CSCI 370 Final Report

Optical Engines Inc: Calibration Station

Drew Cornmesser

Carson Cramer

Elian Estrada

Revised June 17, 2022

CSCI 370 Summer 2022

Ms. Donna Bodeau

Table 1: Revision history

Revision	Date	Comments
Rev-1	May 20, 2022	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done XI. Team Profile References Appendix A – Key Terms
Rev-2	May 27, 2022	Updated Sections: I. Introduction II. Risks III. Definition of Done Completed Sections: IV. System Architecture
Rev-3	June 3, 2022	Updated Sections: I. System Architecture Completed Sections: II. Software Test and Quality III. Project Ethical Considerations
Rev-4	June 10, 2022	Updated Sections: I. Software Test and Quality Completed Sections: II. Project Completion Status III. Future Work IV. Lessons Learned
New	June 14, 2022	Updated Sections: I. Introduction II. Functional Requirements III. Non-Functional Requirements IV. System Architecture V. Software Test and Quality VI. Project Ethical Considerations VII. Project Completion Status VIII. Future Work IX. Lessons Learned Completed Sections: X. Technical Design

Table of Contents

I. Introduction

Optical Engines, Inc is a high-powered laser development company based out of Colorado Springs, Colorado. The company strives to develop high quality solutions in both hardware and software by utilizing areas such as the Spyro software “BEAMS”. This project aimed to extend functionality of the existing BEAMS software through three main phases that consist of modernizing the GUI on the front end in order to make the product more visually appealing and conducting mathematical analysis on the back end. The idealized result and last phase of the project was to modify and change the Arduino firmware to be labeled from current label “Arduino Due” to “Spyro Cam”. Optical Engines offers valuable laser services to clients, and it was this team’s hope that through the implementation of these phases we can help the company in their pursuit to create a marketable product so that they can grow as a company.

The software that the team updated is in a repository on GitHub. The project is written in C# on the back end, and then XAML and WPF on the front end. The code was developed primarily by Optical Engines employees, Matthew Schulz and Donald Sipes. The need for continued effort stems from the fact that the current user interface is visually unaesthetic, while the implementation and calculation of the laser math on the back end is not fully formed and very shallow in development. With regards to the labeling for the Arduino, changing the labeling of the firmware to “Spyro Cam” will allow for Optical Engines to have naming rights when it comes towards marketing the finalized product [1].

There was trouble with ordering both the lasers and the Arduinos that the team used for testing and data sources. The orders were back logged and because of that it was be challenging to fully test the products from a data collection standpoint. The main source of the data was collected from Optical Engines laboratory in their main office in the Colorado Springs. Optical Engines contact liaison Matthew did send us sampling data so that we could manually test various data points. While this was ultimately not an idealized version of testing it gave the team a rough idea on the progress being made.

The hardware interface being used was an Arduino. The Arduino controlled various functionality in both the software and the laser systems. However, the predominant use of the Arduino was to contain firmware libraries that are implemented in the various phases of the project.

With regards to abbreviations and acronyms, Optical Engines does utilize terminology that is important to understand. BEAMS is used to show laser and particle beam measurements. To extend functionality on the hardware side, the Arduino was utilized to add a tab for both reading and writing EEPROM (Electronically Erasable Programmable Read Only Memory). EEPROM is ultimately “a type of non-volatile ROM that enables individual bytes of data to be erased and reprogrammed” [2]. The current functionality has a thermopile sensor detecting an emitted infrared laser, and after the backend software calculates specified laser calculations, those laser calculations are stored in memory (EEPROM) on the Arduino. The team was tasked with utilizing the Arduino to recognize these various memory addresses of interest so that with future analytical interest these memory addresses can either be read for data collection and interpretation or written over and replaced by other calculations of interest.

Optical Engines performs laser consulting methods for a wide variety of clients. These stakeholders include those who are involved in laser optics research institutions such as scientific firms, colleges, or universities. Recently Optical Engines, through their usage of lasers, has created a more efficient version of plastic welding. This laser welding technology introduces itself to potential new stakeholders in the manufacturing industry as a laser solution for more precise and efficient welding methods. Potential stakeholders include defense system corporations as well. Optical Engines was approached by various companies to potentially weaponize lasers in hopes of shooting down drones, planes, or enemy combatants in the field. If any of these potential stakeholders are interested in laser solutions to their specified problems, then the product that this team will create will result in a viable solution.

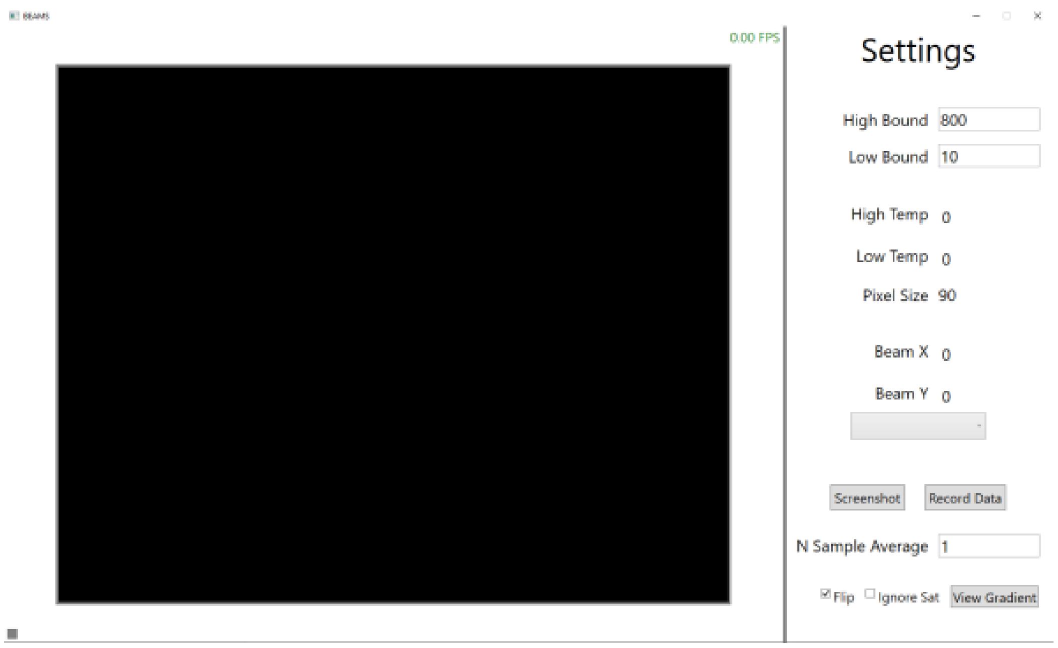
Creating a project such as this required intensive maintenance. From a team perspective the software was maintained through the implementation of GitHub and branches associated with Git. It was important that the team adhered to Optical Engines branch naming schemes. The naming schemes of created branches were+ as follows:

