5-2-2020

Creation of a Graphical User Interface for Reflectance and Transmission Mode Quantitative Polarized Light Imaging

Ethan Meitz
Washington University in St. Louis

Abstract:

Throughout the spring 2020 semester I worked in Dr. Lake's Musculoskeletal Soft Tissue lab developing custom software and a graphical user interface (GUI) for polarized light image acquisition. The Lake lab recently purchased a commercially available polarization camera produced by FLIR, the Blackfly S USB 3.0 with SONY IMX250MZR sensor which required code development for polarization image acquisition. The Lake lab has previously used quantitative polarized light imaging (QPLI) in reflectance and transmission mode to evaluate dynamic collagen fiber alignment in musculoskeletal soft tissues. They aim to use the reflectance based QPLI technique (rQPLI) to measure the alignment of collagen fibers in the ulnar collateral ligament (UCL) during fatigue loading to visualize region specific microstructural damage before failure. In QPLI, the Lake Lab has historically used a custom division of focal plane polarization sensor for rapid characterization of the Stokes parameters. Establishing the use of a commercially available polarization camera makes technology transfer and application scale up simpler than with the previous custom hardware. Development of an image acquisition software suite for the Blackfly camera will allow images and video taken by the commercially available sensor to be compared to the previously validated sensor currently in use for rQPLI. Development began with extensive reading of documentation and software examples. From there, I created a software suite resulting in a functional GUI that could interface with the camera. Some present limitations on acquisition capability and data storage persist and will be addressed in future semesters.

Introduction:

The UCL is a connective tissue found in the elbow which helps connect the upper arm to the forearm. The UCL is thought to fail due to damage from repeated, cyclic loading, like the repetitive throwing motion of a professional baseball player [1]. After an acute failure event, the UCL typically cannot heal on its own, so surgery is often required to fully repair the ligament to its normal functionality [1]. There is no accepted method capable of detecting the progression or onset of fatigue damage in these tissues. Therefore, surgeons are unable to provide any informed preventative care to their patients until an ultimate injury event. A real-time, quantitative imaging modality that can visualize the dynamic collagen microstructure of the UCL is required to get this information. Clinical imaging techniques such as MRI and CT scans are limited in their spatial and temporal resolution and cannot distinguish dynamic collagen microstructure to provide information about dynamic UCL damage [2].

Previous studies have shown that transmission mode quantitative polarized light imaging (QPLI) can be used *ex vivo* to measure the alignment of collagen fibers in the UCL [3]. QPLI utilizes a camera equipped with a division-of-focal-plane polarization sensor that can quantify the polarization state of light. Ligaments are birefringent, which means they can polarize light in proportion to their collagen fiber alignment [4]. This polarized light can either be quantified after it passes through tissue (transmission mode) or through the light the tissue reflects (reflectance mode). In transmission mode samples must be thinned to allow for light penetration, limiting the physiological relevance of the technique, so reflectance mode is preferred as the samples do not need to be processed prior to imaging [5]. Therefore, by analyzing the UCL with reflectance mode QPLI during mechanical loading, the behavior and failure mechanism of its collagen fiber structure can be studied in a more physiologically relevant loading environment.

The Lake lab historically performed transmission and reflectance mode imaging with a custom division-of-focal plane polarization sensor developed by Prof. Viktor Gruev capable of calculating the first three Stokes parameters pixelwise for each frame [6]. Recently, SONY commercialized an analogous polarization sensor design with improved spatial and temporal resolution to the custom sensor. Use of a commercially available camera will also allow for ease of translation and scale-up of the imaging modality in future applications. Therefore, the Lake lab recently purchased a Blackfly S USB 3.0 camera with a Sony IMX250MZR polarization sensor from FLIR. This camera and accompanying sensor were not sold as a "plug-and-play" model; a custom acquisition software suite needed to be developed in order to extract polarization information from the sensor.

The FLIR camera requires custom code because for optimal QPLI performance the input parameters must be customizable (i.e. desired frame rate, resolution, file name, path, stokes parameters) and the camera operation flexible for different applications. Most importantly, the raw camera polarization data must be accessible for calibration, analysis and post-processing. The FLIR camera outputs a greyscale image in its most basic acquisition mode, but to use the polarization capabilities the Stokes parameters S_0, S_1, and S_2 need to be calculated from the intensity of light on each pixel of the sensor. Then, the angle of polarization (AoP) and degree of linear polarization (DoLP) can be calculated from the Stokes parameters. These are the parameters allow determination of the collagen fiber alignment. Researchers must be able to see the camera view in order to ensure the sample is in frame and in focus prior to the start of image acquisition. A GUI interfaced with the FLIR camera was chosen as the ideal way to solve these problems as a GUI provides a user-friendly experience that does not require in depth knowledge to operate.

