
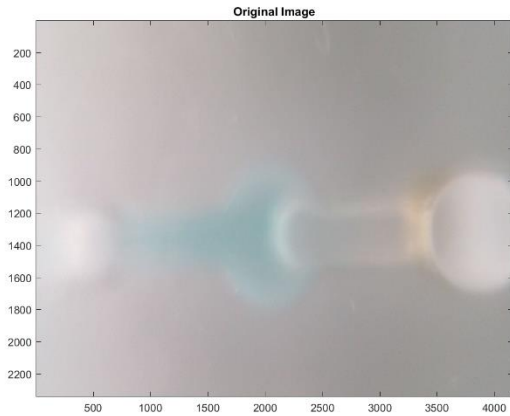


Figure A.9. Region of interest results for image number 9.

Appendix B - Other Images

The following images are images that were received a few months after the original two sets of images categorized in Appendix A. All twelve images offered unsuccessful results when the region of interest finding algorithm was applied to them.



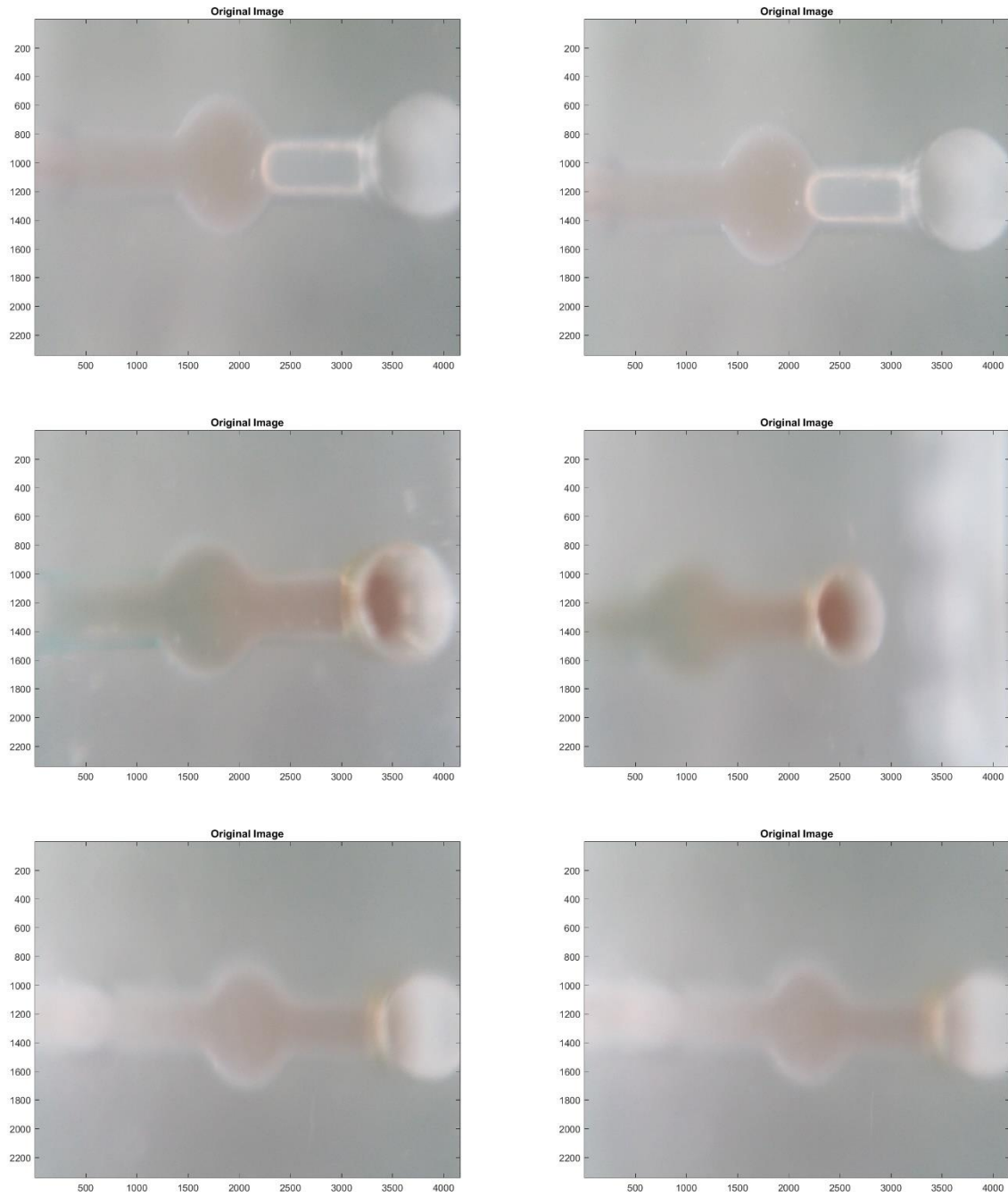


Figure B.10. Images for which the region of interest algorithm was unsuccessful.

Appendix C - Test Images Created for the Enhanced Model Tests

The following images were created using Adobe Illustrator. Figure C.11 illustrates the color scale that was created to facilitate the assignment of a scalar value to each RGB color used for model validation. The model coefficients were calculated after assigning RGB values at each of the scale lines (10, 8, 6, 4, 2, and 0). Figure C.12 contains the images used to test the models.