<name of person working on specified task>-<specified task that is in development>

This naming scheme was relatively straightforward however it provided a level of communication with both this team and Optical Engines as to who was working on what task. This way the team could allocate time and attention to each task in a thoughtful and more intuitive manner. Once a team member believes that a task is complete, said team member created a pull request on GitHub for that respective branch. Optical Engines main client liaison Matt would run various tests in the laboratories at Optical Engines headquarters and upon Matt’s discretion the pull request was either approved, and that respective branch merged into the master branch, or the pull request was denied with feedback provided, and at that point the team would implement the requested changes. Utilizing GitHub in this manner allowed for effective maintenance and easy access on Optical Engines' end upon the conclusion of Field Session. Once Field Session concludes, Matt was to serve as the primary software maintenance operator on the Optical Engines side.

II. Functional Requirements

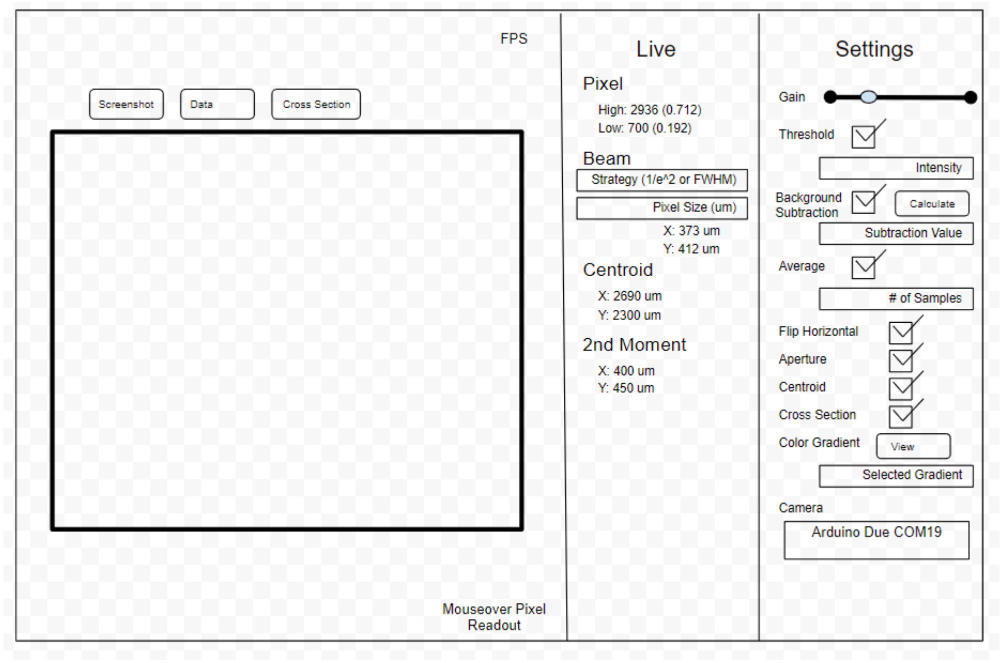
The initial model that Optical Engines utilized is shown in Figure 2. The team added functionality to Figure 1 so that it corresponded with Figure 2.



Settings

High Bound 800
Low Bound 10
High Temp 0
Low Temp 0
Pixel Size 90
Beam X 0
Beam Y 0
N Sample Average 1
☑ Flip ☐ Ignore Sat

Figure 1. Initial GUI



Live

Pixel
High: 2936 (0.712)
Low: 700 (0.192)
Beam
Strategy (1/e^2 or FWHM)
Pixel Size (um)
X: 373 um
Y: 412 um
Centroid
X: 2690 um
Y: 2300 um
2nd Moment
X: 400 um
Y: 450 um

Settings

Gain
Threshold ✓
Intensity
Background Subtraction ✓
Subtraction Value
Average ✓
# of Samples
Flip Horizontal ✓
Aperture ✓
Centroid ✓
Cross Section ✓
Color Gradient
Selected Gradient
Camera
Arduino Due COM19

Figure 2. Design Intention for Finalized GUI

To ensure full functionality as shown in Figure 2 the team considered functional requirements on both the frontend and the backend of the project as well as the relationships between the two.

Front End Requirements:

1. Move the “Screenshot” and “Record Data” buttons from the far-right column to the center of the far-left column.
2. Make three columns on the user interface with one showing the laser output, another showing “Live” settings, and lastly a “Settings” column so that the user may update the settings on the laser in real time.
3. Add a slider to the “Threshold” label so that the user can dynamically see the updates being made to the laser.
4. Add a “Background Subtract” button without the beam or the sensor so that the software can calculate the average pixel value.
5. Add a threshold value for pixels on the laser image so that the user can understand high and low bounds of value and their corresponding pixel value.
6. Display the cartesian coordinates of the centroid to the user in real time.
7. Display the cartesian coordinates of the second order moment radius to the user in real time.

Backend Requirements:

8. Aperture the software so that all pixels outside of a 99% power enclosure of the beam are voided.
9. Set the bitrate to 16 for measurements and a lower value (Optical Engines has yet to determine appropriate value) for alignment work.
10. Incorporate the ability to average calculations for a specified number of frames (either a straight average or a rolling average).
11. Calculate the centroid (first order moment) in cartesian coordinates.
12. Calculate the second order moment radius in cartesian coordinates.