Based on the needs identified, my objective for the semester was to research, design and create a GUI to acquire raw data, display live AoP, DoLP, and greyscale images, and control the camera functions with easy to use push buttons.

Methods:

Polarization Sensor

The Blackfly S USB 3.0 camera which utilizes a Sony IMX250MZR polarization sensor to measure the intensity and direction of polarized light was the primary piece of equipment. This camera only works within the USB 3.0 port of a computer and requires a special lens to tune the focal length.

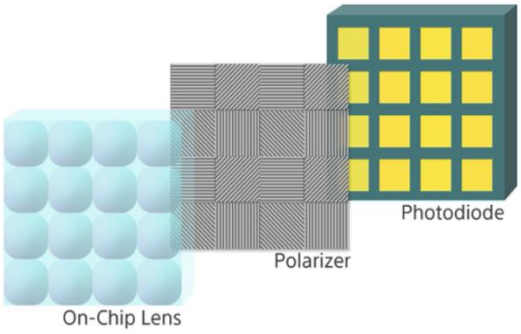

Figure 1. The 3 layers of the SONY filter [7]

Within the SONY polarization sensor, each individual pixel is a linear polarizer that sits on top of a photodiode (**Figure 1**) [7]. There are four orientations 0°,45° 90°, and 135° which form a 2x2 grid called a super-pixel. Each pixel measures the intensity of each type of linearly polarized light in that particular direction.

Calculating the Stokes Parameters:

Using the data from each pixel the first three Stokes parameters can be calculated using the following equations, where I_0 is the intensity of light in the 0° quadrant [6].

$$S_0 = I_0 + I_{90}$$

$$S_1 = I_0 - I_{90}$$

$$S_2 = I_{45} - I_{135}$$

Calculating the AoP and DoLP:

The Angle of Polarization (AoP) is defined as the average angle of orientation (0-180°) of the polarized light and the degree of linear polarization (DoLP) is defined as the percentage of light incident on the sensor that is linearly polarized, expressed as a decimal from 0-1 [8]. The equations for AoP and DoLP are shown below which are calculated using the Stokes parameters [6]:

$$DoLP = \frac{\sqrt{S_1^2 + S_2^2}}{S_0}$$

$$AoP = \frac{1}{2}\tan^{-1}\left(\frac{S_2}{S_1}\right)$$

The average DoLP (AVG DoLP) and standard deviation of AoP (STD AoP) are two values which can be calculated from the DoLP and AoP data within a region of interest [9]. These values have been shown to accurately describe the microstructure of collagen in a variety of tendons and ligaments [9]. The AVG DoLP is a measurement that can be used to infer the strength of collagen fiber alignment and the STD AoP is inversely proportional to the uniformity of collagen fiber alignment [9].

Software Development

To create a functional GUI, snippets of existing code were taken from the Spinnaker SDK and combined into one code base. The code from the SDK allowed the camera to connect to the computer and acquire images. Furthermore, Qt, an open source C++ GUI library, provided a base to build the GUI up from without having to write code for simple functions such as buttons and text boxes. By starting with existing code, I could work quickly and get a working sample after only a few weeks. From this base custom code could be written to extract the raw data and display

the live images. As the project developed, I worked closely with members of the Lake lab to incorporate functions that would be necessary to complete experiments. For example, I included fields for customization of the resolution and FPS. Certain applications do not require high resolution images and therefore having the ability to tune the resolution down to prevent unnecessarily large files is desirable.

Once a working sample was created, I could test the GUI by simply connecting the camera. In early models, there was no GUI window and just a command window popped up, but it allowed me to see any bugs that occurred. Once the code properly saved data I began working on the GUI. After a new feature was added the camera could be tested. This iterative approach was successful, and I would only work on one piece until it worked.

Results:

I first identified requirements for the custom GUI by speaking to members of the Lake Lab who perform QPLI and by looking at the current software suite for image acquision. The needs I identified were as follows:

1. Acquire raw pixel intensity data
2. Calculate and acquire AoP and DoLP data
3. Display live AoP, DoLP and greyscale images
4. Start and stop image acquisition on command
5. Change image frames per second (FPS) and resolution

Each item was identified as essential to use the GUI in an experimental environment. The greyscale, AoP, and DoLP pixel data were required for post processing analyses and statistics

whereas the GUI needed to allow full control over the camera functions so the GUI could adapt to any application

The Blackfly S came equipped with the Spinnaker software development kit (SDK) and documentation which outlined the included functions and contained several code examples. A GUI called SpinView came with the camera; however, it could not extract the raw pixel data from each image or show live AoP and DoLP images. The code from this GUI could not be edited as it was proprietary, my objective was to create a new GUI to extract raw pixel data and show the AoP and DoLP while retaining SpinView's positive attributes such as a live greyscale camera view.

