THE DESIGN, FABRICATION, AND EVALUATION OF MOBILE POINT-OF-CARE SYSTEMS FOR CELLULAR IMAGING IN MICROFLUIDIC CHANNELS

By

MARK LEWANDOWSKI

Submitted in partial fulfillment of the requirements for the degree of

Master of Science

Mechanical Engineering

CASE WESTERN RESERVE UNIVERSITY

January, 2018

CASE WESTERN RESERVE UNIVERSITY SCHOOL

OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

Mark Lewandowski

candidate for the degree of Master of Science.

Committee Chair

Dr. Umut Gurkan

Committee Member

Dr. Ozan Akkus

Committee Member

Dr. Clare Rimnac

Date of Defense

November 29, 2017

*We also certify that written approval has been obtained for any proprietary material contained therein.

TABLE OF CONTENTS

List of Tables

List of Figures

Acknowledgements

I would like to acknowledge the following team members for their contributions towards advancing this multi-part project: Arwa Fraiwan, Yunus Alapan, M. Noman Hasan, Erdem Kucukal, Yuncheng Man, Jonathan Koss, Jane Little, Erina Quinn, Charlotte Yuan, and Umut Gurkan. A special thank you is due to Dr. Gurkan, who provided the opportunity to work on this project and contributed his time to guide this project and my career as a graduate student at Case Western Reserve University. I would also like to thank Dr. Deepa Manwani for her collaboration at the Children Hospital at Montefiore. Lastly, I thank my committee members Dr. Akkus and Dr. Rimnac for their participation in my thesis defense. This project was made possible by the funding from The National Heart Lung and Blood Institute, the Case-Coulter Translational Research Partnership, and the services and expertise found in CWRU's think[box].

List of Abbreviations

Beta Defensin Index	BDI
Case Western Reserve University	CWRU
Computer Aided Design	CAD
Fibronectin	FN
Field of View	FOV
Fluorescent Smartphone Imaging Device	FSID
Institutional Review Board	IRB
Laminin	LN
Light Emitting Diode	LED
Pearson-product-moment Correlation Coefficient	PCC
Poly(methyl methacrylate)	PMMA
Point-of-Care	POC
Red Blood Cell	RBC
Revolutions Per Minute	RPM
Sickle Cell Disease	SCD
Smartphone Imaging Device	SID
Thrombospondin	TSP

Abstract

The Design, Fabrication, and Evaluation of Mobile Point-of-Care Systems for Cellular Imaging in Microfluidic Channels

By
MARK LEWANDOWSKI

The utilization of mobile imaging built on Smartphone platforms as an inexpensive tool is becoming a popular research area. Many diseases require a point-of-care device, that can quickly screen cellular samples for analysis, to alleviate financial burden, provide quick diagnosis, and monitor disease conditions. Two diseases, Sickle Cell Disease and Oral Cancer would benefit from a portable imaging device. Sickle Cell Disease affects millions of people worldwide and causes morbidity and mortality, especially in children. Currently there is no portable method of monitoring disease severity through abnormal cellular adhesion, which is at the root of serious SCD symptoms. There is also no quick and inexpensive method of screening for oral cancer during a routine dental checkup. Presented here is a clinically appropriate device that captures cellular images of microchannels, with both bright field and fluorescent capabilities, to help monitor SCD severity and detect the presence of cancer in oral lesions.

1) Smartphone Bright Field Imaging

1.1 Introduction

i) Mobile Bright Field Imaging

Utilizing Smartphone technology as an inexpensive imaging tool is quickly becoming a popular research area. Researchers have used Smartphone imaging systems with bright field capabilities to image various microparticles via different methods. Researchers from University of Connecticut have explored the use of a normal bright-field enabled Smartphone attachment, paired with magnets, to observe the magnetic levitation of micro particles in a suspension [**1**]. Mobile bright field imaging on Smartphone based platforms have allowed for new portable and inexpensive systems that are capable of cellular screening.

ii) Sickle Cell Disease

SCD is a disease that affects the formation of hemoglobin [**2-4**] in RBCs. This defect causes many changes to the properties of the cell such as: shape, cellular deformability [**5-8**], but most importantly abnormal adhesion [**7, 9-13**]. Vaso-occlusions are blood vessel blockages formed from accumulated adhered RBCs. In SCD, disease severity [**14-17**], organ damage [**18**], painful crises, and early mortality have all been attributed to vaso-occlusions caused by adhered cells to endothelium [**19**].

iii) Need and Approach

Even though abnormal cellular adhesion to endothelial proteins is at the root of vaso-occlusions, which are the clinical hallmarks of SCD symptoms [**13, 20**], there is no clinically relevant tool to evaluate cellular adhesion as a clinical biomarker for disease severity. A validated method of quantifying abnormal RBC adhesion has been demonstrated by researchers at CWRU by introducing patient blood into a endothelial protein coated channels of a microfluidic chip, called the SCD Biochip, imaging adhered cells with a scanning microscope, and manually counting the cells in Photoshop [**21**]. The

aim of this project is to replicate these results utilizing a more portable and cost-efficient method of imaging. In SID, we utilize a Smartphone to aid in the capture and analysis of cellular images.

1.2 Materials and Methods for a SCD Monitoring Device

i)μ Materials

The POC system is comprised of a microfluidic chip (SCD Biochip), a Samsung Galaxy S4, an optical Smartphone attachment, and a custom android application that digitally processes cellular images **(Fig. 1)**. The POC system's optical attachment incorporates a bright field feature.

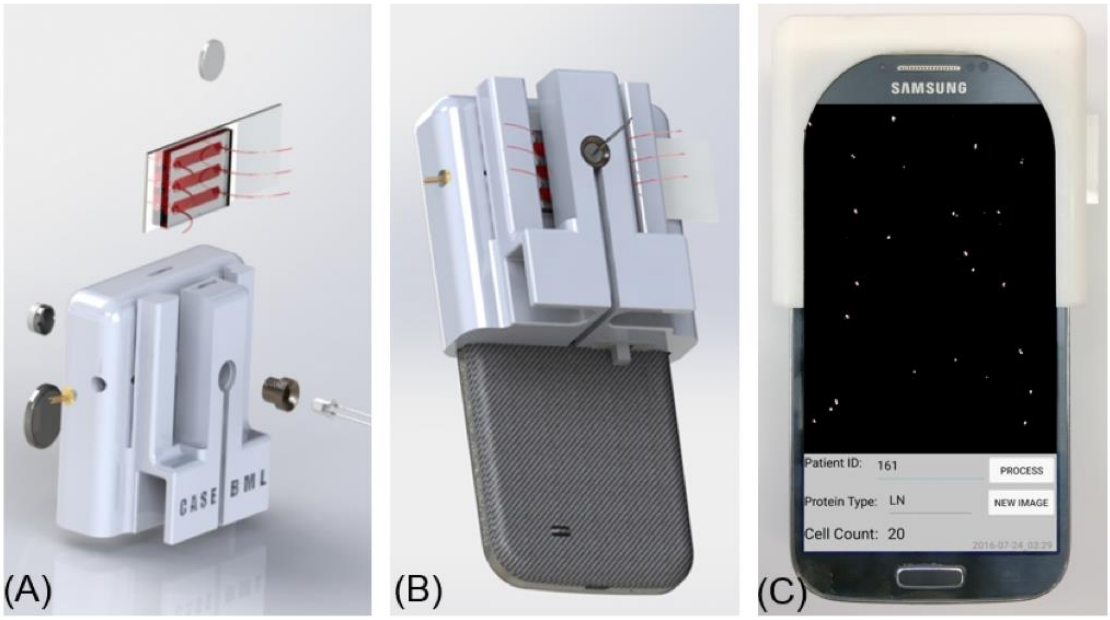


Figure 1: A. The respective components of the SCD POC device. **B.** The POC device paired with a Samsung Galaxy S4. **C.** A custom android application, CellVision, showing results from a successful test.

A 3v coin cell battery powers a bright white LED, controlled by a 3-terminal slide switch. The LED light passes through a diffuser, polarizing the light, and illuminates one of channels in the SCD Biochip, demonstrated by the optics diagram (**Fig. 2**). The light is collected onto an external ball lens with a diameter of 10mm and

received on the camera lens of the Smartphone, distanced to one focal length away from the sample for image clarity. The light from the Smartphone lens is submitted to the embedded CMOS sensor chip that creates the image of cells to display on the Smartphone screen. Fine focusing is achieved through the auto-focusing programming of the custom android application, CellVision.

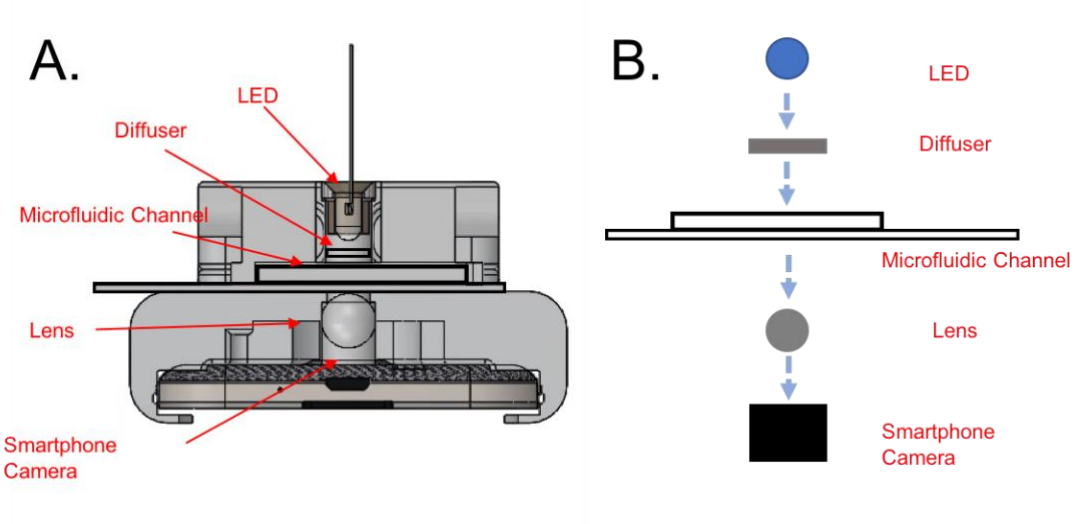


Figure 2: A. Cross section view of optical design from **B.** initial optical schematics.

Initial designs included a 6.33mm diameter aspheric lens. Due to limited FOV from that lens, larger lenses were investigated. Other lenses that were considered were a 7.2mm diameter lens and a 10mm diameter ball lens (**Fig. 3**). The 10mm diameter ball lens provided a larger and clearer FOV, extending the FOV to the channel walls at the standard Smartphone magnification. The 10mm diameter ball lens was used in all the subsequent patient trials. The purpose of this optical attachment is to image RBC's which range from 5-10 μm, so the system must be able to detect particles of this size. However, this optical attachment has the ability to image down to 1 μm (**Fig. 21**) and can easily detect RBC's. The pixel resolution of this system is 1080 by 1920, or a 16:9 ratio.

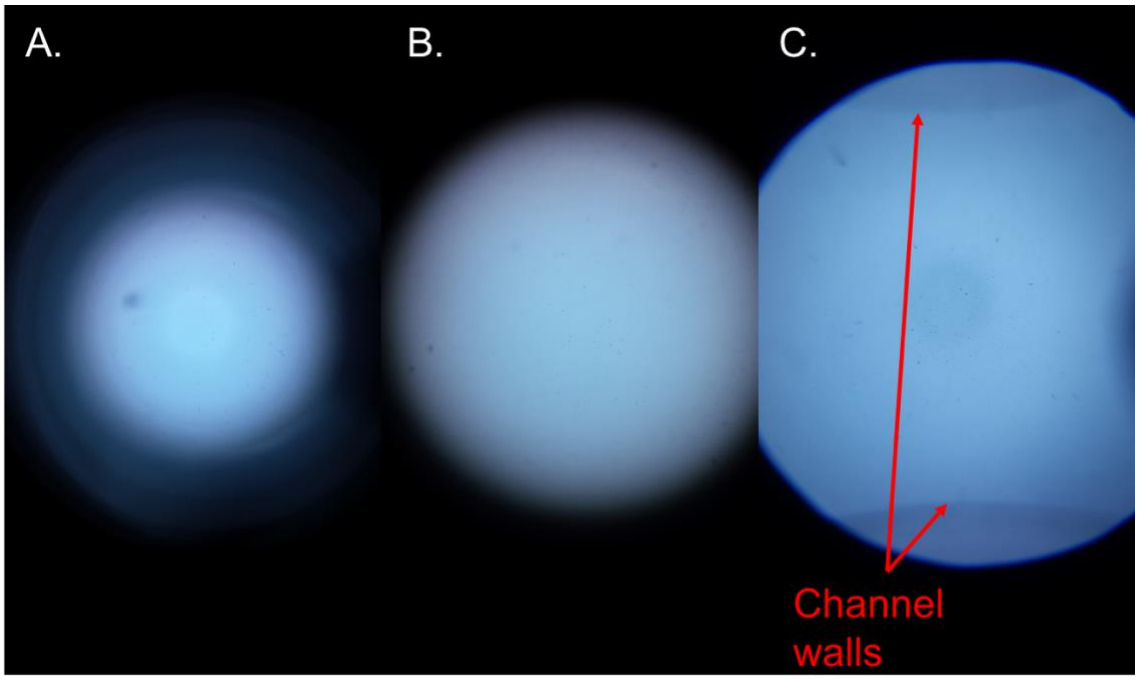


Figure 3: Changes in FOV for the optical attachment at minimum magnification with **A.** 6.33mm aspheric lens **B.** 7.2mm aspheric lens **C.** 10mm ball lens.

ii) SCD POC Device Fabrication and Assembly

The 3D optical attachment (Figure 1) was designed using a 3D modeling software (SolidWorks) and built by a 3D printer (Fortus 400mc) in think[box] at CWRU. All fabrication on the 3D printed housing was also conducted in think[box], including milling, soldering, and assembly of optics and circuitry components. Our optical attachment is compatible with a Samsung Galaxy S4 for sickle cell disease monitoring. The Samsung Galaxy S4 was chosen as it provided less design constraints than other phones on the market. The camera is located in the center and far enough away from any edges of the Smartphone. The location of the camera mattered in the design of this device, because the SCD biochip is considerably large compared to the body of the phone. In designing a docking location for the biochip it made the most sense for the sample loading area to be located in the center of the device which lines up nicely to the camera on the Samsung Galaxy S4.

iii) Experimental Design

Blood samples were obtained from University Hospital's Hematology and Oncology Division under IRB approval. For information regarding blood flow steps through the SCD Biochip, please refer to the publication [**21**]. Cellular images from the microscope were uploaded to Photoshop and manually counted. The microfluidic chip was removed from the microscope and inserted into SID. SID captured a 1mm by 1mm image from the center of each channel, processed, and recorded by our CellVision android application (**Fig. 4**).

iv) **Automated Image Processing**

Image capturing and digital image processing was performed with a custom Android application developed by Jonathan Koss, an electrical engineering graduate student at Columbia University. The first screen of the application previews the camera and allows the user to capture an image. This custom camera application allows many of the camera properties to be fixed to ensure consistency between images. After an image is taken, the image is displayed so the user can check the image quality and then choose to process the image or take another. The image is automatically time stamped and the user then inputs patient information and proceeds to processing the image.

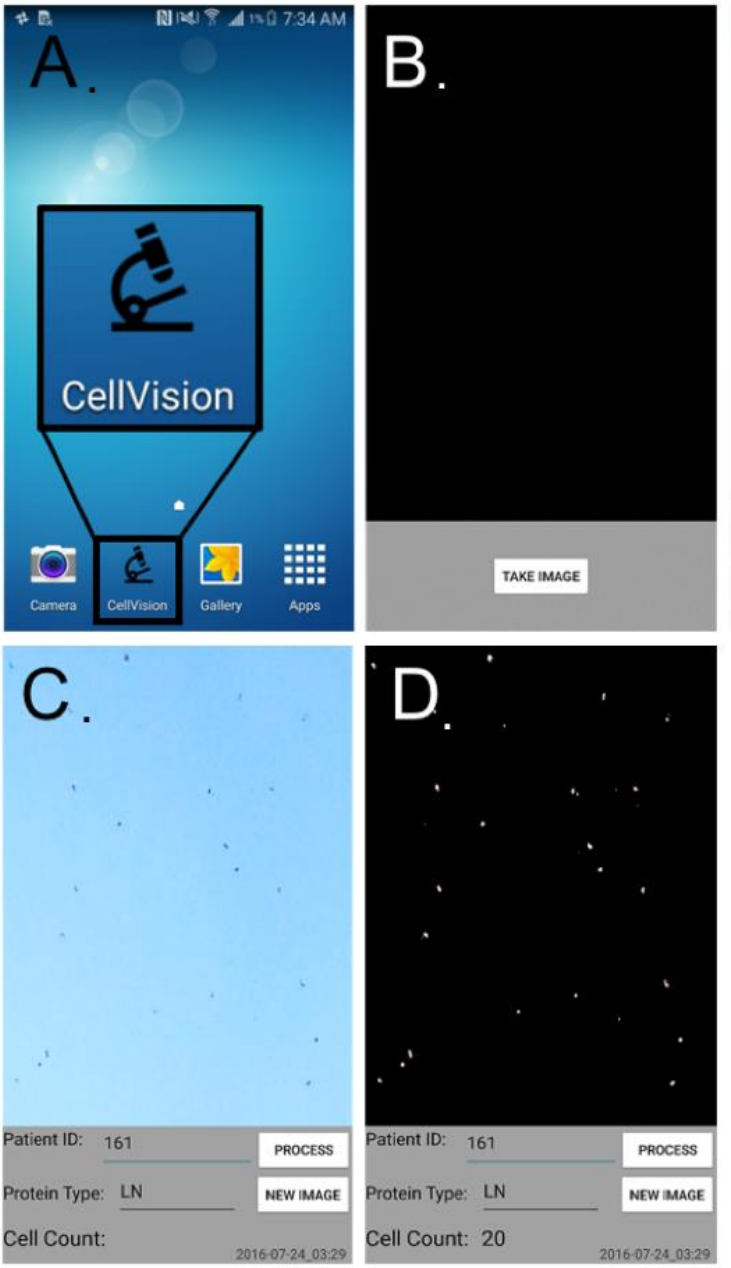


Figure 4: Work flow of the automated cell counting android application, *CellVision*. **A.** Locating the application on the home screen. **B.** The camera screen opens. **C.** Taking an image of the cells and inputting patient information. **D.** Processing the image and saving the results.

Processing is done by utilizing the OpenCV library for Android. The workflow for CellVision is presented in **Figure 5**. The processing begins by cropping the image to a 1mm x 1mm square within the microchannel. Then, an adaptive threshold was used to

turn the potential cells white and the background black. Next, all white contours were identified and filled in with white color as well. Cell counting is the performed using the OpenCV simple blob feature detector labeling all properly sized objects in the image as cells. The number of cells detected by the application is then displayed on the screen along with the patient information. The application automatically saves a screen shot of the experimental results to the google photos folder of the android operating system. When connected to Wi-Fi, the photos automatically upload to an online patient database which can be viewed by members of the team anywhere around the world in a matter of minutes. Patients are given identification codes so that their identities are kept secret.

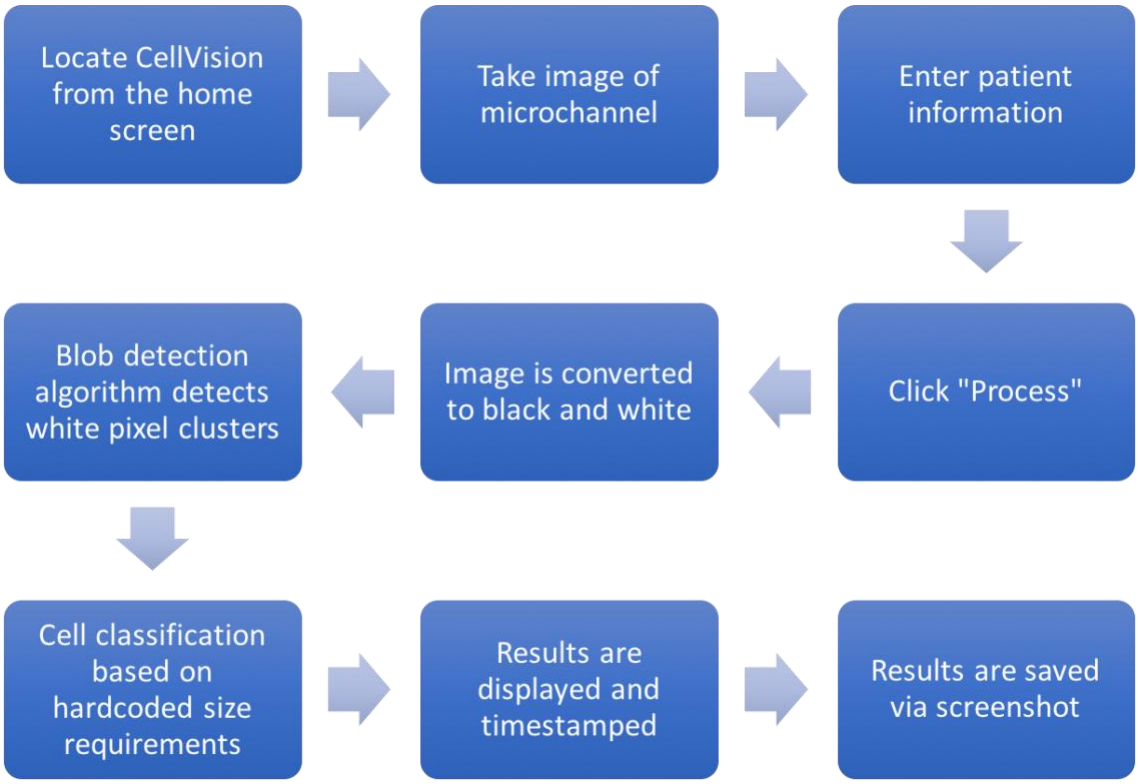


Figure 5: CellVision application workflow.

v) Statistical Analysis

Statistical analysis was completed on the retrieved data. The number of cells that our Smartphone imaging platform counted was plotted against a traditional microscopy

approach. A linear fit was added to all data plots to demonstrate the linear trend of the data. Also, a Pearson product-moment correlation coefficient was generated for every plot to determine the level of correlation the portable imaging system cell counts were compared to manual counts from the scanning microscope images.

1.3 Results

i) SCD POC Device Design

The phone attachment housing was 3D printed on a Fortus 400mc printer in CWRU's think[box]. The lens, diffuser, slide switch, battery, LED were hand assembled and installed into the device. This Smartphone imaging system allowed for the portable quantification of cellular adhesion in SCD blood samples and showed strong correlation to experiments completed with traditional scanning microscopy. This device was also cost efficient, effectively costing only $131.68 to create a functional prototype (**Appendix 1**). For approximately $231, was able to generate the following cellular FOV within the microchannels (**Fig. 6A**). The RBC's are easily distinguishable for both manual and automated counting.

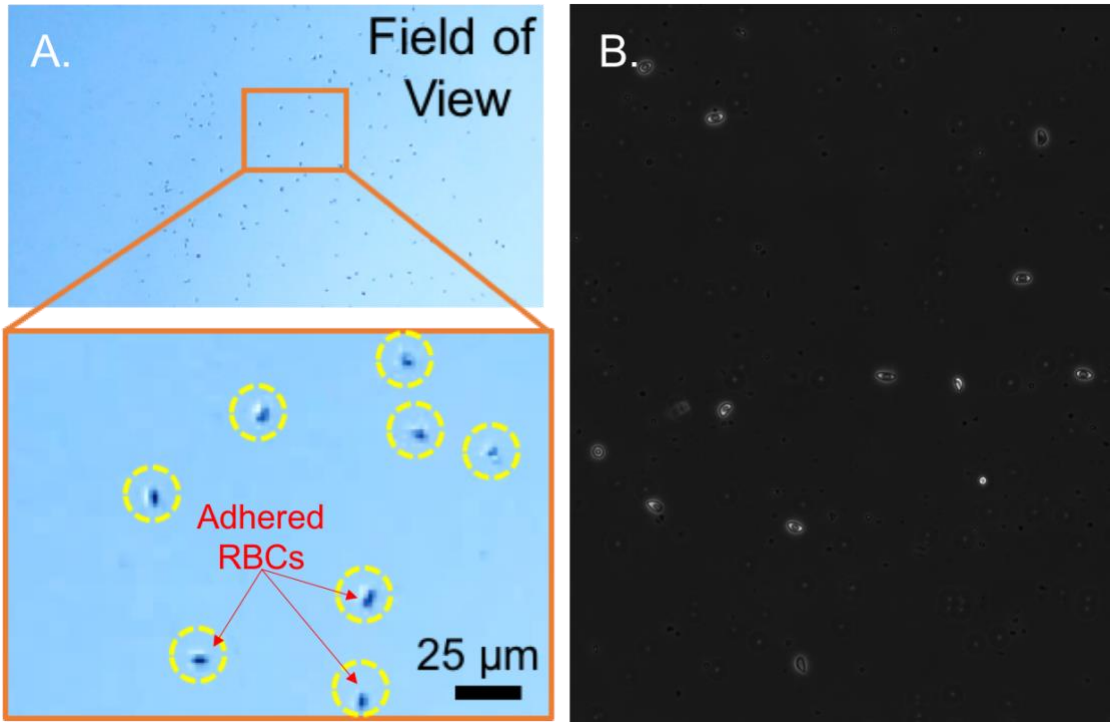


Figure 6: Typical cellular images as seen on the **A.** POC system compared to those on **B.** an expensive scanning microscope.

ii) SCD POC Device Laboratory Validation

Our side-by-side experiments between an inverted microscope (Olympus IX83) and our Smartphone imaging system yielded cell counts from the microscope and Smartphone imaging system **(Fig. 7)**. Laminin coated channels showed homogeneous adhesion through the entire microchannel and therefore held the highest correlation of 0.96. RBC's flowed through Fibronectin and Thrombospondin coated channels behaved slightly less homogeneous, but nevertheless had a strong correlation of 0.81 each.

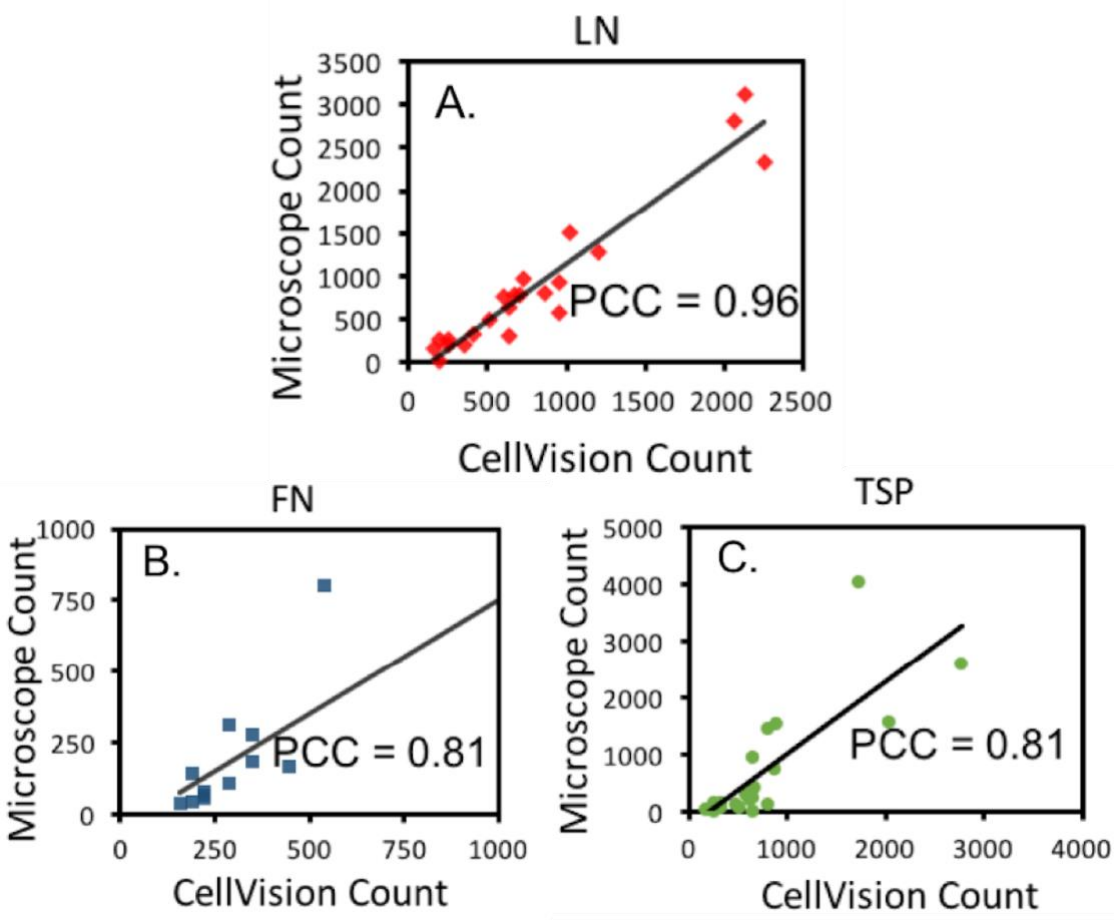


Figure 7: Scatter plots of cellular counts from our Smartphone imaging system against an inverted microscope (Olympus IX83). **A.** Laminin coated channels. **B.** Fibronectin coated channels. **C.** Thrombospondin coated channels.

These data points where collected from a $1mm^2$ image generated from the SID system. The data was standardized by multiplying by 31.8 and compared to a $31.8mm^2$ microscope image. Areas of $1.5mm^2$ and $2mm^2$, with standard multipliers of 21.2 and 15.9 respectively, were also investigated to determine the optimal area size to sample data from (**Fig. 8**). Data was promising for the use of larger sized FOVs for LN. However, FN and TSP channels had very low correlations to the microscope counts. $1mm^2$ area showed the best results across all proteins and therefore all subsequent patient data was taken from a $1mm^2$ sample area.

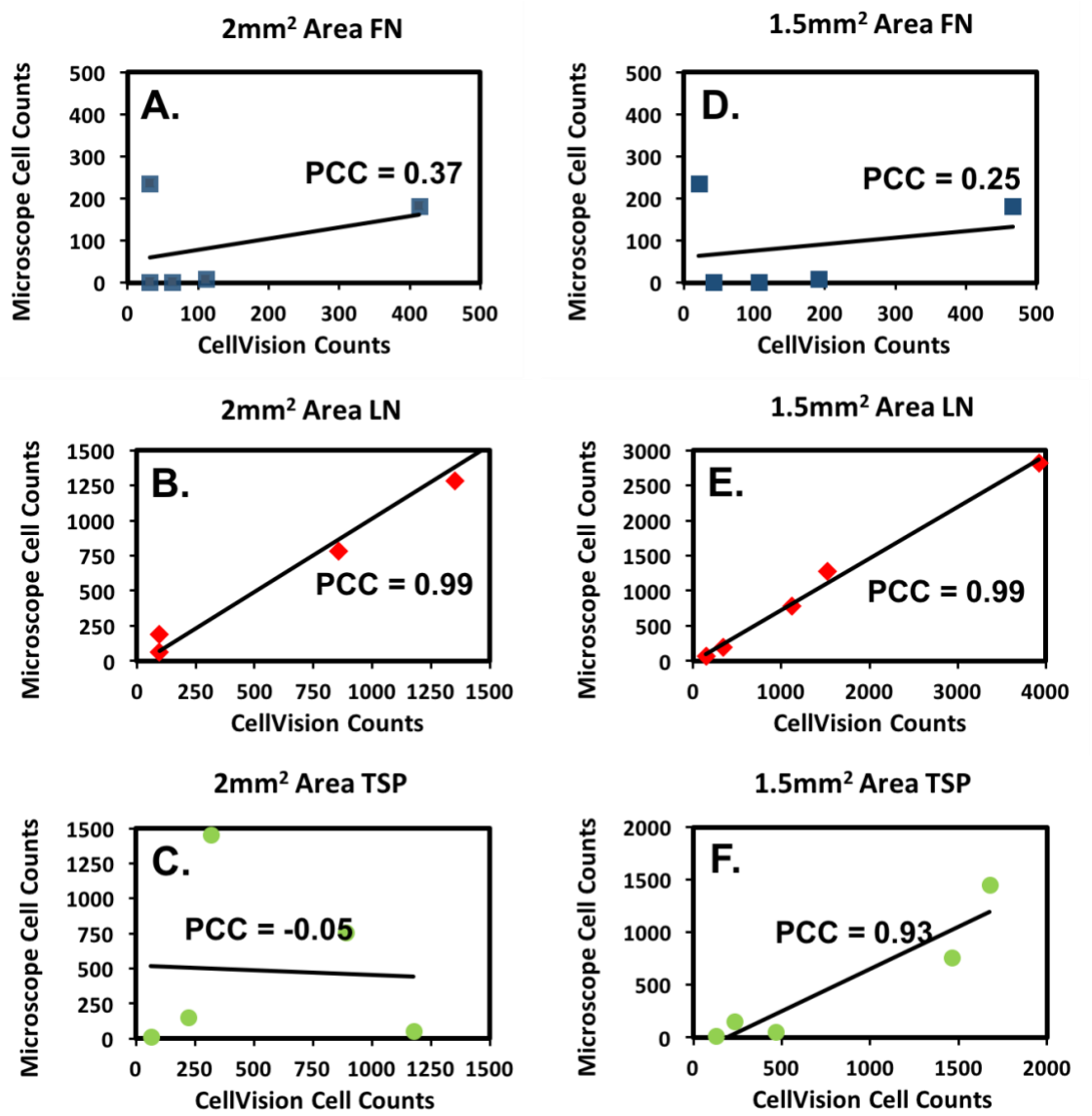


Figure 8: Scatter plots showing the correlations from larger sized FOVs with respective proteins.

iii) SCD POC Device Clinical Validation

Due to its promising in-lab results, our POC SCD device has been implemented in a research partnership with Dr. Deepa Manwani and the Sickle Cell Anemia Program at The Children's Hospital at Montefiore. We trained nurses on the functionality of the device and how to obtain cellular adhesion data with the device. We also provided an instructional video and written protocol (**Appendix 4**) on how to run the adhesion experiments with the POC system. This data is aiding Dr. Manwani's research group in

their studies aimed on the effects of anti-adhesion therapies on RBC adhesion in SCD patients. It was decided that Laminin coated channels would be used in the research at Montefiore because our studies have shown that Laminin is the most reliable protein to obtain data with our system. Laminin has demonstrated the ability to capture a wide range of adhesion levels. Our collaborators at the Children's Hospital collect consented patient blood samples and conduct experiments on their own SID system, supplied by CASE-BML. The Montefiore team then sends the blood sample to the CASE-BML team, who then analyzes the same sample with their microscope and their SID system and have compared the results (**Fig. 9)**

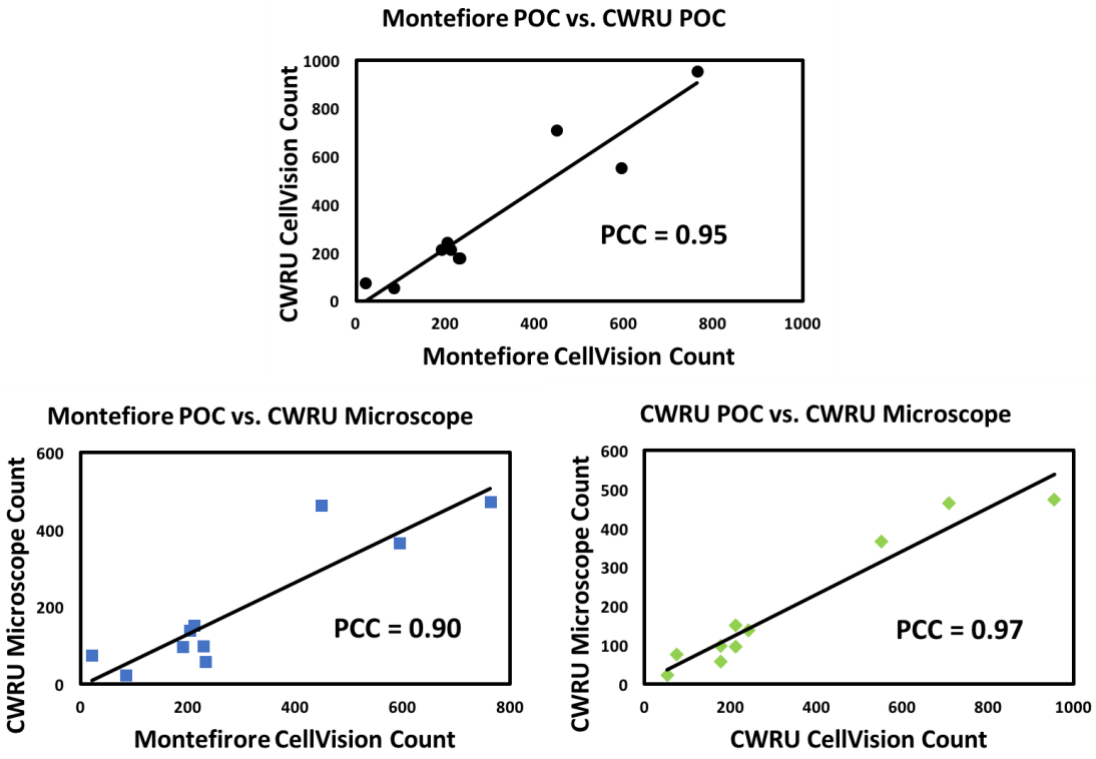


Figure 9: Scatter plots of cellular counts from the Smartphone system **A.** at both Montefiore and CWRU **B.** at Montefiore against CWRU Microscope **C**. at CWRU against CWRU Microscope. All data points are LN and of a 1mm^2 sample area.