Arduino Requirements:

13. EEPROM values are verified by outputting memory addresses that correspond to various calculations to the user.
14. The naming scheme on the Arduino changes from “Arduino Due” to “SpyroCam”.

Critical Relationships.

There were critical relationships amongst the following requirements:

- Requirement 3 and Requirement 5.
    - It was necessary to obtain the visual for the Threshold Slider task (specified in Requirement 3) to properly gauge whether the pixel thresholding value was being calculated appropriately.
- Requirement 6 and Requirement 11.
    - In order to display the centroid value to the user (specified in Requirement 6) the team first had to calculate the centroid of a specified laser.
- Requirement 7 and 12.
    - In order to display the second order moment radius value to the user (specified in Requirement 7) the team had to first calculate the second order moment radius value to the user.
- The Arduino requirements and the Frontend and Backend Requirements.
    - In order to ensure that specified EEPROM values are accurate the team had to first create functional software on the backend and frontend side of the project.

III. Non-Functional Requirements

To further ensure full functionality as shown in Figure 1 the team considered (in addition to the frontend and backend) non-functional requirements with regards to the interface utilized, code maintenance, as well as operating systems.

Frontend.

1. The interface must be aesthetically pleasing.
2. Code on the front end must be written in WPF.

Backend.

3. Code on the backend must be written in C#.

Interface.

4. The project must be compiled using Visual Studio 2022.

Code Maintenance.

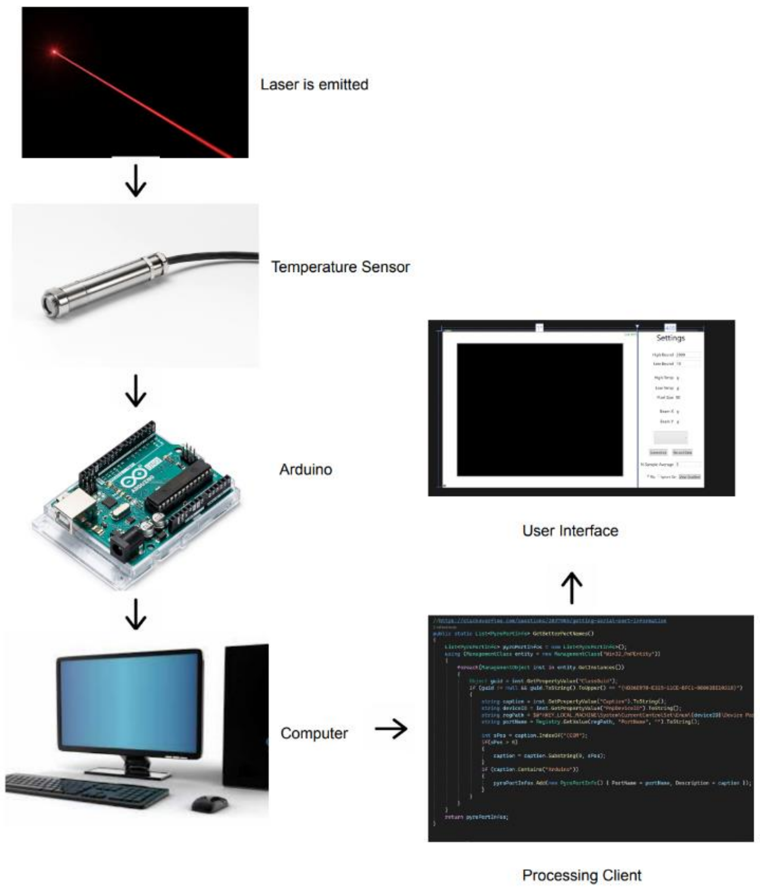
5. The master branch of code must be stored in a repository on GitHub.

Operating Systems.

6. Since the code for the interface is written in WPF, the machine that a team member compiles on must be of a Windows type operating system.

VI. System Architecture

Figure 3 depicts the step-by-step process of the program from an external and internal perspective. With regards to Figure 3, the team was involved in the development of the processing client and the user interface. Understanding the entire scope of the program and the components surrounding where to add functionality allowed for a faster learning curve and a more holistic understanding of the program.



```
//https://stackoverflow.com/questions/[illegible]/getting-serial-port-information
public static List<PyroPortInfo> GetBetterPortNames()
{
    List<PyroPortInfo> pyroPortInfos = new List<PyroPortInfo>();
    using (ManagementClass entity = new ManagementClass("Win32_PnPEntity"))
    {
        foreach(ManagementObject inst in entity.GetInstances())
        {
            Object guid = inst.GetPropertyValue("ClassGuid");
            if (guid != null && guid.ToString().ToUpper() == "{[illegible]}")
            {
                string caption = inst.GetPropertyValue("Caption").ToString();
                string deviceID = inst.GetPropertyValue("PnpDeviceID").ToString();
                string regPath = $@"HKEY_LOCAL_MACHINE\System\CurrentControlSet\Enum\{deviceID}\Device Pa
                string portName = Registry.GetValue(regPath, "PortName", "").ToString();

                int sPos = caption.IndexOf("(COM");
                if(sPos > 0)
                {
                    caption = caption.Substring(0, sPos);
                }
                if (caption.Contains("Arduino"))
                {
                    pyroPortInfos.Add(new PyroPortInfo() { PortName = portName, Description = caption });
                }
            }
        }
    }
    return pyroPortInfos;
}
```

Figure 3. Design Model

The program begins when an infrared laser emitted from a cathode is detected by a temperature sensor. The readings from that sensor are sent to an Arduino where they are stored, and the Arduino subsequently uploads these readings to the computer in the form of a CSV file. The processing client reads in a specified CSV file and then various laser calculations are conducted based on that data. Subsequently, these calculations are then displayed to the user in the user interface so that stakeholders can discern meaningful data in real time from the simulation.

