2014

INTEGRATION OF INSTRUMENTATION AND PROCESSING SOFTWARE OF A LASER SPECKLE CONTRAST IMAGING SYSTEM

Jacob James Carrick
Michigan Technological University

INTEGRATION OF INSTRUMENTATION AND PROCESSING SOFTWARE OF A LASER SPECKLE CONTRAST IMAGING SYSTEM

By

Jacob J. Carrick

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Electrical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2014

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Electrical Engineering.

Department of Electrical and Computer Engineering

Thesis Advisor: *Dr. Sean Kirkpatrick*

Committee Member: *Dr. Michael Roggemann*

Committee Member: *Dr. Jingfeng Jiang*

Department Chair: *Dr. Daniel Fuhrmann*

TABLE OF CONTENTS

1. Abstract

Laser speckle contrast imaging (LSCI) has the potential to be a powerful tool in medicine, but more research in the field is required so it can be used properly. To help in the progression of Michigan Tech's research in the field, a graphical user interface (GUI) was designed in Matlab to control the instrumentation of the experiments as well as process the raw speckle images into contrast images while they are being acquired. The design of the system was successful and is currently being used by Michigan Tech's Biomedical Engineering department. This thesis describes the development of the LSCI GUI as well as offering a full introduction into the history, theory and applications of LSCI.

2. Introduction

2.1 Theory Overview

Laser speckle is a phenomenon brought about by the coherence of laser light. There are two types of speckle: subjective and objective, this paper focuses on the former. When the laser light illuminates an object the light penetrates it and the light waves are scattered by the particles that make up the object. The light waves that are collected by a photo-detector form an interference pattern. The interference pattern is generated because the waves traveling from the object to the detector have different path lengths, from scattering at different depths in the object, creating a relative phase difference between the light waves. The phase difference between the waves causes constructive and destructive interference to occur, generating the random interference pattern called speckle[1].

As stated above the speckle pattern is dependent on the path length differences of the light illuminating the object, therefore if there is motion in the area that is illuminated, the speckle pattern will fluctuate. The statistics of the speckle pattern contain information about the motion that causes this to happen. To make the fluctuation useful,

the raw speckle images are converted to speckle contrast images by the following formula:

$$K = \frac{\sigma}{\langle I \rangle}$$

Where K is contrast, σ is standard deviation and $\langle I \rangle$ is the mean intensity. The contrast formula is applied to a group of raw speckle pixels to create one speckle contrast pixel. This grouping is done throughout the raw speckle image to create a speckle contrast image. The size of the grouping of pixels that the processing software uses is important. If the group is too small then the calculated statistics are compromised. If the group is too large then the spatial resolution of the image will suffer. Groupings of 7x7 or 5x5 pixels have been found to be an acceptable compromise[2]. The grouping used in this paper's system is 7x7 pixels. Some LSCI systems instead of acquiring the statistics in the spatial domain gather statistics in the temporal domain. This is done by taking multiple images and using the same pixel in each image to calculate statistics. The temporal systems have good spatial resolution, but lack temporal resolution. The size of the speckle is also a contributing factor in acquiring good statistics. In LSCI systems the size of the speckles is determined by the wavelength of the laser (λ), the f-number of the lens system (*f*#) and the magnification (M) as shown in the equation:

$$d = 2.44\lambda f\#(M + 1)$$

The size of the speckle, as shown by Kirkpatrick and Duncan, must satisfy Nyquist criteria[3]. That is to say that the size of the speckle must be at least twice the size of the detectors pixel. A common preconception is that the size of the speckle should be the same as a pixel. Kirkpatrick and Duncan proved that at 1 pixel per speckle the maximum contrast is in the 0.7 to 0.8 range. Only when there are two or more pixels per speckle does the contrast approach the theoretical maximum of 1. Goodman developed a formula in 1965 and later corrected it in 1985 describing the relationship between the variance of the time-averaged speckle pattern and the temporal fluctuation statistics given by:

$$\sigma^2(T) = \frac{2}{T}\int_0^T \left(1 - \frac{\tau}{T}\right) C_i(\tau) d\tau$$

Where T is the exposure time, also called the integration time or shutter speed, of the camera, and $C_i(\tau)$ is the covariance of the instantaneous intensity. The equation for contrast can be expanded into the form:

$$K = \frac{\sigma}{\langle I \rangle} = \left\{\frac{\tau_c}{2T}\left[2 - \frac{\tau_c}{T}\left(1 - e^{\frac{-2T}{\tau_c}}\right)\right]\right\}^{1/2}$$

Assuming that there are a large number of scatterers and the appropriate model for the motion is a Lorentzian, meaning the flow is completely homogeneous. The decorrelation time, τ_c is related to the velocity of the scattering particles by:

$$v_c = \frac{\lambda}{2\pi\tau_c}$$

As pointed out by Duncan and Kirkpatrick, the motion can be seen as completely inhomogeneous, making the appropriate model a Gaussian[4]. This changes the equation for the contrast to:

$$K = \frac{\sigma}{\langle I \rangle} = \left\{\frac{\tau_c}{2T}\left[\sqrt{2\pi}\,\mathrm{erf}\left(\frac{\sqrt{2}T}{\tau_c}\right) - \frac{\tau_c}{T}\left(1 - e^{-2\left(\frac{T}{\tau_c}\right)^2}\right)\right]\right\}^{1/2}$$

The actual behavior of the particles is a combination of these two independent statistical models. This behavior is modeled by a convolution of a Gaussian and a Lorentzian, which is a Voigt profile.

In biological tissue the primary moving scatterers are red blood cells. Therefore, when imaging biological tissue using LSCI, the contrast (K) will be lower in the area of the image where blood is flowing. The lower the contrast, the faster the blood cells are moving. This property gives LSCI the ability to give a semi-quantitative analysis of blood flow in the imaging region.

2.2 History

In the 1960s, when coherent imaging was being researched, speckle was viewed as noise and research focused on how to minimize its effect on the image. In time, research of speckle shifted to discovering applications for laser speckle. Then in the early 1980s, the theory of using laser speckle to image retinal vasculature was developed by Fercher and Briers[5]. The team attempted to discover a technique that would non-invasively diagnose problems in the eye. One part of the diagnostic method was measurement of blood flow in the retina. At the time, the traditional method for viewing retinal blood flow was to inject a fluorescent dye and wait for the dye to be visible in the retinal blood vessels. This method left much to be desired, as it not only involved injection of potently dangerous chemicals, but required a wait period and provided only a small window of time to view to blood vessels. A new technique that could capture the blood flow non-invasively would prove to be very beneficial. The research led them to the development of what they called "single-exposure speckle photography." They were able to design a system that produced promising results, but the technology of the time limited the progress of the research. In the 1990s, the advancement in digital technology allowed research in the field of LSCI to continue[2]. Researchers were able to develop relatively simple systems that included a laser, CCD or CMOS camera, and custom processing software. As a result there have been many contributions to the field including optimization of exposure time, noise reduction, and improvement in LSCI theory. In recent years, research in LSCI has uncovered possible uses for the technique in fields such as retinal imaging, skin perfusions, neurophysiology and many more.

2.3 Applications

2.3.1 Retinal Imaging

LSCI started being developed because of the need for better retinal imaging techniques. While today there are variety of imaging techniques available for imaging

blood flow in the retina, LSCI has advantages over other systems. One such technique is fundus photography, which uses the light absorption properties of blood to view the vasculature, but gives no visualization of the degree of flow. LSCI is able to view the blood vessels and indicate relative flow values. While LSCI has its advantages, there are challenges with retinal imaging. The use of the laser must be carefully administered, as to not damage the retinal tissue. Any movement of the eye will decrease image quality[6].

2.3.2 Skin Perfusion Imaging

Skin perfusion is the distribution of blood in the skin. While there are other techniques that can view skin perfusion, they require injection of radioactive material or a scanning of the desired surface. Despite the scattering nature of skin limiting the depth of viewing, LSCI has shown promise with laser therapy of vascular lesions. LSCI has the ability to show skin perfusion in real-time before, during, and after laser therapy of the tissue[6].

2.3.3 Neurophysiology

Non-optical functional imaging techniques such as fMRI and PET scans have helped increase our understanding of the human brain. However, the spatial resolutions of these techniques are limited in comparison to optical imaging technology. While there are other optical techniques besides LSCI, they rely on contrast dyes and the absorption properties of oxy/deoxy hemoglobin. LSCI allows researchers to view the superficial blood vessels of the brain with high quality spatial resolution. LSCI has the potential to be a useful tool in not only functional imaging, but in other areas of Neurophysiology such as stroke, migraine, and cortical spreading depressions[6].

2.3.4 Other Applications

The above three section are just a few examples of applications for LSCI. In the field of dentistry LSCI could be used for the assessment of pulp. Studies have shown that LSCI techniques could diagnose burn depths. Speckle contrast techniques also have

applications beyond the medical field, such as vehicle velocity, and the assessment of drying paint.

2.4 Objectives

As described in the previous sections, LSCI has the potential to be a powerful tool in medicine. However, more research in the area of LSCI is required so that it can be used to the fullest degree capable. The research of LSCI at Michigan Tech is important to the field and for the research to progress efficiently a new system was required. The system needed to be all-inclusive and require only limited programing knowledge to use. My knowledge of programing and optics, and a desire to work in biomedical optics, made me an ideal candidate to design such a system.