Montefiore ships their samples to CASE-BML and tests the blood within 24 hours after is has been drawn. To study the effects that time has on the adhesive properties,

blood samples from two patients were taken and tested over 0, 24, 48, and 72 hours after the sample was obtained. After 72 hours patient A's blood experienced a 64% decrease in cellular adhesion over 72 hours. Similarly, patient B's blood experienced a 62% decrease in cellular adhesion over the same time span (**Fig. 10**). After time, both patients' blood clotted more easily, decreasing the number of cells that are exposed to the protein coated channels. The blood clotting can potentially explain decreasing cellular adhesion within the channels. In conclusion, this two patient study suggests that the blood we are receiving from the Children's Hospital is prone to less adhesion than freshly drawn blood and should be accounted for moving forward in the study.

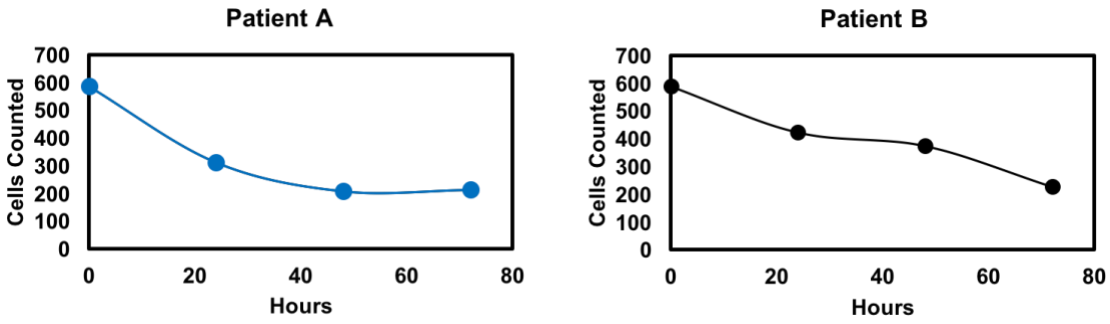


Figure 10: Data demonstrating that the cells exhibit less adhesion the longer the duration between blood draw and experimentation.

1.4 Discussion

i) SCD POC Device Laboratory Performance

There is currently no clinically relevant tool to evaluate disease severity for SCD patients. We have introduced a Smartphone imaging technology which provides an easy to use platform on which SCD patients could monitor their disease severity through the evaluation of RBC adhesion. RBC adhesion to FN, LN, and TSP was measured using our Smartphone imaging system with the SCD-biochip, at or above physiological flow shear stresses of post capillary venules (1–5 dyne/cm2) [**13, 22-24**]. These patient studies on LN, FN, and TSP have demonstrated that LN is the best protein to capture data with inside

the microchannels. LN experiments on SID held the highest correlation against our scanning microscope, with a PCC value of 0.96. A PCC value was calculated for the datasets as it demonstrates the linearity between the two variables, microscope cell counts and POC system cell counts. Also, LN channels showed the ability to capture a wide range of adhesion levels. From less than 200 cells to 2500 cells, the SID system can accurately determine the total number of cells that exist in the microchannels. It is important to be able to accurately capture low, medium, and high cellular adhesion because these different adhesion levels correspond to various disease states. Very high adhesion levels could relate to a patient being in crisis, when SCD patients have a very high likelihood of experiencing pain from numerous vascular occlusions. Whereas low cellular adhesion may relate to a patient's "baseline", or their disease in a relaxed state. These levels are also important to track so that patients and healthcare providers can record the progress of the disease after certain treatments, a longitudinal analysis.

We saw variable correlations across the 3 proteins and observed that LN had very homogenous adhesion along the entire channel, which improved the LN correlation to the microscope a great deal. FN and TSP had less homogenous adhesion tendencies. Also, for unknown reasons yet to be explored, some adhesion experiments completed with FN and TSP would result in an aggregation of RBCs within the channel. In such instances, CellVision would be unable to correctly predict the clotting. The clot could potentially be too large and would recognize the singular cells. Even if the app is capable of distinguishing singular cells, the app can only detect cells that are on the surface of the microchannel. In clots, there is an extra dimension of cells beneath the surface that our Smartphone and software combination would be unable to detect. The clotting in these experiments may be attributed to the platelets being activated and causing extra

adhesion, but it is currently unknown why platelets might be active in these particular adhesion experiments, but presents the opportunity for future investigation. For these reasons, it was decided that LN coated channel would be used in subsequent experiments and with our clinical partners.

ii) SCD POC Device Clinical Performance

To investigate the clinical feasibility of our POC device, we collaborated with Dr. Deepa Manwani and the Sickle Cell Anemia Program at the Children's hospital at Montefiore. Data (**Fig. 9**) is now being retrieved from Montefiore and the results look very promising. Initial data lacked accuracy as the team in Montefiore grew accustomed to handling the POC system and suggested valuable user feedback to improve the functionality of the device. Originally in the adhesion experiment, three syringes would be used, one for each channel. Feedback from the Montefiore team included that three syringes were a lot to handle. Therefore, one of the improvements was to incorporate a manifold that split blood flow from one syringe into three microchannels, instead of three different syringes attached to three microchannels. Also, to improve the accuracy of the results, for each patient sample three Laminin coated channels were analyzed. Also, three images were taken per channel in different locations and the sum of number of cells were averaged. This technique has significantly increased the accuracy of the study, the two POC systems are now recording very similar data with a PCC of 0.95. The POC system at Montefiore is retrieving values that have a PCC value of 0.90 to data points collected from the microscope at CWRU. The POC system located at CWRU is also now reporting very strongly correlated values to the microscope with a PCC of 0.97. Prior to the added images and added channels the data was unreliable, but by averaging many data points we have eliminated the effect of potential outliers and are seeing stronger correlations.

We expect that the data will continue to be promising as the Montefiore team becomes more comfortable operating the system.

The nurses at Montefiore are happy with the protocol changes that we have made. They have confirmed that the changes have had stark impact on their experience using our system. Most of the Montefiore team's concerns stemmed from steps from the adhesion experiment. The nurses have found that the POC system, meaning the optical attachment and software, are easy to use. Compared to the normal protocol using the microscope, the POC system addition is extremely user friendly.

1.5 Conclusion

Our Smartphone imaging system will allow for data retrieval at multiple research centers and in the field, resulting in faster turnaround time on research pertaining to cellular adhesion in SCD patients and contribute to the progression of microfluidic research. SID can only detect particles down to 1 μm in size, however RBCs range from 5-10 μm in size, therefor we can easily detect and distinguish adhered RBCs in our patient samples.

Preliminary designs and builds were tested for a next generation **(Fig. 11)** of SID. This iteration features a built-in programmable syringe pump. The syringe pump is driven off stepper motors, which are coupled to a lead screw that provides ample torque to drive the plunger along steel guide rods. The plunger is fit to the steel guide rods with a "slip" clearance, so that plunger can move freely. The pump is controlled by a control panel, featuring an LCD screen and button pad that communicates with an Arduino micro-controller. The microcontroller sends appropriate signals to the stepper motors through a stepper driver, controlling the rotational speed and therefore the translational plunger speed. Syringes are connected to the SCD Biochip and once blood and bluffer have been

successfully dispensed through the microchannels, the device's internal LED will illuminate the sample and the Smartphone will capture magnified images of the adhered cells within the SCD Biochip. A Smartphone application will then analyze the cellular images. The syringe pump is powered from a barrel jack that is externally accessible. Also, a USB A-B port is accessible to provide software upgrades to the Arduino. The housing for the imaging system and syringe pump comes in two halves and is secured together by a rugged clasp, ensuring the alignment of all critical components within the device.

The goal for a product like SID would be to enable the point of care at the patient home, but this largely depends on the types of anti-adhesion therapies and the status of the anti-adhesion therapies in the future. It may be more likely that the point of care will remain at the clinic or hospital, which will result in a different end design. This SCD POC system will help drive research efforts in the areas of anti-adhesion therapies and in microfluidics.

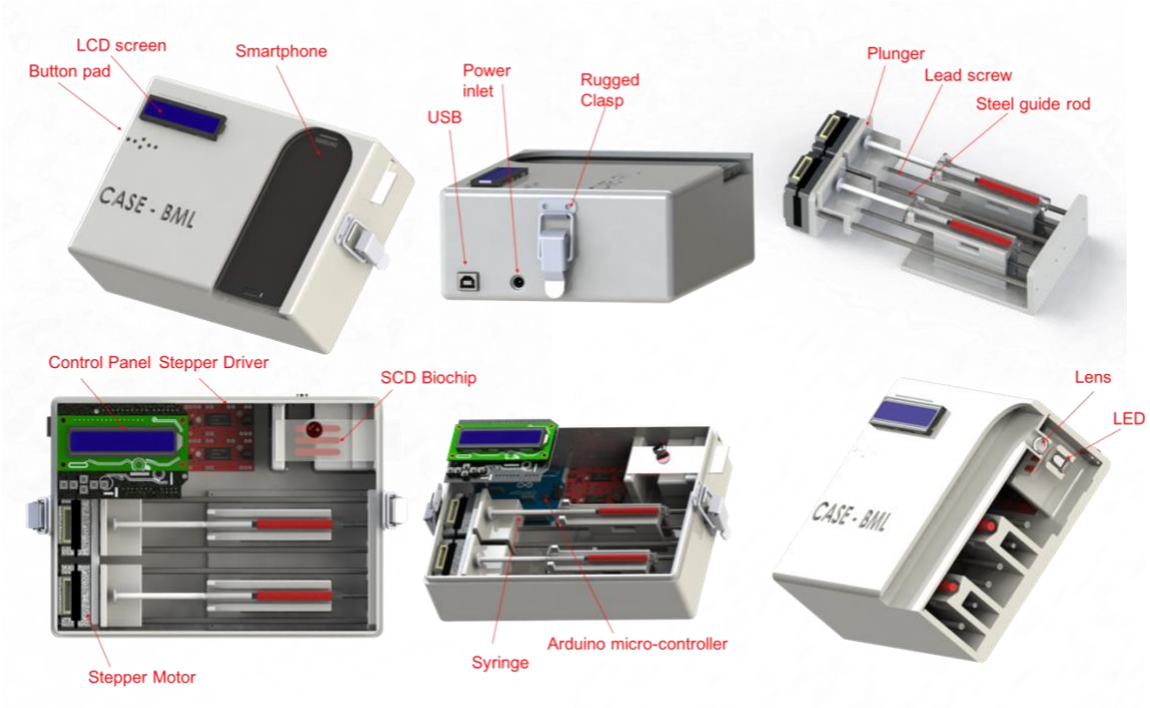


Figure 11: Images of the next generation SCD POC device, incorporating a programmable syringe pump.

2) Smartphone Fluorescent Imaging

2.1 Introduction

i) Mobile Fluorescent Imaging

The use of mobile imaging platforms has moved passed simple bright field illumination. Researchers are now pairing Smartphone technology with lasers, LEDs, and emission filters of certain wavelengths and have achieved the ability to excite fluorescent particles and capture the emission spectrum onto Smartphone camera lenses. For example, a team at the University of California at Los Angeles have incorporated fluorescent imaging features that have the ability to image nanoparticles and viruses tagged with fluorescent molecules [**25**]. Fluorescent microscopy is a heavily used tool among the scientific community which has allowed for the evaluation of biological fluorescent stains and proteins in modern diagnostics. A mobile POC device equipped with fluorescent imaging capabilities has many applications in the portable detection of biological matter labeled with fluorescent dyes.

ii) Oral Cancer

CIS, also known as oral intraepithelial carcinoma, of the oral cavity is a common cause of oral leukoplakia. If the cancerous lesions are not detected or biopsied, the cancer may metastasize to other organs. A late stage diagnosis is can be fatal or require expensive surgeries such as the removal of parts from the tongue, jaw, and cheek. The School of Dental Medicine at CWRU have identified that an over expression in the biomarker, hBD-3, exists in tumor cells in oral CIS lesions (**Fig. 12**) [**26**], while expression of the biomarker hBD-2 is absent. Furthermore, in healthy oral epithelia both biomarkers are expressed somewhat evenly.

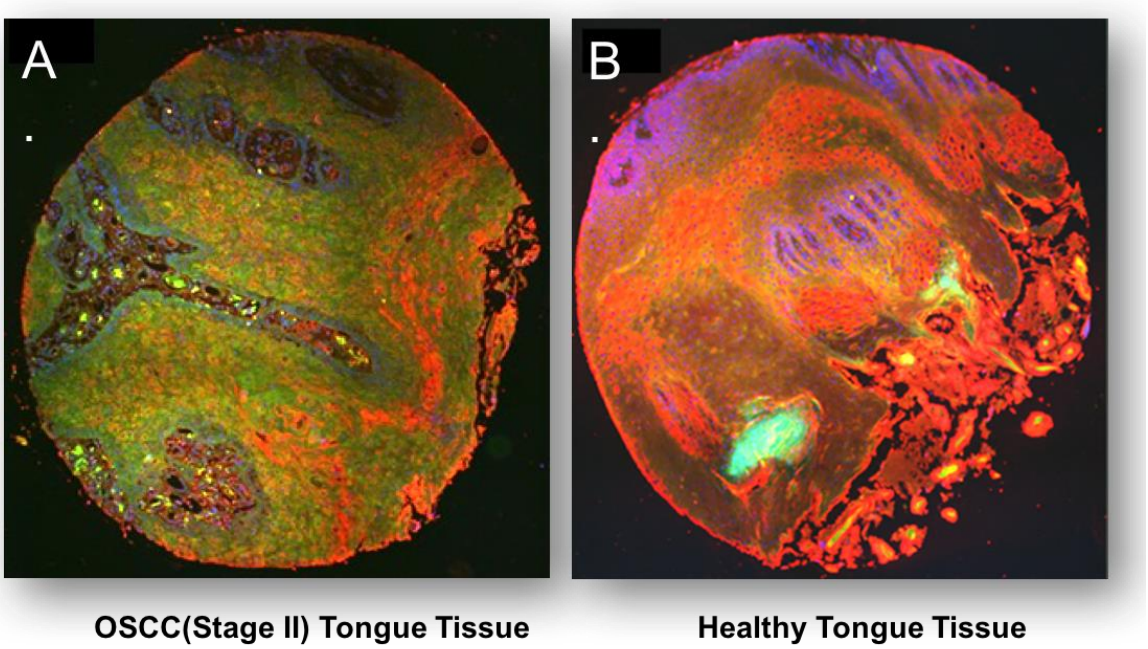


Figure 12: A. The over expression of hBD-3 (green fluorescence) in tongue tumor cells. **B.** Fluorescence from healthy tongue tissue. Figure from [**26**].

iii) Need and Approach

If detected early, an estimated 40,500 lives can be saved annually from oral cancer. The ability to diagnose a disease in an early state is possible by screening the general population during their normal check-ups. With oral cancer, early detecting is possible through the implementation of a POC device that can achieve a diagnosis while the patient attends a dental appointment as part of a normal screening procedure. Currently there is no POC device that can quickly and affordably diagnose lesions as cancerous. The Dental School at CWRU has published a method of identifying the overexpression of hBD-3 using ELISA [**26**]. Theoretically, identifying the abnormal hBD-3 expression by using immunofluorescence microscopy is possible. By labeling hBD-3 and hBD-2 with fluoresce dye-conjugated secondary proteins and measuring the intensity of the emission, a ratio of potential lesion fluorescence site over healthy epithelia fluorescence can prove the

existence of tumor cells. Smartphone imaging platforms have achieved the ability to perform fluorescent microscopy. Therefore, the aim of this project is to test the theory of immunofluorescence in oral cancer detection with microscope images and fabricate a proof of concept Smartphone imaging device that has the ability to capture fluorescent data. In the form of a fluorescent imaging device utilizing Smartphone technology, the hope is that FSID can achieve similar results compared to the trusted ELISA method with further development.

2.2 Materials and Methods

i) Materials

The FSID system **(Fig. 13)** is comprised of a microfluidic chip, a Samsung Galaxy S4, an optical Smartphone attachment fitted with fluorescent imaging capabilities, and a custom android application that digitally processes cellular images.

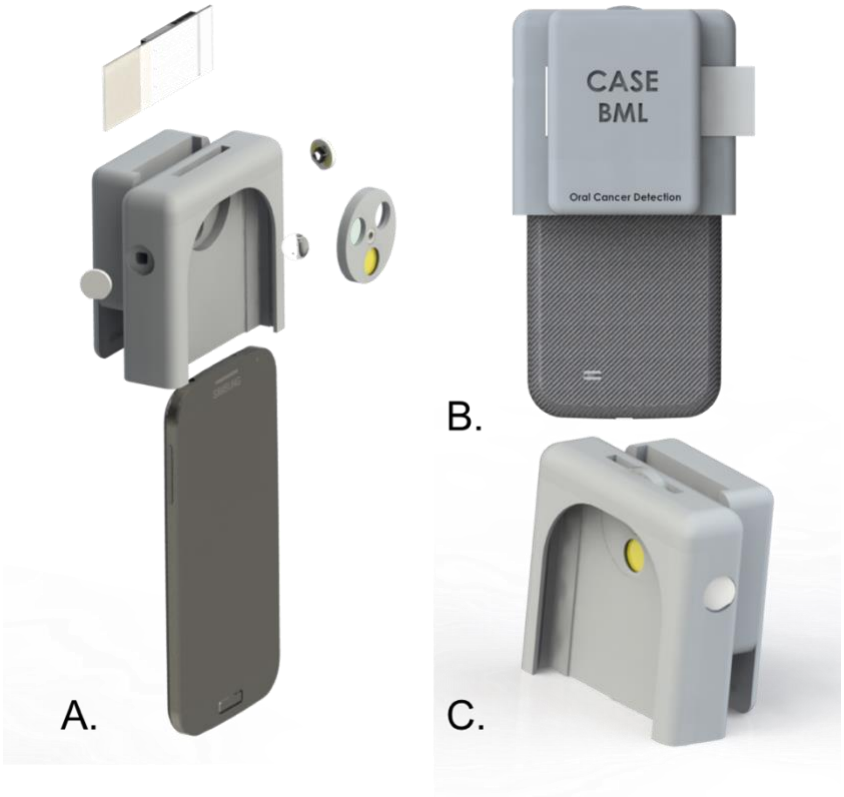


Figure 13: Various views of the FSID. **A.** Exploded view showing the various components, featuring a rotating filter wheel. **B.** Front view of the FSID. **C.** Rear view showing the placement of LED's and filter wheel within the assembly.

The FSID system's optical attachment incorporates a fluorescent imaging feature, where a 3v coin cell battery powers either a blue or red LED excitation source, controlled by an on-off-on toggle switch. These diodes illuminate stained particles from a 15° incidence angle, which reduces extra noise [**25**]. Emission filters are used to filter out noise and only transmit a specific range of wavelengths of emission to the Smartphone camera. The light is collected onto an external ball lens with a diameter of 10mm and received on the camera lens of the Smartphone. For specific details regarding the optical components, please refer to **Table 1**. The light from the Smartphone lens is submitted to the embedded CMOS sensor chip that creates the image of cells to display on the Smartphone screen. The optical design is demonstrated in **Figure 14**, which was

modeled for the specific use of certain conjugated proteins in the experiments which can be viewed in a spectra-viewer (**Fig. 15**).

Company	Cat. Number	Item	Additional Information
Edmund Optics	#64-621	675nm emission filter	Longpass filter
Edmund Optics	#67-017	535 CWL filter	Fluorescence filter, 43nm bandwidth
Edmund Optics	#32-784	Lens	Ball lens, 10mm diameter
Chanzon	B01AUI4WC8	Red LED	620-625nm wavelength
Chanzon	B01AUI4VYW	Blue LED	450-455nm wavelength

Table 1: Optical components in FSID

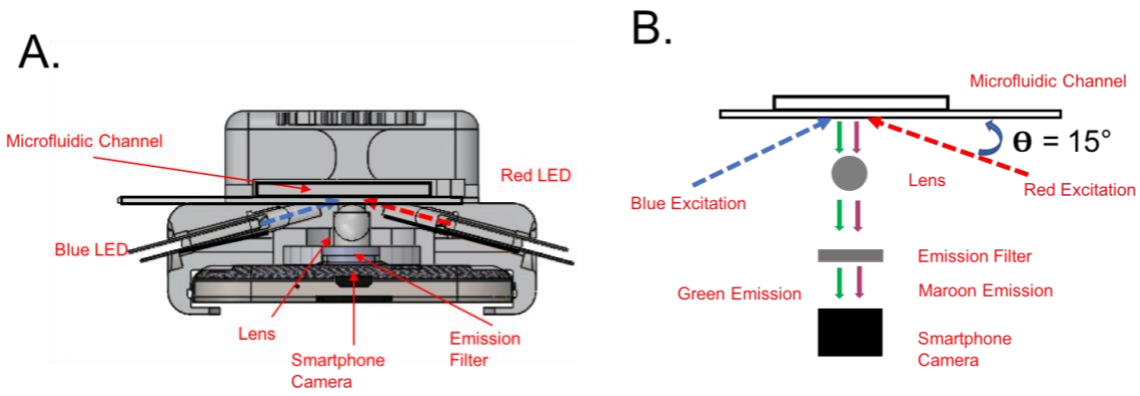


Figure 14: A. Oral Cancer POC top-cross section view constructed from **B.** initial optical schematics.

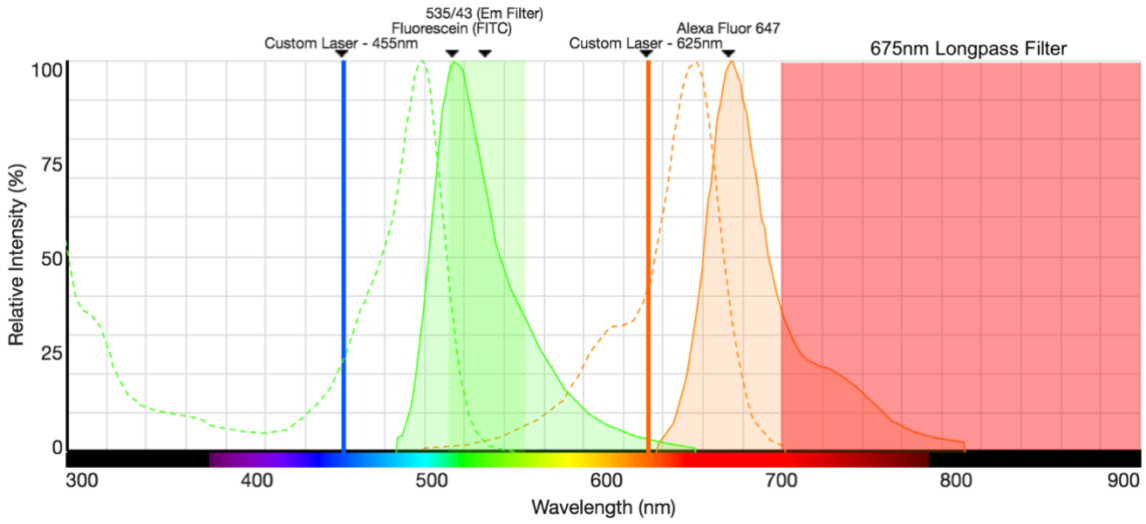


Figure 15: Spectra-viewer from ThermoFischer Scientific, showing the excitation ranges/sources for the two conjugated protein stains used on hBD2 and hBD3. Also, the spectra-viewer shows the emission ranges and filters used in our mobile POC device.

ii) Oral Cancer Device Fabrication and Assembly

The 3D optical attachment (**Fig. 13**) was designed using a 3D modeling software (SolidWorks) and built by a 3D printer (Fortus 400mc) in think[box] at CWRU. All fabrication on the 3D printed housing was also conducted in think[box], including milling, soldering, and assembly of optics and circuitry components. Our optical attachment is compatible with a Samsung Galaxy S4 for oral cancer screening.

iii) Sample Preparation

Staining Solution

To prepare the staining solution, 1µL of triton-x (i.e. 0.1%) was mixed in 1 mL PBS. Next, 10 µL of hBD3 protein and 10 µL of hBD2 protein were added to the triton and PBS mixture. Lastly, the tube was covered with foil.

Cell Collection

Cytobrush samples from consented patients are collected into separate receptacles ("left" and "right" for healthy individuals; "lesional" and "normal" site for patients) with the brushes immersed in 500 µL of fluid. The tubes where gently shaken to release cells from the teeth of the brush. The tubes were centrifuged for 30 seconds at 6000 rpm to extract the palettes. The supernatant fluid was carefully aspirated, freeing the palette. 250 µL of the staining solution was added to each tube. The tubes where wrapped in foil and gently shaken for 30 minutes on a shaker set to 20 rpm.

Chip Preparation

The microfluidic chip used in our experiments is a slight variation of the SCD Biochip, with two non-functionalized microchannels. To review the fabrication of the

SCD Biochip please refer to the cited work. Once the channels are washed with PBS and ethanol, 30 µL of the stained cell solution was injected into corresponding channels ("left" and "right" for healthy individuals; "lesional" and "normal" site for patients).

iv) Experimental Design

Healthy individual control samples and suspicious lesion patient samples were collected from patients, given patient identification numbers so that the identities are kept classified, and fluorescently imaged in the FSID (**Fig. 16**) using the custom Oral Cancer Detection android application to capture data. The oral cancer team is blind, meaning that we do not know whether or not the suspicious lesion samples are in fact cancerous. The same samples were placed on an Olympus IX83 microscope and microscopic images where uploaded to the Oral Cancer Detection application. BDI data is collected and added to a table and stored in a secured online folder. Sample microscope images of healthy cells and cancerous samples are present in **Figure 27.** The CWRU School of Dentistry takes the same patient sample and tests it using their ELISA diagnostic method to determine whether or not the sample is cancerous. Data from all three experiments were collected to compare the microfluidic and fluorescent imaging detection methods to the already existing ELISA detection method to determine the feasibility of a portable oral cancer detection system using fluorescent imaging.

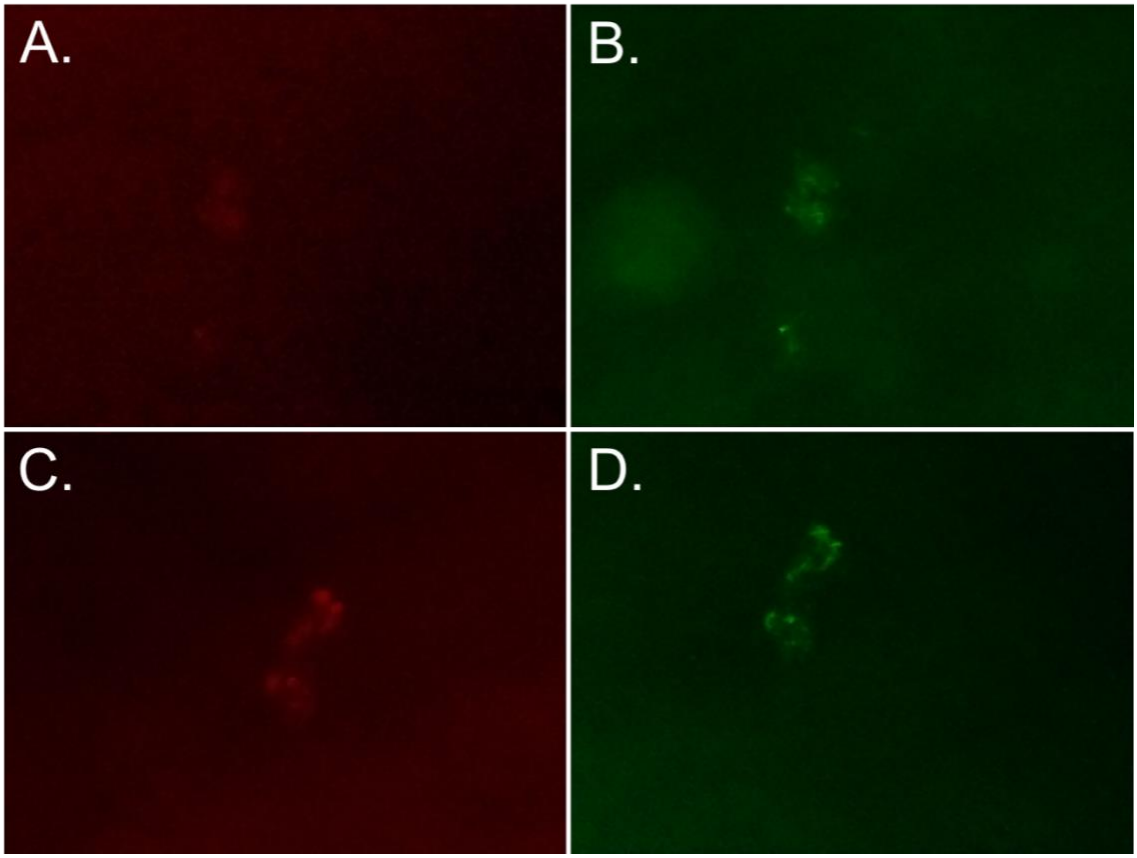


Figure 16: Fluorescent images taken on the FSID of a control patient sample (**A, C**) hBD-2, and (**B, D**) hBD-3.