The first two requirements identified above could be achieved using the Spinnaker SDK. After careful review of the documentation the "Acquisition" and "Polarization" code examples were chosen. The acquisition example provided a code base to access the Blackfly camera and begin taking photos. The polarization example provided functions which converted the images captured by acquisition into AoP and DoLP images. From there custom code was written to extract the data and write it to a file for post processing.

The last three goals required a GUI library. Qt is an open source GUI library written in C++ [10]. This particular library was chosen because it interfaced well with Visual Studio which was the Integrated Development Environment (IDE) used to compile the "Acquisition" and "Polarization" examples. Qt provides a plethora of tools with documentation to create a custom GUI. Essentially, it comes with pre-written snippets of code that can be implented easily. For example, to add and item to a drop down menu, simply type dropDown.addItem("Item Name"). Without Qt, doing something like this would be infeasable if not impossible. Qt saves times so that pertinent problems such as data extraction can be tackled.

To display the AoP, DoLP, and greyscale images the raw data was sent to Qt and reconstructed into an image. To start and stop acquisition pieces from the acquisition code example were placed into Qt. Qt provides a unique signals and slots framework which allows a button (signal) to execute a command (slot) instanteously without the need for extra code to connect the two.

Figure 2. Demonstration of the signals and slots framework used by Qt.

For example, if "Object 1" is a button and "Object 2" is an image (**Figure 2**). When the button is pressed it emits a "signal" to the image and executes a block of code. This signal might display the image or shrink the image. This idea can be applied to every item within the GUI such as buttons, drop boxes, images, and text labels. The signals and slots are very powerful and allow the user to enact instanteous feedback within their GUI. Signals and slots were used to change the FPS and resolution and begin acquisition. However, the FPS of the camera could not be controlled directly, so a mathematical function was created to calculate the appropriate amount of time to pause the program given the chosen FPS. The equation is:

$$Pause\ Time\ = \frac{1}{Desired\ FPS} - \frac{1}{75\ FPS}$$

Where 75 FPS is the maximum the Blackfly can achieve. This pause time is added to every frame that is captured thus slowing the frame rate to the desired frame rate.

The first two goals were successfully completed prior to the semester being shifted to exclusively remote work due to the COVID-19 pandemic. Figure 2 shows the first image reconstructed using the acquisition code. This image is a pseudo-color representation of pixel intensity data acquired in a single acquired frame. The reconstructed image was exactly what the viewfinder showed before image acquisition and had the correct resolution (2448 by 2048).

Figure 3. A pseudo-color image created using raw pixel data reconstructed into an image using MATLAB

Multi-frame data acquisition also worked as expected. The data was saved after image acquisition due to the large file size. Data was saved to a specified folder as a text file, with a unique time stamped name for each image acquired. Each text file (1 per frame) was 5 MB when the resolution was set to 2448 by 2048. This was expected because each pixel requires 8 bits to store (1 byte) so 2448*2048 = 5,013,504 bytes or 5 MB.

While the functionality of the GUI could not be fully tested it successfully connected to the camera and functionality was added to every button. An image of the GUI is shown in Figure 4.