The purpose of this Master's thesis is to investigate the possibility of controlling the instruments of the LSCI experiment with an easy to use Graphical User Interface (GUI). Research has been done to investigate if the GUI can be made to process the images from the experiment while the experiment is in production. This interface will be very useful to the Biomedical Engineering department's LSCI research. Students researching in this field will be able to use the experiment setup and change parameters without the need to have programming knowledge. The ability of the GUI to process images while the images are being collected will benefit the research by allowing faster data analysis.

2.5 Background

The previous LSCI experiment setup was very rough and not user friendly. The pump was controlled by a potentiometer; the images were collected using the Flycapture software and then imported into Matlab to be processed. The proposed solution to control the pump was to use Labview combined with an Arduino. The experiment used an Instech P625 peristaltic pump as shown in Figure 1.

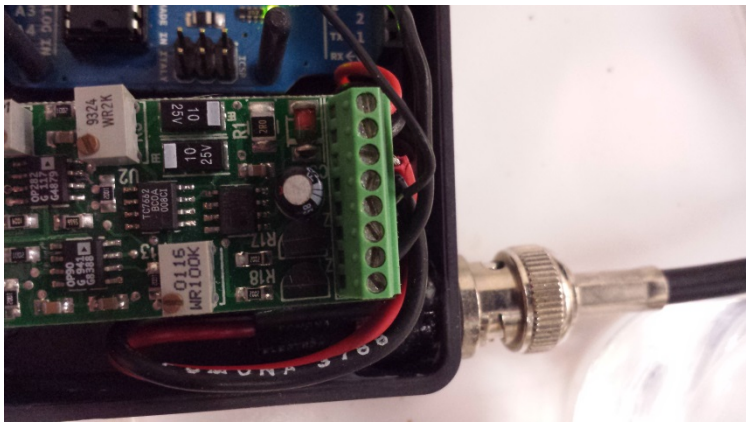


Figure 1: Instech P625 peristaltic pump

This pump is equipped with an MC200 Motor control card. At the rear of the card there are pins that control the motor of the pump. The pins are as follows starting at the pin on the bottom in Figure 1:

1. Negative reference voltage
2. Speed control input
3. Positive reference voltage
4. Power ground
5. Power ground
6. DC Power input
7. Negative Motor input
8. Positive Motor input

By studying the pin layout, the solution to control the pump became clear. A DC power supply was connected to pins 5 and 6. Pin 2 was connected to a PWM (Pulsed-Width Modulated) pin on the Arduino and pin 4 to the Arduino's ground pin. The Arduino and pump configuration is shown in Figure 2. This method was tested by writing code into the Arduino, telling it to send a PWM with a specified duty cycle to the speed control pin. The PWM was successful in controlling the pump's motor; however the method of telling the Arduino to send a PWM was not useful. Instructions, downloadable Arduino code, and Labview modules were discovered on National Instruments website for use in controlling the Arduino using Labview[7]. Using the available modules, a simple GUI

was constructed in Labview that could control the velocity at which the motor spun. As research progressed, Labview became less desirable to use because the image processing code was already written in Matlab and Michigan Tech's license for Labview did not include modules for image acquisition at the start of the research. The University's license for Matlab included the Image Acquisition Toolbox as well as the Image Processing Toolbox. These facts led to the decision that preforming the image acquisition and processing in Matlab would best serve the goal of the project. The GUI itself was made using Matlab's GUIDE tool, which allows the user to create custom GUIs using a layout editor where the user creates buttons, sliders, etc. and then automatically generates code for the user's features.

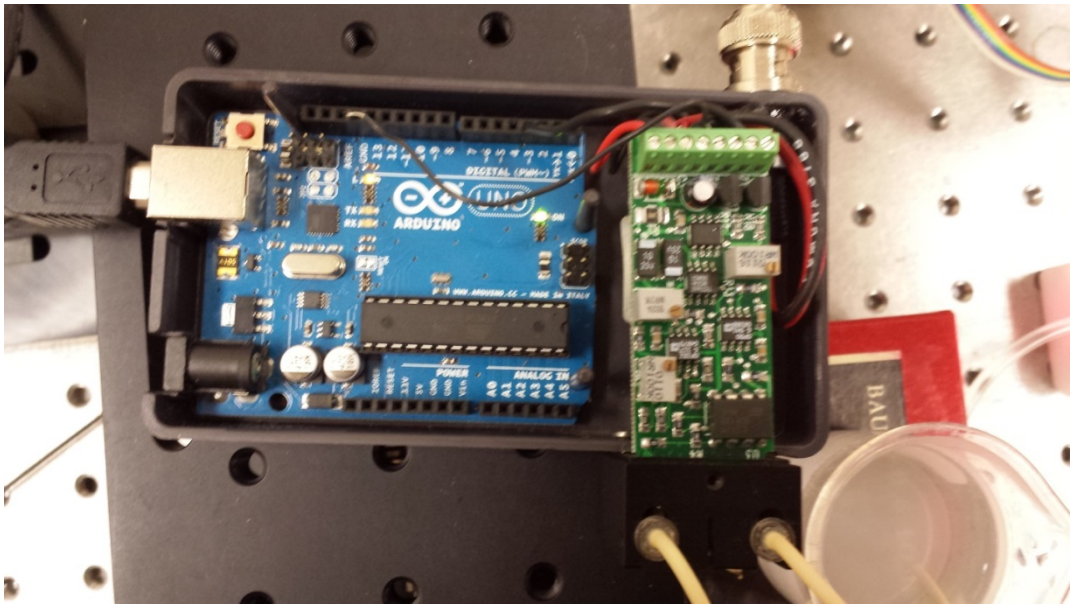


Figure 2: Pump and Arduino configuration

Investigation of Matlab's compatibility with the Dragonfly express camera yielded that the camera was compatible with Matlab's Image Acquisition Toolbox provided that the CMU Driver[8] was downloaded and installed into the camera[9]. For future experiments the camera needed to be able to capture images at frame rates approaching 200 frames per second (fps). For the camera to achieve the desired frame rate several conditions were required. First, the camera had to be connected to the computer using an IEEE-1394b cable and a compatible host card. The lab was equipped

with both the cable and host card meeting this condition. Second, the computer's operating system needed to allow the cable and card to transfer data at maximum capacity (800 Mb/s). The second condition proved to be a challenge as the research computer's operating system was Windows XP with service package 3 which limited the data speed to 100 Mb/s. However, Point Grey Research provided a free download application that forces Windows XP to downgrade to service package 1, which does not limit the data speed. Condition 2 was later avoided when the computer's hard drive started to fail and Information Technology (IT) installed a new hard drive with Windows 7. Third, the camera needed to be configured to the partial image capture configuration Format_7, Mode_0, Mono8, with 648x484 pixels[10]. The camera configuration is performed in the GUI code inside the Initialize camera call back. The previously described configuration used two properties to control the camera's frame rate: the Normalized Bytes per Packet (currently set at 56) and the Region of Interest (currently set at 640x480). The current values for the two properties allows for a frame rate of 110 fps. However, higher frame rates are possible by decreasing the Region of Interest and increasing the Normalized Bytes per Packet.

3. Methods

The first step taken to create the GUI was to simply get it to initialize the camera and have the video feed displayed inside the GUI. First, an area in the GUI was needed for displaying the images. This was accomplished by creating an axis area using the axis tool inside GUIDE. The properties of the axis area were then changed so that the area was made of 640x480 pixels. A control panel was created in the GUI area to house the various camera controls that would be added later. This control panel was named "Camera Control Panel." A button was then added to the control panel for initializing the camera, these features can be seen in Figure 3. By saving the layout of the GUI in GUIDE an m-file for the code of the GUI was automatically generated. The m-file included code that will initialize the GUI, a function that runs before the GUI appears on the screen, a function that runs when the GUI is closed, and a function for the "Initialize Camera" button that was created. In the opening function, the imaqreset command is

called to reset all image acquisition functions, a variable for the video input object and a variable for the camera information were defined (camobj and camsrc respectively). In the closing function, code was added to delete the camera object to insure that the next time the GUI was opened there would be no left over information from the previous session. In the initialize camera function, code was written to setup the camera object.