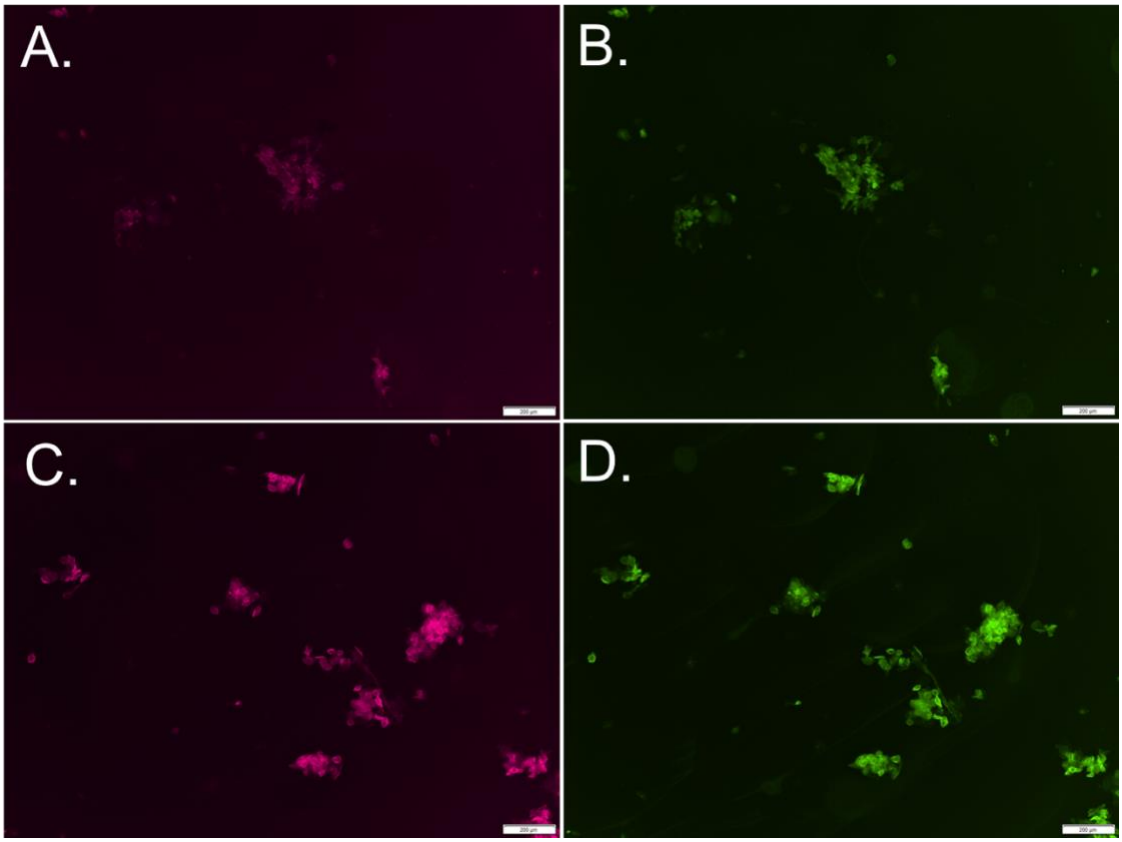


Figure 17: Fluorescent images of **A.** hBD-2 of potential cancerous cells, **B.** hBD-3 of potential cancerous cells, **C.** hBD-2 of healthy cells and **D.** hBD-3 of healthy cells.

v) Automated Image Processing

A custom Android application (**Fig. 18**) was created by Jonathan Koss, to capture images from the microfluidic chip and report the BDI score of a patient (lesion). The app has one interactive screen with two sections. To calculate the BDI, two fluorescent images that capture the intensity of the fluorescence emitted from the labeled hBD-2 and hBD-3 biomarkers are obtained from each of the two microchannels (left; normal epithelia and right; cancerous lesion), for a total of four images. The top section contains four quadrants (one for each of the four required images) which each contain a thumbnail square to display the image as well as two buttons to either capture an image or load one already saved to the phone. The bottom section contains a button to calculate the BDI which can

only be done when all four images have been filled. After the button is pressed, the score is displayed in this area.

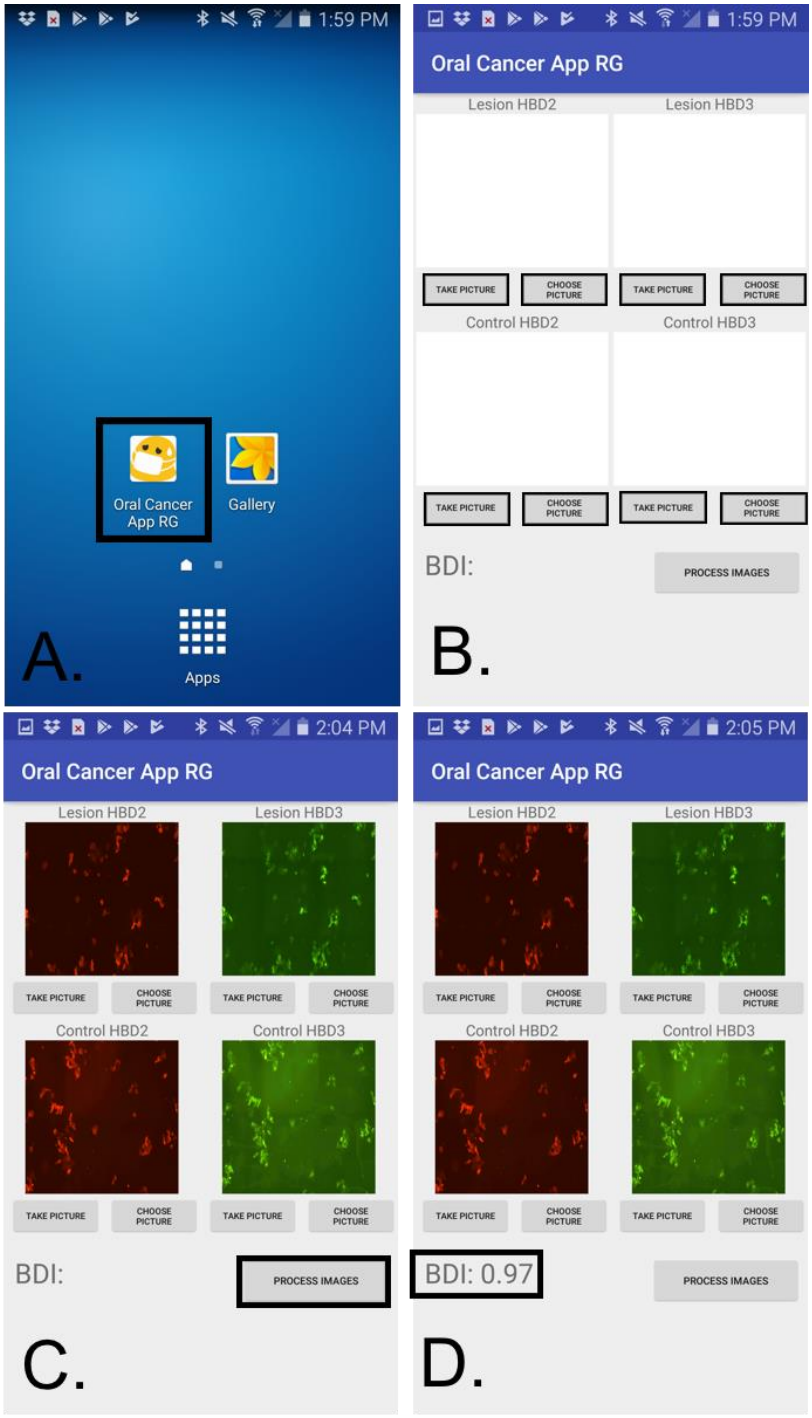


Figure 18: Work flow of the Oral Cancer Detection android application. **A.** Locating the application on the home screen. **B.** The home screen opens, either take or chose an image for each protein/location. **C.** Click the "Process Image" button that determines fluorescence intensity. **D.** BDI is displayed.

First, the concentration of the biomarkers is measured from each image. For each protein, the intensity of a given fluorescence (red, green, blue) must be calculated from the images. Each image is represented by a matrix for each color (red, green, blue) where each value (0-255) is the intensity of that color for a specific pixel for that given picture. The intensity for an entire picture is calculated by summing the matrix that represents the specific color fluorescence from that image. Since all images are the same number of pixels and only the relative values matter, this sum does not need to be scaled. Work flow for the oral cancer detection app is presented in **Figure 19**.

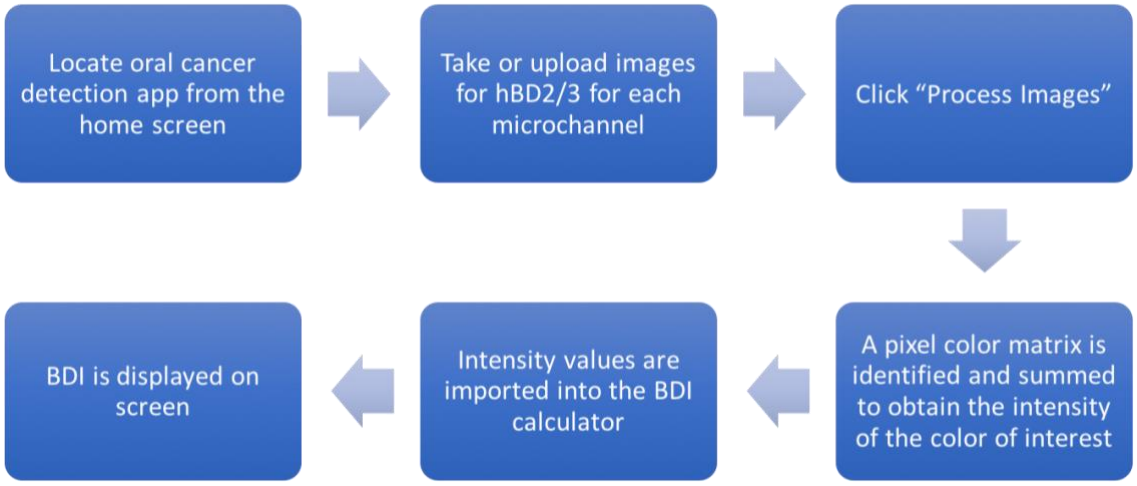


Figure 19: Oral cancer detection application workflow.

The biomarkers can be labeled with multiple different fluorescent stains that emit different colors, so depending on what stains are used, a different version of the app could be used to identify different color combinations. Then, the ratio of hBD-3/hBD-2 for the left and right channels must be calculated. BDI is then equal to the left-ratio/right-ratio

(**Fig. 20**). A BDI of over 1 signifies potential cancerous lesions, whereas a BDI of ~ 1 signifies healthily epithelial.

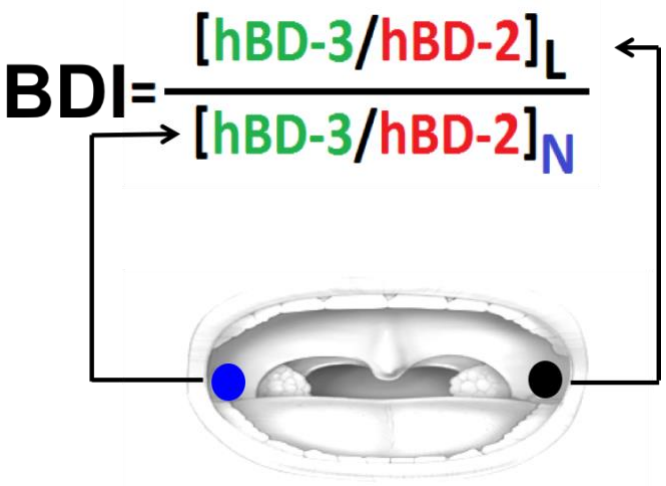


Figure 20: Equation for the BDI calculation.

2.3 Results

i) Oral Cancer POC Design

The Oral Cancer POC attachment housing was 3D printed on a Fortus 400mc printer in CWRU's think[box]. The LED's, filters, toggle switch, filter wheels were all hand assembled and machined as necessary and installed into the device. This fluorescent Smartphone imaging system shows the potential for mobile Oral Cancer screening using immunofluorescent imaging. This device was also cost efficient, effectively costing only $131.68 to create a functional prototype (**Appendix 2**). For $131.68 our mobile fluorescent imaging system can easily detect particles down to 1 μm in diameter (**Fig. 21**).

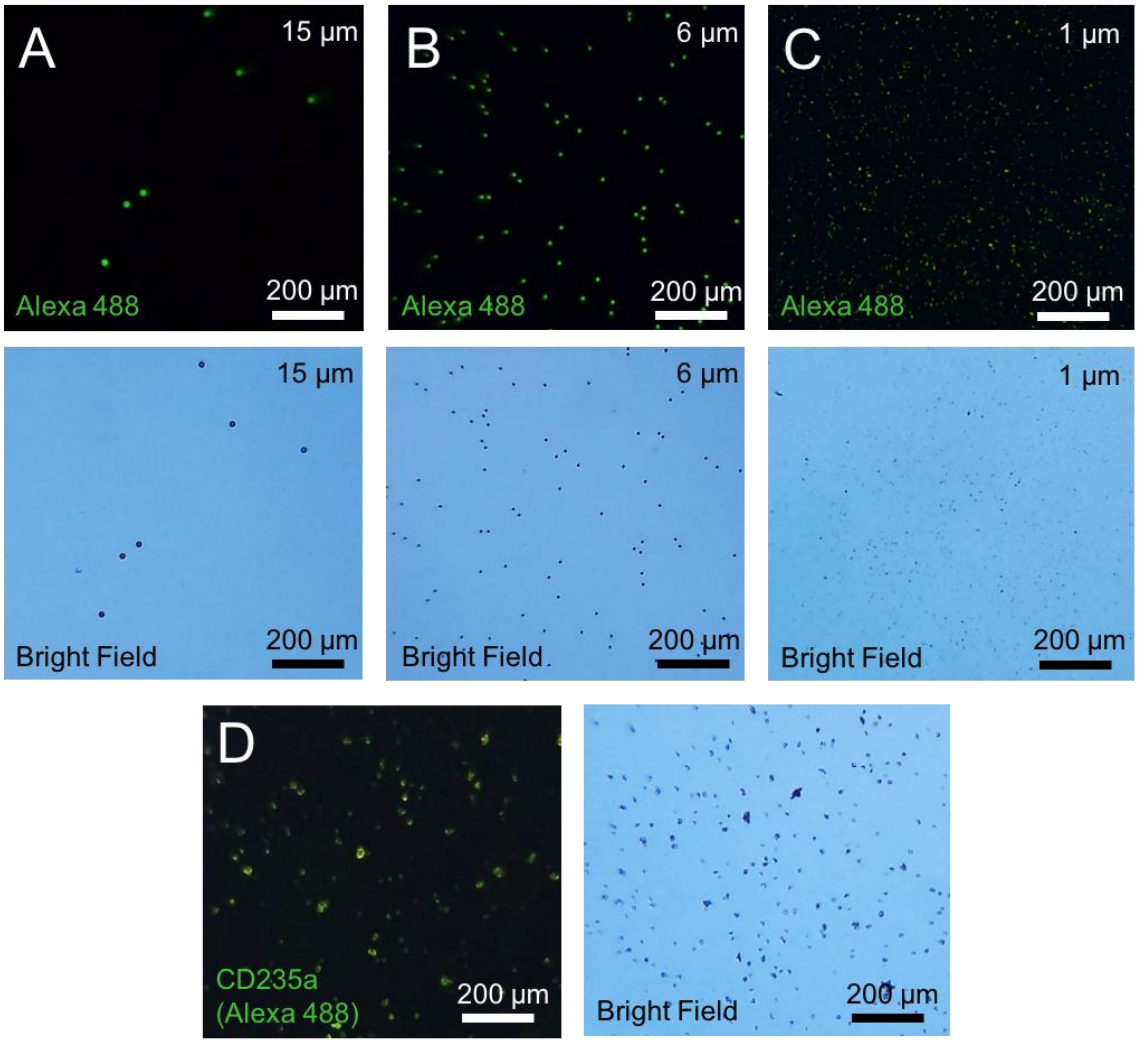


Figure 21: Our POC system has the ability to image various sized fluorescent particles. Here are Alexa 488 labeled **A.** 15um diameter silica beads **B.** 6um diameter silica beads **C.** 1um diameter silica beads **D.** Epithelia cells.

ii) Oral Cancer POC Device Validation

The FSID was used to capture florescent images of cheek cells from consented control (healthy) samples and consented cancerous patient samples. Fluorescent mages where also obtained from a scanning microscope from each sample. Both types of images where imported to an android application that determines the intensity of the emission fluorescence from both hBD-2 and hBD-3 conjugated proteins. The results from the experiments are shown in **Table 2**.

Sample Category	Sample Number	Swab Type	Test Date	Microscope BDI	Cell Phone BDI
Control	1	Cheek	10/31/2017	4.39	0.51
Control	2	Cheek	10/31/2017	1.08	1.94
Control	3	Cheek	10/31/2017	0.36	1.31
Control	4	Cheek	10/31/2017	0.61	0.98
Control	5	Cheek	10/31/2017	1.55	Poor imaging
Suspicious Lesion	Z-01-A-057	Cheek	11/2/2017	0.98	1.57
Control	6	Tongue	11/7/2017	0.05	0.91
Control	7	Tongue	11/7/2017	0.61	1.44
Control	8	Tongue	11/7/2017	0.08	1.27
Control	9	Tongue	11/7/2017	0.48	Poor imaging
Control	10	Tongue	11/7/2017	1.09	1.35
Suspicious Lesion	Z-01-A-059	Floor of the Mouth	11/9/2017	0.84	1.72

Table 2: Summary of experiments comparing BDI's from the microscope images and images obtained from FSID.

2.4 Discussion

i) Oral Cancer POC Device Performance

This POC device is in its very early stages, however it has already shown some promising results. FSID reported expected BDI values for multiple healthy patient samples (~1) and the two lesion samples (>1). FSID would be more reliable with more development and stable optics that could obtain more clear images. FSID was able to capture fluorescent emission in all but two patients, and with fine tuning of the optics the device could capture accurate BDI ratings and reflect a diagnosis.

2.5 Conclusion

This POC device, FSID, has proven its ability to capture fluorescence from hBD-2 and hBD-3 fluorescence from conjugated proteins and shows promise in calculating right BDI range based on control or diseased samples. The FSID system can only detect fluorescent particles down to 1 μm in size, however epithelial cells range from 25-100 μm in size, therefor the FSID system vastly surpasses the optical requirements for this type of imaging. More development is needed to stabilize optics to obtain clearer images. Therefor future work is aimed at progressing image clarity with FSID. The microscope also shows some promising results. However, it was quickly investigated that

some of the extremely low BDI values are a result of background noise. Any extra pixels in the background can drastically alter the BDI. Images that were clean reported nice accurate BDI ranges. To avoid detecting background noise in both microscope and POC system, it may be possible to implement the blob detection functionality from the SCD POC system's software and apply the code to the fluorescent images. If the blob detection software can also determine the shape of cheek cells, it may allow the software to just detect pixel color matrices from the fluorescence from just the cell. Our microfluidic channels are made out of PMMA. PMMA has a fluorescence spectrum which could be another cause for the background noise in both microscope and POC system images. Studies show that photobleaching plastic prior to the use in imaging can reduce background noise in fluorescent images. Another possibility is to explore plastics that have lower fluoresce levels in the design of our microfluidic systems, for example PDMS has a lower autofluoresence than PMMA and could lower or eliminate the background noise [27].

3) Conclusion

Presented here is a clinically appropriate, portable imaging device that captures cellular images of microfluidic channels, fitted with bright field and fluorescent imaging features. The device has shown promise in determining SCD severity through RBC adhesion analysis in its clinical validation, capturing fluorescence from stained oral cancer patient samples, and reporting correct intensity levels.

These types of devices may have other uses outside of the medical field. Hand held imaging systems have many uses in many disciplines. For example, these imaging systems could be used in material science as well. Insert a fractured material sample into SID and the magnification would potentially allow the user to observe fracture sites and with the help of a slightly different software package, perhaps it could decipher between fracture types. There is a large field of Smartphone imaging technology in today's research community, a review of the technology can be found in **Table 3,** ranging from detecting fluoride in water to imaging the human iris.

Application	Light Source	Optical Components	Type of Phone	Image Processing	Detection Limit	Reference
E. Coli detection in water	UV LED	External lens	Sony-Ericsson U10i Aino	External, ImageJ	~5 to 10 cfu mL^{-1}	[35]
rbST antibody detection in milk	UV LED, white LED	External lens, longpass optical filter	Samsung Galaxy SII	No	625 nm	[30]
Lactose/galactose detection in food samples	Blue light	Blue light optical filter	iPhone 4	External, ImageJ	0.01 mM	[32]
Fluoride detection in water	Smartphone LED	Light chamber	Asus Zenfone, Moto G, Samsung DUOS	No	0.2 ppm	[29]
Catechol detection in water	White LED	NA	Samsung Galaxy A8	Yes	NA	[33]
Antibiotic residue detection in milk	Fluorescent lamp	NA	iPhone 5s	YES	0.5 µg mL^{-1}	[31]
Aflatoxin B1 detection in maize	White LED	External lens	Samsung Galaxy S2	No	5 µg/kg	[28]
Nanoparticles & Virus detection	Blue laser	Interference optical filter, external lens	Nokia PureView 808	No	100 nm	[25]
Quantification of Giardia lamblia cysts	Blue LEDs	External lens, emission filter, excitation filter	Nokia Lumia 1020	Yes	4 µm	[36]
Magnetic levitation for measuring densities in microspheres	White LED	External lens	Samsung Galaxy s4	Yes	5 µm	[1]
Detection of a specific miRNA sequence	Green laser	Diffraction grating, collimator, external lens, collecting lens,	iPhone 4	Yes	>100 nm	[34]
Detecting clinically active trachoma	External flashlight	External Lens	iPhone 4	No	NA	[37]
Detecting quantum dots and nanoparticles	Blue laser diode	External Lens, linear polarizer, bandpass filter	Nokia Lumia 1020	Yes	50 nm	[38]
Detection of seborrheic dermatitis on the scalp	White LED	Diced linear variable filters, external CMOS sensor	Samsung Galaxy 7 edge	No	NA	[39]
Optic disk imaging	Smartphone flash	Prism	Samsung SIII GT-I9300	No	NA	[40]
Fluorescent imaging of quantum dots	Red LED, blue LED, white LED	Lens, filters	NA	External, ImageJ	1 µm	[41]
Detecting fluorescent beads	Blue LED	Color filter	Sony-Ericsson U10i Aino	External	10 µm	[42]
Lens-less imaging	Ambient illumination	Infrared filter	Samsung Galaxy S3	Yes	500 nm	[43]
Global health applications	Blue LED	Objective lens, emission filter, excitation filter, condenser lens, collector lens	Nokia N73	Yes	400 nm	[44]
Rapid Diagnostic Test reader	White LED	Plano convex-lens,	Samsung Galaxy s2	Yes	~1-2 µm	[45]
Blood pathogen detection	Smartphone flash	Lens, mirror	iPhone 4	External	1 pg/mL	[46]
Lens-less microscopy	White LED	Chip	MotoZine ZN5	External	2 µm	[47]
Photoplethysmography	Smartphone LED	Band-pass filter	Nokia model E63)	No	NA	[48]
Fluorimeter for pH Measurements of Environmental Water	White LED	Color filter	Kogan Agora 8	No	NA	[49]
Visible iris recognition	Near-infra-red	NA	Iphone 5s, Nokia Lumia 1020	Yes	NA	[50]
Adhesion analysis in SCD	White LED	Diffuser, Lens	Samsung Galaxy s4	Yes	1 µm	NA
Oral Cancer detection	Blue LED, Red LED	Long-pass filter, fluorescent filter	Samsung Galaxy s$	Yes	1 µm	NA

Table 3: A review of current Smartphone imaging technology

Currently our POC devices are limited to the Samsung Galaxy s4 and newer models. Whereas some Smartphone imaging technologies currently available are compatible with other types of Smartphones. Our designs can be easily modified for a "puzzle piece" design in which you can attach different cases to the Smartphone of choice and the external features of the case would fit the features on the optical attachment. Another limitation of our imaging system is that we can only image particles down to 1 μm in size, while other technologies have the ability to detect nanometer sized particles. For our application, a 1 μm detection limit is acceptable because we are imaging cells that are greater than 5 μm.

Despite the limitations of the two POC systems presented here, SID and FSID fill a gap in the current technologies by providing a full system, including imaging devices and automated image processing applications for SCD and oral cancer. The oral cancer POC device is the only device that incorporates a filter wheel to easily switch between filters and different color excitation sources, for fluorescent imaging of cells, that has the ability to do in house image processing. While the SCD POC device is the only bright field imaging device that is used in the imaging of RBCs in microfluidic channels.

Appendix

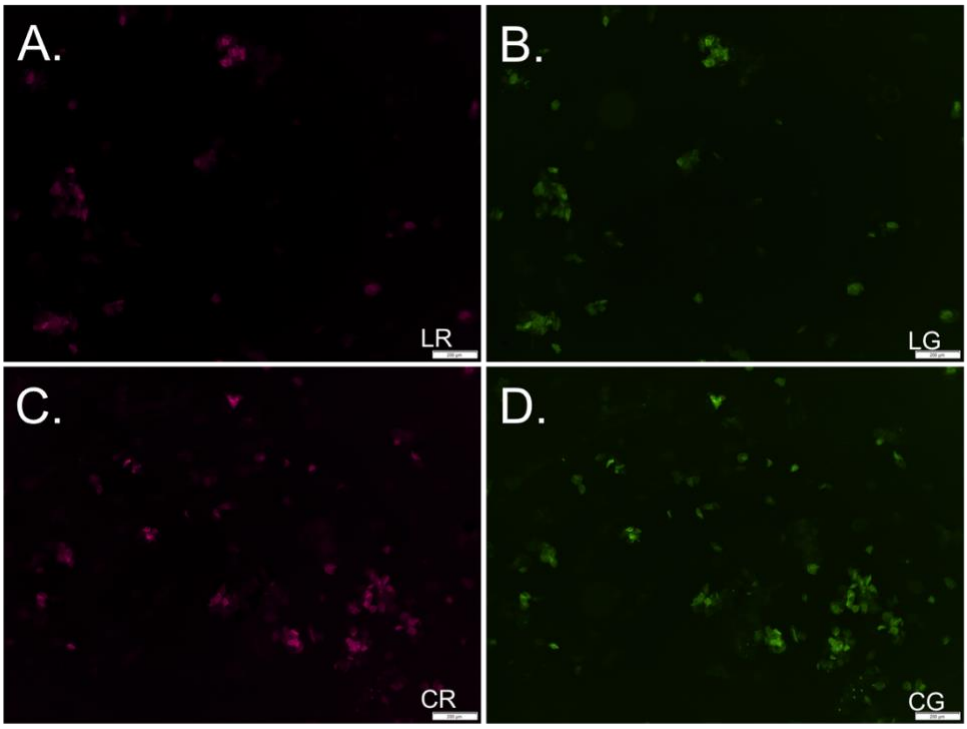
Appendix 1. SCD Device Material Cost

Company	Cat. Number	Item	Cost/Exp.
CWRU think[box]	N/A	3D Printed Housing	$80
Amazon (Addicore)	0885623082493	Bright White LED	$0.10
Amazon (Parts Express)	0844632012549	100 Ohm Resistor	$0.63
Amazon (Uxcell)	0700955171571	Slide Switch	$0.22
Edmund Optics	32-748	10mm Ball Lens	$32
Edmund Optics	DG05-1500	Glass Diffuser	$17
Mouser	658-CR2032	2032 Coin Battery	$0.33
TE Connectivity	120591-1	Battery Holder	$1.40
		Total Material Cost:	$131.68

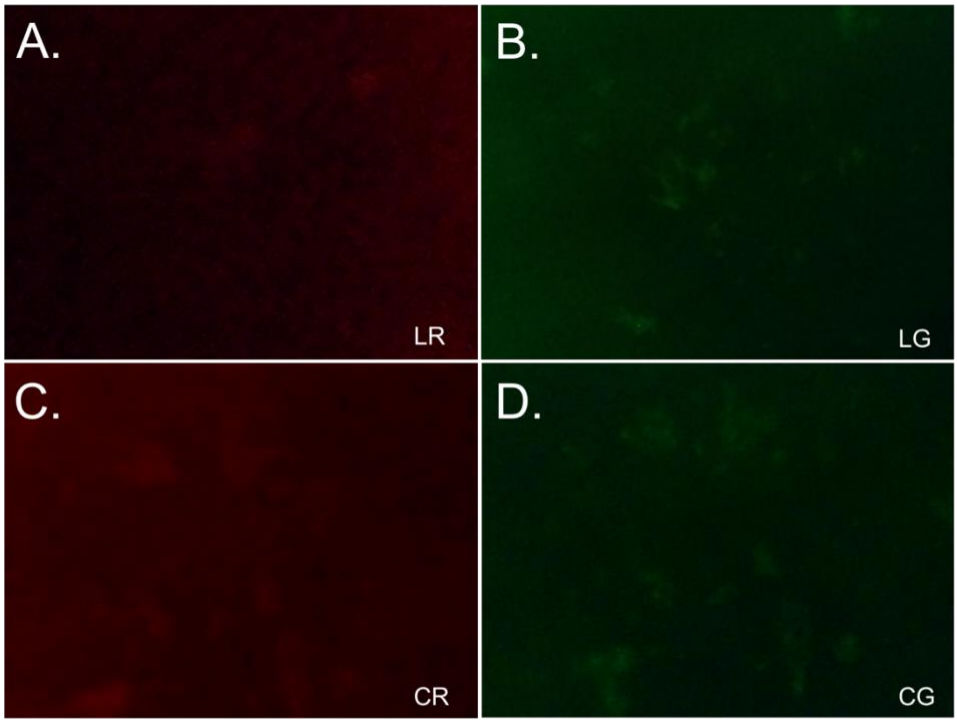
Appendix 2. Oral Cancer Device Material Cost

Company	Cat. Number	Item	Cost/Chip
CWRU think[box]	N/A	3D Printed Housing	$80
CWRU think[box]	N/A	3D Printed Filter Wheel	$0.50
Chanzon	B01AUI4WC8	Super Bright Red LED	$0.65
Chanzon	B01AUI4VYW	Super Bright Blue LED	$0.66
Edmund Optics	64-621	Longpass Filter	$89
Edmund Optics	67-017	Fluorescence Filter	$185
Edmund Optics	32-748	10mm Ball Lens	$32
Mouser	658-CR2032	2032 Coin Battery	$0.33
TE Connectivity	120591-1	Battery Holder	$1.40
McMASTER-CARR	5972K197	Bearing	$15.62
McMASTER-CARR	8116K34	Steel Rod	$1.59
UXCELL	s12101100am0009	on-off-on Toggle Switch	$1.59
		Total Material Cost	$329.14