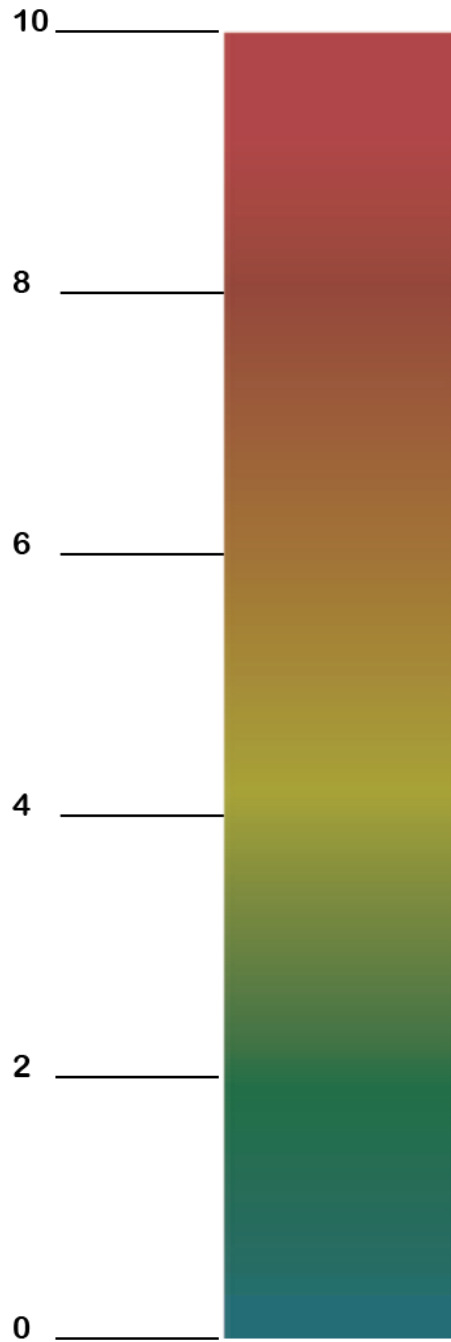
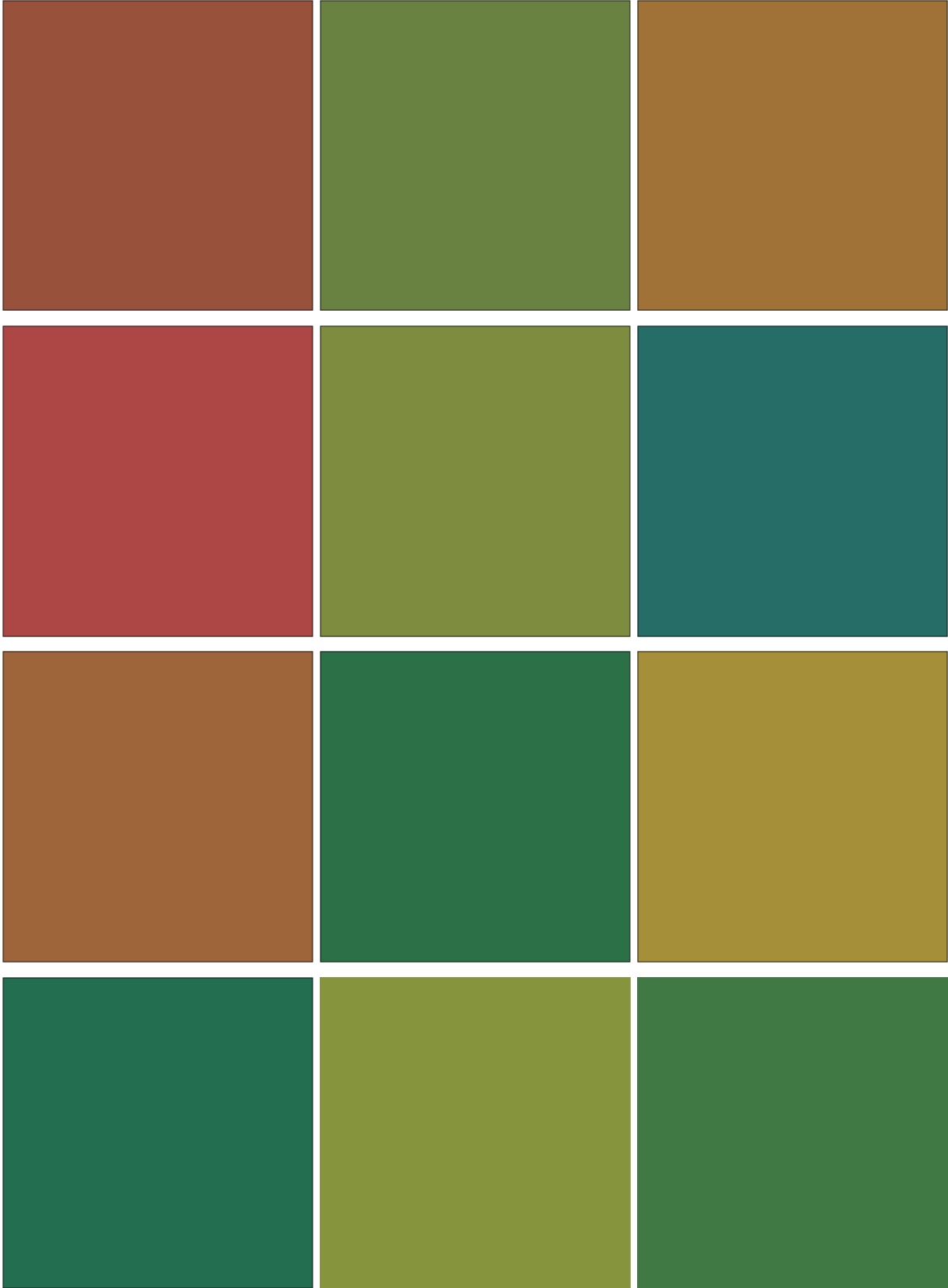


Figure C.11. Color gradient scale created to assign scalar values to RGB triplets.