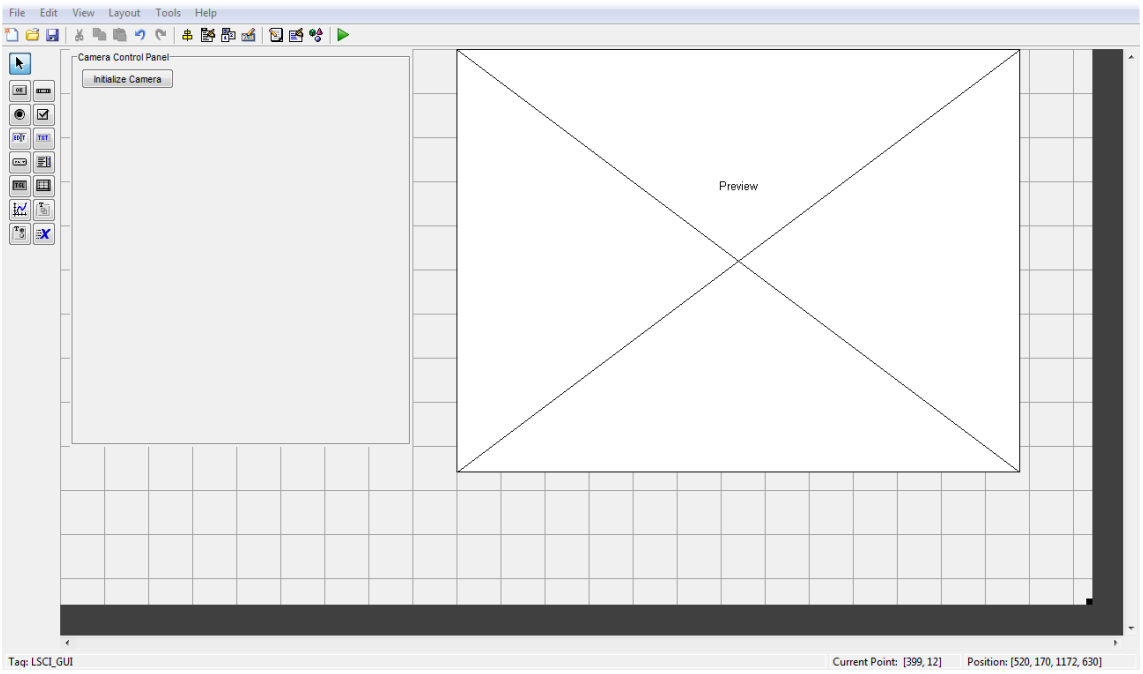


Figure 3: GUI Layout 1

What was needed next was code that would stream images from the camera to the image space in the GUI's layout. The Image Acquisition Toolbox includes the "Preview" command which can be used to that desired end. The following code was then written to place the live video stream in the image area:

```
*[ vidRes = get(camobj, 'ROIPosition');
set(handles.Preview,'Position',[450.0+vidRes(1)-4 150.0+vidRes(2)-2 vidRes(3)
vidRes(4)]);
% image width
imWidth = vidRes(3);
% image height
imHeight = vidRes(4);
% create an empty image container and show it on Preview
hImage = image(zeros(imHeight, imWidth), 'parent', handles.Preview);
% begin preview
preview(camobj, hImage);]*
```

The above code saved the height and width of the camera's region of interest (ROI) to variables. A white image the size of the camera's ROI was created in the image area of the GUI by referencing the area's tag (the tag was titled "Preview"). The image space was then used with the preview command to place the video feed inside itself.

The frame rate that the camera was operating at was important to the LSCI research, therefore a way to calculate and display the camera's frame rate was needed. A "Measure Frame rate" button and a small text area were created. The background color of the text area was changed to cyan to create a box in order to set it apart from the other text in the panel. The new panel design is shown in Figure 4. Code was written in the measure frame rate function that started the camera and collected 50 images. The timestamp of the images was then used to calculate the frame rate of the camera. The calculated frame rate was then displayed in the text area.

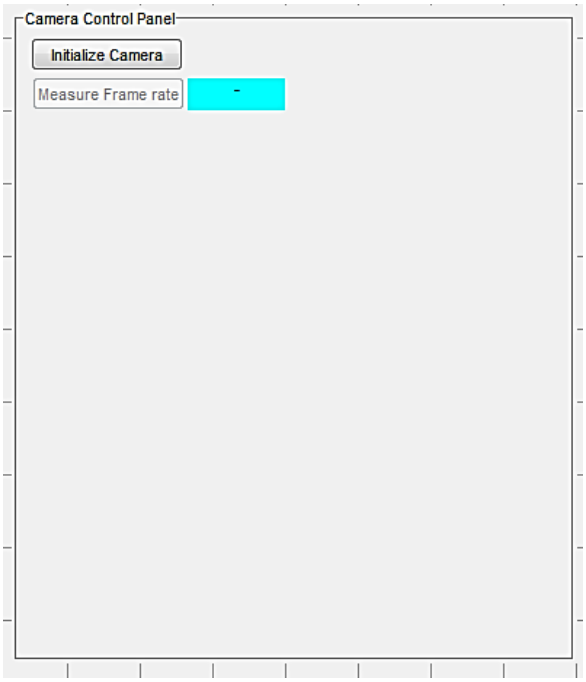


Figure 4: GUI Layout 2

The next objective to be worked on was the GUI's ability to collect and process images. The key to achieving this was to understand how Matlab collects images and how the user can access the stored images. When Matlab collects images, they are stored in a section of memory specifically set aside. Also when the command is called to collect images, the function starts the acquisition process and continues down the line of code. When the "getdata" command is used to access an image, that image is removed from the acquisition memory and placed into a different section of memory. When "getdata" is used to access only one image it takes the oldest image in the acquisition memory. To test the above information the acquisition and processing function was written to start acquiring an infinite amount of images and then an infinite loop was coded to get one image at a time, process the image, and display the contrast image in the GUI's image space. The test was successful at displaying contrast images until the program ran out of memory. To control how many frames were acquired an edit box was added to the GUI with text asking how many frames the user would like to acquire. The processing function was edited to set the frames per trigger property of the camera and the number of times the processing loop would run equal to the number in the edit box at the time the

processing button was pressed. Now that the GUI could process images and display them in the image space of the GUI, a button was added to change the image space to display the live video feed. Changes to the control panel are shown in Figure 5. The function for this button had the same code as the code that achieved this purpose in the initialize camera function.

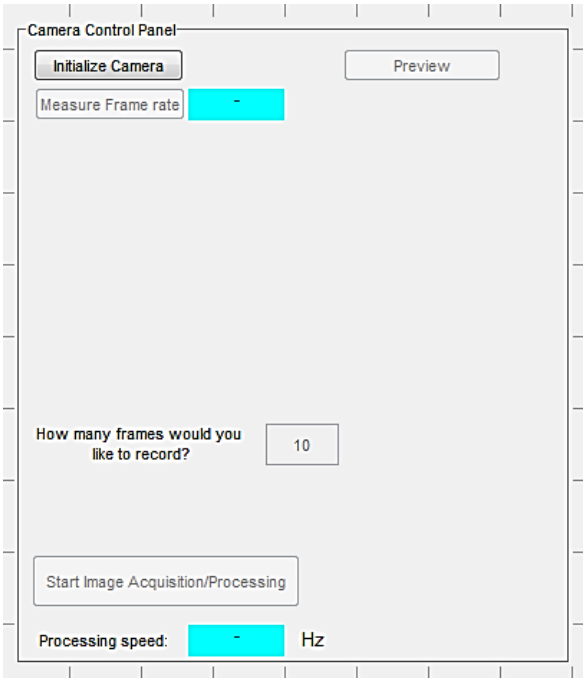


Figure 5: GUI Layout 3

The ability of the GUI to save both the raw speckle and contrast images was expressed. To achieve this, two arrays were created: one for raw speckle images and another for the contrast images. The raw speckle image was saved to the array inside the processing loop before any matrix manipulation was performed and once the image was converted to a contrast image it was saved again but to the other array. The images could have been saved to image files inside the processing loop, but this would slow down the GUI's ability to process images. To prevent lag in the image processing a "Save Images" button was added to the GUI. When pressed the button would run a loop saving the raw speckle and contrast images to individual image files. In order to prevent errors the "Save Images" button was only enabled after all the images had been processed. The initial design of the "Save Images" button simply saved the images to the folder that the

Matlab directory was currently in and if another set of images were saved to this same folder the previous set of images would be overwritten.

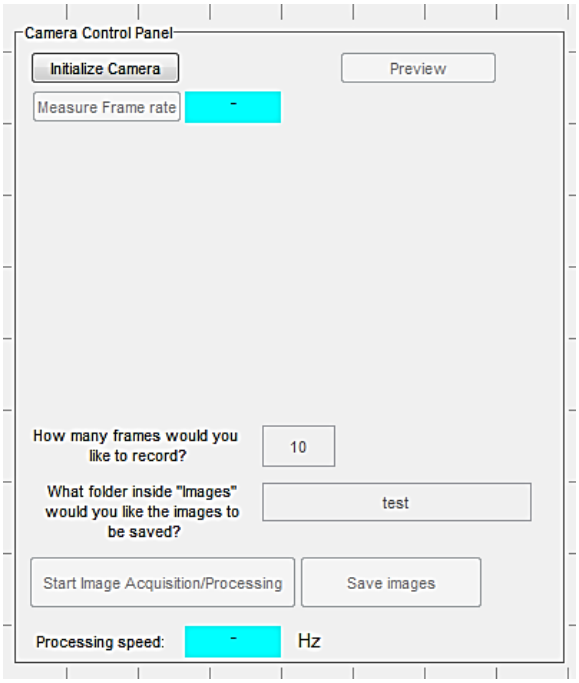


Figure 6: GUI Layout 4

A way to tell the GUI where to save the images was attempted. An edit box was added to the GUI (see Figure 6) allowing the user to enter in the path name of the folder they wished to save the images in. String addition was then used to combine the path name of the folder and the name of the image file. The testing of this method resulted in errors in Matlab. The text that was written in the edit box was rapped to the next line part way through. To fix the problem the majority of the path name was written in the code and then the folder the user wanted to save to was taken from the edit box. This method worked but, the user needed to create the folder they wanted the images saved to before pressing the "Save Images" button.