Appendix 3. BDI Experiments

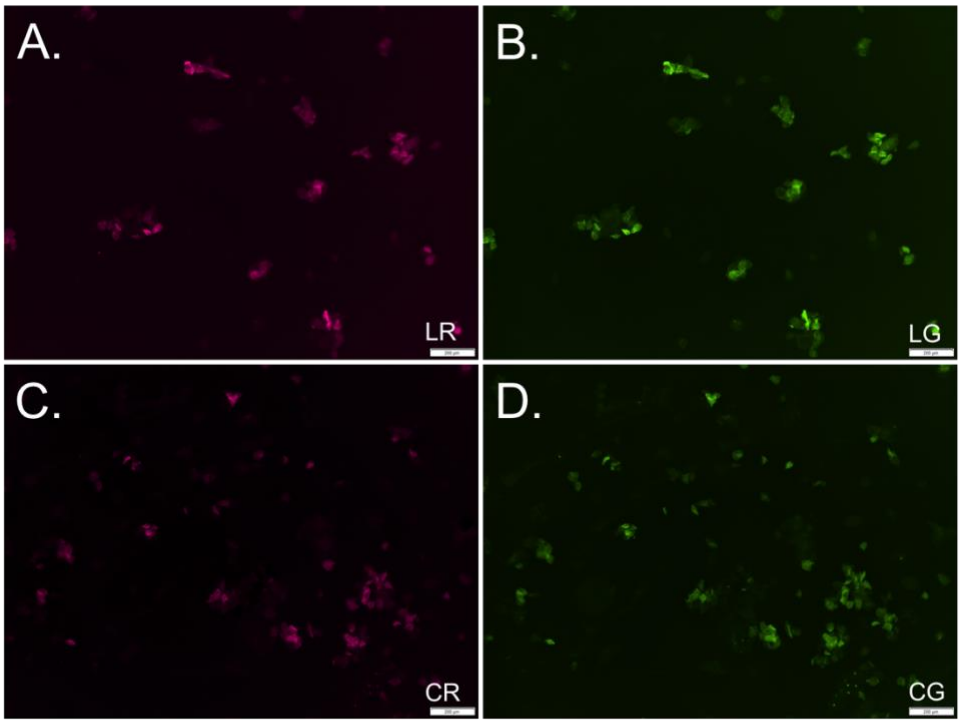


BDI = 4.39

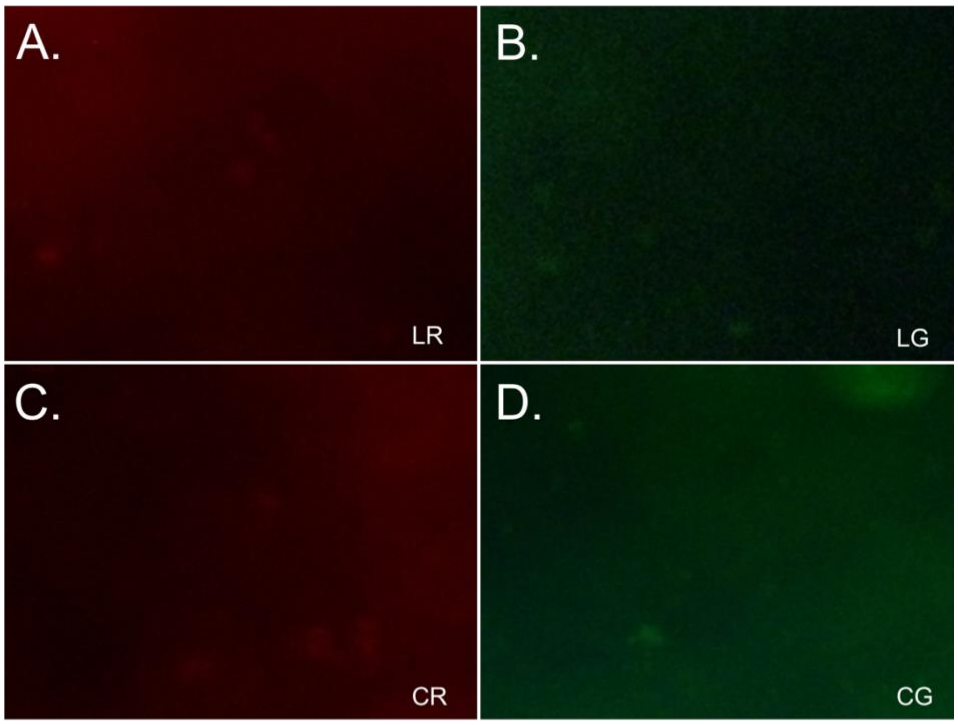


BDI = 0.51

Microscope and POC BDI Control Sample 1

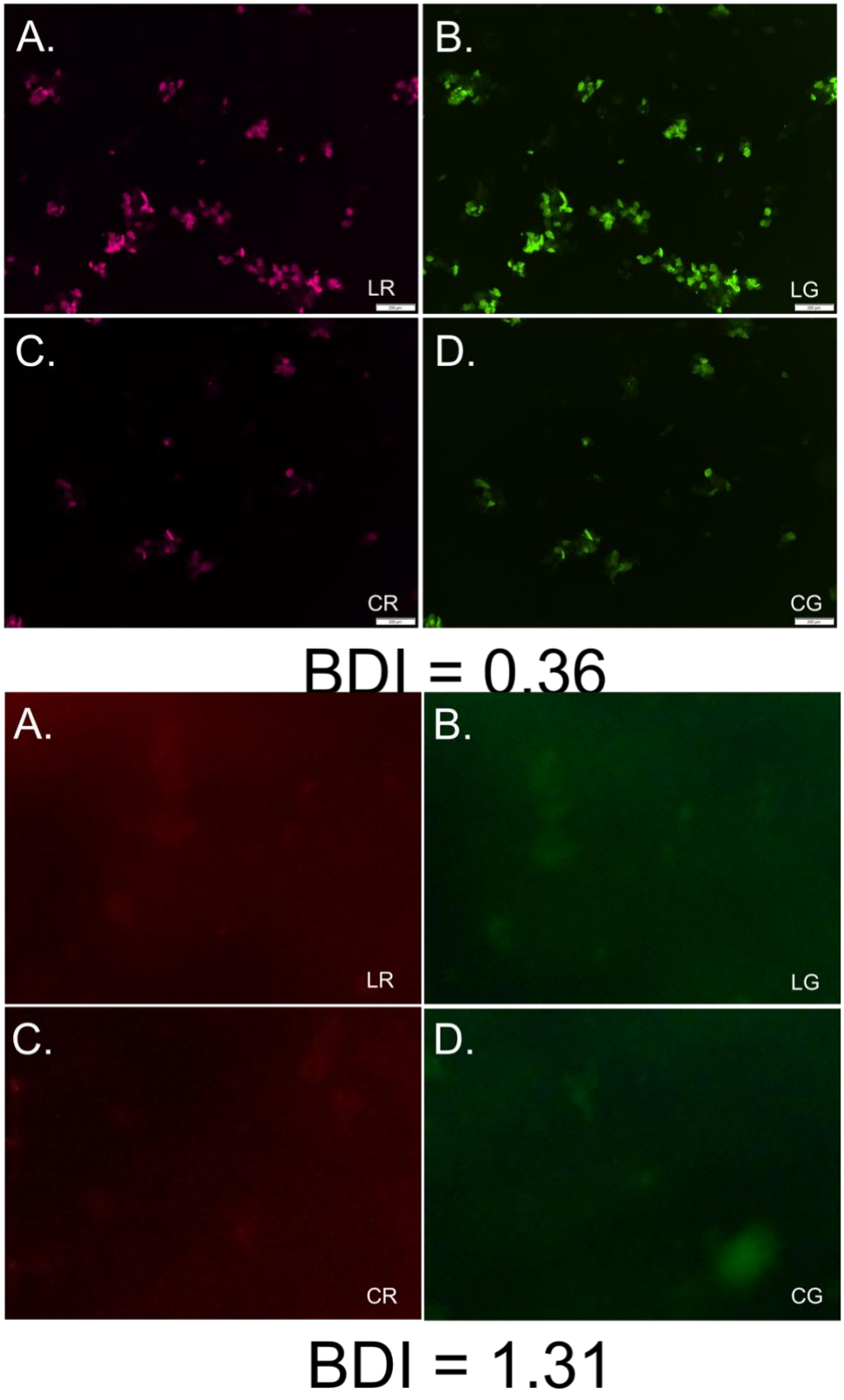


BDI = 1.08

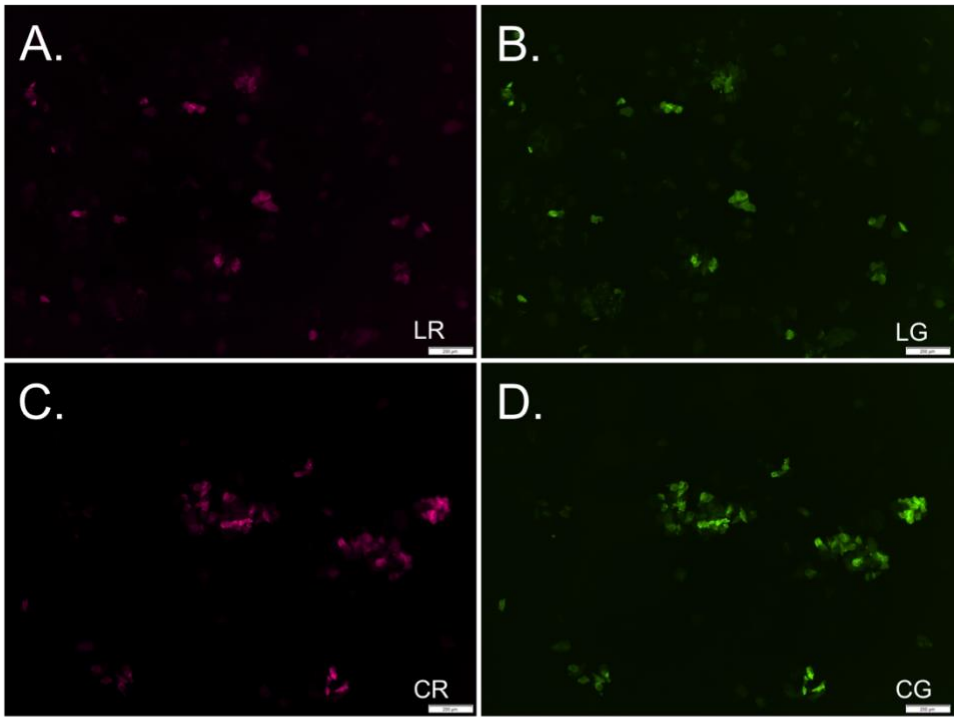


BDI = 1.94

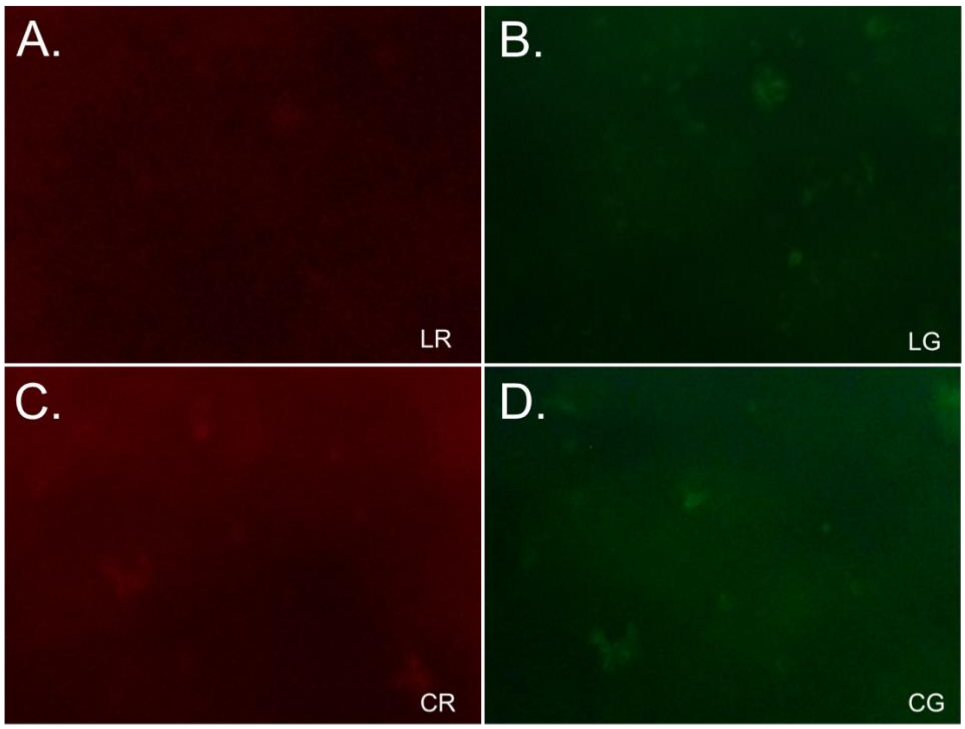
Microscope and POC BDI Control Sample 2



Microscope and POC BDI Control Sample 3

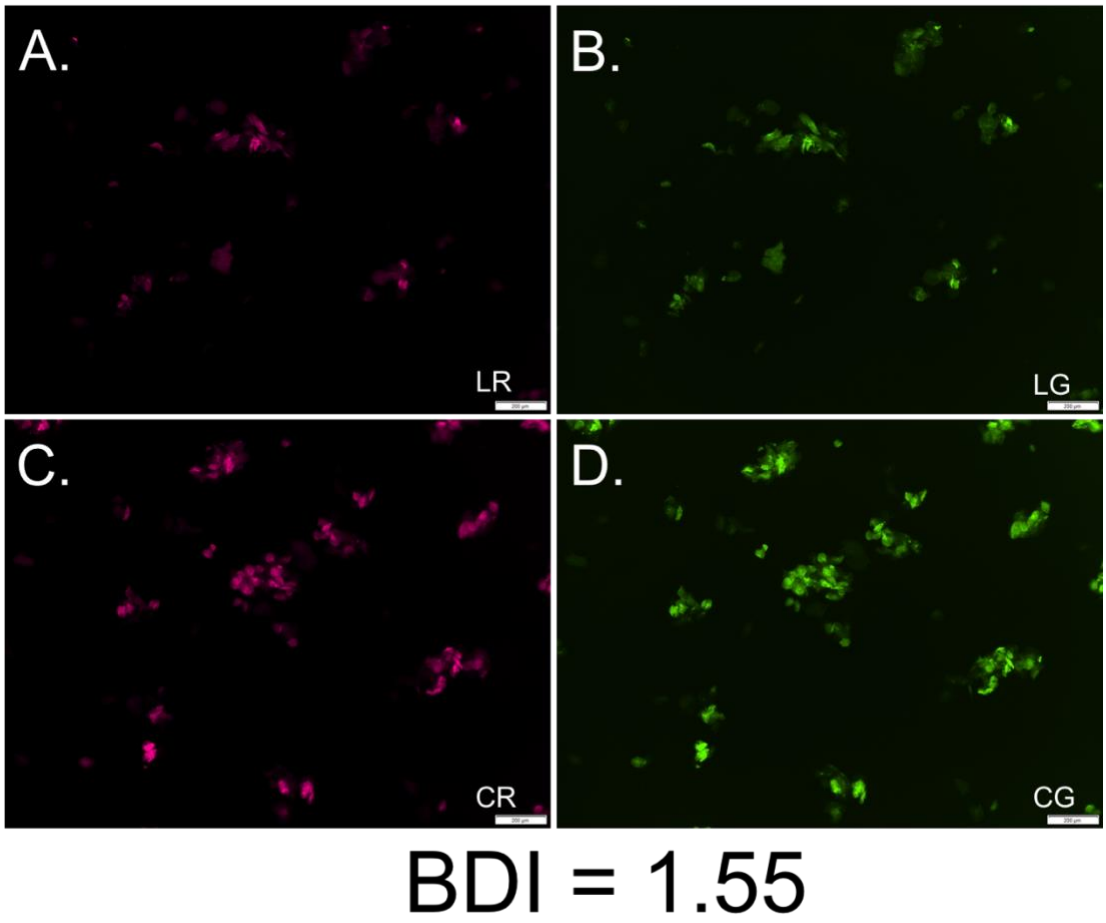


BDI = 0.61

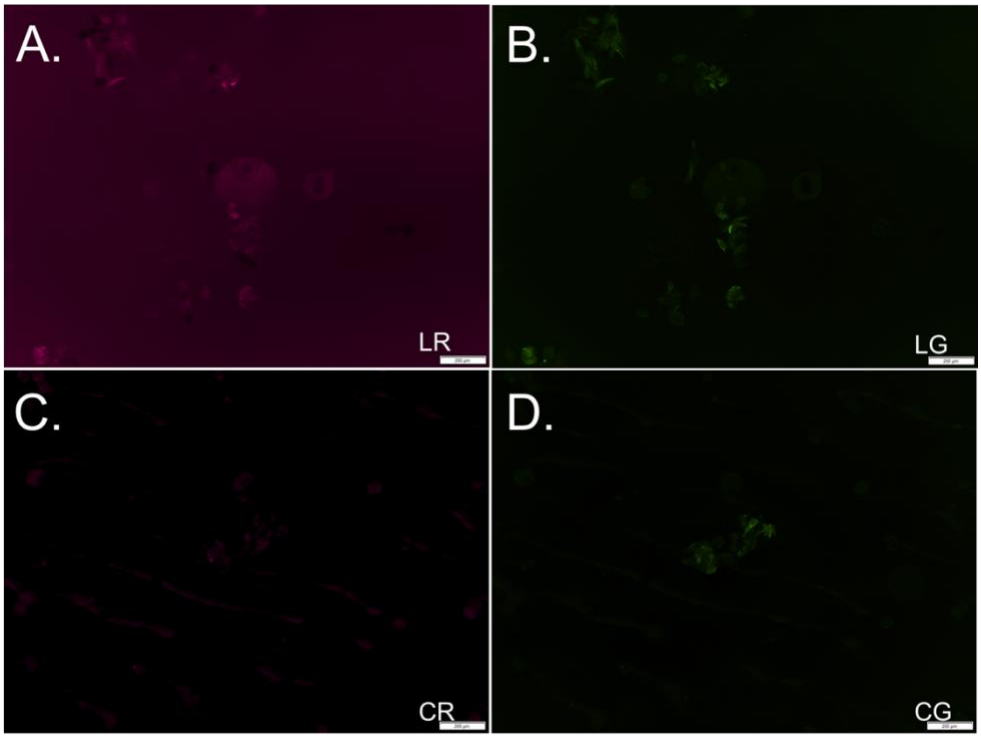


BDI = 0.98

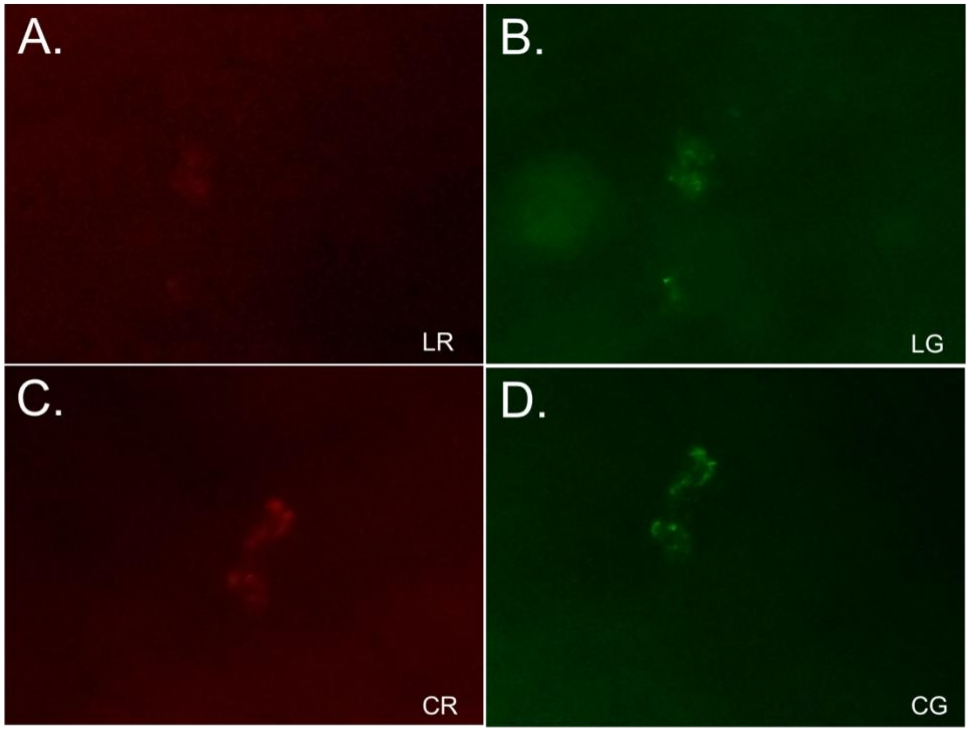
Microscope and POC BDI Control Sample 4



Microscope BDI Control Sample 5

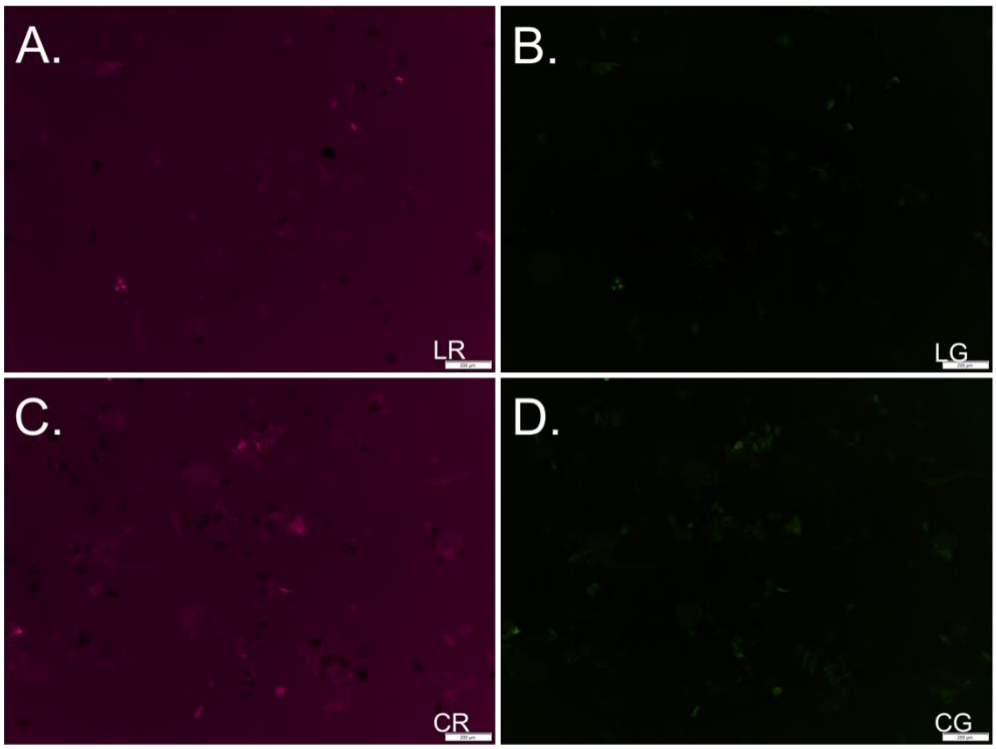


BDI = 0.05

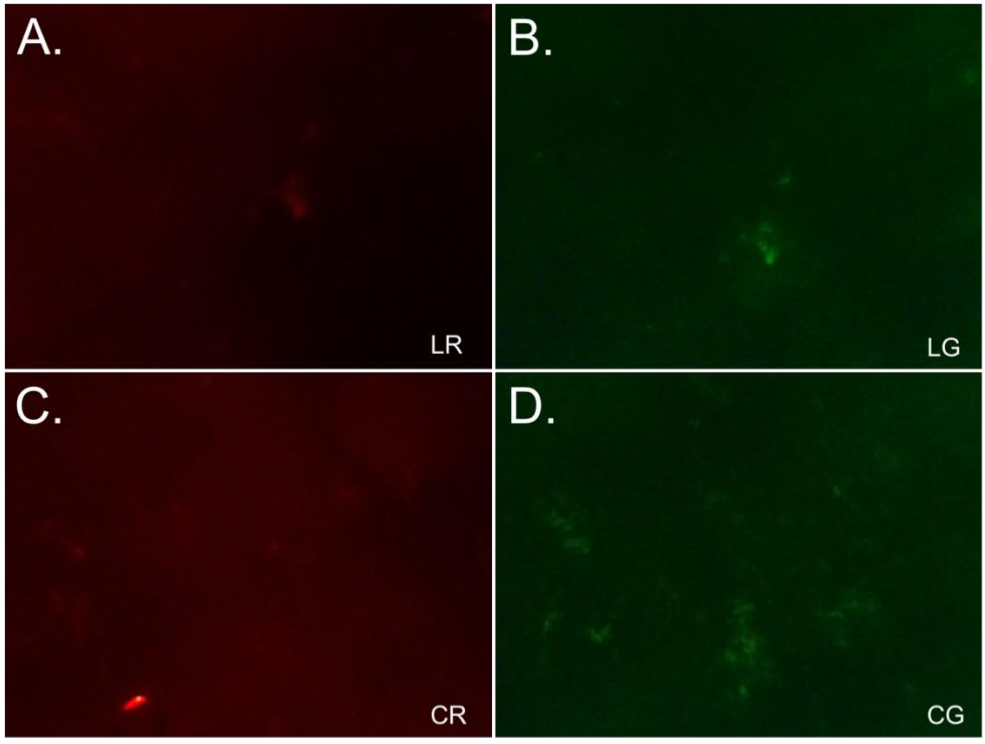


BDI = 0.91

Microscope and POC BDI Control Sample 6

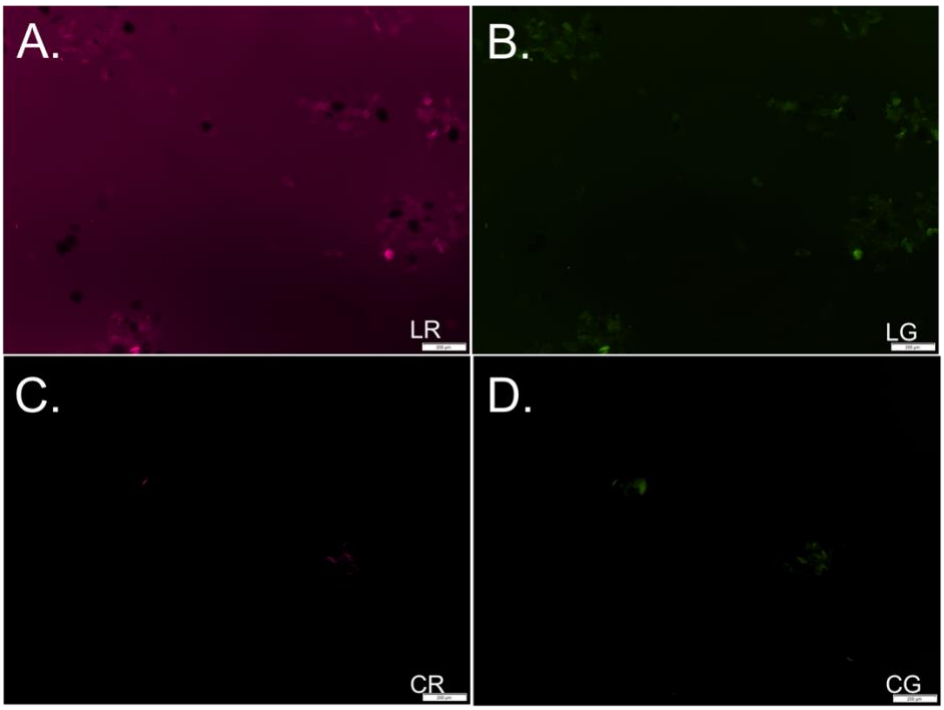


BDI = 0.61

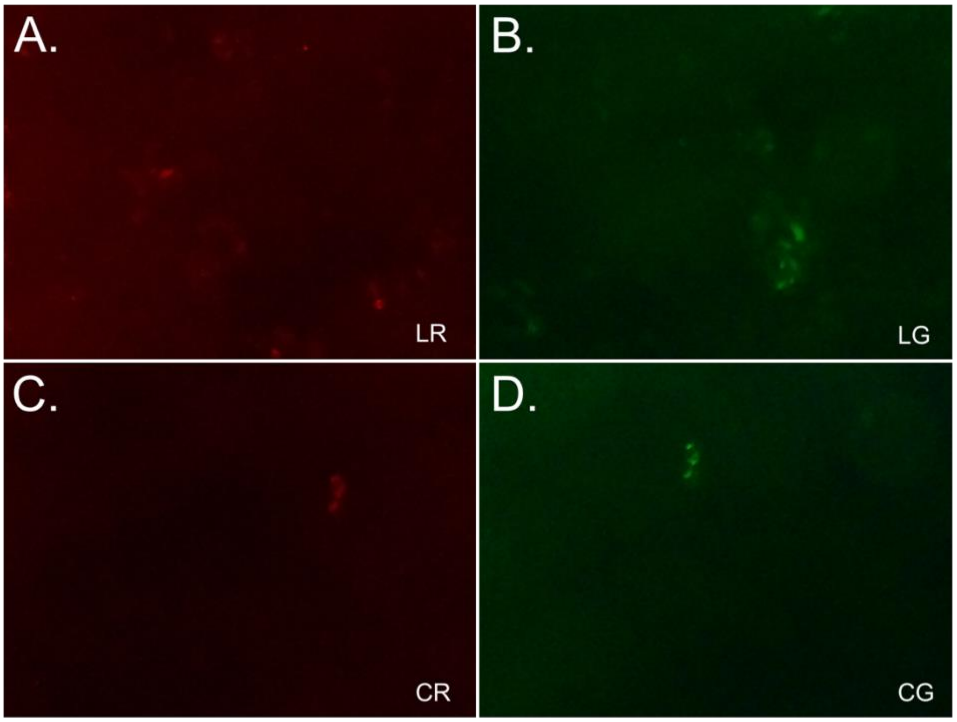


BDI = 1.44

Microscope and POC BDI Control Sample 7

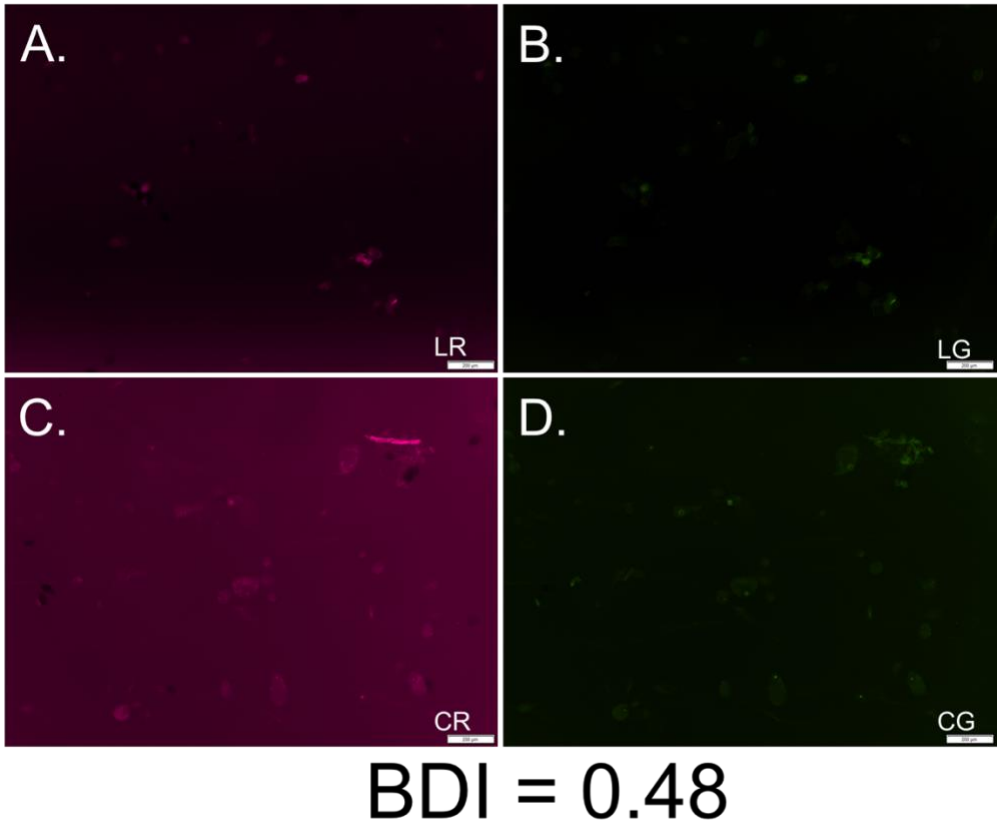


BDI = 0.08

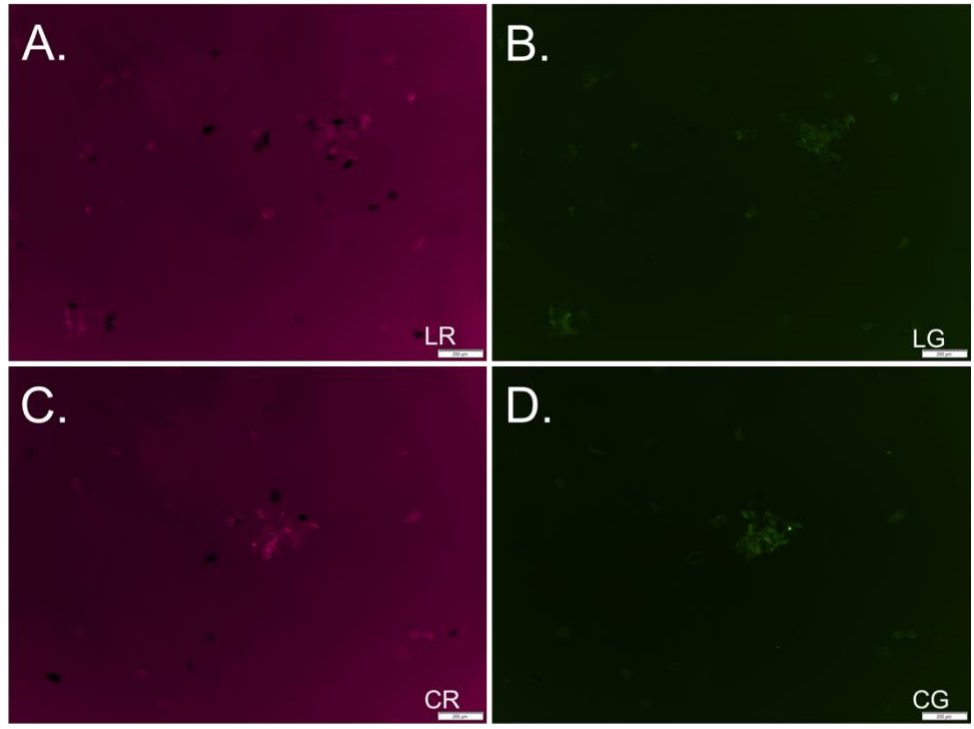


BDI = 1.27

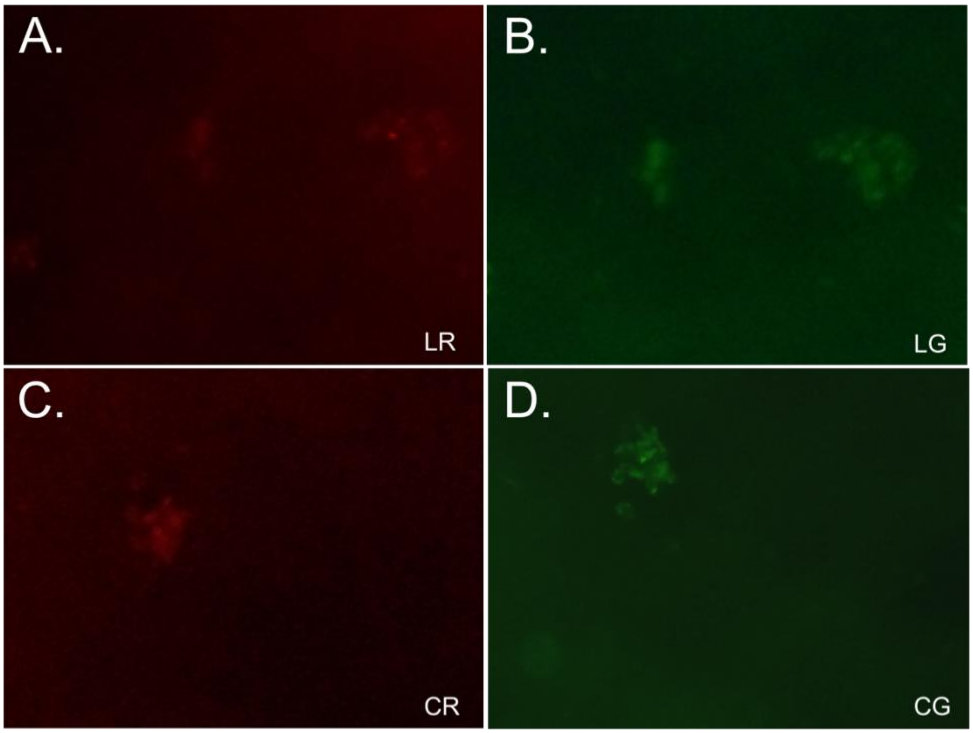
Microscope and POC BDI Control Sample 8



Microscope BDI Control Sample 9

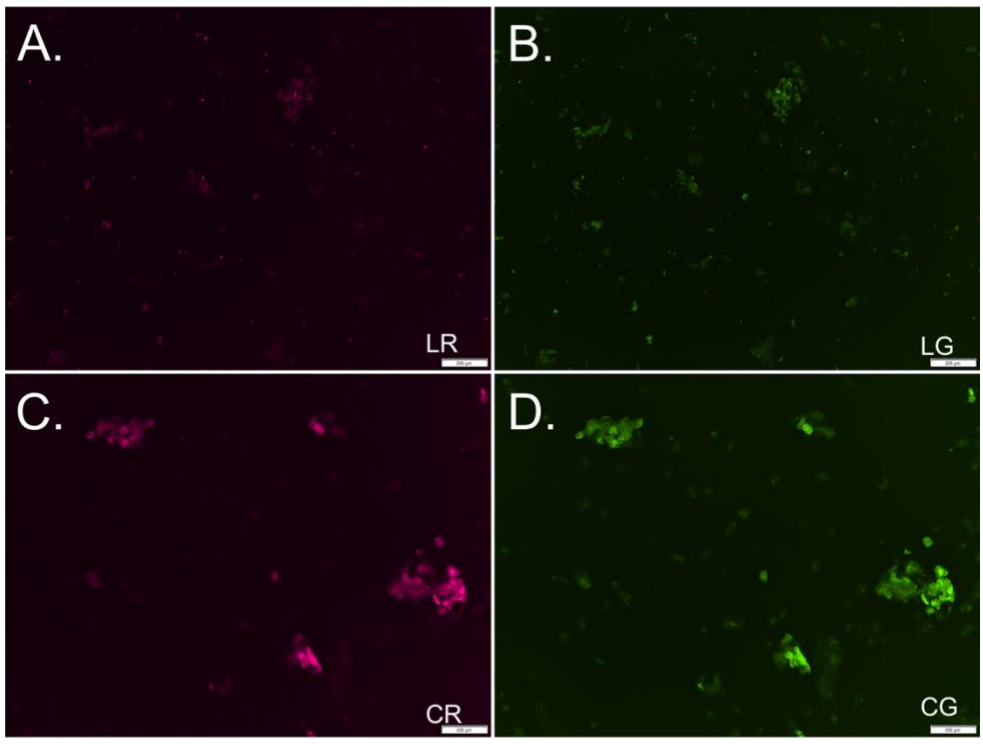


BDI = 1.09

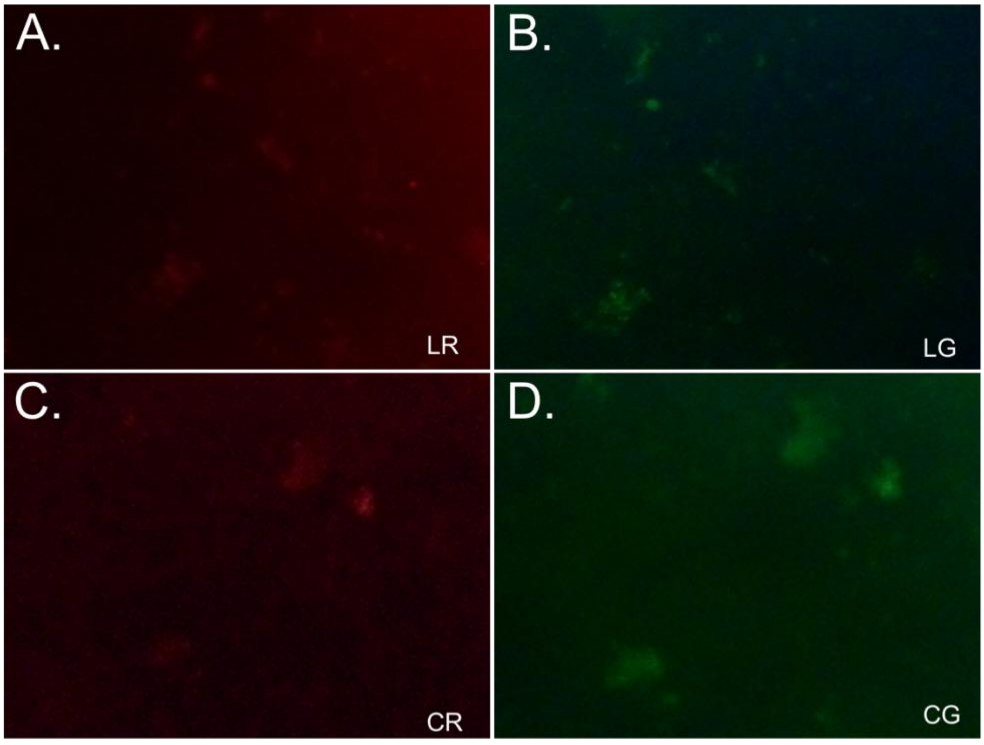


BDI = 1.35

Microscope and POC BDI Control Sample 10

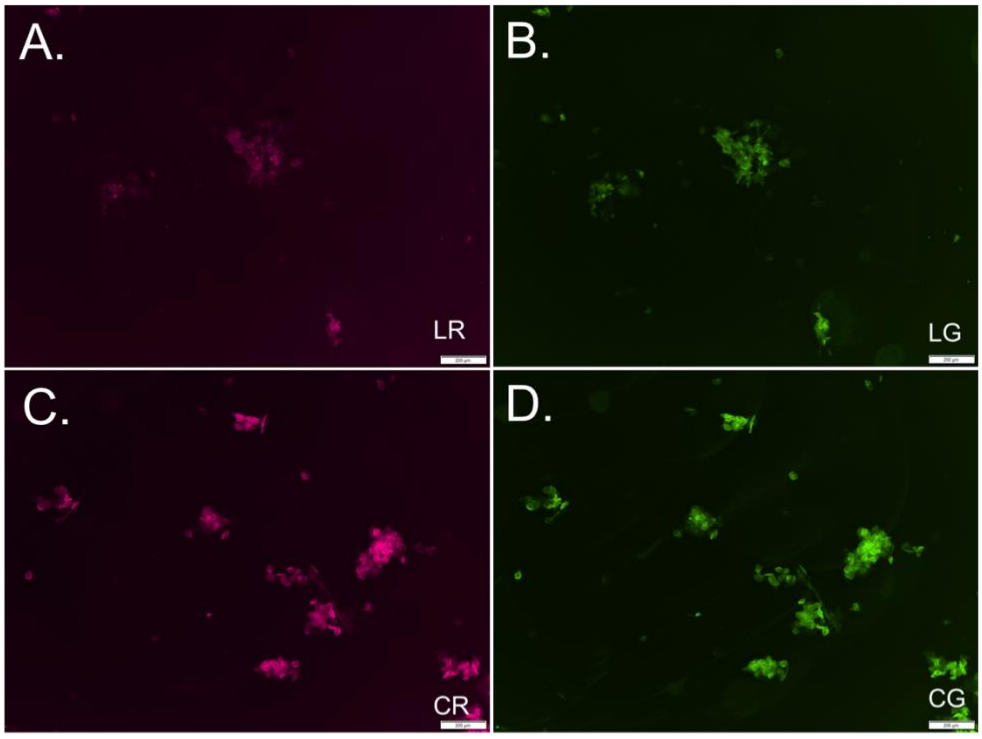


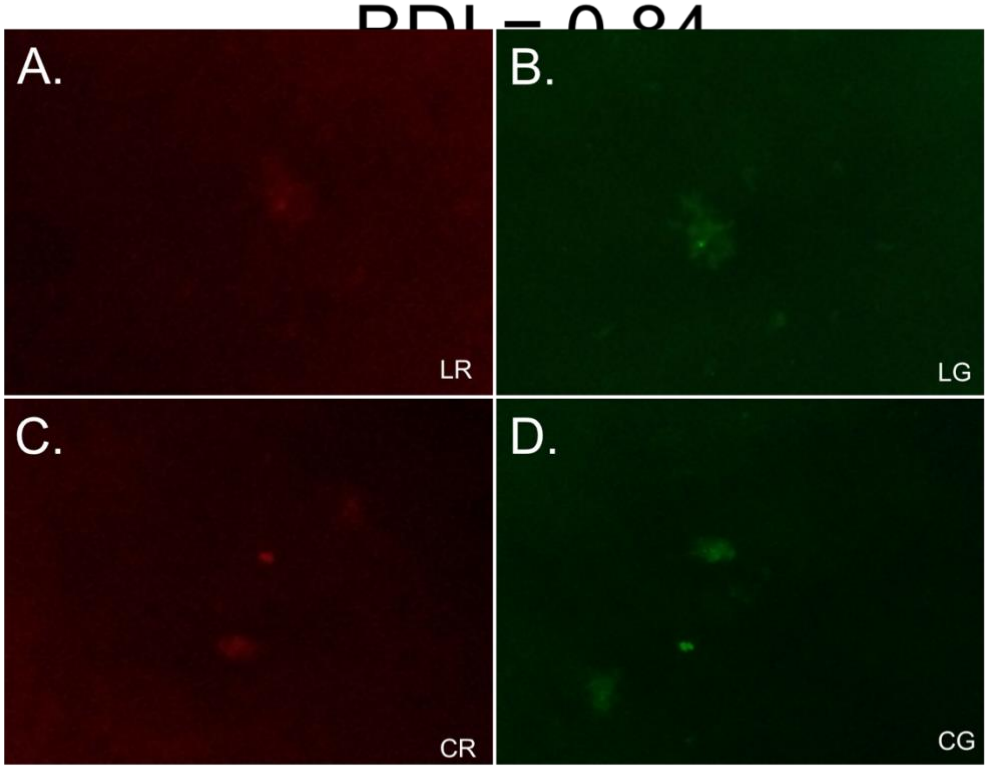
BDI = 0.98



BDI = 1.57

Microscope and POC BDI Disease Sample 1





BDI = 1.72

Microscope and POC BDI Disease Sample 2

Appendix 4.

Complete Adhesion Experiment Protocol

1. Chip Microfabrication and Functionalization (DAY 1)

1. Clean the holes of the PMMA by passing a wire from the top to bottom. Make sure to clarify which side is which.
2. Take out the bottom cover (brown), and put a double-sided adhesive (DSA) film completely on the bottom side of PMMA. (Note: Ensure the DSA is not blocking any holes and is as straight as possible. Remove all stray fibers with forceps from the DSA before attaching to PMMA). Use the back of the small forceps to attach DSA completely to the PMMA.
3. Use the gas duster to remove all dust and fibers from the slide and PMMA/ DSA.
4. Take out the outlier and attach/align it to an aminosilane coated microscope glass slide.
5. Completely attach the PMMA onto the slide. Squeeze out any bubbles by pushing them together, starting from the middle.
6. Remove the top cover paper using forceps.
7. Make 5 cm tubings (outlets) with an angle on one side. Cut tubing at a 45 degree angle.
8. Insert tubings to the outlet holes so that the ends with angles would face the flow direction. Seal the tubings with epoxy. (Note: Dispense epoxy on petri dish, mix with pipette tip, and put on tubing so that it will flow down to holes) Wait for at least 10 minutes until the epoxy has been solidified.
9. Put PBS droplets on the inlet holes so that they will flow into channels through capillary action. (Inlets should always be covered with PBS until sealed with epoxy)
10. Start washing channels.

    30uL PBS

    30uL ETOH

20uL GMBS (5 min Incubation Room Temperature)

20uL GMBS (15 min Incubation Room Temperature)

30+30uL ETOH

30+30uL PBS

30uL ETOH (Only if bubbles are present in channel)

30uL PBS (Only if bubbles are present in channel)

Completely remove ETOH from the channel by wiping away puddle containing ETOH from top of slide and putting on more PBS. Flow another 30 uL of PBS to make sure there is no bubbles in the channel.

22uL Laminin, Fibronectin, or Thrombospondin

Slowly transfer chip in the Petri dish and cover with aluminum foil (1.5 hour incubation at room temperature)

30uL BSA (Overnight incubation in 4°C fridge)

After incubation period (DAY 2)

1. Wash each channel with 30 uL of PBS.
2. Cut 40 cm long inlet tubings, and make a 45 degree angle cut on one side.
3. Take out a 1 mL syringe and fill it with PBS using the 18 gauge needle.
4. Insert a 30 gauge blunt needle (pink) to the syringe and connect it to the tubing.
5. Run PBS through the tubing using the syringe (make sure there is no bubble).
6. Insert the tubing (with a small PBS droplet at the tip so that no bubbles are introduced) in the inlet holes.
7. Clip the tubing using a binder clip so that there is no flow through the channels and then disconnect the 30 gauge blunt needle from the syringe.

8. Wipe the puddle in the inlet with Kimwipe. Wipe surface top/bottom of chip with ethanol and dry with Kimwipe (For good image quality)
9. Epoxy all tubings onto the chip. (Note: Dispense epoxy on petri dish, mix with pipette tip, and put on tubing so that it will flow down to holes)
10. Remove bubbles in the syringe needle tip with a pipette. Fill the syringe tip completely with PBS.

2. Passing Blood through Channels:

1. Fill a 1 mL syringe with blood up to the 0.3 mL line. (Use a blunt gauge tip needle)
2. Shake it up and down and push the syringe until the 0.2 mL line. (Make sure you occasionally hit the syringe with your fingers to let the bubbles move into the tip)
3. Make sure 200 uL blood is in the syringe with no bubbles. (Note: Make sure syringe pump is plugged in, turned on, set the rate and clear the volume when you change the rate)
4. Clip inlet and outlet tubing.
5. Remove the 18 gauge needle tip and connect the syringe to the 30 gauge blunt needle tip that is connected to the inlet tubing.
6. Place the syringe in the pump and wait for 5 minutes before running the sample.
7. Unclip the tubing and start the pump. Fill the channel with blood at a flow rate of 18.5 uL/min until blood exits the chip (make sure the tubing is uncoiled as much as possible for smooth flow).
8. As soon as you see a droplet of blood coming out of the outlet tubing, stop the pump - set the volume to 0 - change the flow rate to 1.85 uL/min - and start the pump again.
9. Run 15 uL of blood at 1.85 uL/min.

10. Stop the pump, clip the inlet and outlet tubing and wait for 7.5 minutes – While waiting, prepare another syringe filled with wash buffer following the steps below:

- Using a new syringe and 18 gauge needle, carefully load the syringe with wash buffer as much as possible. (the buffer tends to create bubbles easily)
- Shake the syringe up and down several times to allow the bubbles to move up towards the tip.
- Slowly push the piston to remove the bubbles out of the syringe. There might be persistent bubbles left within the needle in which case an impact loading of the piston could be more useful than a continuous push. (It's not necessary to eliminate all the bubbles at this stage)
- Take out the 18 gauge needle and connect a 30 gauge blunt needle one to the syringe.
- Slowly push the piston to fill the needle.
- Shake the syringe up and down, which will let the remaining bubbles accumulate at the top of the needle.
- Impact load the piston to apply high pressure which will remove the bubbles.

NOTE: For persistent bubbles, it might be necessary to finger-hit the needle several times. However, hitting too hard and/or too many times is very likely to create new bubbles.

11. Remove the tubing from the blood syringe and cut a small portion (2-3 cm) using a razor designated only for biohazard applications.

12. Connect the wash buffer to the inlet tubing.

13. Submerge the outlet tubing in PBS in case it is clogged by dried blood.

14. Unclip the outlet and inlet tubings, start the pump and wash the channel with (at least) 180 uL of wash buffer at uL/min. Ideally, you should maintain the flow until all the unbound cells have been washed off.

3. POC Device

1. Insert the chip into the portable imaging system as seen in the image, so that the notch on the chip aligns with the bottom groove in the sample loading area. (The top most channel is now in the camera's field of view)
2. Insert the Samsung Galaxy S4 into the portable imaging system so that the top of the phone is flush against the top of the Smartphone loading area. (The camera will now be aligned with the channel)
3. Provide illumination to the chip by turning on the LED with the switch located at the bottom of the portable imaging system.
4. Open the application "CellVision" on your Samsung Galaxy S4.
5. Press the "Take Image" button (Note: Be sure the cells are in focus before pressing "Take Image")
6. Enter the patient identification number and the protein type that coats the channel in the field of view.
7. Press "Process", the application will quantify the number of cells in the image and display them. A copy of the results will be saved to the image folder of the phone.
8. Move the chip into the next position, so that the second channel is in the field of view.
9. Repeat steps 5-7.
10. Move the chip to its final position, so that the third channel is in the field of view.
11. Repeat steps 5-7.
12. Turn off the LED using the switch located at the base of the portable imaging system.
13. Place the chip, tubing, and syringes into the appropriate biohazard receptacles.

Appendix 5.
SCD IRB Approval

University Hospitals Cleveland Medical Center

IRB ADMINISTRATION OFFICE
11100 Euclid Avenue, Lakeside 1400
Cleveland, Ohio 44106
Phone: (216) 844-1529

IRB APPROVAL NOTIFICATION

The University Hospitals Institutional Review Board (IRB) has reviewed the following submission:

Principal Investigator: Jane Little
Protocol Title: "Sickle Cell Disease (SCD) Biochip: Towards a Simple and Reliable Way to Monitor Sickle Cell Disease."
UHCMC IRB number: 05-14-07C

Submission Type: Continuing Review

Review Type: On Agenda
Date of Committee Review: 02/07/2017

As such, the UHCMC IRB has determined that with respect to the rights and welfare of the individuals, the appropriateness of the methods used to obtain informed consent and the risks and potential medical benefits of the investigation, the current submission is acceptable under Federal Human Subject Protection regulations promulgated under 45 CFR 46 and 21 CFR 50 and 56.

Date of Approval: 02/07/2017
The current expiration date for this study is: 02/06/2018
(The expiration date is the last day that a protocol has IRB approval)

- Per Federal regulation, changes MAY NOT be made to any element of the current research without prior IRB approval, except to eliminate an immediate and apparent hazard to subjects enrolled in the trial.
- Per Federal regulation, the research may not continue without IRB approval. You must submit a request for continuation at least 6-8 weeks prior to the expiration date noted above. Once the study is complete, the IRB requires prompt notification of study closure.
- Failure to retain current IRB approval may result in archiving of the current study and human subjects non-compliance allegations.

Documents reviewed and/or approved as part of this submission:

Title	Version Number	Version Date
Departmental Approval_Medicine_Little 26Jan2017	Version 1.0	01/27/2017
Departmental Approval_Pediatrics _Little IRB 05-14-07C_25Jan2017	Version 1.0	01/25/2017

Human Risk: [Risk for adults] Not Greater Than Minimal Risk [Risk for minors] Not Greater Than Minimal Risk (45 CFR 46.404/ 21 CFR 50.51)
Vulnerable populations approved for inclusion: Minors, Student of House Staff under the supervision of investigators, Employees of UHHS or Case, Illiterate Individuals, Non-English Speaking Individuals
Funding Source: Jane Little, MD/ Department of Medicine (DOM)
Other information: HIPAA Authorization
Approval Signature:

UHCMC IRB Chairperson
(Signature was applied by the IRB Administration Office)

The UHCMC IRB operates under the HHS Federal Wide Assurance of Compliance number 00003937 and IRB registration numbers 00000684, 00001691 and 00008600

p. 1 of 1 for Jane Little iRIS Reference # 067360

Appendix 6.
Oral Cancer IRB Approval

IRB ADMINISTRATION OFFICE
11100 Euclid Avenue, Lakeside 1400
Cleveland, Ohio 44106
Phone: (216) 844-1529

IRB APPROVAL NOTIFICATION

The University Hospitals Institutional Review Board (IRB) has reviewed the following submission:

Principal Investigator: Chad Zender
Protocol Title: Comprehensive Head and Neck Database (CHND)
UHCMC IRB number: 07-15-03C

Submission Type: Addendum
Other, explain:Submission of Sub-Project Protocol to use materials from Biorepository/Database
Review Type: Expedite

As such, the UHCMC IRB has determined that with respect to the rights and welfare of the individuals, the appropriateness of the methods used to obtain informed consent and the risks and potential medical benefits of the investigation, the current submission is acceptable under Federal Human Subject Protection regulations promulgated under 45 CFR 46 and 21 CFR 50 and 56.

Date of Approval: 10/18/2017
The current expiration date for this study is: 05/24/2018