Since the team was adding functionality to a working product there was an extensive code base that the team will expand upon. To allow for as seamless introduction to the existing code as possible the team utilized the class diagram shown in Figure 4 to serve as a reference when needing to update functionality in a certain class within the existing codebase.

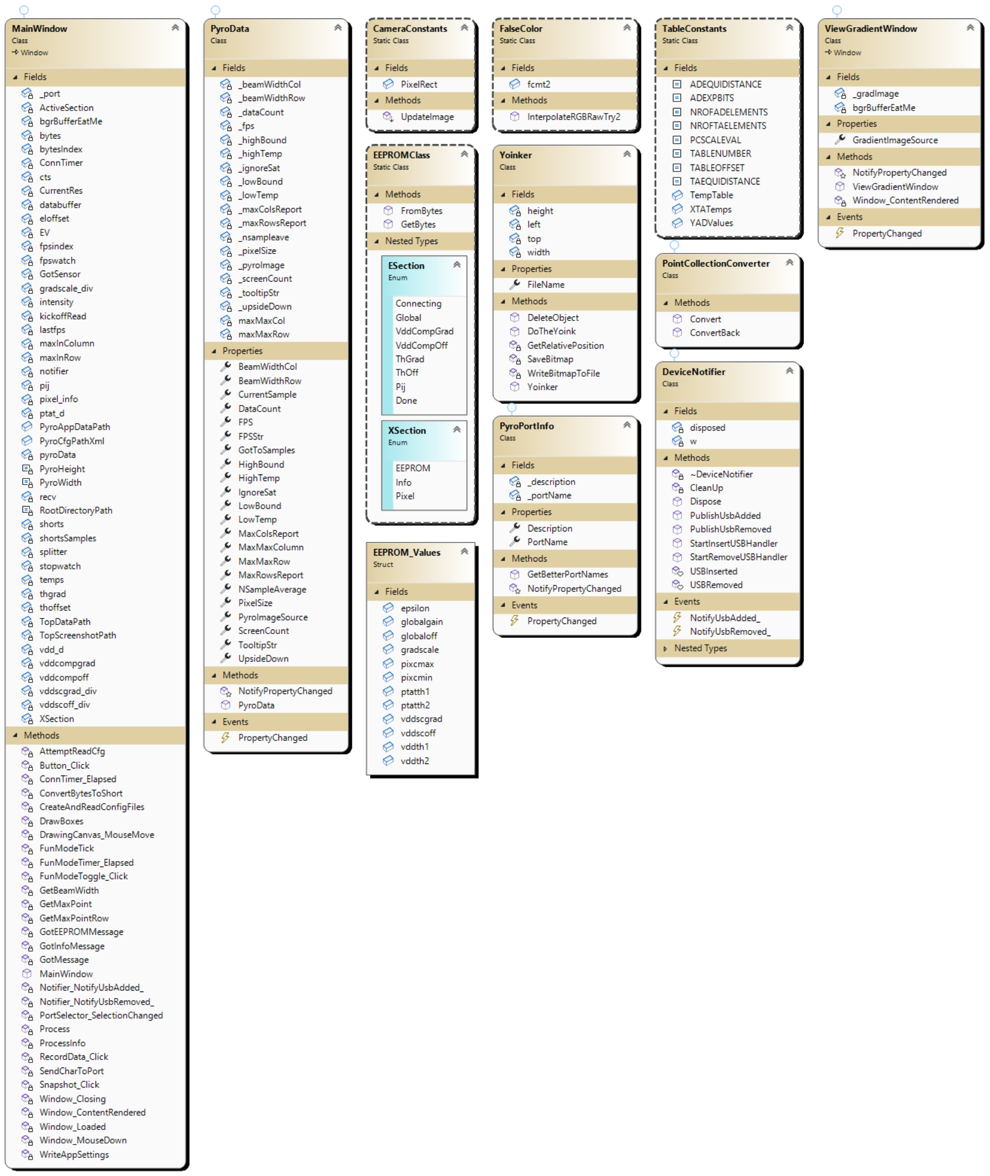
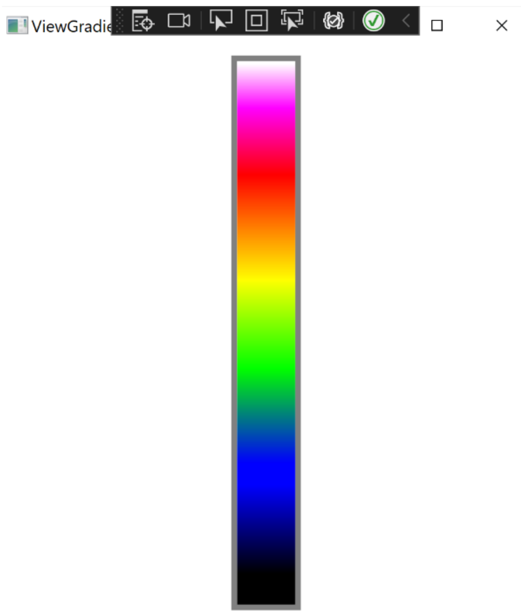


Figure 4. Class Diagram

The team was mainly concerned with the MainWindow and the PyroData classes. There were two main components to the MainWindow class: MainWindow.xaml and MainWindow.xaml.cs. The MainWindow.xaml component utilized WPF and was what displayed the GUI depicted in Figure 1. The MainWindow.xaml.cs component utilized C# and was where the calculations on the backend were implemented. The PyroData class was the class that handled the simulation itself. The team utilized the PyroData class primarily for getters and setters that were called in MainWindow.xaml.cs so that calculations could be carried out and then ultimately displayed using MainWindow.xaml.

The team implemented a greyscale option that the user could select to change the laser coloring scheme from a standard heatmap scale. Changing the coloring scheme required the team to work primarily with the ViewGradientWindow and FalseColor classes. The ViewGradientWindow class displayed a gradient example that showed the user the breakdown of the chosen color scheme. If the user desired a standard coloring scheme, then the ViewGradientWindow class would display Figure 5 and if the user desired a greyscale coloring scheme the ViewGradientWindow class would display Figure 6.