For the contrast images to be useful to the research, the various settings for the camera needed to be controlled manually by the GUI's user. The camera settings included exposure, gain, shutter, gamma, and brightness. To control the settings, an edit box and slider tool was created for each of the five settings. Inside the camera initialization function, code was added to change the mode of each setting to "manual"

and the value type to “absolute”. The “absolute” value type was used so that the user could know the real values of each setting instead of the relative value. Using the properties of each setting code was added to the initialize camera function to set the maximum and minimum value of each slider. The default values for each setting were defined in the initialize camera function and then used to set the starting value of each edit box, slider, and camera setting. The function of each edit box was coded to take the newly entered value equal to the camera setting value and then change the position of the corresponding slider to that of the new value. The slider functions were written so when the slider was moved to a new value the setting was changed to that value and the corresponding edit box displayed the new value. So that the setting could be returned to their starting values, the “Set Default Setting” button was created in the GUI. The function for this button was written to return the values of each setting, edit box, and slider to the default values defined in the initialize camera function. The functionality of the edit boxes and sliders were tested by changing the values of the settings inside the GUI while the image space was showing live video.

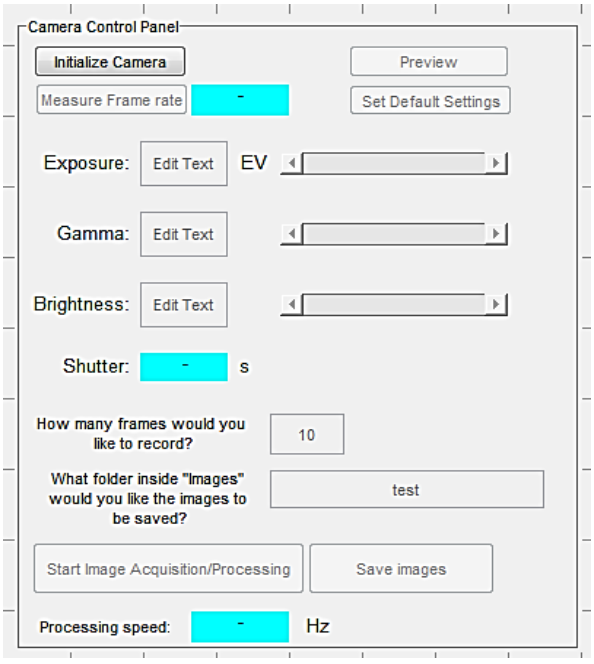


Figure 7: GUI Layout 5

Changing the values seemed to work, however there were some irregularities. One such anomaly was that if the shutter was changed in the GUI, the preview screen would go black until the shutter time was returned to its starting value. If the starting value of the shutter speed was changed in the GUI's code there were no problems. A solution or explanation for this phenomenon was never discovered but, the issue was found to be acceptable. The slider for the shutter time was removed and the edit box was changed to a display only text box (Figure 7). A more serious problem was also identified during testing; the changes in the camera settings were not appearing in the image files. As proof, two sets of images were saved, one where the gamma was set to maximum and the other with the gamma at the minimum. The image quality of both sets of images was identical. The problem appeared to be that when the start command was called, the camera was changing the camera settings to their automatic mode. However, Matlab indicated that each setting was still in manual. This lead to the conclusion that the camera was resetting the controls to their automatic modes after the start command in Matlab was executed. By reading through the Dragonfly technical reference manual[11], a possible explanation was discovered. In the manual it explains that when exposure is set to manual the camera's embedded programming changes the value of the shutter speed and/or gain to achieve the user's desired exposure value. This discovery indicated that because all three of those controls are set to manual, they could cause an error in the camera's programming causing it to reset all three controls to their automatic mode before capturing images. In an attempt to fix the problem, the gain controls were deleted from the interface and the code for the gain controls were commented out. The gain code was commented out and not deleted so that in the future if controlling the gain was found to be more important than the exposure, the controls could simply be added to the interface without retyping code. Even with these changes to the GUI, the problem was not fixed. Point Grey Research (PGR) was then contacted to see if they could come up with an explanation for the problem. While waiting for a response from PGR, a programming solution was found. By setting the camera object's trigger type to manual, the camera would wait to acquire images until the trigger command was executed. The values for each camera control were set after the start command and then the trigger

command was called to collect images. This solution was implemented wherever the start or preview commands were used in the program.

With all of the tasks related to the camera completed, the ability for the GUI to control the experiments pump was investigated. A search on MathWork's website yielded downloadable server code for the Arduino and Matlab functions for controlling the Arduino[12]. A control panel to house the pump controls was created in the GUI's interface. The following features were placed inside the control panel: an edit box so the user could enter their desired speed, a button to set the speed of the pump, a text box to display the actual speed of the pump, and a button to stop the pump, as shown in Figure 8.

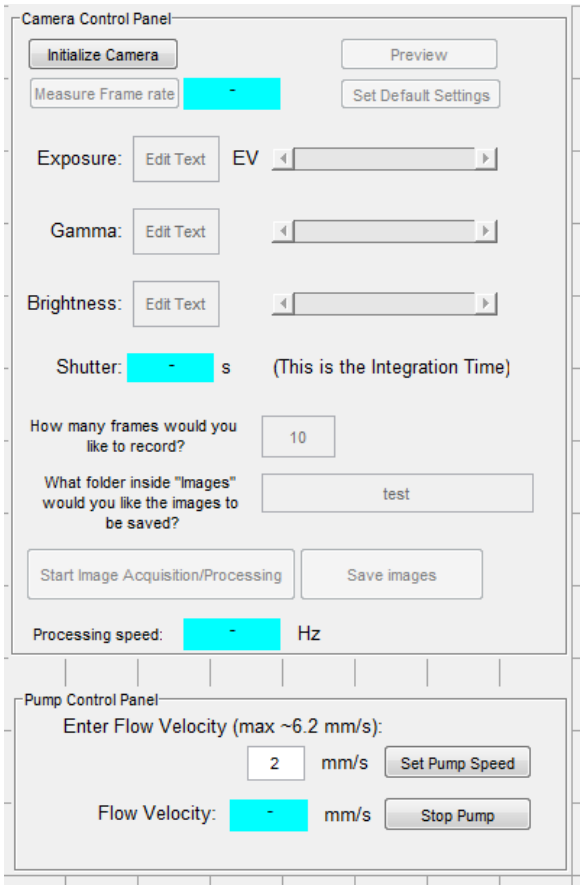


Figure 8: GUI Layout 6

To create a connection between Matlab and the Arduino, the following code was added to the opening function of the GUI:

[handles.arduino=arduino('COM3');]

Code was written into the "Set Pump Speed" button's function to grab the value in the edit box and convert it to the equivalent duty cycle value. This was done by first relating the flow velocity to the appropriate voltage on the speed control pin to achieve that velocity. The relationship was found experimentally by an undergraduate student that works in the lab. His experimental data is shown in Figure 9. The Relationship does not span over the whole voltage range. Based on observation, the speed of the pump reaches a maximum of approximately 6.2 mm/s, despite an increase in voltage.

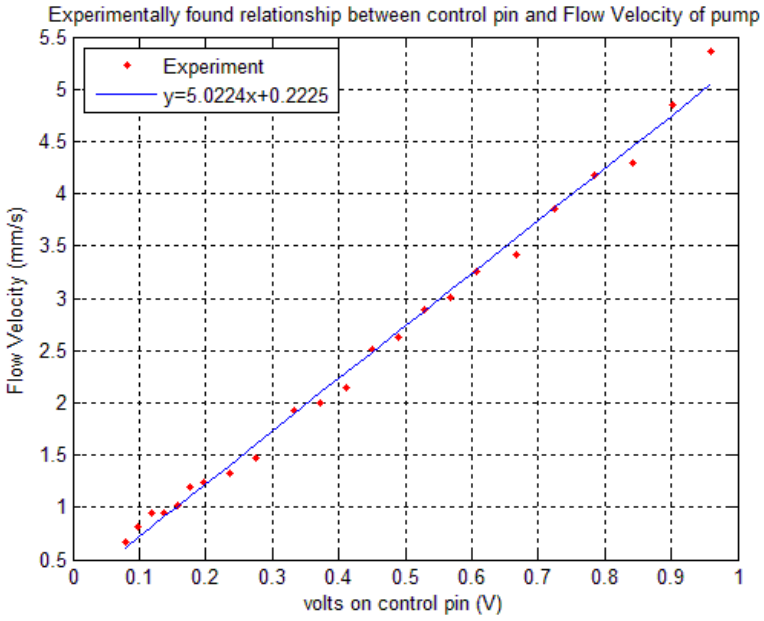


Figure 9: Flow Velocity Graph

The voltage was then converted to a duty cycle value by the following equation:

$$duty\ cycle = \frac{255}{5} * voltage$$

The duty cycle value must be an integer, so the calculated value is rounded and then used to send a PWM from the Arduino to the speed control pin on the pump. Because the value is rounded, it is converted back to a speed value and displayed in the GUI. Code was then added to the "Stop Pump" button's function to send a zero voltage PWM. This

code was also added to the GUI's closing function. With this, the programming of the GUI was completed.