(The expiration date is the last day that a protocol has IRB approval)

- Per Federal regulation, changes MAY NOT be made to any element of the current research without prior IRB approval, except to eliminate an immediate and apparent hazard to subjects enrolled in the trial.
- Per Federal regulation, the research may not continue without IRB approval. You must submit a request for continuation at least 6-8 weeks prior to the expiration date noted above. Once the study is complete, the IRB requires prompt notification of study closure.
- Failure to retain current IRB approval may result in archiving of the current study and human subjects non-compliance allegations.

Documents reviewed and/or approved as part of this submission:

Title	Version Number	Version Date
CASE4315-013_Protocol (Weinberg-Gurkan) 12Oct2017	Version 1.0	10/12/2017

Human Risk Determination: Greater Than Minimal Risk

If you have any questions or concerns, please contact the UHCMC IRB office at (216) 844-1529. The UHCMC IRB appreciates your commitment towards the ethical conduct of human subject's research.

The UHCMC IRB operates under the HHS Federal Wide Assurance of Compliance number 00003937 and IRB registration numbers 00000684, 00001691 and 00008600

p. 1 of 1 for Chad Zender iRIS Reference # 075777

Appendix 7.
CellVision App Code
Link to Online Repository
https://github.com/jonathankoss/scdapp

AutoFitTextureView.java

```
package com.example.android.camera2basic;

import android.content.Context;
import android.util.AttributeSet;
import android.view.TextureView;

/**
 * A {@link TextureView} that can be adjusted to a specified aspect ratio.
 */
public class AutoFitTextureView extends TextureView {

    private int mRatioWidth = 0;
    private int mRatioHeight = 0;

    public AutoFitTextureView(Context context) {
        this(context, null);
    }

    public AutoFitTextureView(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public AutoFitTextureView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }

    /**
```

```
     * Sets the aspect ratio for this view. The size
of the view will be measured based on the ratio
     * calculated from the parameters. Note that the
actual sizes of parameters don't matter, that
     * is, calling setAspectRatio(2, 3) and
setAspectRatio(4, 6) make the same result.
     *
     * @param width  Relative horizontal size
     * @param height Relative vertical size
     */
    public void setAspectRatio(int width, int height)
{
        if (width < 0 || height < 0) {
            throw new IllegalArgumentException("Size
cannot be negative.");
        }
        mRatioWidth = width;
        mRatioHeight = height;
        requestLayout();
    }


    @Override
    protected void onMeasure(int widthMeasureSpec,
int heightMeasureSpec) {
        super.onMeasure(widthMeasureSpec,
heightMeasureSpec);
        int width =
MeasureSpec.getSize(widthMeasureSpec);
        int height =
MeasureSpec.getSize(heightMeasureSpec);
        if (0 == mRatioWidth || 0 == mRatioHeight) {
            setMeasuredDimension(width, height);
        } else {
            if (width < height * mRatioWidth /
mRatioHeight) {
                setMeasuredDimension(width, width *
mRatioHeight / mRatioWidth);
            } else {
                setMeasuredDimension(height *
mRatioWidth / mRatioHeight, height);
            }
        }
    }
```

```
}
```

BitmapHolder.java

```
package
com.example.android.camera2basic;

import android.graphics.Bitmap;

/**
 * Created by Emilio on 2/5/17.
 */

public class BitmapHolder {

    static Bitmap bitmap;
}
```

BlobDetection.java

```
package
com.example.android.camera2basic;

/**
 * Created by Jonathan on 5/19/2016.
 */
    import java.util.ArrayList;
    import java.util.List;

    import android.graphics.Bitmap;
    import android.graphics.Color;

    public class BlobDetection {

        private byte[][] COLOUR_ARRAY = {{(byte)103,
(byte)121, (byte)255},
                {(byte)249, (byte)255, (byte)139},
                {(byte)140, (byte)255, (byte)127},
```

```
                {(byte)167, (byte)254, (byte)255},
                {(byte)255, (byte)111, (byte)71}};


        private int width;
        private int height;


        public static int[] labelBuffer;
        public static int[][] labelBufferCoordinates;
        public static int[] labelTable;
        public static int[] xMinTable;
        public static int[] xMaxTable;
        public static int[] yMinTable;
        public static int[] yMaxTable;
        public static int[] massTable;
        public static List<Blob> blobList ;
        public static class Blob
        {
            public int xMin;
            public int xMax;
            public int yMin;
            public int yMax;
            public int mass;


            public Blob(int xMin, int xMax, int yMin,
int yMax, int mass)
            {
                this.xMin = xMin;
                this.xMax = xMax;
                this.yMin = yMin;
                this.yMax = yMax;
                this.mass = mass;
            }


            public String toString()
            {
                return String.format("X: %4d -> %4d,
Y: %4d -> %4d, mass: %6d", xMin, xMax, yMin, yMax,
mass);
            }
        }
```

```
        public BlobDetection(Bitmap bitmap) {
            int width = bitmap.getWidth();
            int height = bitmap.getHeight();
            this.width = width;
            this.height = height;


            labelBuffer = new int[width * height];
            labelBufferCoordinates = new
int[width][height];
            // The maximum number of blobs is given by
an image filled with equally spaced single pixel
            // blobs. For images with less blobs,
memory will be wasted, but this approach is simpler
and
            // probably quicker than dynamically
resizing arrays
            int tableSize = width * height / 4;


            labelTable = new int[tableSize];
            xMinTable = new int[tableSize];
            xMaxTable = new int[tableSize];
            yMinTable = new int[tableSize];
            yMaxTable = new int[tableSize];
            massTable = new int[tableSize];
        }


        public Bitmap  getBlob(Bitmap bitmap) {
            // This is the neighbouring pixel pattern.
For position X, A, B, C & D are checked
            // A B C
            // D X


            int srcPtr = 0;
            int aPtr = -width - 1;
            int bPtr = -width;
            int cPtr = -width + 1;
```

```
            int dPtr = -1;


            int label = 1;


            int minBlobMass = 0;
            int maxBlobMass = -1;


            Blob blobMayor = null;
            blobList = new ArrayList<Blob>();


            //First pass
            for (int y = 0; y< bitmap.getHeight()
;y++) {
                for (int x = 0;x<bitmap.getWidth();x++
) {
                    // if data[row][col] is not
Background
                    if ( bitmap.getPixel(x, y) ==
Color.BLACK ) {
                        labelBuffer[srcPtr] = 0;
                        labelBufferCoordinates[x][y] =
0;
                        // Find label for neighbours
(0 if out of range)
                        int aLabel = (x > 0 && y > 0)
                      ? labelTable[labelBuffer[aPtr]]
: 0;
                        int bLabel = (y > 0)
                                    ?
labelTable[labelBuffer[bPtr]] : 0;
                        int cLabel = (x < width-1 && y
> 0)   ? labelTable[labelBuffer[cPtr]] : 0;
                        int dLabel = (x > 0)
                                    ?
labelTable[labelBuffer[dPtr]] : 0;


                        // Look for label with least
value
                        int min = Integer.MAX_VALUE;
```

```
                        if (aLabel != 0 && aLabel <
min) min = aLabel;
                        if (bLabel != 0 && bLabel <
min) min = bLabel;
                        if (cLabel != 0 && cLabel <
min) min = cLabel;
                        if (dLabel != 0 && dLabel <
min) min = dLabel;


                        // If no neighbours in
foreground
                        if (min == Integer.MAX_VALUE)
                        {
                            labelBuffer[srcPtr] =
label;

labelBufferCoordinates[x][y] = label;
                            labelTable[label] = label;


                            // Initialise min/max x,y
for label
                            yMinTable[label] = y;
                            yMaxTable[label] = y;
                            xMinTable[label] = x;
                            xMaxTable[label] = x;
                            massTable[label] = 1;


                            label ++;
                        }


                        // Neighbour found
                        else
                        {
                            // Label pixel with lowest
label from neighbours
                            labelBuffer[srcPtr] = min;

labelBufferCoordinates[x][y] = min;
                            // Update min/max x,y for
label
```

```
                            yMaxTable[min] = y;
                            massTable[min]++;
                            if (x < xMinTable[min])
xMinTable[min] = x;
                            if (x > xMaxTable[min])
xMaxTable[min] = x;


                            if (aLabel != 0)
labelTable[aLabel] = min;
                            if (bLabel != 0)
labelTable[bLabel] = min;
                            if (cLabel != 0)
labelTable[cLabel] = min;
                            if (dLabel != 0)
labelTable[dLabel] = min;
                        }
                    }


                    srcPtr ++;
                    aPtr ++;
                    bPtr ++;
                    cPtr ++;
                    dPtr ++;
                }
            }


            // Iterate through labels pushing min/max
x,y values towards minimum label
            if (blobList == null) blobList = new
ArrayList<Blob>();


            for (int i=label-1 ; i>0 ; i--)
            {
                if (labelTable[i] != i)
                {
                    if (xMaxTable[i] >
xMaxTable[labelTable[i]]) xMaxTable[labelTable[i]] =
xMaxTable[i];
```

```
                    if (xMinTable[i] <
xMinTable[labelTable[i]]) xMinTable[labelTable[i]] =
xMinTable[i];
                    if (yMaxTable[i] >
yMaxTable[labelTable[i]]) yMaxTable[labelTable[i]] =
yMaxTable[i];
                    if (yMinTable[i] <
yMinTable[labelTable[i]]) yMinTable[labelTable[i]] =
yMinTable[i];
                    massTable[labelTable[i]] +=
massTable[i];


                    int l = i;
                    while (l != labelTable[l]) l =
labelTable[l];
                    labelTable[i] = l;
                }
                else
                {
                    // Ignore blobs that butt against
corners
                    if (i == labelBuffer[0]) continue;
                                                    //
Top Left
                    if (i == labelBuffer[width])
continue;                                       // Top
Right
                    if (i ==
labelBuffer[(width*height) - width + 1]) continue; //
Bottom Left
                    if (i ==
labelBuffer[(width*height) - 1]) continue;
       // Bottom Right


                    if (massTable[i] >= minBlobMass &&
(massTable[i] <= maxBlobMass || maxBlobMass == -1))
                    {
                        Blob blob = new
Blob(xMinTable[i], xMaxTable[i], yMinTable[i],
yMaxTable[i], massTable[i]);
                        if ( blobMayor == null ||
blob.mass >= blobMayor.mass )
```

```
                            blobMayor = blob;
                        blobList.add(blob);
                    }
                }
            }
            int maximoBlob = 0;
            int posicion = 0;
            for (int i = 0;i<massTable.length;i++) {
                if ( massTable[i] > maximoBlob ) {
                    posicion = i;
                    maximoBlob = massTable[i];
                }


            }
            Bitmap finalBitmap =
bitmap.copy(Bitmap.Config.ARGB_8888, true);
            for (int y = 0; y< bitmap.getHeight()
;y++) {
                for (int x = 0;x<bitmap.getWidth();x++
) {
                    if (
labelTable[labelBufferCoordinates[x][y]] == posicion )
{
                        finalBitmap.setPixel(x, y,
Color.BLACK);
                    }   else  if (
labelTable[labelBufferCoordinates[x][y]] != 0) {
                        finalBitmap.setPixel(x, y,
Color.GREEN);
                    } else {
                        finalBitmap.setPixel(x, y,
Color.WHITE);
                    }
                }
            }
            return finalBitmap;
        }
    }
```