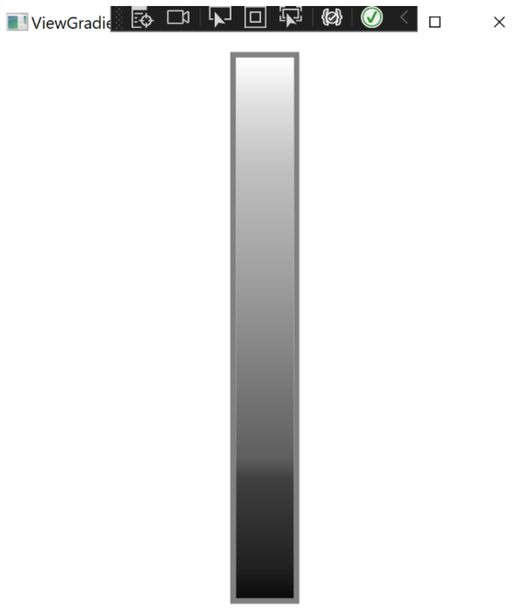


Figure 5. Standard Coloring Scheme   Figure 6. Greyscale Coloring Scheme

The ViewGradientWindow had two main components: ViewGradientWindow.xaml.cs and ViewGradientWindow.xaml. The ViewGradientWindow.xaml.cs component utilized C# and was where the coloring schemes on the backend were calculated. These were calculated using multi-dimensional arrays that hold RGB values to obtain a desired color. The RGB values inside these arrays could be changed depending on the desired coloring scheme. The ViewGradientWindow.xaml component utilizes WPF and was what displayed the desired color schemes shown in Figure 5 and Figure 6.

The FalseColor class utilized C# and through the usage of getters and setters held the user's choice of coloring scheme. This class had a relationship with both the ViewGradientWindow and the MainWindow classes as the chosen coloring scheme is denoted in MainWindow.xaml.cs, and then subsequently displayed in MainWindow.xaml. The chosen color scheme that was held in the FalseColor class was then denoted in ViewGradientWindow.xaml.cs and then displayed in ViewGradientWindow.xaml. This breakdown of the relationships between classes that the team will utilized is shown in Figure 7.

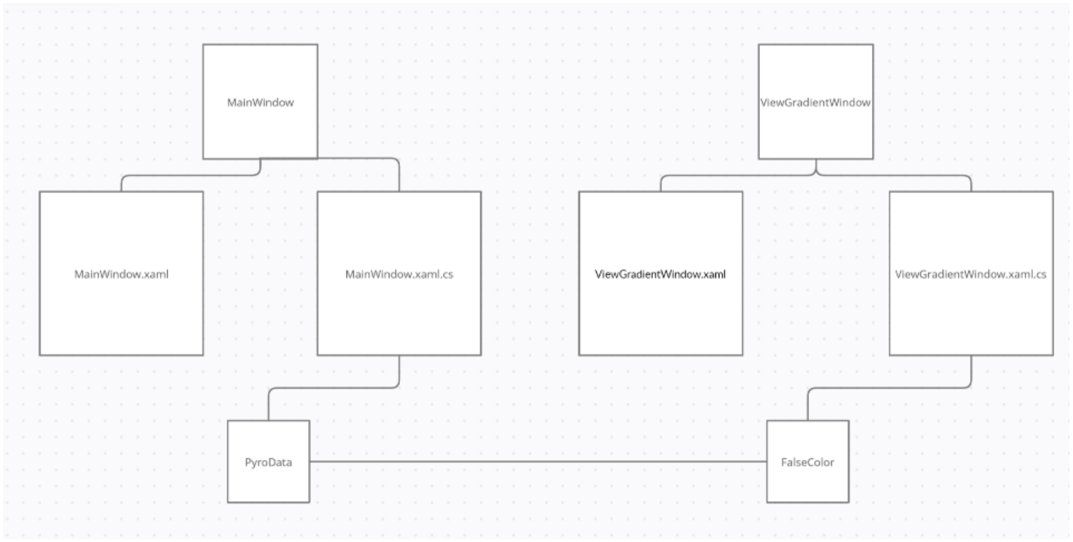


Figure 7. Relationship Between Relevant Classes

In addition to the gradient the team did implement calculations that adjusted how the laser could be viewed based on user specification:

Pixel Thresholding:

Each pixel that was displayed in the laser shown in the GUI had a specific intensity value. It was necessary for the user to be able to select only certain pixels above a defined specificity value. Figure 8 shows the laser in the GUI when pixel thresholding is turned off. The laser in Figure 8 is what the heat sensor is detecting with no additional parameters added to it.

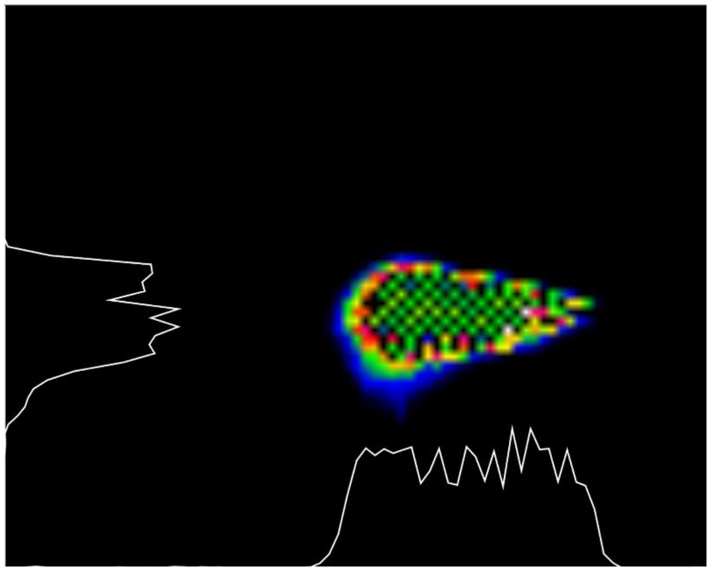
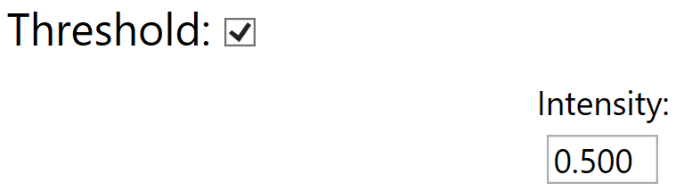


Figure 8. Laser Without Pixel Thresholding

When pixel thresholding was turned on and a subsequent intensity level was set, the laser would only show pixels with an intensity value at or above the specified intensity value. Figure 9 shows the GUI checkbox implementation to enable pixel thresholding with the intensity value set to 0.5 (50% intensity). Figure 10 shows the laser in the GUI when pixel thresholding is enabled.