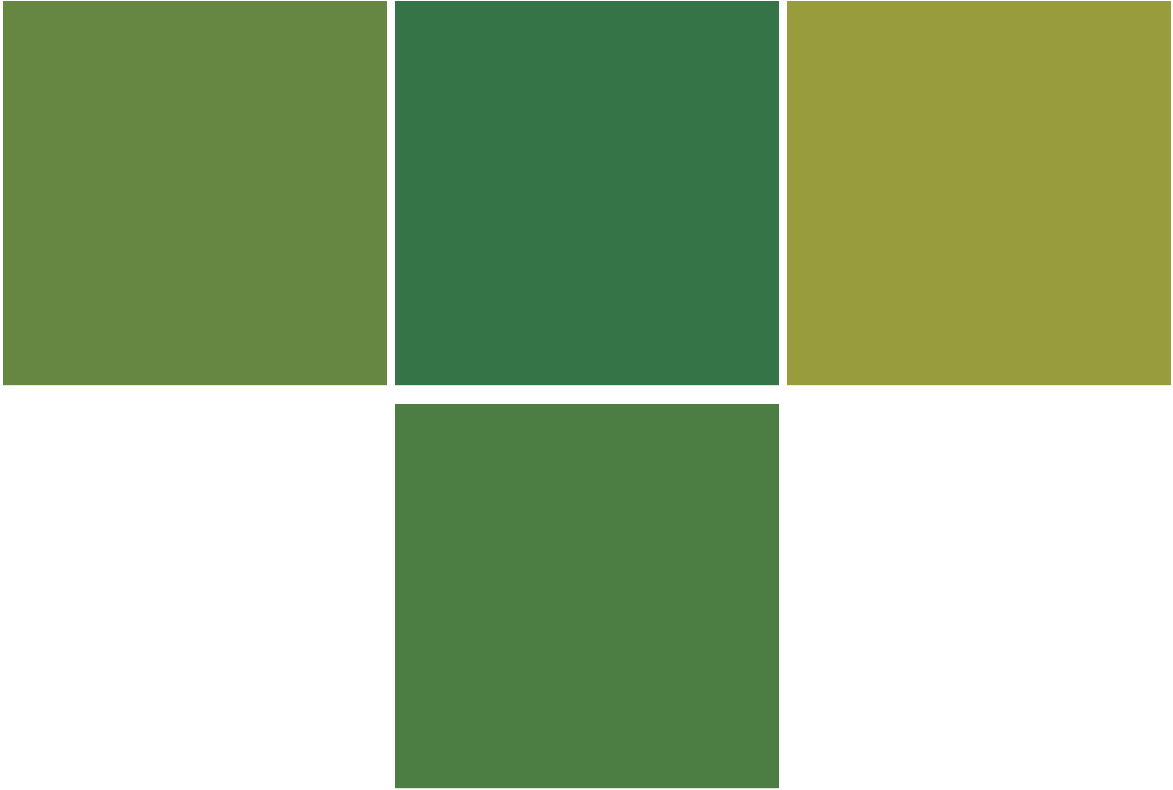


Figure C.12. Single-value RGB images created to validate the model calculations.

Appendix D - Sample Images for Model Definitions

The following images are images that were received to assist with Hgb model validation. Each image has a corresponding, experimentally determined Hgb level.



Figure D.13. Subject 1 images for an Hgb level of 9.8. First image – from a test tube. Second image – from a microfluidic chip.



Figure D.14. Subject 2 images for an Hgb level of 9.7. First three images – from a test tube. Last two images – from a microfluidic chip.