Camera2BasicFragment.java

```
//package
com.example.android.ca
mera2basic;
                        //
                        //import android.Manifest;
                        //import android.app.Activity;
                        //import android.app.AlertDialog;
                        //import android.app.Dialog;
                        //import android.app.DialogFragment;
                        //import android.app.Fragment;
                        //import android.app.usage.UsageEvents;
                        //import android.content.Context;
                        //import android.content.DialogInterface;
                        //import android.content.Intent;
                        //import android.content.pm.PackageManager;
                        //import android.content.res.Configuration;
                        //import android.database.Cursor;
                        //import android.graphics.ImageFormat;
                        //import android.graphics.Matrix;
                        //import android.graphics.Point;
                        //import android.graphics.Rect;
                        //import android.graphics.RectF;
                        //import android.graphics.SurfaceTexture;
                        //import android.hardware.Camera;
                        //import android.hardware.camera2.CameraAccessException;
                        //import android.hardware.camera2.CameraCaptureSession;
                        //import android.hardware.camera2.CameraCharacteristics;
                        //import android.hardware.camera2.CameraDevice;
                        //import android.hardware.camera2.CameraManager;
                        //import android.hardware.camera2.CameraMetadata;
                        //import android.hardware.camera2.CaptureRequest;
                        //import android.hardware.camera2.CaptureResult;
                        //import android.hardware.camera2.TotalCaptureResult;
                        //import android.hardware.camera2.params.MeteringRectangle;
                        //import android.hardware.camera2.params.StreamConfigurationMap;
                        //import android.media.Image;
                        //import android.media.ImageReader;
                        //import android.net.Uri;
                        //import android.os.Bundle;
                        //import android.os.Handler;
                        //import android.os.HandlerThread;
                        //import android.provider.MediaStore;
                        //import android.support.annotation.NonNull;
```

```
//import android.support.annotation.Nullable;
//import android.support.v13.app.FragmentCompat;
//import android.support.v4.content.ContextCompat;
//import android.util.Log;
//import android.util.Size;
//import android.util.SizeF;
//import android.util.SparseIntArray;
//import android.view.LayoutInflater;
//import android.view.MotionEvent;
//import android.view.Surface;
//import android.view.TextureView;
//import android.view.View;
//import android.view.ViewGroup;
//import android.widget.AdapterView;
//import android.widget.ArrayAdapter;
//import android.widget.Spinner;
//import android.widget.Switch;
//import android.widget.Toast;
//
//import org.apache.commons.io.FileUtils;
//
//import java.io.File;
//import java.io.FileOutputStream;
//import java.io.IOException;
//import java.nio.ByteBuffer;
//import java.text.DateFormat;
//import java.util.ArrayList;
//import java.util.Arrays;
//import java.util.Collections;
//import java.util.Comparator;
//import java.util.Date;
//import java.util.List;
//import java.util.concurrent.Semaphore;
//import java.util.concurrent.TimeUnit;
//
//import static android.app.Activity.RESULT_OK;
//
//public class Camera2BasicFragment extends Fragment
//        implements View.OnClickListener,
FragmentCompat.OnRequestPermissionsResultCallback {
//
//    private static final int RESULT_LOAD_IMAGE = 1;
//
//    /**
```

```
//     * Conversion from screen rotation to JPEG orientation.
//     */
//    private static final SparseIntArray ORIENTATIONS = new
SparseIntArray();
//    private static final int REQUEST_CAMERA_PERMISSION = 1;
//    private static final String FRAGMENT_DIALOG = "dialog";
//    private static final int FOCUS_AREA_SIZE = 10;
//
//    private Spinner spinner;
//    public imageTypeOnItemSelectedListener imageTypeListener;
//
//
//
//    static {
//        ORIENTATIONS.append(Surface.ROTATION_0, 90);
//        ORIENTATIONS.append(Surface.ROTATION_90, 0);
//        ORIENTATIONS.append(Surface.ROTATION_180, 270);
//        ORIENTATIONS.append(Surface.ROTATION_270, 180);
//    }
//
//    /**
//     * Tag for the {@link Log}.
//     */
//    private static final String TAG = "Camera2BasicFragment";
//
//    /**
//     * Camera state: Showing camera preview.
//     */
//    private static final int STATE_PREVIEW = 0;
//
//    /**
//     * Camera state: Waiting for the focus to be locked.
//     */
//    private static final int STATE_WAITING_LOCK = 1;
//
//    /**
//     * Camera state: Waiting for the exposure to be precapture
state.
//     */
//    private static final int STATE_WAITING_PRECAPTURE = 2;
//
//    /**
//     * Camera state: Waiting for the exposure state to be
something other than precapture.
```

```
//     */
//    private static final int STATE_WAITING_NON_PRECAPTURE = 3;
//
//    /**
//     * Camera state: Picture was taken.
//     */
//    private static final int STATE_PICTURE_TAKEN = 4;
//
//    /**
//     * Max preview width that is guaranteed by Camera2 API
//     */
//    private static final int MAX_PREVIEW_WIDTH = 1920;
//
//    /**
//     * Max preview height that is guaranteed by Camera2 API
//     */
//    private static final int MAX_PREVIEW_HEIGHT = 1080;
//
//    /**
//     * {@link TextureView.SurfaceTextureListener} handles
several lifecycle events on a
//     * {@link TextureView}.
//     */
//    private final TextureView.SurfaceTextureListener
mSurfaceTextureListener
//            = new TextureView.SurfaceTextureListener() {
//
//        @Override
//        public void onSurfaceTextureAvailable(SurfaceTexture
texture, int width, int height) {
//            openCamera(width, height);
//        }
//
//        @Override
//        public void onSurfaceTextureSizeChanged(SurfaceTexture
texture, int width, int height) {
//            configureTransform(width, height);
//        }
//
//        @Override
//        public boolean
onSurfaceTextureDestroyed(SurfaceTexture texture) {
//            return true;
//        }
```

```
//
//        @Override
//        public void onSurfaceTextureUpdated(SurfaceTexture
texture) {
//        }
//
//    };
//
//    /**
//     * ID of the current {@link CameraDevice}.
//     */
//    private String mCameraId;
//
//    /**
//     * An {@link AutoFitTextureView} for camera preview.
//     */
//    private AutoFitTextureView mTextureView;
//
//    private Switch mDemoModeSwitch;
//
//    private CameraCharacteristics characteristics;
//
//
//    private CaptureRequest.Builder mPreviewBuilder;
//
//    /**
//     * A {@link CameraCaptureSession } for camera preview.
//     */
//    private CameraCaptureSession mCaptureSession;
//
//    /**
//     * A reference to the opened {@link CameraDevice}.
//     */
//    private CameraDevice mCameraDevice;
//
//    /**
//     * The {@link android.util.Size} of camera preview.
//     */
//    private Size mPreviewSize;
//
//    /**
//     * {@link CameraDevice.StateCallback} is called when
{@link CameraDevice} changes its state.
//     */
```

```
//    private final CameraDevice.StateCallback mStateCallback =
new CameraDevice.StateCallback() {
//
//        @Override
//        public void onOpened(@NonNull CameraDevice
cameraDevice) {
//            // This method is called when the camera is
opened.  We start camera preview here.
//            mCameraOpenCloseLock.release();
//            mCameraDevice = cameraDevice;
//            createCameraPreviewSession();
//        }
//
//        @Override
//        public void onDisconnected(@NonNull CameraDevice
cameraDevice) {
//            mCameraOpenCloseLock.release();
//            cameraDevice.close();
//            mCameraDevice = null;
//        }
//
//        @Override
//        public void onError(@NonNull CameraDevice
cameraDevice, int error) {
//            mCameraOpenCloseLock.release();
//            cameraDevice.close();
//            mCameraDevice = null;
//            Activity activity = getActivity();
//            if (null != activity) {
//                activity.finish();
//            }
//        }
//
//    };
//
//    /**
//     * An additional thread for running tasks that shouldn't
block the UI.
//     */
//    private HandlerThread mBackgroundThread;
//
//    /**
//     * A {@link Handler} for running tasks in the background.
//     */
```

```
//    private Handler mBackgroundHandler;
//
//    /**
//     * An {@link ImageReader} that handles still image
capture.
//     */
//    private ImageReader mImageReader;
//
//    /**
//     * This is the output file for our picture.
//     */
//    private File mFile;
//
//    /**
//     * This a callback object for the {@link ImageReader}.
"onImageAvailable" will be called when a
//     * still image is ready to be saved.
//     */
//    private final ImageReader.OnImageAvailableListener
mOnImageAvailableListener
//            = new ImageReader.OnImageAvailableListener() {
//
//        @Override
//        public void onImageAvailable(ImageReader reader) {
//            System.out.println("EMILIO: image available");
//            mBackgroundHandler.post(new
ImageSaver(reader.acquireNextImage(), mFile));
//        }
//
//    };
//
//    /**
//     * {@link CaptureRequest.Builder} for the camera preview
//     */
//    private CaptureRequest.Builder mPreviewRequestBuilder;
//
//    /**
//     * {@link CaptureRequest} generated by {@link
#mPreviewRequestBuilder}
//     */
//    private CaptureRequest mPreviewRequest;
//
//    /**
//     * The current state of camera state for taking pictures.
```

```
//     *
//     * @see #mCaptureCallback
//     */
//    private int mState = STATE_PREVIEW;
//
//    /**
//     * A {@link Semaphore} to prevent the app from exiting
before closing the camera.
//     */
//    private Semaphore mCameraOpenCloseLock = new Semaphore(1);
//
//    /**
//     * Whether the current camera device supports Flash or
not.
//     */
//    private boolean mFlashSupported;
//
//    /**
//     * Orientation of the camera sensor
//     */
//    private int mSensorOrientation;
//
//    /**
//     * A {@link CameraCaptureSession.CaptureCallback} that
handles events related to JPEG capture.
//     */
//    private CameraCaptureSession.CaptureCallback
mCaptureCallback
//            = new CameraCaptureSession.CaptureCallback() {
//
//        private void process(CaptureResult result) {
//            switch (mState) {
//                case STATE_PREVIEW: {
//                    // We have nothing to do when the camera
preview is working normally.
//                    break;
//                }
//                case STATE_WAITING_LOCK: {
//                    Integer afState =
result.get(CaptureResult.CONTROL_AF_STATE);
//                    if (afState == null) {
//                        captureStillPicture();
//                    } else if
(CaptureResult.CONTROL_AF_STATE_FOCUSED_LOCKED == afState ||
```

```
//
CaptureResult.CONTROL_AF_STATE_NOT_FOCUSED_LOCKED == afState) {
//                        // CONTROL_AE_STATE can be null on
some devices
//                        Integer aeState =
result.get(CaptureResult.CONTROL_AE_STATE);
//                        System.out.println("EMILIO: aeState =
"+aeState);
//                        if (aeState == null ||
//                                aeState ==
CaptureResult.CONTROL_AE_STATE_CONVERGED) {
//                            mState = STATE_PICTURE_TAKEN;
//                            captureStillPicture();
//                        } else {
//                            runPrecaptureSequence();
//                        }
//                    } else {
//                        runPrecaptureSequence();
//                    }
//                    break;
//                }
//                case STATE_WAITING_PRECAPTURE: {
//                    // CONTROL_AE_STATE can be null on some
devices
//                    Integer aeState =
result.get(CaptureResult.CONTROL_AE_STATE);
//                    if (aeState == null ||
//                            aeState ==
CaptureResult.CONTROL_AE_STATE_PRECAPTURE ||
//                            aeState ==
CaptureRequest.CONTROL_AE_STATE_FLASH_REQUIRED) {
//                        mState = STATE_WAITING_NON_PRECAPTURE;
//                    }
//                    break;
//                }
//                case STATE_WAITING_NON_PRECAPTURE: {
//                    // CONTROL_AE_STATE can be null on some
devices
//                    Integer aeState =
result.get(CaptureResult.CONTROL_AE_STATE);
//                    if (aeState == null || aeState !=
CaptureResult.CONTROL_AE_STATE_PRECAPTURE) {
//                        mState = STATE_PICTURE_TAKEN;
//                        captureStillPicture();
```

```
//                    }
//                    break;
//                }
//            }
//        }
//
//        @Override
//        public void onCaptureProgressed(@NonNull
CameraCaptureSession session,
//                                        @NonNull
CaptureRequest request,
//                                        @NonNull CaptureResult
partialResult) {
//            process(partialResult);
//        }
//
//        @Override
//        public void onCaptureCompleted(@NonNull
CameraCaptureSession session,
//                                       @NonNull CaptureRequest
request,
//                                       @NonNull
TotalCaptureResult result) {
//            process(result);
//        }
//
//    };
//
//    /**
//     * Shows a {@link Toast} on the UI thread.
//     *
//     * @param text The message to show
//     */
//    private void showToast(final String text) {
//        final Activity activity = getActivity();
//        if (activity != null) {
//            activity.runOnUiThread(new Runnable() {
//                @Override
//                public void run() {
//                    Toast.makeText(activity, text,
Toast.LENGTH_SHORT).show();
//                }
//            });
//        }
```

```
//    }
//
//    /**
//     * Given {@code choices} of {@code Size}s supported by a
camera, choose the smallest one that
//     * is at least as large as the respective texture view
size, and that is at most as large as the
//     * respective max size, and whose aspect ratio matches
with the specified value. If such size
//     * doesn't exist, choose the largest one that is at most
as large as the respective max size,
//     * and whose aspect ratio matches with the specified
value.
//     *
//     * @param choices           The list of sizes that the
camera supports for the intended output
//     *                          class
//     * @param textureViewWidth  The width of the texture view
relative to sensor coordinate
//     * @param textureViewHeight The height of the texture view
relative to sensor coordinate
//     * @param maxWidth          The maximum width that can be
chosen
//     * @param maxHeight         The maximum height that can be
chosen
//     * @param aspectRatio       The aspect ratio
//     * @return The optimal {@code Size}, or an arbitrary one
if none were big enough
//     */
//    private static Size chooseOptimalSize(Size[] choices, int
textureViewWidth,
//            int textureViewHeight, int maxWidth, int
maxHeight, Size aspectRatio) {
//
//        // Collect the supported resolutions that are at least
as big as the preview Surface
//        List<Size> bigEnough = new ArrayList<>();
//        // Collect the supported resolutions that are smaller
than the preview Surface
//        List<Size> notBigEnough = new ArrayList<>();
//        int w = aspectRatio.getWidth();
//        int h = aspectRatio.getHeight();
//        for (Size option : choices) {
```

```
//            if (option.getWidth() <= maxWidth &&
option.getHeight() <= maxHeight &&
//                    option.getHeight() == option.getWidth() *
h / w) {
//                if (option.getWidth() >= textureViewWidth &&
//                    option.getHeight() >= textureViewHeight) {
//                    bigEnough.add(option);
//                } else {
//                    notBigEnough.add(option);
//                }
//            }
//        }
//
//        // Pick the smallest of those big enough. If there is
no one big enough, pick the
//        // largest of those not big enough.
//        if (bigEnough.size() > 0) {
//            return Collections.min(bigEnough, new
CompareSizesByArea());
//        } else if (notBigEnough.size() > 0) {
//            return Collections.max(notBigEnough, new
CompareSizesByArea());
//        } else {
//            Log.e(TAG, "Couldn't find any suitable preview
size");
//            return choices[0];
//        }
//    }
//
//    public static Camera2BasicFragment newInstance() {
//        return new Camera2BasicFragment();
//    }
//
//    @Override
//    public View onCreateView(LayoutInflater inflater,
ViewGroup container,
//                             Bundle savedInstanceState) {
//
//        View view =
inflater.inflate(R.layout.fragment_camera2_basic, container,
false);
//
//        return view;
//    }
```

```
//
//    private void tapToFocus(CaptureRequest.Builder
requestBuilder, MotionEvent event) {
//        Rect rect;
//        Size size;
//        if
(CameraCharacteristics.SENSOR_INFO_ACTIVE_ARRAY_SIZE != null) {
//            showToast("sensor info valid");
//            rect =
characteristics.get(CameraCharacteristics.SENSOR_INFO_ACTIVE_ARR
AY_SIZE);
//        } else {
//            rect = new Rect(0, 0, 3000, 4096);
//        }
//        Log.i("onAreaTouchEvent",
"SENSOR_INFO_ACTIVE_ARRAY_SIZE,,,,,,,,rect.left--->" + rect.left
+ ",,,rect.top--->" + rect.top + ",,,,rect.right--->" +
rect.right + ",,,,rect.bottom---->" + rect.bottom);
//        if (CameraCharacteristics.SENSOR_INFO_PIXEL_ARRAY_SIZE
!= null) {
//            size =
characteristics.get(CameraCharacteristics.SENSOR_INFO_PIXEL_ARRA
Y_SIZE);
//        } else {
//            size = new Size(3000, 4096);
//        }
//        Log.i("onAreaTouchEvent",
"mCameraCharacteristics,,,,size.getWidth()--->" +
size.getWidth() + ",,,size.getHeight()--->" + size.getHeight());
//        int areaSize = 64;
//        //Rect focusRect = calculateTapArea(event.getX(),
event.getY(), 1f);
//        int right = rect.right;
//        int bottom = rect.bottom;
//        int viewWidth = mTextureView.getWidth();
//        int viewHeight = mTextureView.getHeight();
//        int ll, rr;
//        Rect newRect;
//        int centerX = (int) event.getX();
//        int centerY = (int) event.getY();
//        ll = ((centerX * right) - areaSize) / viewWidth;
//        rr = ((centerY * bottom) - areaSize) / viewHeight;
//        int focusLeft = clamp(ll, 0, right);
//        int focusBottom = clamp(rr, 0, bottom);
```

```
//        Log.i("focus_position", "focusLeft--->" + focusLeft +
",,,focusTop--->" + focusBottom + ",,,focusRight--->" +
(focusLeft + areaSize) + ",,,focusBottom--->" + (focusBottom +
areaSize));
//        newRect = new Rect(focusLeft, focusBottom, focusLeft +
areaSize, focusBottom + areaSize);
//        MeteringRectangle meteringRectangle = new
MeteringRectangle(newRect, 100);
//        MeteringRectangle[] meteringRectangleArr =
{meteringRectangle};
//        showToast("metering Rect: " + meteringRectangle);
//
////        int regions_af =
characteristics.get(CameraCharacteristics.CONTROL_MAX_REGIONS_AF
);
////        int regions_ae =
characteristics.get(CameraCharacteristics.CONTROL_MAX_REGIONS_AE
);
////        int regions_awb =
characteristics.get(CameraCharacteristics.CONTROL_MAX_REGIONS_AW
B);
////
////        String af = String.valueOf(regions_af);
////        String ae = String.valueOf(regions_ae);
////        String awb = String.valueOf(regions_awb);
////
////        String characts =
String.valueOf(characteristics.get(CameraCharacteristics.INFO_SU
PPORTED_HARDWARE_LEVEL));
////
////        showToast("char: " + characts);
////        showToast("af: " + af);
////        showToast("ae: " + ae);
////        showToast("awb: " + awb);
//
//        requestBuilder.set(CaptureRequest.CONTROL_AF_TRIGGER,
CameraMetadata.CONTROL_AF_TRIGGER_CANCEL);
//        requestBuilder.set(CaptureRequest.CONTROL_AF_REGIONS,
meteringRectangleArr);
//        requestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
CaptureRequest.CONTROL_AF_MODE_AUTO);
//        requestBuilder.set(CaptureRequest.CONTROL_AF_TRIGGER,
CameraMetadata.CONTROL_AF_TRIGGER_START);
//        try {
```

```
//            mCaptureSession.capture(requestBuilder.build(),
null, null);
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        }
//    }
//
//    private void updatePreview() {
//        try {
//
mCaptureSession.capture(mPreviewRequestBuilder.build(),
mCaptureCallback, mBackgroundHandler);
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        } catch (Exception e) {
//            e.printStackTrace();
//            Log.i("updatePreview",
"ExceptionExceptionException");
//        }
//    }
//
//    private Rect calculateTapArea(final float x, final float
y, final float coefficient) {
//        int areaSize = Float.valueOf(FOCUS_AREA_SIZE *
coefficient).intValue();
//
//        int left = clamp((int) x - areaSize / 2, 0,
mTextureView.getWidth() - areaSize);
//        int top = clamp((int) y - areaSize / 2, 0,
mTextureView.getHeight() - areaSize);
//
//        RectF rectF = new RectF(left, top, left + areaSize,
top + areaSize);
//
//        return new Rect(Math.round(rectF.left),
Math.round(rectF.top), Math.round(rectF.right),
Math.round(rectF.bottom));
//    }
//
//    @Override
//        public void onViewCreated(final View view, Bundle
savedInstanceState) {
//
view.findViewById(R.id.picture).setOnClickListener(this);
```

```
//
view.findViewById(R.id.button_choose).setOnClickListener(new
View.OnClickListener() {
//            @Override
//            public void onClick(View view) {
//                Intent intent = new Intent(
//                        Intent.ACTION_PICK,
//
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
//                startActivityForResult(intent,
RESULT_LOAD_IMAGE);
//            }
//        });
//
//        mTextureView = (AutoFitTextureView)
view.findViewById(R.id.texture);
//
//        mTextureView.setOnTouchListener(new
View.OnTouchListener() {
//            public boolean onTouch(View v, MotionEvent event)
{
//
//                switch (event.getAction()) {
//                    case MotionEvent.ACTION_DOWN:
//
//
////                        try {
//                            // Auto focus should be continuous
for camera preview.
////
mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
////
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
//                            // Flash is automatically enabled
when necessary.
//                            tapToFocus(mPreviewRequestBuilder,
event);
//
//                            //updatePreview();
////
////                              // Finally, we start displaying
the camera preview.
////                              //mPreviewRequest =
mPreviewRequestBuilder.build();
```

```
////
//mCaptureSession.capture(mPreviewRequest,
////                                    // mCaptureCallback,
mBackgroundHandler);
////                        } catch (CameraAccessException e) {
////                            e.printStackTrace();
////                        }
//
//
//                        break;
//                    case MotionEvent.ACTION_UP:
//                        break;
//                }
//                return true;
//            }
//        });
//
//        spinner = (Spinner)
view.findViewById(R.id.pickImageType);
//        mDemoModeSwitch = (Switch)
view.findViewById(R.id.demo_mode);
//
//        List<String> list = new ArrayList<String>();
//        list.add("Brightfield");
//        list.add("Fluorescent");
//        ArrayAdapter<String> dataAdapter = new
ArrayAdapter<String>(getActivity(),
//                android.R.layout.simple_spinner_item, list);
//
dataAdapter.setDropDownViewResource(android.R.layout.simple_spin
ner_dropdown_item);
//        spinner.setAdapter(dataAdapter);
//
//        imageTypeListener =  new
imageTypeOnItemSelectedListener();
//        spinner = (Spinner)
view.findViewById(R.id.pickImageType);
//        spinner.setOnItemSelectedListener(imageTypeListener);
//
//    }
//
//    public File imageFilePath;
//
//    @Override
```

```
//    public void onActivityCreated(Bundle savedInstanceState) {
//        super.onActivityCreated(savedInstanceState);
//
//        long time= System.currentTimeMillis();
//
//        String timeStamp = String.valueOf(time);
//
//        String picName = "pic" + timeStamp + ".jpg";
//
//        mFile = new
File(getActivity().getExternalFilesDir(null), picName);
//
//    }
//
//    @Override
//    public void onActivityResult(int requestCode, int
resultCode, Intent data) {
//        super.onActivityResult(requestCode, resultCode, data);
//
//        if (requestCode == RESULT_LOAD_IMAGE && resultCode ==
RESULT_OK && data != null) {
//            Uri selectedImage = data.getData();
//            String[] filePathColumn = {
MediaStore.Images.Media.DATA };
//
//            Cursor cursor =
getActivity().getContentResolver().query(selectedImage,
//                    filePathColumn, null, null, null);
//            cursor.moveToFirst();
//
//            int columnIndex =
cursor.getColumnIndex(filePathColumn[0]);
//            String picturePath =
cursor.getString(columnIndex);
//            cursor.close();
//
//            Intent nextScreen = new
Intent(getActivity().getApplicationContext(),
DisplayImage.class);
//
//            nextScreen.putExtra("imagefilepath",
//                    mDemoModeSwitch.isChecked() ?
"/storage/emulated/0/DCIM/cellPictures/TSP.png" : picturePath);
//
```

```
//            startActivity(nextScreen);
//        }
//    }
//
//    @Override
//    public void onResume() {
//        super.onResume();
//        startBackgroundThread();
//
//        // When the screen is turned off and turned back on,
the SurfaceTexture is already
//        // available, and "onSurfaceTextureAvailable" will not
be called. In that case, we can open
//        // a camera and start preview from here (otherwise, we
wait until the surface is ready in
//        // the SurfaceTextureListener).
//        if (((AutoFitTextureView)
(getActivity().findViewById(R.id.texture))).isAvailable()) {
//            openCamera(((AutoFitTextureView)
(getActivity().findViewById(R.id.texture))).getWidth(),
((AutoFitTextureView)
(getActivity().findViewById(R.id.texture))).getHeight());
//        } else {
//            ((AutoFitTextureView)
(getActivity().findViewById(R.id.texture))).setSurfaceTextureLis
tener(mSurfaceTextureListener);
//        }
//    }
//
//    @Override
//    public void onPause() {
//        closeCamera();
//        stopBackgroundThread();
//        super.onPause();
//    }
//
//    private void requestCameraPermission() {
//        if
(FragmentCompat.shouldShowRequestPermissionRationale(this,
Manifest.permission.CAMERA)) {
//            new
ConfirmationDialog().show(getChildFragmentManager(),
FRAGMENT_DIALOG);
//        } else {
```

```
//            FragmentCompat.requestPermissions(this, new
String[]{Manifest.permission.CAMERA},
//                    REQUEST_CAMERA_PERMISSION);
//        }
//    }
//
//    @Override
//    public void onRequestPermissionsResult(int requestCode,
@NonNull String[] permissions,
//                                           @NonNull int[]
grantResults) {
//        if (requestCode == REQUEST_CAMERA_PERMISSION) {
//            if (grantResults.length != 1 || grantResults[0] !=
PackageManager.PERMISSION_GRANTED) {
//
ErrorDialog.newInstance(getString(R.string.request_permission))
//                        .show(getChildFragmentManager(),
FRAGMENT_DIALOG);
//            }
//        } else {
//            super.onRequestPermissionsResult(requestCode,
permissions, grantResults);
//        }
//    }
//
//    /**
//     * Sets up member variables related to camera.
//     *
//     * @param width  The width of available size for camera
preview
//     * @param height The height of available size for camera
preview
//     */
//    private void setUpCameraOutputs(int width, int height) {
//        Activity activity = getActivity();
//        CameraManager manager = (CameraManager)
activity.getSystemService(Context.CAMERA_SERVICE);
//        try {
//            for (String cameraId : manager.getCameraIdList())
{
//                characteristics
//                        =
manager.getCameraCharacteristics(cameraId);
//
```

```
//                // We don't use a front facing camera in this
sample.
//                Integer facing =
characteristics.get(CameraCharacteristics.LENS_FACING);
//                if (facing != null && facing ==
CameraCharacteristics.LENS_FACING_FRONT) {
//                    continue;
//                }
//
//                StreamConfigurationMap map =
characteristics.get(
//
CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
//                if (map == null) {
//                    continue;
//                }
//
//                // For still image captures, we use the
largest available size.
//                Size largest = Collections.max(
//
Arrays.asList(map.getOutputSizes(ImageFormat.JPEG)),
//                        new CompareSizesByArea());
//                mImageReader =
ImageReader.newInstance(largest.getWidth(), largest.getHeight(),
//                        ImageFormat.JPEG, /*maxImages*/2);
//                mImageReader.setOnImageAvailableListener(
//                        mOnImageAvailableListener,
mBackgroundHandler);
//                System.out.println("EMILIO: image reader
set");
//
//                // Find out if we need to swap dimension to
get the preview size relative to sensor
//                // coordinate.
//                int displayRotation =
activity.getWindowManager().getDefaultDisplay().getRotation();
//                //noinspection ConstantConditions
//                mSensorOrientation =
characteristics.get(CameraCharacteristics.SENSOR_ORIENTATION);
//                boolean swappedDimensions = false;
//                switch (displayRotation) {
//                    case Surface.ROTATION_0:
//                    case Surface.ROTATION_180:
```

```
//                        if (mSensorOrientation == 90 ||
mSensorOrientation == 270) {
//                            swappedDimensions = true;
//                        }
//                        break;
//                    case Surface.ROTATION_90:
//                    case Surface.ROTATION_270:
//                        if (mSensorOrientation == 0 ||
mSensorOrientation == 180) {
//                            swappedDimensions = true;
//                        }
//                        break;
//                    default:
//                        Log.e(TAG, "Display rotation is
invalid: " + displayRotation);
//                }
//
//                Point displaySize = new Point();
//
activity.getWindowManager().getDefaultDisplay().getSize(displayS
ize);
//                int rotatedPreviewWidth = width;
//                int rotatedPreviewHeight = height;
//                int maxPreviewWidth = displaySize.x;
//                int maxPreviewHeight = displaySize.y;
//
//                if (swappedDimensions) {
//                    rotatedPreviewWidth = height;
//                    rotatedPreviewHeight = width;
//                    maxPreviewWidth = displaySize.y;
//                    maxPreviewHeight = displaySize.x;
//                }
//
//                if (maxPreviewWidth > MAX_PREVIEW_WIDTH) {
//                    maxPreviewWidth = MAX_PREVIEW_WIDTH;
//                }
//
//                if (maxPreviewHeight > MAX_PREVIEW_HEIGHT) {
//                    maxPreviewHeight = MAX_PREVIEW_HEIGHT;
//                }
//
//                // Danger, W.R.! Attempting to use too large a
preview size could  exceed the camera
```

```
//                // bus' bandwidth limitation, resulting in
gorgeous previews but the storage of
//                // garbage capture data.
//                mPreviewSize =
chooseOptimalSize(map.getOutputSizes(SurfaceTexture.class),
//                        rotatedPreviewWidth,
rotatedPreviewHeight, maxPreviewWidth,
//                        maxPreviewHeight, largest);
//
//                // We fit the aspect ratio of TextureView to
the size of preview we picked.
//                int orientation =
getResources().getConfiguration().orientation;
//                if (orientation ==
Configuration.ORIENTATION_LANDSCAPE) {
//                    mTextureView.setAspectRatio(
//                            mPreviewSize.getWidth(),
mPreviewSize.getHeight());
//                } else {
//                    mTextureView.setAspectRatio(
//                            mPreviewSize.getHeight(),
mPreviewSize.getWidth());
//                }
//
//                // Check if the flash is supported.
//                Boolean available =
characteristics.get(CameraCharacteristics.FLASH_INFO_AVAILABLE);
//                mFlashSupported = available == null ? false :
available;
//
//                mCameraId = cameraId;
//                return;
//            }
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        } catch (NullPointerException e) {
//            // Currently an NPE is thrown when the Camera2API
is used but not supported on the
//            // device this code runs.
//
ErrorDialog.newInstance(getString(R.string.camera_error))
//                    .show(getChildFragmentManager(),
FRAGMENT_DIALOG);
//        }
```

```
//    }
//
//    /**
//     * Opens the camera specified by {@link
Camera2BasicFragment#mCameraId}.
//     */
//    private void openCamera(int width, int height) {
//        if (ContextCompat.checkSelfPermission(getActivity(),
Manifest.permission.CAMERA)
//                != PackageManager.PERMISSION_GRANTED) {
//            requestCameraPermission();
//            return;
//        }
//        setUpCameraOutputs(width, height);
//        configureTransform(width, height);
//        Activity activity = getActivity();
//        CameraManager manager = (CameraManager)
activity.getSystemService(Context.CAMERA_SERVICE);
//        try {
//            if (!mCameraOpenCloseLock.tryAcquire(2500,
TimeUnit.MILLISECONDS)) {
//                throw new RuntimeException("Time out waiting
to lock camera opening.");
//            }
//            manager.openCamera(mCameraId, mStateCallback,
mBackgroundHandler);
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        } catch (InterruptedException e) {
//            throw new RuntimeException("Interrupted while
trying to lock camera opening.", e);
//        }
//    }
//
//    /**
//     * Closes the current {@link CameraDevice}.
//     */
//    private void closeCamera() {
//        try {
//            mCameraOpenCloseLock.acquire();
//            if (null != mCaptureSession) {
//                mCaptureSession.close();
//                mCaptureSession = null;
//            }
```

```
//            if (null != mCameraDevice) {
//                mCameraDevice.close();
//                mCameraDevice = null;
//            }
//            if (null != mImageReader) {
//                mImageReader.close();
//                mImageReader = null;
//            }
//        } catch (InterruptedException e) {
//            throw new RuntimeException("Interrupted while
trying to lock camera closing.", e);
//        } finally {
//            mCameraOpenCloseLock.release();
//        }
//    }
//
//    /**
//     * Starts a background thread and its {@link Handler}.
//     */
//    private void startBackgroundThread() {
//        mBackgroundThread = new
HandlerThread("CameraBackground");
//        mBackgroundThread.start();
//        mBackgroundHandler = new
Handler(mBackgroundThread.getLooper());
//    }
//
//    /**
//     * Stops the background thread and its {@link Handler}.
//     */
//    private void stopBackgroundThread() {
//        mBackgroundThread.quitSafely();
//        try {
//            mBackgroundThread.join();
//            mBackgroundThread = null;
//            mBackgroundHandler = null;
//        } catch (InterruptedException e) {
//            e.printStackTrace();
//        }
//    }
//
//    /**
//     * Creates a new {@link CameraCaptureSession} for camera
preview.
```

```
//     */
//    private void createCameraPreviewSession() {
//        try {
//            SurfaceTexture texture =
mTextureView.getSurfaceTexture();
//            assert texture != null;
//
//            // We configure the size of default buffer to be
the size of camera preview we want.
//
texture.setDefaultBufferSize(mPreviewSize.getWidth(),
mPreviewSize.getHeight());
//
//            // This is the output Surface we need to start
preview.
//            Surface surface = new Surface(texture);
//
//            // We set up a CaptureRequest.Builder with the
output Surface.
//            mPreviewRequestBuilder
//                    =
mCameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW
);
//            mPreviewRequestBuilder.addTarget(surface);
//
//            // Here, we create a CameraCaptureSession for
camera preview.
//
mCameraDevice.createCaptureSession(Arrays.asList(surface,
mImageReader.getSurface()),
//                    new CameraCaptureSession.StateCallback() {
//
//                        @Override
//                        public void onConfigured(@NonNull
CameraCaptureSession cameraCaptureSession) {
//                            // The camera is already closed
//                            if (null == mCameraDevice) {
//                                return;
//                            }
//
//                            // When the session is ready, we
start displaying the preview.
//                            mCaptureSession =
cameraCaptureSession;
```

```
//                            try {
//                                // Auto focus should be
continuous for camera preview.
////
mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
////
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
//                                // Flash is automatically
enabled when necessary.
//
setAutoFlash(mPreviewRequestBuilder);
//
//                                // Finally, we start
displaying the camera preview.
//                                mPreviewRequest =
mPreviewRequestBuilder.build();
//
mCaptureSession.capture(mPreviewRequest,
//                                        mCaptureCallback,
mBackgroundHandler);
//                                System.out.println("EMILIO:
try block finished");
//                            } catch (CameraAccessException e)
{
//                                e.printStackTrace();
//                            }
//                        }
//
//                        @Override
//                        public void onConfigureFailed(
//                                @NonNull CameraCaptureSession
cameraCaptureSession) {
//                            showToast("Failed");
//                        }
//                    }, null
//            );
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        }
//    }
//
//    /**
//     * Configures the necessary {@link
android.graphics.Matrix} transformation to `mTextureView`.
```

```
//     * This method should be called after the camera preview
size is determined in
//     * setUpCameraOutputs and also the size of `mTextureView`
is fixed.
//     *
//     * @param viewWidth  The width of `mTextureView`
//     * @param viewHeight The height of `mTextureView`
//     */
//    private void configureTransform(int viewWidth, int
viewHeight) {
//        Activity activity = getActivity();
//        if (null == mTextureView || null == mPreviewSize ||
null == activity) {
//            return;
//        }
//        int rotation =
activity.getWindowManager().getDefaultDisplay().getRotation();
//        Matrix matrix = new Matrix();
//        RectF viewRect = new RectF(0, 0, viewWidth,
viewHeight);
//        RectF bufferRect = new RectF(0, 0,
mPreviewSize.getHeight(), mPreviewSize.getWidth());
//        float centerX = viewRect.centerX();
//        float centerY = viewRect.centerY();
//        if (Surface.ROTATION_90 == rotation ||
Surface.ROTATION_270 == rotation) {
//            bufferRect.offset(centerX - bufferRect.centerX(),
centerY - bufferRect.centerY());
//            matrix.setRectToRect(viewRect, bufferRect,
Matrix.ScaleToFit.FILL);
//            float scale = Math.max(
//                    (float) viewHeight /
mPreviewSize.getHeight(),
//                    (float) viewWidth /
mPreviewSize.getWidth());
//            matrix.postScale(scale, scale, centerX, centerY);
//            matrix.postRotate(90 * (rotation - 2), centerX,
centerY);
//        } else if (Surface.ROTATION_180 == rotation) {
//            matrix.postRotate(180, centerX, centerY);
//        }
//        mTextureView.setTransform(matrix);
//    }
//
```

```
//    /**
//     * Initiate a still image capture.
//     */
//    private void takePicture() {
//        lockFocus();
//    }
//
//    /**
//     * Lock the focus as the first step for a still image
capture.
//     */
//    private void lockFocus() {
//        try {
//            // This is how to tell the camera to lock focus.
////
mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_TRIGGER,
////
CameraMetadata.CONTROL_AF_TRIGGER_START);
//            // Tell #mCaptureCallback to wait for the lock.
//            mState = STATE_WAITING_LOCK;
//
mCaptureSession.capture(mPreviewRequestBuilder.build(),
mCaptureCallback,
//                    mBackgroundHandler);
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        }
//    }
//
//    /**
//     * Run the precapture sequence for capturing a still
image. This method should be called when
//     * we get a response in {@link #mCaptureCallback} from
{@link #lockFocus()}.
//     */
//    private void runPrecaptureSequence() {
//        try {
//            // This is how to tell the camera to trigger.
//
mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AE_PRECAPTURE_
TRIGGER,
//
CaptureRequest.CONTROL_AE_PRECAPTURE_TRIGGER_START);
```

```
//            // Tell #mCaptureCallback to wait for the
precapture sequence to be set.
//            mState = STATE_WAITING_PRECAPTURE;
//
mCaptureSession.capture(mPreviewRequestBuilder.build(),
mCaptureCallback,
//                    mBackgroundHandler);
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        }
//    }
//
//    /**
//     * Capture a still picture. This method should be called
when we get a response in
//     * {@link #mCaptureCallback} from both {@link
#lockFocus()}.
//     */
//    private void captureStillPicture() {
//        System.out.println("EMILIO: captureStillPicture
called");
//        try {
//            final Activity activity = getActivity();
//            if (null == activity || null == mCameraDevice) {
//                return;
//            }
//            // This is the CaptureRequest.Builder that we use
to take a picture.
//            final CaptureRequest.Builder captureBuilder =
//
mCameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_STILL_C
APTURE);
//
captureBuilder.addTarget(mImageReader.getSurface());
//
//            // Use the same AE and AF modes as the preview.
////
captureBuilder.set(CaptureRequest.CONTROL_AF_MODE,
////
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
//            setAutoFlash(captureBuilder);
//
//            // Orientation
```

```
//            int rotation =
activity.getWindowManager().getDefaultDisplay().getRotation();
//
captureBuilder.set(CaptureRequest.JPEG_ORIENTATION,
getOrientation(rotation));
//
//            CameraCaptureSession.CaptureCallback
CaptureCallback
//                    = new
CameraCaptureSession.CaptureCallback() {
//
//                @Override
//                public void onCaptureCompleted(@NonNull
CameraCaptureSession session,
//                                               @NonNull
CaptureRequest request,
//                                               @NonNull
TotalCaptureResult result) {
//                    showToast("Saved: " + mFile.toString());
//                    Log.d(TAG, mFile.toString());
//                    unlockFocus();
//                }
//            };
//
////            mCaptureSession.stopRepeating();
//            mCaptureSession.capture(captureBuilder.build(),
CaptureCallback, null);
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        }
//    }
//
//    /**
//     * Retrieves the JPEG orientation from the specified
screen rotation.
//     *
//     * @param rotation The screen rotation.
//     * @return The JPEG orientation (one of 0, 90, 270, and
360)
//     */
//    private int getOrientation(int rotation) {
//        // Sensor orientation is 90 for most devices, or 270
for some devices (eg. Nexus 5X)
```

```
//        // We have to take that into account and rotate JPEG
properly.
//        // For devices with orientation of 90, we simply
return our mapping from ORIENTATIONS.
//        // For devices with orientation of 270, we need to
rotate the JPEG 180 degrees.
//        return (ORIENTATIONS.get(rotation) +
mSensorOrientation + 270) % 360;
//    }
//
//    /**
//     * Unlock the focus. This method should be called when
still image capture sequence is
//     * finished.
//     */
//    private void unlockFocus() {
//        try {
//            // Reset the auto-focus trigger
////
mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_TRIGGER,
////
CameraMetadata.CONTROL_AF_TRIGGER_CANCEL);
//            setAutoFlash(mPreviewRequestBuilder);
//
mCaptureSession.capture(mPreviewRequestBuilder.build(),
mCaptureCallback,
//                    mBackgroundHandler);
//            // After this, the camera will go back to the
normal state of preview.
//            mState = STATE_PREVIEW;
//
mCaptureSession.setRepeatingRequest(mPreviewRequest,
mCaptureCallback,
//                    mBackgroundHandler);
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        }
//    }
//
//    @Override
//    public void onClick(View view) {
//        switch (view.getId()) {
//            case R.id.picture: {
//                takePicture();
```

```
//
//                try{
//                    Thread.sleep(2000);
//                }
//
//                catch(InterruptedException e){
//
//                }
//
//
//
//                Intent nextScreen = new
Intent(getActivity().getApplicationContext(),
DisplayImage.class);
//
//                String imageFilePath = mFile.getParent() + "/"
+ mFile.getName();
//
//                nextScreen.putExtra("imagefilepath",
imageFilePath);
//                nextScreen.putExtra("demo_mode",
mDemoModeSwitch.isChecked());
//
//                startActivity(nextScreen);
//                break;
//            }
//
//
//        }
//    }
//
//    private void setAutoFlash(CaptureRequest.Builder
requestBuilder) {
//        if (mFlashSupported) {
//
//            int imageType = imageTypeListener.imageType;
//
//            requestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
//                    CaptureRequest.CONTROL_AE_MODE_ON);
//
//            showToast("pos:" + imageType);
//
//            //1 refers to Fluorescent
//            if (imageType == 1) {
```

```
//
requestBuilder.set(CaptureRequest.SENSOR_SENSITIVITY, 200);
//            }
//            Rect cropRect = new Rect(871,1549, 1451, 2580);
//
//
requestBuilder.set(CaptureRequest.SCALER_CROP_REGION, cropRect);
//        }
//    }
//
//    public static int clamp(int val, int min, int max) {
//        if (val > max) {
//            return max;
//        }
//        if (val < min) {
//            return min;
//        }
//        return val;
//    }
//
//    /**
//     * Saves a JPEG {@link Image} into the specified {@link
File}.
//     */
//    private static class ImageSaver implements Runnable {
//
//        /**
//         * The JPEG image
//         */
//        private final Image mImage;
//        /**
//         * The file we save the image into.
//         */
//        private final File mFile;
//
//        public ImageSaver(Image image, File file) {
//            mImage = image;
//            mFile = file;
//        }
//
//        @Override
//        public void run() {
//            ByteBuffer buffer =
mImage.getPlanes()[0].getBuffer();
```

```
//            byte[] bytes = new byte[buffer.remaining()];
//            buffer.get(bytes);
//            FileOutputStream output = null;
//            try {
//                output = new FileOutputStream(mFile);
//                output.write(bytes);
//            } catch (IOException e) {
//                e.printStackTrace();
//            } finally {
//                mImage.close();
//                if (null != output) {
//                    try {
//                        output.close();
//                    } catch (IOException e) {
//                        e.printStackTrace();
//                    }
//                }
//            }
//        }
//
//    }
//
//    /**
//     * Compares two {@code Size}s based on their areas.
//     */
//    static class CompareSizesByArea implements
Comparator<Size> {
//
//        @Override
//        public int compare(Size lhs, Size rhs) {
//            // We cast here to ensure the multiplications
won't overflow
//            return Long.signum((long) lhs.getWidth() *
lhs.getHeight() -
//                    (long) rhs.getWidth() * rhs.getHeight());
//        }
//
//    }
//
//    /**
//     * Shows an error message dialog.
//     */
//    public static class ErrorDialog extends DialogFragment {
//
```

```
//        private static final String ARG_MESSAGE = "message";
//
//        public static ErrorDialog newInstance(String message)
{
//            ErrorDialog dialog = new ErrorDialog();
//            Bundle args = new Bundle();
//            args.putString(ARG_MESSAGE, message);
//            dialog.setArguments(args);
//            return dialog;
//        }
//
//        @Override
//        public Dialog onCreateDialog(Bundle
savedInstanceState) {
//            final Activity activity = getActivity();
//            return new AlertDialog.Builder(activity)
//
.setMessage(getArguments().getString(ARG_MESSAGE))
//                    .setPositiveButton(android.R.string.ok,
new DialogInterface.OnClickListener() {
//                        @Override
//                        public void onClick(DialogInterface
dialogInterface, int i) {
//                            activity.finish();
//                        }
//                    })
//                    .create();
//        }
//
//    }
//
//    /**
//     * Shows OK/Cancel confirmation dialog about camera
permission.
//     */
//    public static class ConfirmationDialog extends
DialogFragment {
//
//        @Override
//        public Dialog onCreateDialog(Bundle
savedInstanceState) {
//            final Fragment parent = getParentFragment();
//            return new AlertDialog.Builder(getActivity())
//                    .setMessage(R.string.request_permission)
```

```
//                    .setPositiveButton(android.R.string.ok,
new DialogInterface.OnClickListener() {
//                        @Override
//                        public void onClick(DialogInterface
dialog, int which) {
//
FragmentCompat.requestPermissions(parent,
//                                    new
String[]{Manifest.permission.CAMERA},
//
REQUEST_CAMERA_PERMISSION);
//                        }
//                    })
//
.setNegativeButton(android.R.string.cancel,
//                            new
DialogInterface.OnClickListener() {
//                                @Override
//                                public void
onClick(DialogInterface dialog, int which) {
//                                    Activity activity =
parent.getActivity();
//                                    if (activity != null) {
//                                        activity.finish();
//                                    }
//                                }
//                            })
//                    .create();
//        }
//
//
//    }
//
//
//
//}
```