Threshold: ☑

Intensity:

0.500

Figure 9. Pixel Thresholding Enabled With Intensity of 50%

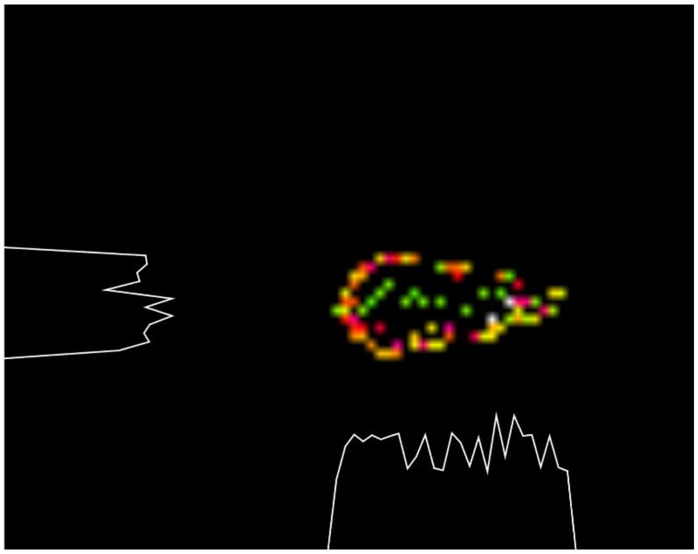


Figure 10. Laser With Pixel Thresholding Enabled

Gain Slider:

When the user desired to increase the pixel intensity shown by the laser it was needed to implement a gain slider so that the user could visually see the laser being updated in real time with increasing or decreasing pixel intensity. Figure 10 shows the gain slider. Figure 11 depicts the laser when the pixel intensity was turned to the maximum setting and refer to Figure 8 to visualize the laser when pixel intensity was turned down.

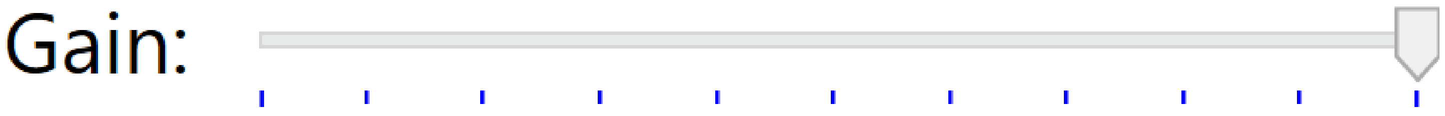


Figure 10. Implemented Gain Slider

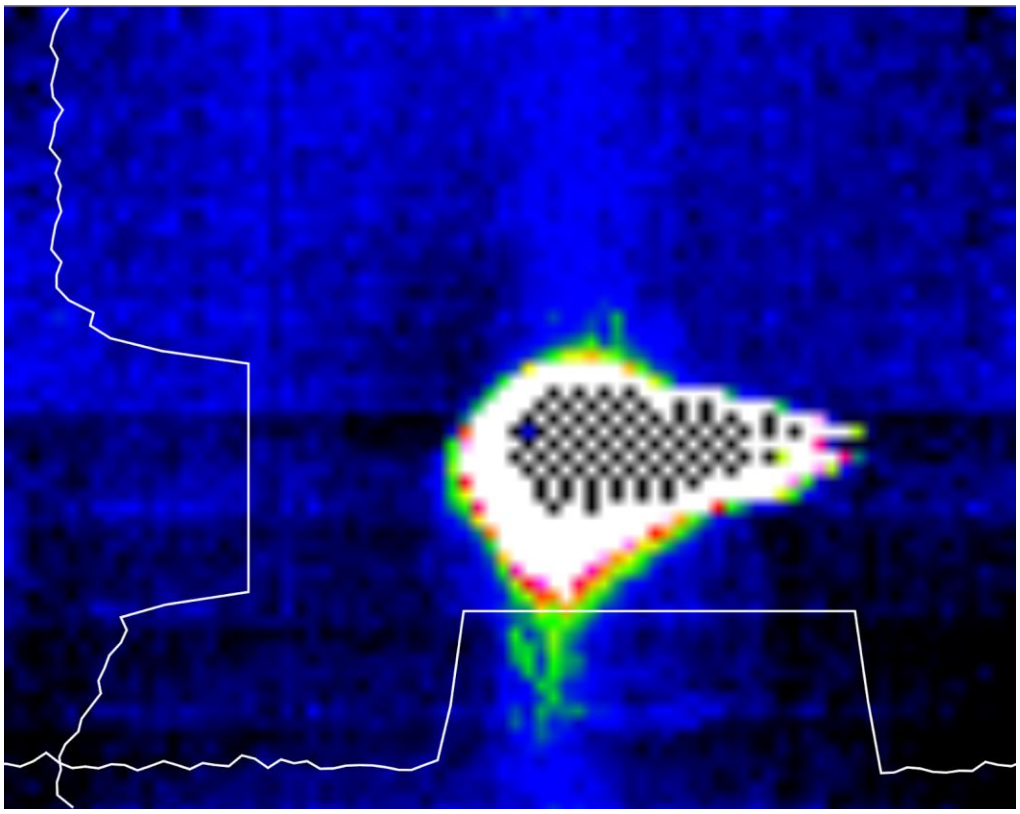


Figure 11. Laser With Gain Slider Set to Maximum Setting

Background Subtraction:

Figure 11 delineates a common problem with temperature sensors as the blue depicted in the background is "background noise". It is extraneous data that does not correspond solely with the laser. This noise can throw off visuals and various calculations. It was needed to incorporate a background subtraction method that calculated and removed this background noise so that the user could obtain an accurate reading with high pixel intensity values. Figure 12 shows GUI for calculating the background subtraction value and Figure 13 shows the same laser depicted in Figure 11 just with background subtraction calculated and that excess noise taken care of.