4. Results

When the LSCI_GUI m-file is executed, Matlab will first attempt the connection between Matlab and the Arduino. If the connection is successful the GUI's window will open, revealing a blank image space; all features in the camera control panel are disabled except for the "Initialize Camera" button and all features of the pump control panel, this is shown in Figure 10.

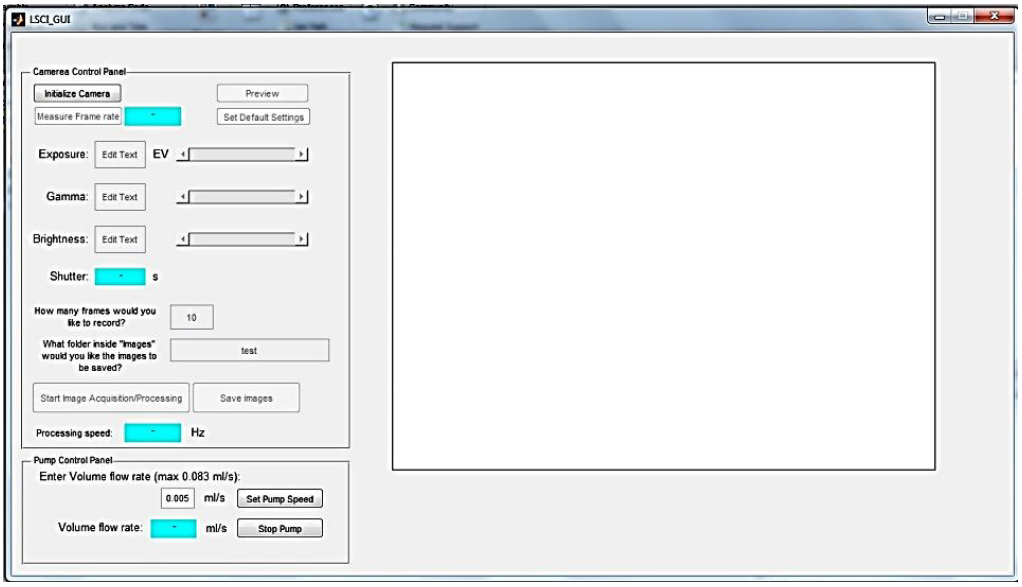


Figure 10: GUI when first opened

Once the "Initialize Camera" button is pressed, the image space area will show a live feed of the camera's view and all features of the camera control panel will be enabled with the exception of the "Save Images" button, the folder edit box, and the "Initialized Camera" button itself, as shown in Figure 11.

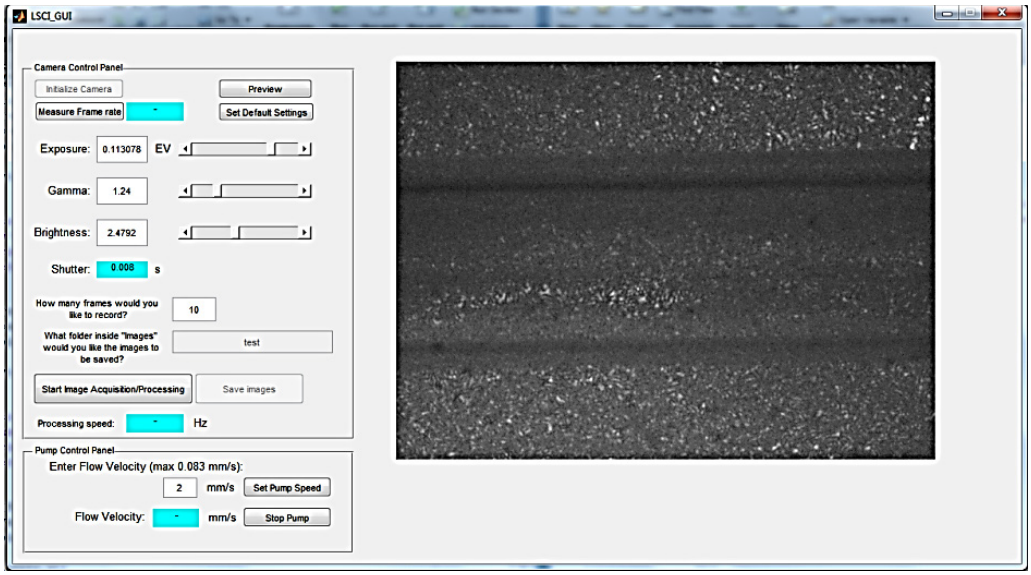


Figure 11: GUI after camera Initialization

The LSCI GUI is now able to control the camera to an extent. The camera's exposure, gamma, and brightness can be controlled inside the GUI by entering numbers into the appropriate edit box or moving the setting's slider tool. The camera's shutter speed can also be changed, but to do so the user must change the shutter's default value on line 64 of the GUI's code.

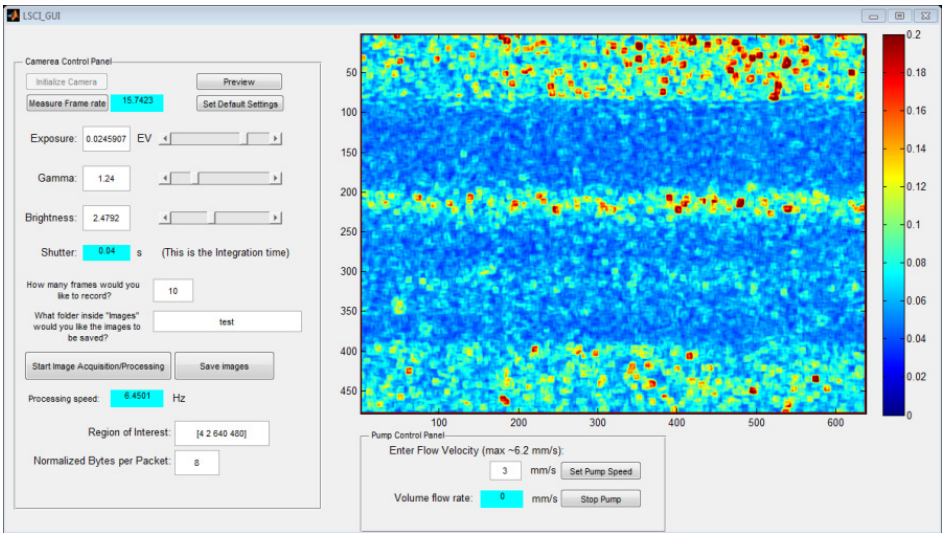
[handles.shutter_default=.002;]

The GUI has a button that will return all camera settings to their starting values when pressed.

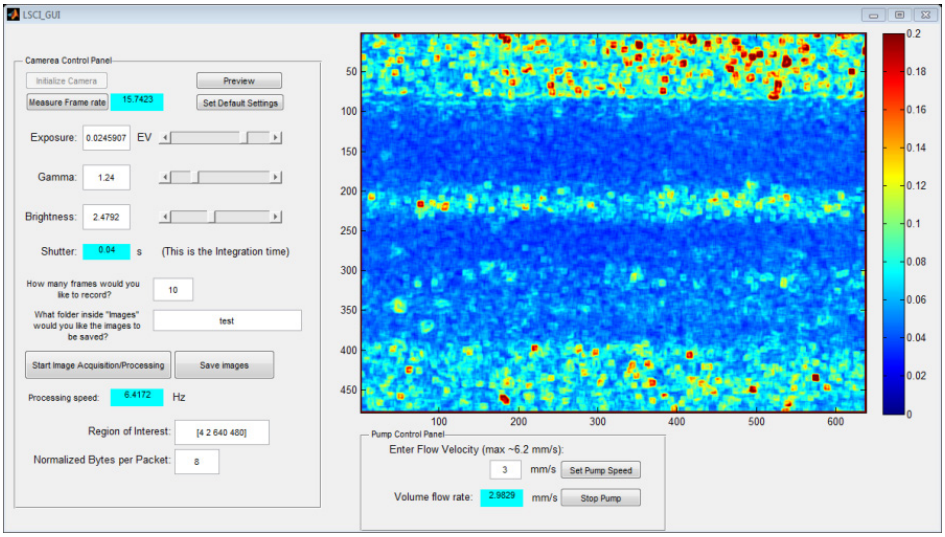
The GUI has the ability for the user to designate the number of frames they wish to collect, then collect and process those frames while the camera is still collecting images. The user can then save the set of raw speckle images and contrast images to a designated folder, given that the designated folder has been created before the save images button is pressed. The image space of the GUI can be changed back to the live video feed by pressing the "Preview" button and the camera's frame rate can be measured at any time (except when the GUI is processing images) by pressing the "Measure Frame rate" button.

The user is able to control the experiment's pump by using the GUI's pump control panel. The speed of the pump can be changed by entering an appropriate flow velocity in the edit box and pressing the "Set Pump Speed" button. The actual flow of the pump is displayed in the text box below the edit box. The pump can be stopped by pressing the "Stop Pump" button or by closing the GUI's window.