CameraActivity.java

```
package
com.example.android.came
ra2basic;

import android.app.Activity;
import android.os.Bundle;
```

```
import android.widget.ArrayAdapter;
import android.widget.Spinner;


import java.util.ArrayList;
import java.util.List;


public class CameraActivity extends Activity {


    private Spinner spinner;
    public int imageType;
    public imageTypeOnItemSelectedListener imageTypeListener;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_camera);
        if (null == savedInstanceState) {
            getFragmentManager().beginTransaction()
                    .replace(R.id.container,
MainFragment.newInstance())
                    .commit();
        }


//        addItemsOnSpinner();
//        addListenerOnSpinnerItemSelection();
    }


    public void addItemsOnSpinner() {


        spinner = (Spinner) findViewById(R.id.pickImageType);


        List<String> list = new ArrayList<String>();
        list.add("Brightfield");
        list.add("Fluorescent");
        ArrayAdapter<String> dataAdapter = new
ArrayAdapter<String>(this,
```

```
                android.R.layout.simple_spinner_item, list);

dataAdapter.setDropDownViewResource(android.R.layout.simple_sp
inner_dropdown_item);
        spinner.setAdapter(dataAdapter);
    }


    public void addListenerOnSpinnerItemSelection() {
        imageTypeListener =  new
imageTypeOnItemSelectedListener();
        spinner = (Spinner) findViewById(R.id.pickImageType);
        spinner.setOnItemSelectedListener(imageTypeListener);



    }


}
```

CameraFragment.java

```
//package
com.example.android.camer
a2basic;

 //
 //import android.app.Activity;
 //import android.app.Fragment;
 //import android.content.Context;
 //import android.content.Intent;
 //import android.content.res.Configuration;
 //import android.graphics.ImageFormat;
 //import android.graphics.Point;
 //import android.graphics.Rect;
 //import android.graphics.SurfaceTexture;
 //import android.hardware.camera2.CameraAccessException;
 //import android.hardware.camera2.CameraCaptureSession;
 //import android.hardware.camera2.CameraCharacteristics;
 //import android.hardware.camera2.CameraDevice;
 //import android.hardware.camera2.CameraManager;
 //import android.hardware.camera2.CameraMetadata;
 //import android.hardware.camera2.CaptureRequest;
 //import android.hardware.camera2.params.MeteringRectangle;
 //import
 android.hardware.camera2.params.StreamConfigurationMap;
 //import android.media.ImageReader;
```

```
//import android.os.Bundle;
//import android.provider.MediaStore;
//import android.support.annotation.Nullable;
//import android.util.Log;
//import android.util.Size;
//import android.view.LayoutInflater;
//import android.view.MotionEvent;
//import android.view.Surface;
//import android.view.TextureView;
//import android.view.View;
//import android.view.ViewGroup;
//import android.widget.Switch;
//
//import java.util.Arrays;
//import java.util.Collections;
//
//
///**
// * Created by Emilio on 2/4/17.
// */
//
//public class CameraFragment extends Fragment {
//
//    private static final int RESULT_LOAD_IMAGE = 1;
//
//    private String mCameraId;
//
//    private AutoFitTextureView mTextureView;
//
//    private Switch mDemoModeSwitch;
//
//    private CameraCharacteristics mCharacteristics;
//
//    private ImageReader mImageReader;
//
//    private CameraDevice mCameraDevice;
//
//    private CameraCaptureSession mCaptureSession;
//
//
//
//    public static CameraFragment newInstance() {
//        return new CameraFragment();
//    }
```

```
//
//    @Override
//    public void onCreate(Bundle savedInstanceState) {
//        super.onCreate(savedInstanceState);
//
//        CameraManager cameraManager = (CameraManager)
getActivity().getSystemService(Context.CAMERA_SERVICE);
//
//        for (String cameraId :
cameraManager.getCameraIdList()) {
//
//            CameraCharacteristics characteristics =
cameraManager.getCameraCharacteristics(cameraId);
//
//            if
(characteristics.get(CameraCharacteristics.LENS_FACING) !=
null
//                    &&
characteristics.get(CameraCharacteristics.LENS_FACING) ==
CameraCharacteristics.LENS_FACING_BACK) {
//
//                StreamConfigurationMap map =
characteristics.get(CameraCharacteristics.SCALER_STREAM_CONFI
GURATION_MAP);
//
//                Size largest = Collections.max(
//
Arrays.asList(map.getOutputSizes(ImageFormat.JPEG)),
//                        new
Camera2BasicFragment.CompareSizesByArea()
//                );
//
//
//            }
//        }
//    }
//
//    /**
//     * Sets up member variables related to camera.
//     *
//     * @param width  The width of available size for camera
preview
//     * @param height The height of available size for
camera preview
```

```
//     */
//    private void setUpCameraOutputs(int width, int height)
{
//        Activity activity = getActivity();
//        CameraManager manager = (CameraManager)
activity.getSystemService(Context.CAMERA_SERVICE);
//        try {
//            for (String cameraId :
manager.getCameraIdList()) {
//                mCharacteristics
//                        =
manager.getCameraCharacteristics(cameraId);
//
//                // We don't use a front facing camera in
this sample.
//                Integer facing =
mCharacteristics.get(CameraCharacteristics.LENS_FACING);
//                if (facing != null && facing ==
CameraCharacteristics.LENS_FACING_FRONT) {
//                    continue;
//                }
//
//                StreamConfigurationMap map =
mCharacteristics.get(
//
CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
//                if (map == null) {
//                    continue;
//                }
//
//                // For still image captures, we use the
largest available size.
//                Size largest = Collections.max(
//
Arrays.asList(map.getOutputSizes(ImageFormat.JPEG)),
//                        new
Camera2BasicFragment.CompareSizesByArea());
//                mImageReader =
ImageReader.newInstance(largest.getWidth(),
largest.getHeight(),
//                        ImageFormat.JPEG, /*maxImages*/2);
//                mImageReader.setOnImageAvailableListener(
//                        new
ImageReader.OnImageAvailableListener() {
```

```
//                            @Override
//                            public void
onImageAvailable(ImageReader imageReader) {
//                                System.out.println("EMILIO:
image available");
//                                mBackgroundHandler.post(new
Camera2BasicFragment.ImageSaver(reader.acquireNextImage(),
mFile));
//                            }
//                        },
//                        mBackgroundHandler);
//                System.out.println("EMILIO: image reader
set");
//
//                // Find out if we need to swap dimension to
get the preview size relative to sensor
//                // coordinate.
//                int displayRotation =
activity.getWindowManager().getDefaultDisplay().getRotation()
;
//                //noinspection ConstantConditions
//                mSensorOrientation =
mCharacteristics.get(CameraCharacteristics.SENSOR_ORIENTATION
);
//                boolean swappedDimensions = false;
//                switch (displayRotation) {
//                    case Surface.ROTATION_0:
//                    case Surface.ROTATION_180:
//                        if (mSensorOrientation == 90 ||
mSensorOrientation == 270) {
//                            swappedDimensions = true;
//                        }
//                        break;
//                    case Surface.ROTATION_90:
//                    case Surface.ROTATION_270:
//                        if (mSensorOrientation == 0 ||
mSensorOrientation == 180) {
//                            swappedDimensions = true;
//                        }
//                        break;
//                    default:
//                        Log.e(TAG, "Display rotation is
invalid: " + displayRotation);
//                }
```

```
//
//                Point displaySize = new Point();
//
activity.getWindowManager().getDefaultDisplay().getSize(displ
aySize);
//                int rotatedPreviewWidth = width;
//                int rotatedPreviewHeight = height;
//                int maxPreviewWidth = displaySize.x;
//                int maxPreviewHeight = displaySize.y;
//
//                if (swappedDimensions) {
//                    rotatedPreviewWidth = height;
//                    rotatedPreviewHeight = width;
//                    maxPreviewWidth = displaySize.y;
//                    maxPreviewHeight = displaySize.x;
//                }
//
//                if (maxPreviewWidth > MAX_PREVIEW_WIDTH) {
//                    maxPreviewWidth = MAX_PREVIEW_WIDTH;
//                }
//
//                if (maxPreviewHeight > MAX_PREVIEW_HEIGHT)
{
//                    maxPreviewHeight = MAX_PREVIEW_HEIGHT;
//                }
//
//                // Danger, W.R.! Attempting to use too
large a preview size could  exceed the camera
//                // bus' bandwidth limitation, resulting in
gorgeous previews but the storage of
//                // garbage capture data.
//                mPreviewSize =
chooseOptimalSize(map.getOutputSizes(SurfaceTexture.class),
//                        rotatedPreviewWidth,
rotatedPreviewHeight, maxPreviewWidth,
//                        maxPreviewHeight, largest);
//
//                // We fit the aspect ratio of TextureView
to the size of preview we picked.
//                int orientation =
getResources().getConfiguration().orientation;
//                if (orientation ==
Configuration.ORIENTATION_LANDSCAPE) {
//                    mTextureView.setAspectRatio(
```

```
//                            mPreviewSize.getWidth(),
mPreviewSize.getHeight());
//                } else {
//                    mTextureView.setAspectRatio(
//                            mPreviewSize.getHeight(),
mPreviewSize.getWidth());
//                }
//
//                // Check if the flash is supported.
//                Boolean available =
characteristics.get(CameraCharacteristics.FLASH_INFO_AVAILABL
E);
//                mFlashSupported = available == null ? false
: available;
//
//                mCameraId = cameraId;
//                return;
//            }
//        } catch (CameraAccessException e) {
//            e.printStackTrace();
//        } catch (NullPointerException e) {
//            // Currently an NPE is thrown when the
Camera2API is used but not supported on the
//            // device this code runs.
//
Camera2BasicFragment.ErrorDialog.newInstance(getString(R.stri
ng.camera_error))
//                    .show(getChildFragmentManager(),
FRAGMENT_DIALOG);
//        }
//    }
//
//    @Nullable
//    @Override
//    public View onCreateView(LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
//
//        View view =
inflater.inflate(R.layout.fragment_camera2_basic, container,
false);
//
//
view.findViewById(R.id.picture).setOnClickListener(new
View.OnClickListener() {
```

```
//            @Override
//            public void onClick(View view) {
//                takePicture();
//            }
//        });
//
//
view.findViewById(R.id.button_choose).setOnClickListener(new
View.OnClickListener() {
//            @Override
//            public void onClick(View view) {
//                Intent intent = new Intent(
//                        Intent.ACTION_PICK,
//
MediaStore.Images.Media.EXTERNAL_CONTENT_URI
//                );
//                startActivityForResult(intent,
RESULT_LOAD_IMAGE);
//            }
//        });
//
//        mTextureView = (TextureView)
view.findViewById(R.id.texture);
//        mTextureView.setOnTouchListener(new
View.OnTouchListener() {
//            @Override
//            public boolean onTouch(View view, MotionEvent
motionEvent) {
//                switch (motionEvent.getAction()) {
//                    case MotionEvent.ACTION_DOWN:
//                        try {
//                            focusCamera(motionEvent);
//                        } catch (CameraAccessException e) {
//                            e.printStackTrace();
//                        }
//                }
//            }
//        });
//
//        return view;
//    }
//
//    private void focusCamera(MotionEvent event) throws
CameraAccessException {
```

```
//        CaptureRequest.Builder requestBuilder =
mCameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREV
IEW);
//        Rect rect;
//        Size size;
//        if
(CameraCharacteristics.SENSOR_INFO_ACTIVE_ARRAY_SIZE != null)
{
//            rect =
characteristics.get(CameraCharacteristics.SENSOR_INFO_ACTIVE_
ARRAY_SIZE);
//        } else {
//            rect = new Rect(0, 0, 3000, 4096);
//        }
//        Log.i("onAreaTouchEvent",
"SENSOR_INFO_ACTIVE_ARRAY_SIZE,,,,,,,,rect.left--->" +
rect.left + ",,,rect.top--->" + rect.top + ",,,,rect.right---
>" + rect.right + ",,,,rect.bottom---->" + rect.bottom);
//        if
(CameraCharacteristics.SENSOR_INFO_PIXEL_ARRAY_SIZE != null)
{
//            size =
characteristics.get(CameraCharacteristics.SENSOR_INFO_PIXEL_A
RRAY_SIZE);
//        } else {
//            size = new Size(3000, 4096);
//        }
//        Log.i("onAreaTouchEvent",
"mCameraCharacteristics,,,,size.getWidth()--->" +
size.getWidth() + ",,,size.getHeight()--->" +
size.getHeight());
//        int areaSize = 64;
//        //Rect focusRect = calculateTapArea(event.getX(),
event.getY(), 1f);
//        int right = rect.right;
//        int bottom = rect.bottom;
//        int viewWidth = mTextureView.getWidth();
//        int viewHeight = mTextureView.getHeight();
//        int ll, rr;
//        Rect newRect;
//        int centerX = (int) event.getX();
//        int centerY = (int) event.getY();
//        ll = ((centerX * right) - areaSize) / viewWidth;
//        rr = ((centerY * bottom) - areaSize) / viewHeight;
```

```
//        int focusLeft = clamp(ll, 0, right);
//        int focusBottom = clamp(rr, 0, bottom);
//        Log.i("focus_position", "focusLeft--->" + focusLeft
+ ",,,focusTop--->" + focusBottom + ",,,focusRight--->" +
(focusLeft + areaSize) + ",,,focusBottom--->" + (focusBottom
+ areaSize));
//        newRect = new Rect(focusLeft, focusBottom,
focusLeft + areaSize, focusBottom + areaSize);
//        MeteringRectangle meteringRectangle = new
MeteringRectangle(newRect, 100);
//        MeteringRectangle[] meteringRectangleArr =
{meteringRectangle};
//
//
requestBuilder.set(CaptureRequest.CONTROL_AF_TRIGGER,
CameraMetadata.CONTROL_AF_TRIGGER_CANCEL);
//
requestBuilder.set(CaptureRequest.CONTROL_AF_REGIONS,
meteringRectangleArr);
//        requestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
CaptureRequest.CONTROL_AF_MODE_AUTO);
//
requestBuilder.set(CaptureRequest.CONTROL_AF_TRIGGER,
CameraMetadata.CONTROL_AF_TRIGGER_START);
//        mCaptureSession.capture(requestBuilder.build(),
null, null);
//    }
//
//    private int clamp(int val, int min, int max) {
//        if (val > max) {
//            return max;
//        }
//        if (val < min) {
//            return min;
//        }
//        return val;
//    }
//}
```

DisplayImage.java

```
package
com.example.android.
camera2basic;
```

```
import android.Manifest;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Bitmap;
import android.nfc.Tag;
import android.os.Bundle;
import android.provider.MediaStore;
import android.support.v4.app.ActivityCompat;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;


import java.io.File;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;


import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.android.Utils;
import org.opencv.core.Mat;
import org.opencv.core.MatOfKeyPoint;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.features2d.FeatureDetector;
import org.opencv.features2d.Features2d;
import org.opencv.imgproc.Imgproc;
import android.view.View.OnClickListener;



/**
 * Created by Jonathan on 5/17/2016.
 *
```

```
 *
 */



public class DisplayImage extends Activity implements
OnClickListener,
ActivityCompat.OnRequestPermissionsResultCallback{


    static{ System.loadLibrary("opencv_java3"); }
    private static final String TAG = "MyActivity";
    public Mat imageMat = new Mat();
    public Bitmap croppedPhoto;
    public Bitmap thresholdedBitmap;
    public Mat filledContourMat = new Mat();
    public Bitmap filledContourBitmap;
    public String dir;
    public Bitmap uncroppedPhoto;
    public File imageFilePath;
    private boolean mDemoMode;


    private BaseLoaderCallback  mOpenCVCallBack = new
BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status) {
                case LoaderCallbackInterface.SUCCESS:
                {
                    Mat imageMat = new Mat();
                } break;
                default:
                {
                    super.onManagerConnected(status);
                } break;
            }


        }
```

```
    };


    public Bitmap cropToSquare(Bitmap bitmap){
        int width  = bitmap.getWidth();
        int height = bitmap.getHeight();


        //Print the size of original photo to toast for debugging
only
        Context context = getApplicationContext();
        //CharSequence text = "width:" + width + "height: " +
height;


        //Initialize the top left corner to begin cropping
        //Real
//        int cropW = 1250;
//        int cropH = 225;


        //Test
//        int cropW = 225;
//        int cropH = 1250;


        //Choose the size of the square to be cropped. Must
represent 1mm x 1mm.
        // TODO: Should be made dynamic based on the width and
height from above.
        int newWidth = (int) Math.floor(0.47238372093*width);
        int newHeight = (int) Math.floor(0.83979328165*height);


        int cropW;
        int cropH;


        if (mDemoMode) {
            //test
            cropW = (width-newWidth)/2;
            cropH = (height-newHeight)/2;
        } else {
```

```
            // real
            cropW = (width-newWidth)/2;
            cropH = (height-newHeight)/2;
        }


        CharSequence text = "width:" + newWidth + "height: " +
newHeight;


        Log.e(TAG,"height:" + newWidth + "new height:" +
newHeight);
        int duration = Toast.LENGTH_LONG;
        Toast toast = Toast.makeText(context, text, duration);
        toast.show();


        Bitmap cropImg = Bitmap.createBitmap(bitmap, cropW, cropH,
newWidth, newHeight);


        return cropImg;
    }


    //makes it black and white
    public Bitmap threshold(Bitmap bitmap){
        Utils.bitmapToMat(bitmap, imageMat);
        Imgproc.cvtColor(imageMat, imageMat,
Imgproc.COLOR_BGR2GRAY);
        //Imgproc.equalizeHist(imageMat,imageMat);
        Imgproc.GaussianBlur(imageMat, imageMat, new Size(21, 21),
0);
        Imgproc.adaptiveThreshold(imageMat, imageMat, 255,
Imgproc.ADAPTIVE_THRESH_GAUSSIAN_C, Imgproc.THRESH_BINARY_INV, 35,
4);



        Utils.matToBitmap(imageMat, bitmap);
        return bitmap;
    }
```

```
    //fills in donut like cells so we can count better
    public Bitmap fillContours(Bitmap bitmap){


        Utils.bitmapToMat(bitmap,filledContourMat);
        Imgproc.cvtColor(filledContourMat, filledContourMat,
Imgproc.COLOR_BGR2GRAY, 3);
        List<MatOfPoint> contours = new ArrayList<>();
        Mat hier = new Mat();
        Scalar white = new Scalar( 255, 255, 255 );
        Scalar red = new Scalar( 255, 0, 0 );
        Imgproc.findContours(filledContourMat,contours,hier,
Imgproc.RETR_EXTERNAL,Imgproc.CHAIN_APPROX_SIMPLE);
        Imgproc.drawContours(filledContourMat,contours,-1,white,-
1);
        Utils.matToBitmap(filledContourMat, bitmap);


        return bitmap;


    }


//    private void takeScreenshot() {
//        Date now = new Date();
//        android.text.format.DateFormat.format("yyyy-MM-
dd_hh:mm:ss", now);
//
//        try {
//            // image naming and path  to include sd card
appending name you choose for file
//            String mPath =
Environment.getExternalStorageDirectory().toString() + now +
".jpg";
//
//            // create bitmap screen capture
//            View v1 = getWindow().getDecorView().getRootView();
//            v1.setDrawingCacheEnabled(true);
//            Bitmap bitmap =
Bitmap.createBitmap(v1.getDrawingCache());
//            v1.setDrawingCacheEnabled(false);
//
```

```
//            File imageFile = new File(mPath);
//
//            FileOutputStream outputStream = new
FileOutputStream(imageFile);
//            int quality = 100;
//            bitmap.compress(Bitmap.CompressFormat.JPEG, quality,
outputStream);
//
//            outputStream.flush();
//            outputStream.close();
//            bitmap.recycle();
//
//        } catch (Throwable e) {
//            // Several error may come out with file handling or
OOM
//            e.printStackTrace();
//        }
//    }


    @Override
    public void onResume()
    {
        super.onResume();


        findViewById(R.id.process).setEnabled(true);


        if (!OpenCVLoader.initDebug()) {
            Log.d("OpenCV", "Internal OpenCV library not found.
Using OpenCV Manager for initialization");

OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_1_0, this,
mOpenCVCallBack);
        } else {
            Log.d("OpenCV", "OpenCV library found inside package.
Using it!");

mOpenCVCallBack.onManagerConnected(LoaderCallbackInterface.SUCCESS
);
        }
```

```
    }


    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[],
int[] grantResults) {


            System.out.println("It worked!");


        switch (requestCode) {
            case 1: {
                // If request is cancelled, the result arrays are
empty.
                if (grantResults.length > 0
                        && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {


                    // permission was granted, yay! Do the
                    // contacts-related task you need to do.


                } else {


                    // permission denied, boo! Disable the
                    // functionality that depends on this
permission.
                }
                return;
            }


            // other 'case' lines to check for other
            // permissions this app might request
        }
    }
```

```
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.display_image);



        if
(!OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_1_0 , this,
mOpenCVCallBack))
        {
            Log.e("TEST", "Cannot connect to OpenCV Manager");
        }
        Button button =(Button) findViewById(R.id.process);
        Button button1 = (Button) findViewById(R.id.newPicture);
        Button button2 = (Button) findViewById(R.id.savePicture);
        button.setOnClickListener(this);
        button1.setOnClickListener(this);
        button2.setOnClickListener(this);



        Date now = new Date();
        CharSequence date =
android.text.format.DateFormat.format("yyyy-MM-dd_hh:mm", now);


        ((TextView)findViewById(R.id.dateText)).setText(date);


        Intent i = getIntent();


        // Receiving the Data
        ActivityCompat.requestPermissions(this,
                new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                1);
```

```
        ImageView image = (ImageView)
findViewById(R.id.capturedImage);


        uncroppedPhoto = BitmapHolder.bitmap;
        croppedPhoto = cropToSquare(uncroppedPhoto);


                    //croppedPhoto = uncroppedPhoto;
                    image.setImageBitmap(croppedPhoto);


//        int count = 0;
//        int maxTries = 100;
//
//            while(true){
//                try {
//
//                    uncroppedPhoto =
BitmapFactory.decodeFile(imageFilePath);
//                    croppedPhoto = cropToSquare(uncroppedPhoto);
//
//                    //croppedPhoto = uncroppedPhoto;
//                    image.setImageBitmap(croppedPhoto);
//                    break;
//                } catch (Exception e) {
//                    if (++count == maxTries) throw e;
//                }
//            }



    }


    @Override
    public void onClick(View v) {


        ImageView image = (ImageView)
findViewById(R.id.capturedImage);
            switch(v.getId())
```

```
            {
                case R.id.process:
                    findViewById(R.id.process).setEnabled(false);


                    thresholdedBitmap = threshold(croppedPhoto);


                    image.setImageBitmap(thresholdedBitmap);


                    MatOfKeyPoint matOfKeyPoints1 = new
MatOfKeyPoint();
                    Mat thresholdedMat = new Mat();


                    Utils.bitmapToMat(thresholdedBitmap,
thresholdedMat);
                    Imgproc.cvtColor(thresholdedMat,
thresholdedMat, Imgproc.COLOR_BGR2GRAY, 3);


                    filledContourBitmap =
fillContours(thresholdedBitmap);


                    FeatureDetector blobDetector =
FeatureDetector.create(FeatureDetector.SIMPLEBLOB);
                    //blobDetector.write("sdcard/blob.xml");


                    Mat croppedMat = new Mat();
                    Bitmap cellsCircledBitmap;
                    Utils.bitmapToMat(croppedPhoto,croppedMat);
                    Scalar red = new Scalar(255, 0, 0);



blobDetector.read("/storage/emulated/0/DCIM/blob.xml");
                    blobDetector.detect(thresholdedMat,
matOfKeyPoints1);

Features2d.drawKeypoints(filledContourMat,matOfKeyPoints1,croppedM
at,red,Features2d.DRAW_RICH_KEYPOINTS);
```

```
                    Utils.matToBitmap(croppedMat,croppedPhoto);
                    image.setImageBitmap(croppedPhoto);


                    Context context = getApplicationContext();
                    CharSequence text = "" +
matOfKeyPoints1.height();




                    int duration = Toast.LENGTH_LONG;
                    Toast toast = Toast.makeText(context, text,
duration);
                    toast.show();



((TextView)findViewById(R.id.cellCountText)).setText(text);
                    try{
                        Thread.sleep(100);
                    }

                    catch(InterruptedException e){


                    }


                    //takeScreenshot();
                    break;


                case R.id.newPicture:
                    Intent nextScreen = new
Intent(DisplayImage.this.getApplicationContext(),
CameraActivity.class);


                    startActivity(nextScreen);
                    break;
```

```
                case R.id.savePicture:
                    if (image.getDrawable() != null) {


                        takeScreenshot();
//
MediaStore.Images.Media.insertImage(getContentResolver(),
//
((BitmapDrawable)image.getDrawable()).getBitmap(),
//
getIntent().getStringExtra("imagefilepath")+"processed.png",
null);
//
//                        String filename =
getIntent().getStringExtra("imagefilepath");
//                        Toast.makeText(getApplicationContext(),
"Saved to "+(filename.substring(0, filename.length() -
3)+"processed.png"), Toast.LENGTH_SHORT).show();
                    }
        }
    }


    private void takeScreenshot() {
        Date now = new Date();
        android.text.format.DateFormat.format("yyyy-MM-
dd_hh:mm:ss", now);


        try {
            // image naming and path  to include sd card
appending name you choose for file
//            String mPath =
Environment.getExternalStorageDirectory().toString() + "/" + now +
".jpg";


            // create bitmap screen capture
            View v1 = getWindow().getDecorView().getRootView();
            v1.setDrawingCacheEnabled(true);
            Bitmap bitmap =
Bitmap.createBitmap(v1.getDrawingCache());
```

```
                    v1.setDrawingCacheEnabled(false);



MediaStore.Images.Media.insertImage(getContentResolver(),
                            bitmap,

getIntent().getStringExtra("imagefilepath")+"processed.png",
null);


                    String filename =
getIntent().getStringExtra("imagefilepath");
                    Toast.makeText(getApplicationContext(), "Saved to
"+(filename.substring(0, filename.length() - 3)+"processed.png"),
Toast.LENGTH_SHORT).show();


                } catch (Throwable e) {
                    // Several error may come out with file handling or
OOM
                    e.printStackTrace();
                }
            }
        }
```