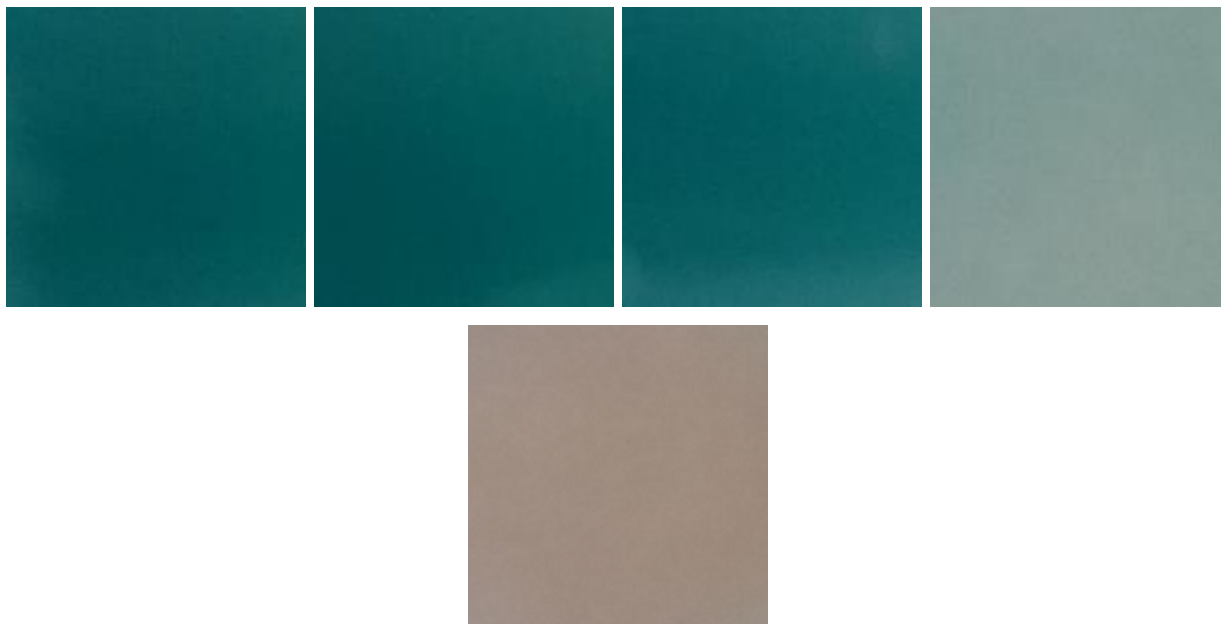


Figure D.15. Subject 3 images for an Hgb level of 9.5. First three images – from a test tube. Last two images – from a microfluidic chip.



Figure D.16. Subject 4 images for an Hgb level of 14.5. First image – from a test tube. Second image – from a microfluidic chip.



Figure D.17. Subject 5 images for an Hgb level of 8.3. First two images – from a test tube. Last image – from a microfluidic chip.

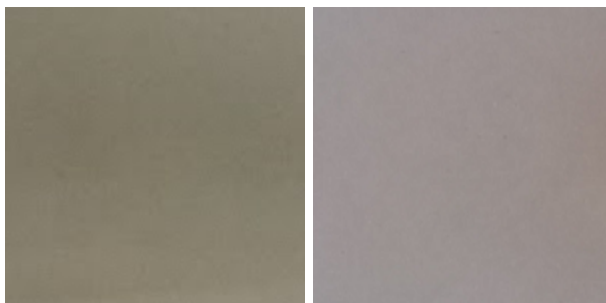


Figure D.18. Subject 6 images for an a Hgb level of 10.2. First image – from a test tube. Second image – from a microfluidic chip.



Figure D.19. Subject 7 images for an Hgb level of 6.0. First image – from a test tube. Second image – from a microfluidic chip.

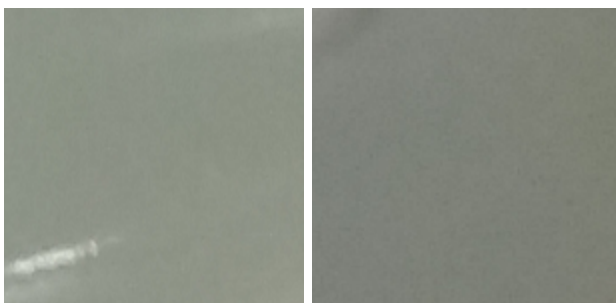


Figure D.20. Subject 8 images for an Hgb level of 10.1. First image – from a test tube. Second image – from a microfluidic chip.

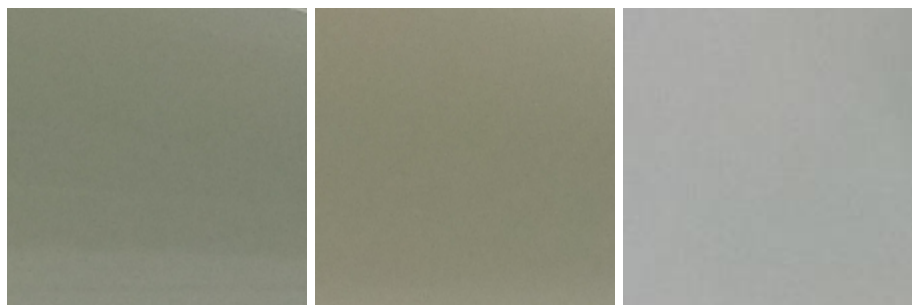


Figure D.21. Subject 9 images for an Hgb level of 14.1. First two images – from a test tube. Last image – from a microfluidic chip.