To test the functionality of the system, a sample with 0.92% concentration of Luxsil glass micro-spheres was pumped through the system. Figure 12a shows the contrast image of the tube full of the sample, but with the pump stopped. Figure 12b shows the contrast image of the tube with the pump moving the sample at 3 mm/s. By visually comparing the two images, image 12b is clearly darker in the tube area, indicating movement of particles.



(a)



(b)

Figure 12: (a) Pump off, (b) Pump on

5. Future Recommendations

While the current LSCI GUI is functioning well, there are improvements that could be made. Parallel processing could be investigated to increase the programs processing speed. Additional features and functions could be added to the system to preform calculations required for the research. For example the GUI could be programed to calculate the speckle correlation time (τ) and plot the contrast (K) vs the quotient of the integration time and correlation time (τ/T). Due to time constraints, these tasks could not

be achieved, but the foundation of the program allows for additional features to be added with minimal difficulty.

6. Conclusion

The objective of the project was to investigate and implement an all-encompassing GUI that would control the instrumentation and processing of the LSCI experiment. All major tasks of the project where accomplished which includes proper setup of the camera, control of the camera settings, acquiring images, processing images, displaying contrast images as they are processed, saving both raw speckle and contrast images to a designated folder, and the ability to control the speed of the experiment's pump. While there are some improvements that can be made to the system in the future, it is working properly and is currently being used by the Biomedical Engineering Department for LSCI research.

7. References

[1] D. A. Boas and A. K. Dunn, "Laser speckle contrast imaging in biomedical optics," *Journal of Biomedical Optics,* vol. 15, pp. 011109-011109-12, 2010.

[2] D. Briers, D. D. Duncan, E. Hirst, S. J. Kirkpatrick, M. Larsson, W. Steenbergen*, et al.*, "Laser speckle contrast imaging: theoretical and practical limitations," *Journal of Biomedical Optics,* vol. 18, pp. 066018-066018, 2013.

[3] S. J. Kirkpatrick, D. D. Duncan, and E. M. Wells-Gray, "Detrimental effects of speckle-pixel size matching in laser speckle contrast imaging," *Optics Letters,* vol. 33, pp. 2886-2888, 2008/12/15 2008.

[4] D. D. Duncan and S. J. Kirkpatrick, "Can laser speckle flowmetry be made a quantitative tool?," *JOSA A,* vol. 25, pp. 2088-2094, 2008.

[5] J. D. Briers, "Laser speckle contrast imaging for measuring blood flow," *Optica Applicata,* vol. 37, 2007.

[6] J. Senarathna, A. Rege, L. Nan, and N. V. Thakor, "Laser Speckle Contrast Imaging: Theory, Instrumentation and Applications," *Biomedical Engineering, IEEE Reviews in,* vol. 6, pp. 99-110, 2013.

[7] N. Instruments. *NI LabVIEW Interface for Arduino Toolkit*. Available: http://sine.ni.com/nips/cds/view/p/lang/en/nid/209835

[8] (1999). *CMU 1394 Digital Camera Driver*. Available: http://www.cs.cmu.edu/~iwan/1394/

[9] P. Grey. Using MATLAB with Point Grey cameras. [Online]. Available: http://www.ptgrey.com/support/kb/index.asp?a=4&q=218&ST=matlab

[10] P. Grey. How to achieve 200 FPS using a Dragonfly Express. [Online]. Available: http://www.ptgrey.com/support/kb/index.asp?a=4&q=236&ST=dragonfly+express

[11] P. G. Research. (2004). *<DragonflyTechnicalReference.pdf>*.

[12] M. C. R. Team. (2011). *MATLAB Support Package for Arduino (aka ArduinoIO Package)*. Available: http://www.mathworks.com/matlabcentral/fileexchange/32374-matlab-support-package-for-arduino-aka-arduinoio-package

8. Appendix A: Matlab Code

```
function varargout = LSCI_GUI(varargin)
% LSCI_GUI MATLAB code for LSCI_GUI.fig
%      LSCI_GUI, by itself, creates a new LSCI_GUI or raises the existing
%      singleton*.
%
%      H = LSCI_GUI returns the handle to a new LSCI_GUI or the handle to
%      the existing singleton*.
%
%      LSCI_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in LSCI_GUI.M with the given input arguments.
%
%      LSCI_GUI('Property','Value',...) creates a new LSCI_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before LSCI_GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to LSCI_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help LSCI_GUI

% Last Modified by GUIDE v2.5 27-Mar-2014 12:36:32

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @LSCI_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @LSCI_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before LSCI_GUI is made visible.
function LSCI_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to LSCI_GUI (see VARARGIN)

%resets image acquisition
imaqreset;
%global variables
global camobj;
global camsrc;

%Frame rate is controlled by NBP (multiples of 8) and ROI
handles.NBP= 64;
handles.ROI=[4 2 640 480];

%set default values for controls CHANGE SHUTTER HERE!!!!!
handles.exposure_default=-3.42625;
handles.shutter_default=.006;
%handles.gain_default=30;
handles.gamma_default=1.24;
handles.brightness_default=2.47922;
%tells matlab what port the arduino is plugged into
handles.arduino=arduino('COM3');
% Choose default command line output for LSCI_GUI
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes LSCI_GUI wait for user response (see UIRESUME)
% uiwait(handles.LSCI_GUI);


% --- Outputs from this function are returned to the command line.
function varargout = LSCI_GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes when user attempts to close LSCI_GUI.
function LSCI_GUI_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to LSCI_GUI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
global camobj;
global camsrc;
%stops pump by sending zero volts to control pin
handles.arduino.analogWrite(3,0);
delete(camobj);
delete(hObject);


% --- Executes on button press in InitCam.
function InitCam_Callback(hObject, eventdata, handles)
% hObject    handle to InitCam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global camobj;
global camsrc;
%enable gui interactive functions
set(handles.ImaqProcess,'Enable','on');
set(handles.MeasFrame,'Enable','on');
set(handles.InitCam,'Enable','off');
set(handles.edit1,'Enable','on');
set(handles.exposure,'Enable','on');
set(handles.region,'Enable','on');
set(handles.packet,'Enable','on');

%set(handles.gain,'Enable','on');
set(handles.gamma,'Enable','on');
set(handles.brightness,'Enable','on');
set(handles.exposure_slider,'Enable','on');
%set(handles.gain_slider,'Enable','on');
set(handles.gamma_slider,'Enable','on');
set(handles.brightness_slider,'Enable','on');
set(handles.PreviewCam,'Enable','on');
set(handles.SetDefault,'Enable','on');
%Create camera object with format
camobj = videoinput('dcam',1,'F7_Y8_648x484');
%variable with information about the camera settings
camsrc = getselectedsource(camobj);
%gets information about camera properties
handles.prop = propinfo(camsrc);
%amount of information per packet (controls frames per second along with
%region of interest
camsrc.NormalizedBytesPerPacket = handles.NBP;
%set region of interest
camobj.ROIPosition = handles.ROI;
%sets camera so frames are acquired after trigger command
triggerconfig(camobj, 'manual');

%set min and max values for slider
%set(handles.gain_slider,'Min',handles.prop.GainAbsolute.ConstraintValue(1));
%set(handles.gain_slider,'Max',handles.prop.GainAbsolute.ConstraintValue(2));
set(handles.gamma_slider,'Min',handles.prop.GammaAbsolute.ConstraintValue(1));
set(handles.gamma_slider,'Max',handles.prop.GammaAbsolute.ConstraintValue(2));
```

```
set(handles.exposure_slider,'Min',handles.prop.AutoExposureAbsolute.ConstraintValue(
1));
set(handles.exposure_slider,'Max',handles.prop.AutoExposureAbsolute.ConstraintValue(
2));
set(handles.brightness_slider,'Min',handles.prop.BrightnessAbsolute.ConstraintValue(1));
set(handles.brightness_slider,'Max',handles.prop.BrightnessAbsolute.ConstraintValue(2))
;

vidRes = get(camobj, 'ROIPosition');
set(handles.Preview,'Position',[450.0+vidRes(1)-4 150.0+vidRes(2)-2 vidRes(3)
vidRes(4)]);
% image width
imWidth = vidRes(3);
% image height
imHeight = vidRes(4);
% create an empty image container and show it on Preview
hImage = image(zeros(imHeight, imWidth), 'parent', handles.Preview);
% begin preview
preview(camobj, hImage);

%set camera controls to manual and absolute
% Gain, Exposure, and Shutter cannot all be set to manuel at the same time
% so gain was left as auto
set(camsrc,'AutoExposureMode','manual');
set(camsrc,'AutoExposureControl','absolute');
set(camsrc,'ShutterMode','manual');
set(camsrc,'ShutterControl','absolute');
%set(camsrc,'GainMode','manual');
%set(camsrc,'GainControl','absolute');
set(camsrc,'GammaControl','absolute');
set(camsrc,'BrightnessControl','absolute');


handles.exposure_value=handles.exposure_default;
handles.shutter_value=handles.shutter_default;
%handles.gain_value=handles.gain_default;
handles.gamma_value=handles.gamma_default;
handles.brightness_value=handles.brightness_default;
```

```
set(handles.exposure_slider,'Value',handles.exposure_value)
%set(handles.gain_slider,'Value',handles.gain_value)
set(handles.gamma_slider,'Value',handles.gamma_value)
set(handles.brightness_slider,'Value',handles.brightness_value)

set(handles.exposure,'String',num2str(handles.exposure_value))
set(handles.shutter,'String',num2str(handles.shutter_value))
%set(handles.gain,'String',num2str(handles.gain_value))
set(handles.gamma,'String',num2str(handles.gamma_value))
set(handles.brightness,'String',num2str(handles.brightness_value))

camsrc.AutoExposureAbsolute=handles.exposure_value;
camsrc.ShutterAbsolute=handles.shutter_value;
%camsrc.GainAbsolute=handles.gain_value;
camsrc.GammaAbsolute=handles.gamma_value;
camsrc.BrightnessAbsolute=handles.brightness_value;



guidata(hObject, handles);

% --- Executes on button press in MeasFrame.
function MeasFrame_Callback(hObject, eventdata, handles)
% hObject    handle to MeasFrame (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global camobj;
global camsrc;


camobj.FramesPerTrigger=50;
%Camera changes settings to auto apon start so settings are changed back to
%manuel after every start command
start(camobj);
camsrc.AutoExposureAbsolute=handles.exposure_value;
camsrc.ShutterAbsolute=handles.shutter_value;
%camsrc.GainAbsolute=handles.gain_value;
camsrc.GammaAbsolute=handles.gamma_value;
camsrc.BrightnessAbsolute=handles.brightness_value;
```

```
trigger(camobj);
pause(3);
[data timestamp]=getdata(camobj);
frame_rate=mean(1./diff(timestamp));
set(handles.text7,'string',num2str(frame_rate));

flushdata(camobj);
stop(camobj);
clear data timestamp

guidata(hObject, handles);
```

```
% --- Executes on button press in ImaqProcess.
function ImaqProcess_Callback(hObject, eventdata, handles)
% hObject    handle to ImaqProcess (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global camobj;
global camsrc;
stoppreview(camobj)


%get and set number of frames user wants to capture
frames=str2double(get(handles.edit1,'String'));
camobj.FramesPerTrigger=frames;
%Starts logging Frames in Memory
%Camera changes settings to auto apon start so settings are changed back to
%manuel after every start command
start(camobj);
camsrc.AutoExposureAbsolute=handles.exposure_value;
camsrc.ShutterAbsolute=handles.shutter_value;
%camsrc.GainAbsolute=handles.gain_value;
camsrc.GammaAbsolute=handles.gamma_value;
```

```
camsrc.BrightnessAbsolute=handles.brightness_value;
pause(1);
trigger(camobj);
%Processing constants
kernel=ones(7,7);
Nk=sum(kernel(:));
fac=Nk/(Nk-1);
s = fix(3/2);
%creates cell array for contrast images to be stored in
handles.Contrast_frames=cell(1,frames);
%creates cell array for Raw speckle images to be stored in
handles.Raw_frames=cell(1,frames);
% loops Processing code for number of frames
axes(handles.Preview);
for i=1:(frames)

    tic;
    data=getdata(camobj,1,'uint8','cell');
    cube=cell2mat(data(1));
    handles.Raw_frames{i}=cube;
    cube=double(cube);
    cube=cube(:,:,1);

    % Spatial contrast

    mu_x=imfilter(cube,kernel,'conv')/Nk;
    x_sq=imfilter(cube.*cube,kernel,'conv')/(Nk-1);
    var_x=sqrt(x_sq - fac*mu_x.^2);
    Cs=var_x./mu_x;

    Cs(1:s,:,:) = [];
    Cs(end-s+1:end,:,:) = [];
    Cs(:,1:s,:) = [];
    Cs(:,end-s+1:end,:) = [];

    imagesc(Cs,[0 0.5]);colormap jet;colorbar

    %updates image
    drawnow
```

```
        handles.Contrast_frames{i}=Cs;
        %shows processing speed
        set(handles.text15,'String',num2str(1/toc));

end

%deletes camera frames stored in memory and stops camera
flushdata(camobj);
stop(camobj)
%enable save button
set(handles.save,'Enable','on');
set(handles.imfile_location,'Enable','on');
guidata(hObject, handles);


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