Background Subtraction: ☑

Figure 12. GUI Implementation for Enabling Background Subtraction

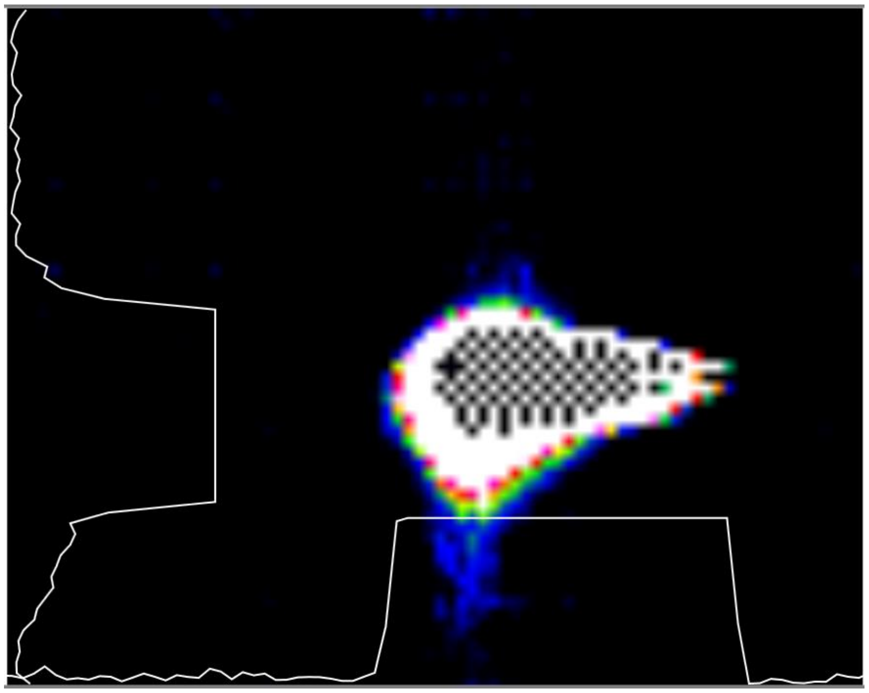


Figure 13. Laser With Background Subtraction Enabled

VII. Software Test and Quality

Testing the product was more nuanced and difficult than in a traditional sense. Due to back ordering issues the team was not able to obtain a temperature sensor in adequate time to test created code in a live setting. Due to this the team was provided bit maps from Optical Engines. These bitmaps were essentially sample data that the team read into the software on the backend. The bitmaps allowed the team to create a "Debug" method to the code so that we could test our calculations. The team was not provided test cases; testing the code operated by a certain team member conducting a calculation then once they got a working value being outputted that member would create a pull request on GitHub for Matthew to test with a live laser. This lack of overall tests made it hard to understand if the methods that the team created were accurate. Since Matthew and the representatives at Optical Engines have a nuanced understanding of the lasers and the mathematical equations that the team was creating validation checking was done on the Optical Engines end. If Matthew decided that the created method looked accurate then that respective branch would be accepted and subsequently merged with the master branch.

In addition to quantitative testing methods, Matthew did conduct periodic code reviews. These code reviews were to check functionality of the GUI on the front end as back-end functionality was tested in the above-described testing methods. During the weekly client meetings, feedback on these reviews was provided so that the team could accurately gauge software quality.

The software testing and quality checks were conducted in a manner that implemented agile development. This allowed for easier maintenance of the project as it allowed for the constant approval and merging of the team's pull requests. It also allowed for the group to be able to simplify the scope of the project and time manage various tasks at a more effective level.

It should be noted that test results were not applicable to this project. Since the tests were being conducted on the client's end and not by the team there were not physical test results that the team was able to provide or validate.

VIII. Project Ethical Considerations

When considering ethics in the development of this product the team referred to ACM/IEEE principles. The following ACM/IEEE principles were found to be applicable to the project:

Principle 1: Public

- Principle 1.01: Software Engineers shall accept full responsibility for their own work.
- Principle 1.03: Software Engineers shall approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate test, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.

These principles were deemed to be applicable to the project as by accepting responsibility for our own work we were able to hold each other accountable and create a better product in the end. It was also necessary to make various pull requests when that respective method was accurate rather than creating a pull request and hoping that Optical Engines would not notice and merge the request regardless.

Principle 2: Client and Employer

- All the subprinciples for Principle 2 were applicable in the development of this product.

Trust between client and employer is critical in the successful creation of a project so by relying on the subprinciples listed in Principle 2 the team was able to successfully maintain a professional and healthy relationship with Optical Engines.

The team did risk violating ACM/IEEE principles. The following ACM/IEEE principle was deemed to be at risk of being violated:

Principle 3: Product

- Principle 3.02: Software Engineers shall ensure proper and achievable goals and objectives for any project on which they work or propose.

The team did have limited knowledge of lasers and optics and it was because of that the team had to recognize our own limitations in developing the final product. If the team were to have been stuck on a certain task and not been able to meet a set deadline, then that client-employer relationship would have diminished. As a result, the team felt it necessary to admit our own shortcomings to the client so that all parties involved could set reasonable deadlines for various tasks.

To successfully navigate these ethics concerns the team incorporated the following ACM/IEEE tests:

The Mirror Test:

- Creating something meaningful as a team allowed for a sense of accomplishment in the software industry. This feeling would elevate our self-efficacy, and as such increase confidence as the team progressed throughout our professional career.