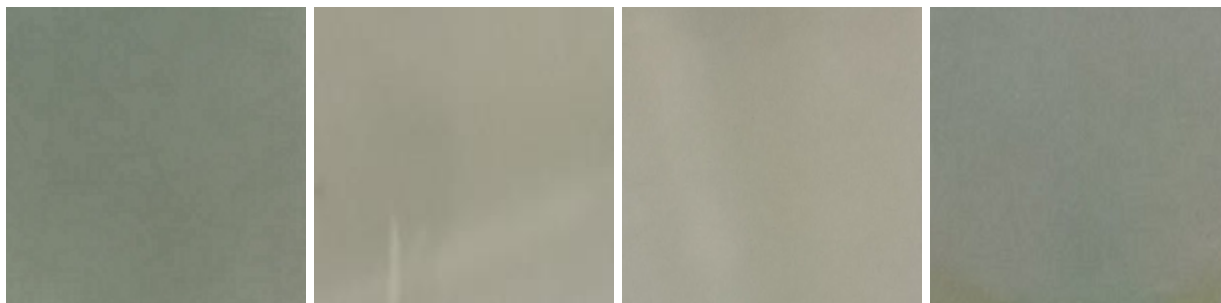


Figure D.22. Subject 9 images for an Hgb level of 7.5. First three images – from a test tube. Last image – from a microfluidic chip.



Figure D.23. Subject 10 images for an Hgb level of 7.3. First three images – from a test tube. Last image – from a microfluidic chip.

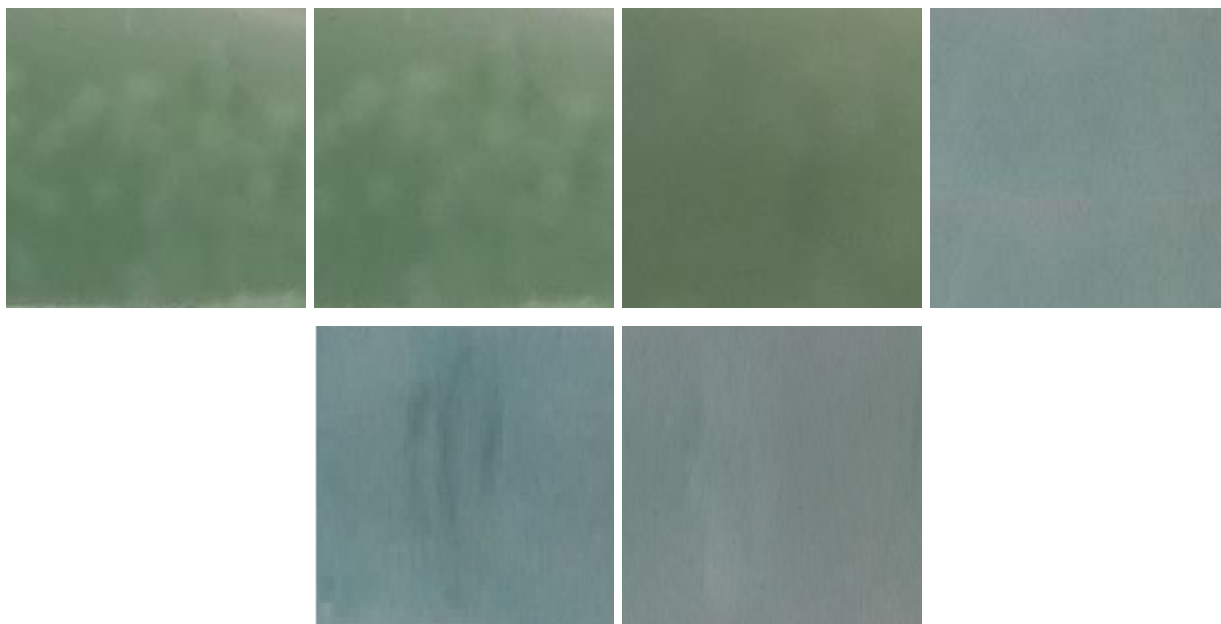


Figure D.24. Subject 11 images for an Hgb level of 9.7. First three images – from a test tube. Last three images – from a microfluidic chip.