```
function exposure_Callback(hObject, eventdata, handles)
% hObject    handle to exposure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of exposure as text
%        str2double(get(hObject,'String')) returns contents of exposure as a double
global camobj;
global camsrc;
handles.exposure_value=str2double(get(hObject,'String'));
set(handles.exposure_slider,'Value',handles.exposure_value);
camsrc.AutoExposureAbsolute=handles.exposure_value;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function exposure_CreateFcn(hObject, eventdata, handles)
% hObject    handle to exposure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



function shutter_Callback(hObject, eventdata, handles)
% hObject    handle to shutter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of shutter as text
%        str2double(get(hObject,'String')) returns contents of shutter as a double
global camobj;
global camsrc;
```

```
handles.shutter_value=str2double(get(hObject,'String'));
set(handles.shutter_slider,'Value',handles.shutter_value);
camsrc.ShutterAbsolute=handles.shutter_value;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function shutter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to shutter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function exposure_slider_Callback(hObject, eventdata, handles)
% hObject    handle to exposure_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global camobj;
global camsrc;
handles.exposure_value=get(hObject,'Value');
camsrc.AutoExposureAbsolute=handles.exposure_value;
set(handles.exposure,'String',handles.exposure_value);
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function exposure_slider_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to exposure_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes during object creation, after setting all properties.
function shutter_slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to shutter_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in SetDefault.
function SetDefault_Callback(hObject, eventdata, handles)
% hObject    handle to SetDefault (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global camobj;
global camsrc;
handles.exposure_value=handles.exposure_default;
%handles.gain_value=handles.gain_default;
handles.gamma_value=handles.gamma_default;
handles.brightness_value=handles.brightness_default;

set(handles.exposure_slider,'Value',handles.exposure_value)
%set(handles.gain_slider,'Value',handles.gain_value)
set(handles.gamma_slider,'Value',handles.gamma_value)
```

```
set(handles.brightness_slider,'Value',handles.brightness_value)

set(handles.exposure,'String',num2str(handles.exposure_value))
%set(handles.gain,'String',num2str(handles.gain_value))
set(handles.gamma,'String',num2str(handles.gamma_value))
set(handles.brightness,'String',num2str(handles.brightness_value))

%camsrc.AutoExposureAbsolute=handles.exposure_value;
%camsrc.GainAbsolute=handles.gain_value;
camsrc.GammaAbsolute=handles.gamma_value;
camsrc.BrightnessAbsolute=handles.brightness_value;
guidata(hObject, handles);


% --- Executes on button press in PreviewCam.
function PreviewCam_Callback(hObject, eventdata, handles)
% hObject    handle to PreviewCam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global camobj;
global camsrc;
vidRes = get(camobj, 'ROIPosition');
set(handles.Preview,'Position',[450.0+vidRes(1)-4 150.0+vidRes(2)-2 vidRes(3)
vidRes(4)]);
% image width
imWidth = vidRes(3);
% image height
imHeight = vidRes(4);
% create an empty image container and show it on Preview
hImage = image(zeros(imHeight, imWidth), 'parent', handles.Preview);
% begin preview
preview(camobj, hImage);

camsrc.AutoExposureAbsolute=handles.exposure_value;
camsrc.ShutterAbsolute=handles.shutter_value;
%camsrc.GainAbsolute=handles.gain_value;
camsrc.GammaAbsolute=handles.gamma_value;
camsrc.BrightnessAbsolute=handles.brightness_value;
```

```
guidata(hObject, handles);



function gain_Callback(hObject, eventdata, handles)
% hObject    handle to gain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gain as text
%        str2double(get(hObject,'String')) returns contents of gain as a double
global camobj;
global camsrc;
handles.gain_value=str2double(get(hObject,'String'));
set(handles.gain_slider,'Value',handles.gain_value);
camsrc.GainAbsolute=handles.gain_value;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function gain_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



function gamma_Callback(hObject, eventdata, handles)
% hObject    handle to gamma (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of gamma as text
%        str2double(get(hObject,'String')) returns contents of gamma as a double
global camobj;
global camsrc;
handles.gamma_value=str2double(get(hObject,'String'));
set(handles.gamma_slider,'Value',handles.gamma_value);
camsrc.GammaAbsolute=handles.gamma_value;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function gamma_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gamma (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function gain_slider_Callback(hObject, eventdata, handles)
% hObject    handle to gain_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global camobj;
global camsrc;
handles.gain_value=get(hObject,'Value');
camsrc.GainAbsolute=handles.gain_value;
set(handles.gain,'String',handles.gain_value);
guidata(hObject, handles);
```

```
% --- Executes during object creation, after setting all properties.
function gain_slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gain_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function gamma_slider_Callback(hObject, eventdata, handles)
% hObject    handle to gamma_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
global camobj;
global camsrc;
handles.gamma_value=get(hObject,'Value');
camsrc.GammaAbsolute=handles.gamma_value;
set(handles.gamma,'String',handles.gamma_value);
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function gamma_slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gamma_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes on button press in save.
function save_Callback(hObject, eventdata, handles)
% hObject    handle to save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
folder=get(handles.imfile_location,'String');
location=strcat('C:\Users\labuser\Documents\MATLAB\LSCI_GUI\Images\',folder,'\');
raw=strcat(location,'raw_vid.avi');

rawavi=VideoWriter(raw);
rawavi.FrameRate=125;
open(rawavi);
for K=1:length(handles.Contrast_frames)
  conimg = cell2mat(handles.Contrast_frames(K));