MainFragment.java

```
package
com.example.android.camera2b
asic;


                import android.app.Activity;
                import android.app.Fragment;
                import android.content.Intent;
                import android.database.Cursor;
                import android.graphics.BitmapFactory;
                import android.net.Uri;
                import android.os.Bundle;
                import android.os.Environment;
                import android.provider.MediaStore;
                import android.support.annotation.Nullable;
                import android.view.LayoutInflater;
                import android.view.View;
                import android.view.ViewGroup;
```

```
import android.widget.Button;


import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;


import butterknife.BindView;
import butterknife.ButterKnife;


/**
 * Created by Emilio on 2/5/17.
 */


public class MainFragment extends Fragment {


    @BindView(R.id.fragment_main_take_button) Button
mTakePictureButton;


    @BindView(R.id.fragment_main_choose_button) Button
mChoosePictureButton;


//    @BindView(R.id.fragment_main_demo_mode_switch)
Switch mDemoModeSwitch;


    private static final int REQUEST_TAKE_PICTURE = 1;


    private static final int REQUEST_CHOOSE_PICTURE = 2;


    private String mCurrentPhotoPath;
```

```
    public static MainFragment newInstance() {
        return new MainFragment();
    }


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            if (mCurrentPhotoPath == null &&
savedInstanceState.getString("photo_path") != null) {
                mCurrentPhotoPath =
savedInstanceState.getString("photo_path");
            }
        }
    }


    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
        View view =
inflater.inflate(R.layout.fragment_main, container,
false);
        ButterKnife.bind(this, view);


        mTakePictureButton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);


                if
(intent.resolveActivity(getActivity().getPackageManager())
!= null) {
                    File photoFile = null;
                    try {
                        photoFile = createImageFile();
                    } catch (IOException e) {
                        e.printStackTrace();
```

```
                    }


                    if (photoFile != null) {

intent.putExtra(MediaStore.EXTRA_OUTPUT,
Uri.fromFile(photoFile));
                        startActivityForResult(intent,
REQUEST_TAKE_PICTURE);
                    }
                }
            }
        });


        mChoosePictureButton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(
                        Intent.ACTION_PICK,

MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
                startActivityForResult(intent,
REQUEST_CHOOSE_PICTURE);
            }
        });
        return view;
    }


    @Override
    public void onSaveInstanceState(Bundle outState) {
        if (mCurrentPhotoPath != null) {
            outState.putString("photo_path",
mCurrentPhotoPath);
        }
        super.onSaveInstanceState(outState);
    }


    @Override
    public void onActivityResult(int requestCode, int
resultCode, Intent data) {
```

```
        super.onActivityResult(requestCode, resultCode,
data);

        if (resultCode == Activity.RESULT_OK) {
            switch (requestCode) {
                case REQUEST_TAKE_PICTURE:
                    try {
                        BitmapHolder.bitmap =
MediaStore.Images.Media.getBitmap(getActivity().getContent
Resolver(), Uri.parse(mCurrentPhotoPath));
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                    startDisplayImageActivity();
                    break;
                case REQUEST_CHOOSE_PICTURE:
                    Uri selectedImage = data.getData();
                    String[] filePathColumn = {
MediaStore.Images.Media.DATA };

                    Cursor cursor =
getActivity().getContentResolver().query(selectedImage,
                            filePathColumn, null, null,
null);
                    cursor.moveToFirst();

                    int columnIndex =
cursor.getColumnIndex(filePathColumn[0]);
                    String picturePath =
cursor.getString(columnIndex);
                    cursor.close();

                    BitmapHolder.bitmap =
BitmapFactory.decodeFile(picturePath);
                    startDisplayImageActivity();
                    break;
                default:
                    break;
            }
        }
```

```
    }


    private File createImageFile() throws IOException {
        // Create an image file name
        String timeStamp = new
SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
        String imageFileName = "JPEG_" + timeStamp + "_";
        File storageDir =
Environment.getExternalStoragePublicDirectory(
                Environment.DIRECTORY_PICTURES);
        File image = File.createTempFile(
                imageFileName,  // prefix
                ".jpg",         // suffix
                storageDir      // directory
        );


        // Save a file: path for use with ACTION_VIEW
intents
        mCurrentPhotoPath = "file:" +
image.getAbsolutePath();
        return image;
    }


    private void startDisplayImageActivity() {
        Intent nextScreen = new
Intent(getActivity().getApplicationContext(),
DisplayImage.class);
        startActivity(nextScreen);
    }
}
```

imageTypeOnItemSelectedListener.java

```
package
com.example.android.camera2basic;


import android.view.View;
import android.widget.AdapterView;
import
android.widget.AdapterView.OnItemSelectedListener;
import android.widget.Toast;
```

```
/**
 * Created by Jonathan on 7/30/2016.
 */
public class imageTypeOnItemSelectedListener
implements OnItemSelectedListener {


    public int imageType;
    public void onItemSelected(AdapterView<?> parent,
View view, int pos,long id) {


        imageType = pos;
        Toast.makeText(parent.getContext(),
                "OnItemSelectedListener : " + pos,
                Toast.LENGTH_SHORT).show();
    }


    @Override
    public void onNothingSelected(AdapterView<?> arg0)
{
        // TODO Auto-generated method stub
    }



}
```

Appendix 8.
Oral Cancer Detection App Code

Link to Online Repository

https://github.com/jonathankoss/oralcapp

DataHolder.java

```
package com.jonkoss.oralcancerdetection;

import android.graphics.Bitmap;

/**
 * Created by Jon on 3/30/17.
 */

public class DataHolder {

    static Bitmap bitmap_left_cy3;
    static Bitmap bitmap_right_cy3;
    static Bitmap bitmap_left_egfp;
    static Bitmap bitmap_right_egfp;

    static Double au_lesion_hbd2;
    static Double au_control_hbd2;
    static Double au_lesion_hbd3;
    static Double au_control_hbd3;

}
```

GetImageFragment.java

```
package com.jonkoss.oralcancerdetection;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.graphics.Bitmap;
```

```
import android.graphics.BitmapFactory;
import android.media.Image;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.provider.ContactsContract;
import android.provider.MediaStore;
import android.support.annotation.Nullable;
import android.support.annotation.StringDef;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;


import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;


import butterknife.BindView;
import butterknife.ButterKnife;


import static android.R.attr.bitmap;
import static android.R.attr.data;
import static android.content.ContentValues.TAG;
import static android.graphics.Color.blue;
import static android.graphics.Color.green;
import static android.graphics.Color.red;




/**
 * A simple {@link Fragment} subclass.
 * Activities that contain this fragment must implement the
```

```
 * to handle interaction events.
 * Use the {@link GetImageFragment#newInstance} factory
method to
 * create an instance of this fragment.
 */
public class GetImageFragment extends Fragment {


    @BindView(R.id.take_picture) Button mTakePictureButton;
    @BindView(R.id.choose_picture) Button
mChoosePictureButton;
    @BindView(R.id.imageView) ImageView mImageView;
    @BindView(R.id.titleText) TextView mTitleText;


    private static final int REQUEST_TAKE_PICTURE = 1;
    private static final int REQUEST_CHOOSE_PICTURE = 2;
    private String mCurrentPhotoPath;


    public String fragmentName;


    public static GetImageFragment newInstance() {
        return new GetImageFragment();
    }




    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            if (mCurrentPhotoPath == null &&
savedInstanceState.getString("photo_path") != null) {
                mCurrentPhotoPath =
savedInstanceState.getString("photo_path");
            }
        }
    }


    @Nullable
```

```
    @Override
    public View onCreateView(LayoutInflater inflater,
ViewGroup container,
                             Bundle savedInstanceState) {
        View view =
inflater.inflate(R.layout.fragment_get_image, container,
false);
        ButterKnife.bind(this, view);


        mTitleText.setText(fragmentName);



        mTakePictureButton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);


                if
(intent.resolveActivity(getContext().getPackageManager())
!= null) {
                    File photoFile = null;
                    try {


                        photoFile =
createImageFile(fragmentName);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }


                    if (photoFile != null) {

intent.putExtra(MediaStore.EXTRA_OUTPUT,
Uri.fromFile(photoFile));
```

```
                        startActivityForResult(intent,
REQUEST_TAKE_PICTURE);
                    }
                }
            }
        });


        mChoosePictureButton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(
                        Intent.ACTION_PICK,

MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
                startActivityForResult(intent,
REQUEST_CHOOSE_PICTURE);
            }
        });


        return view;
    }


    @Override
    public void onActivityResult(int requestCode, int
resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode,
data);
        Bitmap bitmap_left_cy3, bitmap_right_cy3,
bitmap_left_egfp, bitmap_right_egfp;


        int targetW = mImageView.getWidth();
        int targetH = mImageView.getHeight();


        if (resultCode == Activity.RESULT_OK) {
            switch (requestCode) {
                case REQUEST_TAKE_PICTURE:
                    try {
```

```
                        Bitmap bmp =
MediaStore.Images.Media.getBitmap(getActivity().getContentR
esolver(), Uri.parse(mCurrentPhotoPath));
                        bmp = cropToSquare(bmp);
                        ImageParams imageParams = new
ImageParams(fragmentName,bmp);
                        new
getAuTask().execute(imageParams);

mImageView.setImageBitmap(Bitmap.createScaledBitmap(bmp,
targetW, targetW, false));
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                    //startDisplayImageActivity();
                    break;
                case REQUEST_CHOOSE_PICTURE:
                    Uri selectedImage = data.getData();
                    String[] filePathColumn =
{MediaStore.Images.Media.DATA};


                    Cursor cursor =
getActivity().getContentResolver().query(selectedImage,
                            filePathColumn, null, null,
null);
                    cursor.moveToFirst();


                    int columnIndex =
cursor.getColumnIndex(filePathColumn[0]);
                    String picturePath =
cursor.getString(columnIndex);
                    cursor.close();


                    Bitmap bmp =
cropToSquare(BitmapFactory.decodeFile(picturePath));
                    ImageParams imageParams = new
ImageParams(fragmentName,bmp);
                    new getAuTask().execute(imageParams);
```

```
mImageView.setImageBitmap(Bitmap.createScaledBitmap(bmp,
targetW, targetW, false));
                    break;
                default:
                    break;
            }
        }
    }


    private Bitmap cropToSquare(Bitmap bitmap){


        Log.e(TAG, "WE CROPPING HERE");
        int width  = bitmap.getWidth();
        int height = bitmap.getHeight();
        // 4128 x 2322


        int cropW;
        int cropH;
        // real
        cropH = (int) Math.floor(height*0.30281008);
        cropW = (int) Math.floor(width*0.09689922);


        //Choose the size of the square to be cropped. Must
represent 1mm x 1mm.
        // TODO: Should be made dynamic based on the width
and height from above.
        int newHeight = (int)
Math.floor(height*0.47238372093);
        int newWidth = (int)
Math.floor(width*0.83979328165);


        Bitmap cropImg = Bitmap.createBitmap(bitmap, cropW,
cropH, newWidth, newHeight);
        Log.e(TAG, "width: " +  String.valueOf(cropW) + "
height: " + String.valueOf(cropH) + "  new width: " +
String.valueOf(newWidth) + "  new height: " +
String.valueOf(newHeight));
        return cropImg;
```

```
    }


    @Override
    public void onSaveInstanceState(Bundle outState) {
        if (mCurrentPhotoPath != null) {
            outState.putString("photo_path",
mCurrentPhotoPath);
        }
        super.onSaveInstanceState(outState);
    }


    private File createImageFile(String image_descriptor)
throws IOException {
        // Create an image file name
        String timeStamp = new
SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
        String imageFileName = image_descriptor + timeStamp
+ ".jpg";
        String storageDir =
Environment.getExternalStoragePublicDirectory(

Environment.DIRECTORY_PICTURES).getAbsolutePath();
        File image = new File(
                storageDir,   // directory
                imageFileName // file
        );


        // Save a file: path for use with ACTION_VIEW
intents
        mCurrentPhotoPath = "file:" +
image.getAbsolutePath();
        Log.e(TAG, mCurrentPhotoPath);
        return image;
    }


    private class getAuTask extends AsyncTask<ImageParams,
Void, Double> {
        @Override
        protected Double doInBackground(ImageParams...
params) {
```

```
            Bitmap bmp = params[0].bmp;
            String type = params[0].type;
            Log.e(TAG,"type: " + type);
            String color_channel = "green";


            switch (type) {
                case "lesion_hbd2":
                    color_channel = "purple";
                    break;
                case "control_hbd2":
                    color_channel = "purple";
                    break;
                case "lesion_hbd3":
                    color_channel = "green";
                    break;
                case "control_hbd3":
                    color_channel = "green";
                    break;
            }


            bmp = bmp.copy(Bitmap.Config.ARGB_8888, true);
            int intArray[], greenArray[], redArray[],
blueArray[], purpleArray[];
            int greenValue, blueValue, redValue,
purpleValue;
            int greenMin = 10000;
            int purpleMin = 10000;
            int greenMax = 0;
            int purpleMax = 0;
            double au, intensity;


            int i;
            int x = bmp.getWidth();
            int y = bmp.getHeight();


            intensity = 0;
            redValue = 0;
            greenValue = 0;
            blueValue = 0;
            purpleValue = 0;
```

```
                intArray = new int[x*y];


                bmp.getPixels(intArray, 0, x, 0, 0, x, y);



            greenArray = new int[x*y];
            blueArray = new int[x*y];
            redArray = new int[x*y];
            purpleArray = new int[x*y];
              switch (color_channel) {


                case "green":
                    for (i = 0; i < intArray.length; i++) {
                        greenArray[i] = (intArray[i] >>  8)
& 0xff;
                        //greenValue = green(intArray[i]);
                        greenMin = Math.min(greenMin,
greenArray[i]);
                        greenMax = Math.max(greenMax,
greenArray[i]);
                    }
                    Log.e(TAG, "min" +
String.valueOf(greenMin));
                    Log.e(TAG, "max" +
String.valueOf(greenMax));
                    for (i = 0; i < intArray.length; i++) {


                        intensity +=
(((double)greenArray[i] -
(double)greenMin)/((double)greenMax - (double)greenMin));
                    }
                    Log.e(TAG, "intensity" +
String.valueOf(intensity));
                    break;
                case "blue":
```

```
                    for (i = 0; i < intArray.length; i++) {
                        blueValue = blue(intArray[i]);
                        intensity += blueValue;
                    }
                    break;
                case "red":
                    for (i = 0; i < intArray.length; i++) {
                        redValue = red(intArray[i]);
                        intensity += redValue;
                    }
                    break;
                case "purple":
                    for (i = 0; i < intArray.length; i++) {
                        purpleArray[i] = blue(intArray[i])
+ red(intArray[i]);
                        purpleMin = Math.min(purpleMin,
purpleArray[i]);
                        purpleMax = Math.max(purpleMax,
purpleArray[i]);
                    }


                    for (i = 0; i < intArray.length; i++) {


                        intensity +=
(((double)purpleArray[i] -
(double)purpleMin)/((double)purpleMax -
(double)purpleMin));
                    }
                    break;
            }


            au = intensity/intArray.length;


            switch (type){


                case "lesion_hbd2":
                    DataHolder.au_lesion_hbd2  = au;

Log.e(TAG,DataHolder.au_lesion_hbd2.toString());
```

```
                    break;
                case "control_hbd2":
                    Log.e(TAG,type);
                    DataHolder.au_control_hbd2 = au;

Log.e(TAG,DataHolder.au_control_hbd2.toString());
                    break;
                case "lesion_hbd3":
                    DataHolder.au_lesion_hbd3 = au;

Log.e(TAG,DataHolder.au_lesion_hbd3.toString());
                    break;
                case "control_hbd3":
                    DataHolder.au_control_hbd3 = au;

Log.e(TAG,DataHolder.au_control_hbd3.toString());
                    break;
            }
            return au;
        }
    }


    private static class ImageParams {
        String type;
        Bitmap bmp;


        ImageParams(String type, Bitmap bmp){
            this.type = type;
            this.bmp = bmp;
        }
    }
}
```

MainActivity.java

```
package
com.jonkoss.oralcancerdetecti
on;


  import android.content.Intent;
  import android.net.Uri;
  import android.os.Environment;
  import android.provider.MediaStore;
```

```
import android.support.v4.app.FragmentManager;
import android.support.v4.content.FileProvider;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;


import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;


public class MainActivity extends AppCompatActivity {


    private static final String LOG_TAG = "MyActivity";
    private FragmentManager fragmentManager;
    static final int REQUEST_IMAGE_CAPTURE = 1;
    static final int REQUEST_TAKE_PHOTO = 1;
    String mCurrentPhotoPath;




    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        FragmentManager manager =
getSupportFragmentManager();
        GetImageFragment fragment_left_cy3 =
(GetImageFragment)
manager.findFragmentById(R.id.get_image_fragment_left_cy3
);
        fragment_left_cy3.fragmentName = "lesion_hbd2";
        fragment_left_cy3.mTitleText.setText("Lesion
HBD2");


        GetImageFragment fragment_left_egfp =
(GetImageFragment)
```

```
manager.findFragmentById(R.id.get_image_fragment_left_egf
p);
        fragment_left_egfp.fragmentName = "control_hbd2";
        fragment_left_egfp.mTitleText.setText("Control
HBD2");


        GetImageFragment fragment_right_cy3 =
(GetImageFragment)
manager.findFragmentById(R.id.get_image_fragment_right_cy
3);
        fragment_right_cy3.fragmentName = "lesion_hbd3";
        fragment_right_cy3.mTitleText.setText("Lesion
HBD3");


        GetImageFragment fragment_right_egfp =
(GetImageFragment)
manager.findFragmentById(R.id.get_image_fragment_right_eg
fb);
        fragment_right_egfp.fragmentName =
"control_hbd3";
        fragment_right_egfp.mTitleText.setText("Control
HBD3");
    }


    private File createImageFile(int imageType) throws
IOException {
        // Create an image file name


        String timeStamp = "";


        if (imageType == 1) {
            timeStamp = new
SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
        }


        String imageFileName = "JPEG_" + timeStamp + "_";
        File storageDir =
Environment.getExternalStoragePublicDirectory(
```

```
                Environment.DIRECTORY_PICTURES);
        File image = File.createTempFile(
                imageFileName,  /* prefix */
                ".jpg",         /* suffix */
                storageDir      /* directory */
        );


        // Save a file: path for use with ACTION_VIEW
intents
        mCurrentPhotoPath = image.getAbsolutePath();
        return image;
    }


}
```

MissingImageAlertFragment.java

```
package
com.jonkoss.oralcancerdetection;


import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;



/**
 * Created by jonko on 5/1/2017.
 */


public class MissingImageAlertFragment extends
DialogFragment {


    @Override
    public Dialog onCreateDialog(Bundle
savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
```

```
                .setIcon(R.drawable.androidhappy)
                // Set Dialog Title
                .setTitle("Alert DialogFragment")
                // Set Dialog Message
                .setMessage("Alert DialogFragment
Tutorial")


                // Positive button
                .setPositiveButton("OK", new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface
dialog, int which) {
                        // Do something else
                    }
                })


                // Negative Button
                .setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface
dialog,int which) {
                        // Do something else
                    }
                }).create();
    }
}
```

ProcessImagesFragment.java

```
package
com.jonkoss.oralcan
cerdetection;


import android.app.Dialog;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.graphics.drawable.ColorDrawable;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.support.v4.app.Fragment;
```

```
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;


import com.google.android.gms.plus.PlusOneButton;


import java.io.File;
import java.io.IOException;


import butterknife.BindView;
import butterknife.ButterKnife;


import static android.content.ContentValues.TAG;
import static java.lang.Double.valueOf;


/**
 * A fragment with a Google +1 button.
 * Activities that contain this fragment must implement the
 * create an instance of this fragment.
 */
public class ProcessImagesFragment extends Fragment {
    // TODO: Rename parameter arguments, choose names that match
    // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
    @BindView(R.id.process_images)Button mChoosePictureButton;
    @BindView(R.id.resultsText) TextView mResultsText;


    public ProcessImagesFragment() {
        // Required empty public constructor
    }
    // TODO: Rename and change types and number of parameters


    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            //mParam1 = getArguments().getString(ARG_PARAM1);
            //mParam2 = getArguments().getString(ARG_PARAM2);
        }
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view =
inflater.inflate(R.layout.fragment_process_images, container,
false);
        ButterKnife.bind(this, view);


        mChoosePictureButton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {


                if (DataHolder.au_lesion_hbd2 == null ||
DataHolder.au_control_hbd2 == null || DataHolder.au_lesion_hbd3 ==
null || DataHolder.au_control_hbd3 == null) {
                    final Dialog fbDialogue = new
Dialog(getActivity(), android.R.style.Theme_Black_NoTitleBar);

fbDialogue.getWindow().setBackgroundDrawable(new
ColorDrawable(Color.argb(100, 0, 0, 0)));

fbDialogue.setContentView(R.layout.missing_picture_popup);
                    fbDialogue.setCancelable(true);
                    Button bt_close =
(Button)fbDialogue.findViewById(R.id.ib_close);
                    bt_close.setOnClickListener(new
View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                            fbDialogue.dismiss();
                        }
```

```
                    });


                    fbDialogue.show();
                } else {

            Log.e(TAG,DataHolder.au_lesion_hbd2.toString());

            Log.e(TAG,DataHolder.au_lesion_hbd3.toString());

            Log.e(TAG,DataHolder.au_control_hbd2.toString());

            Log.e(TAG,DataHolder.au_control_hbd3.toString());
                    double BDI =
            (DataHolder.au_lesion_hbd3/DataHolder.au_lesion_hbd2)/(DataHolder.a
            u_control_hbd3/DataHolder.au_control_hbd2);
                    mResultsText.setText("BDI: " +
            String.format("%.2f", BDI));
                    //mResultsText.setText(String.format("%.0f",
            DataHolder.au_lesion_hbd3) + ", " + String.format("%.0f",
            DataHolder.au_lesion_hbd2) + ", " + String.format("%.0f",
            DataHolder.au_control_hbd3) + ", " + String.format("%.0f",
            DataHolder.au_control_hbd2));


                }
            }
        });
        return view;
    }
}
```

ThumbnailHolder.java

```
package
com.jonkoss.oralcancerdetection;


import android.graphics.Bitmap;


/**
 * Created by jonko on 3/31/2017.
 */
```

```
public class ThumbnailHolder {


    static Bitmap bitmap_left_cy3;
    static Bitmap bitmap_right_cy3;
    static Bitmap bitmap_left_egfp;
    static Bitmap bitmap_right_egfp;
}
```

References

[1] Knowlton, S., Yu, C., Jain, N., Ghiran, I. and Tasoglu, S. (2015). Smart-Phone Based Magnetic Levitation for Measuring Densities. *PLOS ONE*, 10(8), p.e0134400.

[2] Hebbel, R. P. Beyond hemoglobin polymerization: the red blood cell membrane and sickle disease pathophysiology. Blood 77, 214–237 (1991).

[3] Ferrone, F. A. Polymerization and sickle cell disease: a molecular view. Microcirculation 11, 115–128 (2004).

[4] Noguchi, C. T. & Schechter, A. N. Sickle hemoglobin polymerization in solution and in cells. Annu Rev Biophys Chem 14, 239–263 (1985).

[5] Nash, G. B., Johnson, C. S. & Meiselman, H. J. Mechanical properties of oxygenated red blood cells in sickle cell (HbSS) disease. Blood 63, 73–82 (1984).

[6] Brandao, M. M. et al. Optical tweezers for measuring red blood cell elasticity: application to the study of drug response in sickle cell disease. Eur J Haematol 70, 207–211 (2003).

[7] Mohandas, N. & Evans, E. Sickle erythrocyte adherence to vascular endothelium - morphologic correlates and the requirement for divalent-cations and collagen binding plasma-proteins. J Clin Inves 76, 1605–1612 (1985).

[8] Byun, H. et al. Optical measurement of biomechanical properties of individual erythrocytes from a sickle cell patient. Acta Biomaterialia 8, 4130–4138 (2012)

[9] Montes, R. A., Eckman, J. R., Hsu, L. L. & Wick, T. M. Sickle erythrocyte adherence to endothelium at low shear: role of shear stress in propagation of vaso-occlusion. Am J Hematol 70, 216–227 (2002).

[10] Hillery, C. A., Du, M. C., Montgomery, R. R. & Scott, J. P. Increased adhesion of erythrocytes to components of the extracellular matrix: isolation and characterization of

a red blood cell lipid that binds thrombospondin and laminin. Blood 87, 4879–4886 (1996).

[11] Kasschau, M. R., Barabino, G. A., Bridges, K. R. & Golan, D. E. Adhesion of sickle neutrophils and erythrocytes to fibronectin. Blood 87, 771–780 (1996).

[12] Bartolucci, P. et al. Erythrocyte density in sickle cell syndromes is associated with specific clinical manifestations and hemolysis. Blood 120, 3136–3141 (2012).

[13] Kaul, D. K., Finnegan, E. & Barabino, G. A. Sickle Red Cell-Endothelium Interactions. Microcirculation 16, 97–111 (2009).

[14] Hebbel RP. Adhesive interactions of sickle erythrocytes with endothelium. J Clin Invest 1997;100:S83–6.

[15] Hebbel RP, Boogaerts MA, Eaton JW, Steinberg MH. Erythrocyte adherence to endothelium in sickle-cell anemia. A possible determinant of disease severity. N Engl J Med 1980;302:992–5.

[16] Ballas SK, Smith ED. Red blood cell changes during the evolution of the sickle cell painful crisis. Blood 1992;79:2154–63.

[17] Connes P, Lamarre Y, Waltz X, et al. Haemolysis and abnormal hemorheology in sickle cell anaemia. Br J Haematol 2014;165: 564–72.

[18] Potoka KP, Gladwin MT. Vasculopathy and pulmonary hypertension in sickle cell disease. Am J Physiol-Lung Cell Mol Physiol 2015;308:L314–24

[19] Stuart MJ, Nagel RL. Sickle-cell disease. Lancet 2004;364: 1343–60.

[20] Kaul DK, Fabry ME, Nagel RL. Microvascular sites and characteristics of sickle cell adhesion to vascular endothelium in shear flow conditions: pathophysiological implications. Proc Natl Acad Sci U S A 1989;86:3356–60.

[21] Alapan Y, Gurkan U, ect.. Sickle cell disease biochip: a functional red blood cell adhesion assay for monitoring sickle cell disease

[22] Sheikh S, Rainger GE, Gale Z, Rahman M, Nash GB. Exposure to fluid shear stress modulates the ability of endothelial cells to recruit neutrophils in response to tumor necrosis factor-alpha: a basis for local variations in vascular sensitivity to inflammation. Blood 2003;102:2828–34.

[23] Shaik SS, Soltau TD, Chaturvedi G, et al. Low intensity shear stress increases endothelial ELR1 CXC chemokine production via a focal adhesion kinase-p38{beta} MAPK-NF-{kappa}B pathway. J Biol Chem 2009;284:5945–55.

[24] Kang H, Kwak HI, Kaunas R, Bayless KJ. Fluid shear stress and sphingosine 1-phosphate activate calpain to promote membrane type 1 matrix metalloproteinase (MT1-MMP) membrane translocation and endothelial invasion into three-dimensional collagen matrices. J Biol Chem 2011;286:42017–26.

[25] Wei, Q., Qi, H., Luo, W., Tseng, D., Ki, S., Wan, Z., Göröcs, Z., Bentolila, L., Wu, T., Sun, R. and Ozcan, A. (2013). Fluorescent Imaging of Single Nanoparticles and Viruses on a Smart Phone. *ACS Nano*, 7(10), pp.9147-9155.

[26] Ghosh, S. (2017). *Human Beta Defensins: Promising Biomarker for Oral Cancer*.

[27] Piruska, A., Nikcevic, I., Lee, S., Ahn, C., Heineman, W., Limbach, P. and Seliskar, C. (2005). The autofluorescence of plastic materials and chips measured under laser irradiation. *Lab on a Chip*, 5(12), p.1348.

[28] Lee, S., Kim, G. and Moon, J. (2013). Performance Improvement of the One-Dot Lateral Flow Immunoassay for Aflatoxin B1 by Using a Smartphone-Based Reading System. *Sensors*, 13(4), pp.5109-5116.

[29] Levin, S., Krishnan, S., Rajkumar, S., Halery, N. and Balkunde, P. (2016). Monitoring of fluoride in water samples using a smartphone. *Science of The Total Environment*, 551-552, pp.101-107.

[30] Ludwig, S., Zhu, H., Phillips, S., Shiledar, A., Feng, S., Tseng, D., van Ginkel, L., Nielen, M. and Ozcan, A. (2014). Cellphone-based detection platform for rbST biomarker analysis in milk extracts using a microsphere fluorescence immunoassay. *Analytical and Bioanalytical Chemistry*, 406(27), pp.6857-6866.

[31] Masawat, P., Harfield, A. and Namwong, A. (2015). An iPhone-based digital image colorimeter for detecting tetracycline in milk. *Food Chemistry*, 184, pp.23-29.

[32] Mora, C., Herzog, A., Raso, R. and Stark, W. (2015). Programmable living material containing reporter micro-organisms permits quantitative detection of oligosaccharides. *Biomaterials*, 61, pp.1-9.

[33] Wang, Y., Li, Y., Bao, X., Han, J., Xia, J., Tian, X. and Ni, L. (2016). A smartphone-based colorimetric reader coupled with a remote server for rapid on-site catechols analysis. *Talanta*, 160, pp.194-204.

[34] Yu, H., Tan, Y. and Cunningham, B. (2014). Smartphone Fluorescence Spectroscopy. *Analytical Chemistry*, 86(17), pp.8805-8813.

[35] Zhu, H., Sikora, U. and Ozcan, A. (2012). Quantum dot enabled detection of Escherichia coli using a cell-phone. *The Analyst*, 137(11), p.2541.

[36] McLeod, E. and Ozcan, A. (2015). Rapid imaging, detection and quantification of Giardia lamblia cysts using mobile-phone based fluorescent microscopy and machine learning. *Lab on a Chip*, 15(5), pp.1284-1293.

[37] Bhosai, S., Amza, A., Beido, N., Bailey, R., Keenan, J., Gaynor, B. and Lietman, T. (2012). Application of smartphone cameras for detecting clinically active trachoma: Table 1. *British Journal of Ophthalmology*, 96(10), pp.1350.1-1351.

[38] Wei, Q., Acuna, G., Kim, S., Vietz, C., Tseng, D., Chae, J., Shir, D., Luo, W., Tinnefeld, P. and Ozcan, A. (2017). Plasmonics Enhanced Smartphone Fluorescence Microscopy. *Scientific Reports*, 7(1).

[39] Kim, M., Kim, S., Hwang, M., Kim, J., Je, M., Jang, J., Lee, D. and Hwang, J. (2017). Multispectral imaging based on a Smartphone with an external C- MOS camera for detection of seborrheic dermatitis on the scalp. *Imaging, Manipulation, and Analysis of Biomolecules, Cells, and Tissues*.

[40] Bastawrous, A., Giardini, M., Bolster, N., Peto, T., Shah, N., Livingstone, I., Weiss, H., Hu, S., Rono, H., Kuper, H. and Burton, M. (2016). Clinical Validation of a Smartphone-Based Adapter for Optic Disc Imaging in Kenya. *JAMA Ophthalmology*, 134(2), p.151.

[41] Kim, J., Joo, H., Kim, T. and Ju, Y. (2015). A smartphone-based fluorescence microscope utilizing an external phone camera lens module. *BioChip Journal*, 9(4), pp.285-292.

[42] Zhu, H., Yaglidere, O., Su, T., Tseng, D. and Ozcan, A. (2011). Cost-effective and compact wide-field fluorescent imaging on a cell-phone. *Lab Chip*, 11(2), pp.315-322.

[43] Lee, S. and Yang, C. (2014). A smartphone-based chip-scale microscope using ambient illumination. *Lab Chip*, 14(16), pp.3056-3063.

[44] Breslauer, D., Maamari, R., Switz, N., Lam, W. and Fletcher, D. (2009). Mobile Phone Based Clinical Microscopy for Global Health Applications. *PLoS ONE*, 4(7), p.e6320.

[45] Mudanyali, O., Dimitrov, S., Sikora, U., Padmanabhan, S., Navruz, I. and Ozcan, A. (2012). Integrated rapid-diagnostic-test reader platform on a cellphone. *Lab on a Chip*, 12(15), p.2678.

[46] Stemple, C., Angus, S., Park, T. and Yoon, J. (2014). Smartphone-Based Optofluidic Lab-on-a-Chip for Detecting Pathogens from Blood. *Journal of Laboratory Automation*, 19(1), pp.35-41.

[47] Tseng, D., Mudanyali, O., Oztoprak, C., Isikman, S., Sencan, I., Yaglidere, O. and Ozcan, A. (2010). Lensfree microscopy on a cellphone. *Lab on a Chip*, 10(14), p.1787.

[48] Jonathan, E. and Leahy, M. (2010). Investigating a smartphone imaging unit for photoplethysmography. *Physiological Measurement*, 31(11), pp.N79-N83.

[49] Hossain, A., Canning, J., Ast, S., Rutledge, P., Teh Li Yen and Jamalipour, A. (2015). Lab-in-a-Phone: Smartphone-Based Portable Fluorometer for pH Measurements of Environmental Water. *IEEE Sensors Journal*, 15(9), pp.5095-5102.

[50] Raja, K., Raghavendra, R., Vemuri, V. and Busch, C. (2015). Smartphone based visible iris recognition using deep sparse filtering. *Pattern Recognition Letters*, 57, pp.33-42