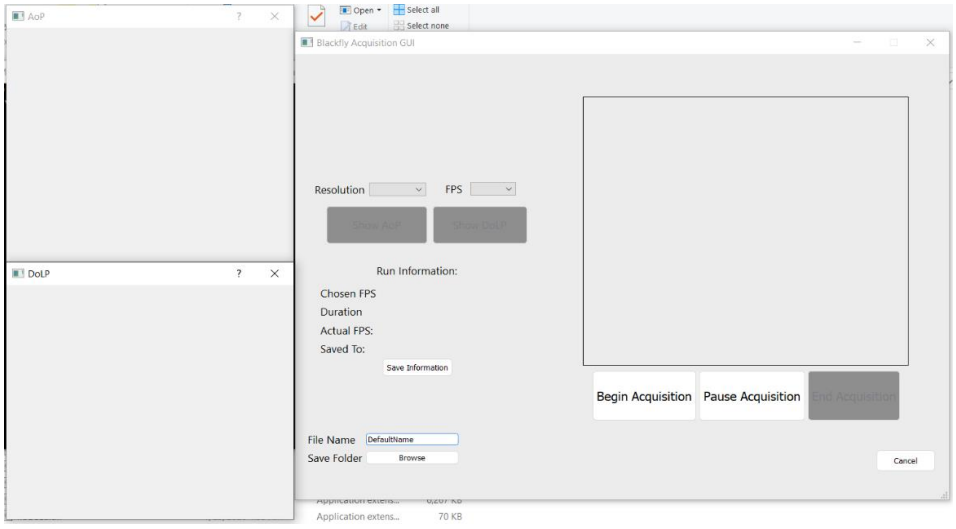

Figure 4. Acquisition GUI with the live views not displaying.

Each of the buttons works as intended. For example, when the AoP window is closed the “Show AoP”, button which is currently greyed out, becomes available to click. The most important button, “Begin Acquisition” does connect to the camera, but currently throws an error which could not be solved and requires future work to debug.

Discussion:

For creation of the software suite to be considered successful I identified that it must (1) read and write data for greyscale, (2) AoP and DoLP images, (3) show the live camera view, (4) start and stop acquisition on command and (5) allow the FPS and resolution to be completely customizable. At the current stage, the GUI can access the camera to read and write data and allows the FPS or resolution to be customized. However, the live images do not display when connected to the camera (**Figure 4**). When “Begin Acquisition” is clicked an error message is thrown to the

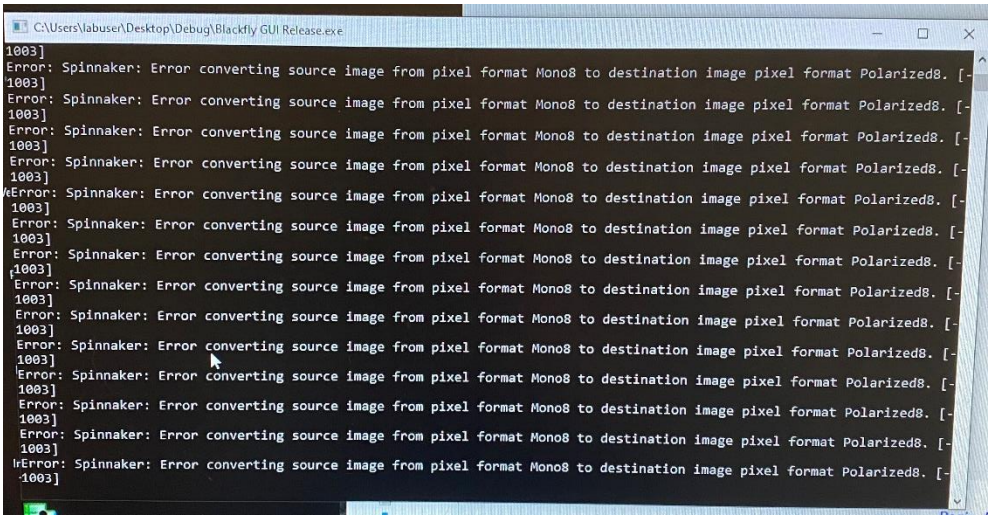

Figure 5 shows the error thrown when "Begin Acquisition" is pressed

command prompt every time an image is taken (**Figure 5**). This error message was coded into the Spinnaker SDK and indicates that the camera cannot convert the image it captured to a specific format. This means the GUI is attempting to convert between two incompatible formats. For example, an excel file (.xlxs) would not open as a word document (.doc). This issue is likely why

the camera view does not show up on the GUI. Since the error throws multiple times, the bug must exist within the camera loop of the acquisition example. More specifically, only two lines use the pixel format: the first use converts the captured image to the Polarized8 and the second uses this converted image to display the AoP and DoLP. From the error message, the former is more likely as the message states the computer cannot convert between the two image types. Pixel formats are extensively documented on FLIR's website under the "Programmer's Guide". I also directly corresponded with FLIR customer support in order to come up with a solution to this bug. The FLIR employee identified a possible issue and recommended that I set the pixel format of the camera at the beginning of the code. This is a logical solution- before I enacted this change the camera was trying to take pictures, but it has no prior knowledge what format the pictures are supposed to be in and would not display anything. However, because of the COVID-19 crisis, my access to the camera was cut short so I was unable to troubleshoot the code any further.