The Harm Test:

- The team needs considered how changing the GUI and adding elements to the laser math on the backend would cause harm. It was the team's belief that these updates would do less harm than the alternative (keeping the software per the initial model). The benefit of changing the GUI was creating a more versatile that could be brought to market. In its initial form, the product was too basic to be sold as a product and by updating it the team was creating more benefits than harm in order to create a marketable product. If the software plan was not properly implemented, then it became easy to forget about various components. If components were not added due to improper planning, then it ultimately lowered the functionality of the product and lowered the cost that it could be sold for.

IX. Project Completion Status

The final project application that the team created meets all functional requirements. The GUI on the front end as well as the mathematical implementation on the back end all meet client satisfaction. It should be noted that various components of the project were scrapped mid-way through the project. These scrapped implementations involved both the Aperture calculation request as well as the EEPROM calculations that provided Arduino work. This allowed the team even further time in developing a well-versed GUI with high functionality on the backend as the team was able to narrow focus.

The GUI does not format correctly on a personal laptop. Optical Engines implements a single Dell computer to run this software and due to this the value for the width and the height of the GUI were hardcoded in to fit the screen on that respective computer. Since none of the team members owned a Dell computer those hardcoded values caused the GUI to be formatted incorrectly on the team's personal machines. The team is unsure of how to correct this formatting issue, however what was of upmost importance to the team was that it displayed properly on the client's end.

X. Future Work

Future work comes in the forms of working with the scrapped components of the project. Being able to work with an Arduino to improve EEPROM functionality would allow for a more versatile product. Being able to change the label on the Arduino from "Arduino Due" to "SpryoCam" would help with potential legal issues when bringing this product to market. Additional research with regards to these legal concerns is needed so that Optical Engines can ensure that they can bring the product to market. Incorporating the Aperture setting would allow for improved user functionality so that additional laser calculations can be performed. One of the client's requests was a way to speed up the software in case the added functionality caused the system to slow. However, the functionality that the team incorporated did not seem to cause a decrease in speed so there was no need to include this request and as such it was not implemented. It is necessary in the future to monitor the speed that the software operates at in case it is needed to incorporate a way to allow the improved functionality at desired speeds.

XI. Lessons Learned

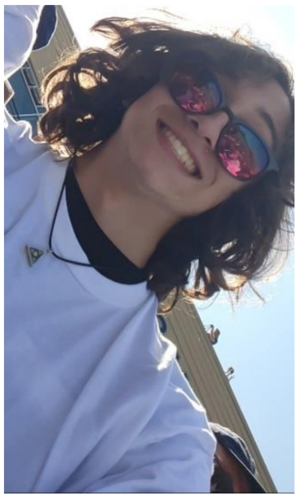
Over the course of working for Optical Engines there were takeaways that the team garnered and can apply to future endeavors in computer science related fields. These lessons learned include:

- It was not enough to understand what to code. To create a better product, it was important to be well versed in what exactly we were coding. There was a learning curve at the beginning of the product as none of the team members had adequate knowledge of lasers or optics, and because of that there was limited understanding in what the team was requested to do.
- The team could never do too much research on what it was that we were doing. The project found the team combing through research papers and sending the client countless emails in order to understand the functionality of the task at hand. There was a learning curve but through the team's desire to know more and understand the requests we were able to build a better product.
- That the software language C makes it easier to implement "getters" and "setters" and these "getters" and "setters" work well with implementation in XAML.
- The team learned how to debug code when a lot of the built-in helper methods from the interface are not provided. The team worked in Visual Studio and when working with XAML in Visual Studio the interface does not warn the user of potential errors in their code. This caused for additional headaches as the team had to track down various bugs or misspelled words or anything that could have caused variables to not be showing properly in the GUI.
- The team learned that Scrum works well in project management. Scrum principles were applied in order to help the team understand what needed to get done during a certain sprint, as well as assigning tasks in a sprint to certain team members. It was helpful to understand that once a request on the backend was complete by a team member, that it went to the next team member who was then tasked with binding the method into the GUI. This sort of assembly type process of having team members focus on either the backend or the frontend and then sending the method off the next person allowed for the team to maximize our efforts.

XII. Team Profile

Drew Cornmesser
Senior
Computer Science
Hometown: Fort Collins, Colorado
Work Experience: CS@Mines Internship, Introduction to Computer Science Teacher Assistant, LEAD Peer Mentor
Sports: Golf, Basketball, Running

I am excited to work towards furthering my knowledge of math and applying it with computer science in order to create a product of significance in the world of optics and lasers.



Elian Estrada
Senior
Computer Science
Hometown: Muskegon, Michigan
Work Experience: Northwestern Mutual, CS@Mines Internship, Cinemark, Whole Foods
Clubs: Sigma Alpha Epsilon Fraternity

I am excited to work on a new area of computer science that I have not done yet.

Carson Cramer
Junior
Computer Science
Hometown: Parker, Colorado
Work Experience: Co-Founder of Fractyl Development LLC.
Sports: Baseball, Golf

I am excited to understand WPF so that I can further my knowledge in working with user interfaces and gain more experience working with a different operating system.

References

[1] “CS Field Session Optical Engines Inc,” *CSCI 370 - Advanced Software Engineering*. [Online]. Available: https://cscourses.mines.edu/csci370/FS2022S/Proposals/OpticalEnginesInc.pdf. [Accessed: 19-May-2022].

[2] R. Awati, “What is EEPROM (electrically erasable programmable read-only memory)?,” *WhatIs.com*, 11-May-2022. [Online]. Available: https://www.techtarget.com/whatis/definition/EEPROM-electrically-erasable-programmable-read-only-memory#:~:text=EEPROM%20is%20a%20type%20of,computing%20and%20other%20electronic%20devices. [Accessed: 19-May-2022].

Appendix A – Key Terms

Term	Definition
EEPROM	Electric Erasable Programmable Read Only Memory is “a type of non-volatile ROM that enables individual bytes of data to be erased and reprogrammed” [2].
BEAMS	BEAMS are just that: beams! In the scope of this project this term is talking about beams of light that are emitted from lasers.