  FileName = sprintf('Contrast image #%d.bmp',K);
  outputFileName=strcat(location,FileName);
  imwrite(conimg,outputFileName, 'bmp' );


  rawimg = cell2mat(handles.Raw_frames(K));
  writeVideo(rawavi,rawimg);
  FileName = sprintf('Raw Speckle image #%d.bmp',K);
  outputFileName=strcat(location,FileName);
  imwrite(rawimg, outputFileName, 'bmp' );

end
close(rawavi);
clear handles.Contrast_frames handles.Raw_frames
set(handles.save,'Enable','off');
set(handles.imfile_location,'Enable','off');
guidata(hObject, handles);


function input_speed_Callback(hObject, eventdata, handles)
% hObject    handle to input_speed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of input_speed as text
%        str2double(get(hObject,'String')) returns contents of input_speed as a double


% --- Executes during object creation, after setting all properties.
function input_speed_CreateFcn(hObject, eventdata, handles)
% hObject    handle to input_speed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in set_speed.
function set_speed_Callback(hObject, eventdata, handles)
% hObject    handle to set_speed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
requested_flow=str2double(get(handles.input_speed,'String'));
%experimentally found equation
volts=0.1979*requested_flow-0.0413;
%duty cycle must be whole number
duty_cycle=round(51*volts);
handles.arduino.analogWrite(3,duty_cycle);
real_flow=(duty_cycle/51+0.0413)/0.1979;
set(handles.real,'String',num2str(real_flow));
guidata(hObject, handles);



function brightness_Callback(hObject, eventdata, handles)
% hObject    handle to brightness (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of brightness as text
%        str2double(get(hObject,'String')) returns contents of brightness as a double
global camobj;
global camsrc;
handles.brightness_value=str2double(get(hObject,'String'));
set(handles.brightness,'Value',handles.brightness_value);
camsrc.BrightnessAbsolute=handles.brightness_value;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function brightness_CreateFcn(hObject, eventdata, handles)
% hObject    handle to brightness (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function brightness_slider_Callback(hObject, eventdata, handles)
% hObject    handle to brightness_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
global camobj;
global camsrc;
handles.brightness_value=get(hObject,'Value');
camsrc.BrightnessAbsolute=handles.brightness_value;
```

```
set(handles.brightness,'String',handles.brightness_value);
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function brightness_slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to brightness_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in stop_pump.
function stop_pump_Callback(hObject, eventdata, handles)
% hObject    handle to stop_pump (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.arduino.analogWrite(3,0);
set(handles.real,'String',num2str(0));
guidata(hObject, handles);



function imfile_location_Callback(hObject, eventdata, handles)
% hObject    handle to imfile_location (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of imfile_location as text
%        str2double(get(hObject,'String')) returns contents of imfile_location as a double


% --- Executes during object creation, after setting all properties.
function imfile_location_CreateFcn(hObject, eventdata, handles)
% hObject    handle to imfile_location (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



function region_Callback(hObject, eventdata, handles)
% hObject    handle to region (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of region as text
%        str2double(get(hObject,'String')) returns contents of region as a double
global camobj
global camsrc
handles.ROI=str2num(get(hObject,'String'));
camobj.ROIPosition = handles.ROI;

vidRes = get(camobj, 'ROIPosition');
set(handles.Preview,'Position',[450.0+vidRes(1)-4 150.0+vidRes(2)-2 vidRes(3)
vidRes(4)]);
% image width
imWidth = vidRes(3);
% image height
imHeight = vidRes(4);
% create an empty image container and show it on Preview
hImage = image(zeros(imHeight, imWidth), 'parent', handles.Preview);
% begin preview
preview(camobj, hImage);

camsrc.AutoExposureAbsolute=handles.exposure_value;
camsrc.ShutterAbsolute=handles.shutter_value;
%camsrc.GainAbsolute=handles.gain_value;
```

```
camsrc.GammaAbsolute=handles.gamma_value;
camsrc.BrightnessAbsolute=handles.brightness_value;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function region_CreateFcn(hObject, eventdata, handles)
% hObject    handle to region (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



function packet_Callback(hObject, eventdata, handles)
% hObject    handle to packet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of packet as text
%        str2double(get(hObject,'String')) returns contents of packet as a double
global camobj
global camsrc

camsrc.NormalizedBytesPerPacket = str2double(get(hObject,'String'));
camsrc.AutoExposureAbsolute=handles.exposure_value;
camsrc.ShutterAbsolute=handles.shutter_value;
%camsrc.GainAbsolute=handles.gain_value;
camsrc.GammaAbsolute=handles.gamma_value;
camsrc.BrightnessAbsolute=handles.brightness_value;
guidata(hObject, handles);
```

```
% --- Executes during object creation, after setting all properties.
function packet_CreateFcn(hObject, eventdata, handles)
% hObject    handle to packet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

9. Appendix B: User Manual

Laser Speckle Contrast Imaging GUI User manual

Step 1: Plug 9-pin 1394b cable into Dragonfly express camera

Step 2: Configure Camera Drivers

- To configure camera drivers first log into labuser.
- Open DriverControlGUI from start menu.
- Select Texas Instruments OHCI Compliant IEEE1394b Host Controller.
- Select Micrsoft Drivers.
- Under the drop down menu select the driver and click Install Driver.
- Close DriverControlGUI.
- Open Control panel->System->Hardware->Device Manager
- Under Point Grey Research Devices right click on PGR Dragonfly Express DX-COL->update driver
- Select Install from a list or specific location (Advance)->next->Don't search I will choose driver to install->next->CMU 1394 Digital Camera Device->next->Continue Anyway->finsh
- To use FlyCap2 you will need to reverse these steps
- Select CMU 1394 Camera Device
- Select Point Grey Drivers
- Select PGR Camera (signed) 2.3.3.59 from drop down menu
- Click install
- Select Texas Instruments OHCI Compliant IEEE1394b Host Controller
- Select Point Grey Drivers
- Select FirePro (signed) 2.3.3.59 from drop down menu
- Click install

Step 3: Open LSCI_GUI

- Log into labuser
- Make sure that arduino's usb cable is plugged into the top usb port on the right side, the power supply is on and set to 12.01V
- Open Matlab
- On the left side of the window under where it says "current window" double click LSCI_GUI folder

- Right click ArduinoIO folder, add to path-> selected folders and subfolders
- Enter LSCI_GUI into command line
- Press Enter

Camera and Processing Controls:

Initialize Camera:

This button formats the camera and the controls of the GUI. After it is pushed the buttons Preview, Measure Frame rate, Set Default settings, and Start Image Acquisition/Processing are enabled. The camera setting controls are also enabled.

Measure Frame rate:

When pushed this button calls a function to start the camera and take 50 images. The time stamps of those images are then used to calculate the frame rate of the camera. The frame rate is then displayed in the box to the right of the button.

Important Note:

The Frame rate is controlled by the Normalized Bytes per Packet and Region of Interest parameters.

Preview:

This button when pressed will return the image window back to a live video feed if the window is not already displaying live video.

Set Default Settings:

Will return all camera settings, sliders, and edit boxes to their default values outlined in the GUI's code.

Camera Settings Sliders:

All sliders when moved will change that setting to the current slider value as well as change the value of its corresponding edit box.

Camera Settings Edit Boxes:

Changing the value and pressing the enter key of one of the camera settings edit boxes will change the camera setting as well as cause the corresponding slider to move to that value.

Important Note:

Shutter (Integration Time) cannot be changed in the GUI but in the code (line 67). Gain must remain on auto because of the embedded code in the camera.

Number of Frames Edit Box:

Located just to the right of the question "How many frames would you like to record?" The number in this box will be the number of frames the GUI will record and Process when the "Start Image Acquisition/Processing" button is pushed. This edit box does not have a call function.

Start Image Acquisition/Processing:

This button will start the recording of the number on designated frames. While the frames are being recorded a loop will grab a frame, save the frame in a cell array, get the frames contrast image, display the contrast image, save the contrast image in a separate cell array, and display the speed of the processing in the green box below the button.

Save Images:

Enabled after processing is finished, this button will save all raw and contrast images as individual jpeg images in the folder designated in the edit box above the save button.

- **Warning: any previously saved images will be overwritten.**
- **Designated folder must be created before the save button is pushed**
- **The folders name in the edit box must be exact or error will appear**

Region of Interest:

Enabled after camera initialization, allows user to change the region of interest of the camera. Combined with the Normalized bytes per packet perimeter these functions control the frame rate of the camera. The layout of the vector is:

[X-offset Y-offset Width Height]

Normalized Bytes per Packet:

Controls how many bytes are transferred at a time. Value must be a multiple of 8. If the value is too large for the ROI then the image will break up or Matlab will show an error.

Pump Control Panel:

Enter the flow rate you would like the pump to do and press the "set pump speed" button. The program will then calculate the appropriate duty cycle value to send to the Arduino and display the actual flow rate in the box below.