This current bug was one of many I ran into during the development of the GUI. One of the largest issues was the speed of data acquisition. The initial design was to write the data from each picture as the image was taken by the camera. That method proved extremely slow as each individual picture was extremely large. Therefore, the pointer, a C++ object defined in the standard template library, was used. Pointers are an object that store an address in memory or "point" to the address. This provides a simple way to keep track of large amounts of data easily. A pointer was created for each image and saved to an array, essentially creating a list of memory addresses. Each pointer can then be accessed later, and the computer will know where the images are located in memory. This way, the data extraction can be done after all the images are collected. I aim to optimize this data acquisition process after finishing debugging the acquisition GUI, which as aforementioned, was cut short due to COVID-19.

Conclusion and Future Directions:

A software development suite was created for a Blackfly S camera from FLIR with SONY IMX250MZR polarization sensor. The intention for this software suite is to aide in acquisition of polarization images for analysis of microstructural damage to the UCL during fatigue loading. The software written can access raw data from images captured by the camera and save the data to a text file for post processing. The Qt program implemented generated a sleek and easy to use GUI which also connects successfully to the camera.

At the premature conclusion to the semester imposed by COVID-19, the software was experiencing a bug that kept it from being able to live stream acquired images directly to the GUI, but the overall functionality of each button included in the GUI works as intended. The first task to complete when the project begins again is to solve this display problem. It is imperative for the user to be able to visualize the field of view in real time prior to performing rQPLI experiments. I also aim to optimize acquisition speed to allow for full frequency testing.

After successful software development, the next step is to evaluate performance of the camera in rQPLI. It will be compared to the existing in house-built division-of-focal-plane polarization sensor currently used in the Lake Lab. Both sensors will be used to analyze the polarization of light manipulated with polarization state generators (PSGs). As the polarization state of light is known when using PSGs, we can know the expected signal calculated using both the FLIR camera and custom camera. Once validated, the FLIR camera can be used in full rQPLI testing of tendons and ligaments, like the UCL, to gain valuable information and insight into structure-function relationships of musculoskeletal soft tissues.

References:

[1] "Ulnar Collateral Ligament Tears | Orthopedics & Sports Medicine." *UConn Health*, 2 Aug. 2017, https://health.uconn.edu/orthopedics-sports-medicine/conditions-and-treatments/where-does-it-hurt/elbow/ulnar-collateral-ligament-tears/.

[2] Martin, S. D., et al. "New Technology for Assessing Microstructural Components of Tendons and Ligaments." *International Orthopaedics*, vol. 27, no. 3, June 2003, pp. 184–89. *DOI.org (Crossref)*, doi:10.1007/s00264-003-0430-4.

[3] Smith, Matthew V., et al. "Mechanical Properties and Microstructural Collagen Alignment of the Ulnar Collateral Ligament During Dynamic Loading." *The American Journal of Sports Medicine*, vol. 47, no. 1, Jan. 2019, pp. 151–57. *DOI.org (Crossref)*, doi:10.1177/0363546518812416.

[4] York, Timothy, et al. "Real-Time High-Resolution Measurement of Collagen Alignment in Dynamically Loaded Soft Tissue." *Journal of Biomedical Optics*, vol. 19, no. 6, SPIE, June 2014, p. 066011. *experts.illinois.edu*, doi:10.1117/1.JBO.19.6.066011.

[5] Smith, Matthew V., et al. "Mechanical Properties and Microstructural Collagen Alignment of the Ulnar Collateral Ligament During Dynamic Loading." The American Journal of Sports Medicine, vol. 47, no. 1, 2019, pp. 151–57. PubMed, doi:10.1177/0363546518812416.

[6] Gruev, Viktor, et al. "CCD Polarization Imaging Sensor with Aluminum Nanowire Optical Filters." *Optics Express*, vol. 18, no. 18, 2010, pp. 19087–94.

[7] "Technology | Image Sensor : Polarization | Products | Sony Semiconductor Solutions Group." *Sony Semiconductor Solutions Group*. *www.sony-semicon.co.jp*, https://www.sony-semicon.co.jp/e/products/IS/polarization/technology.html.

[8] Foster, James J., et al. *Polarization Vision—Overcoming Challenges of Working with a Property of Light We Barely See*. preprint, Animal Behavior and Cognition, 24 Oct. 2017. *DOI.org (Crossref)*, doi:10.1101/207217.

[9] Skelley, Nathan W., et al. "Differences in the Microstructural Properties of the Anteromedial and Posterolateral Bundles of the Anterior Cruciate Ligament." *The American Journal of Sports Medicine*, vol. 43, no. 4, Apr. 2015, pp. 928–36. *PubMed*, doi:10.1177/0363546514566192.

[10] Qt Creator Manual. Qt Documentation. The Qt Company. 2020. https://doc.qt.io/qtcreator/index.html