uOttawa

Tracking System
&
Wireless Transmission

Allami, Noor	7515405
Han, Shuo Tony	7056419
Liu, Tsa-Chun Zac	6484583
Zurkiyeh, Anas	7340941

April 7, 2017

ELG4913B Electrical Engineering Design Project II

Contents

List of Figures

List of Tables

Abstract

This report is designed to demonstrate the progress that has been made on a project decided upon in the Fall 2016 semester. The purpose of the project is to develop an electrical engineering solution for a company developing a visual prosthesis system. iBIONICS is a start-up company that is currently working on developing a solution to people with retinal degenerative diseases. Their developed retinal prosthesis system, The "Diamond Eye", will require the wireless transmission of data by the use of laser as well as the tracking of a target. The target is a photodiode that will receive the data transmitted by the laser and convert it to an electrical signal. Although the design developed is proposed for a company developing a retinal prosthesis system, it can also be implemented in other systems such as those that require redirecting a wireless beam onto a moving target.

Throughout the implementation stage of the project, there have been many changes made to the initial plan in terms of parts and scheduling. The project is delayed due to the late arrival of the MEMs mirror, which for the time being have been replaced by a temporary mirror system. In addition, there have been changes to the type of target which will be used. The data collected throughout the implementation stage are included in this report to illustrate the progress made by the team members.

The design of the wireless transmission system, as well as the tracking method developed by the students will be the intellectual property of the students. Existing intellectual property belonging to iBIONICS may be used for the report component of the project, but not to the extent of violating the Non-Disclosure Agreement.

DISCLAIMER: The students will not conduct any human experiments or interact with a human eye. The project designed will only consist of electrical components set up at SUNLAB.

1 Introduction

1.1 Scope (Noor)

The scope of this project will consist of designing a wireless communication system between a camera and a photodiode by using laser to transmit images and power the target. A system will also be designed to implement tracking the movement of the target to redirect the laser beam using a MEMs mirror.

Since the communication system and the tracking system designed could be implemented by iBIONICS as part of a bigger project, the Diamond Eye, there will be certain tasks that we will not deal with and are out of our scope. The group members will not be responsible for the transfer between the PV chip and the electric chip that it is attached to. The design of the glasses as well as integrating the system onto the glasses will be out of the scope of this project. not be involved in any human experiments or any biological trials.

1.2 Goal (Noor)

1.2.1 Primary Goal

Tracking a target that rotates along the surface of a sphere within the range of $\pm 30^{\circ}$ in any direction from the origin and redirecting the laser beam onto the target's new position using a MEMs mirror.

1.2.2 Secondary Goal

Wirelessly transmitting data that is a decoded image sent as a modulated signal to the laser driver, as well as demodulating and displaying the received data.

1.3 Overview (Noor & Anas)

This project is designed to solve an electrical engineering problem for a company developing a visual prosthesis system. Although the design developed is proposed for a company developing a retinal prosthesis system, it can also be implemented in other systems such as those that require redirecting a wireless beam onto a moving target. The team will only be developing systems and solutions related to electrical engineering.

Currently under design is the "Diamond Eye" by a company called iBIONICS. The "Diamond Eye" is a system, if successfully implemented, will revolutionize the field of visual prostheses. Designed to be implemented as a wireless system, the technology will alleviate many disadvantages currently present in the Argus II product, the only solution currently available for people with retinal degenerative diseases. The "Diamond Eye" will provide people with blindness, due to diseases, the ability to see without the risks that are associated with current technologies. This project consists of developing electrical engineering solutions to some features of the "Diamond Eye", which are outlined in further detail in the scope of the project. Once successfully implemented, using wireless technology will enable iBIONICS to further enhance the product easily and in the least intrusive way possible.

The opportunity to collaborate on this project was provided to the group through SUNLAB at the University of Ottawa. SUNLAB obtained this project as a research opportunity from iBIONICS. Although the project is facilitated by SUNLAB, iBIONICS will be funding the project by providing all the necessary parts to build the prototype. The details of the design which is currently being implemented will be discussed in subsequent sections. Our project will consist of a camera that will capture data to be transmitted to the photodiode (target). The data will be processed and converted to a digital stream of bits using a microcontroller and an encoder. The stream of data will drive a laser into transmitting a corresponding trail of bits which simulate the digital signal. The stream of bits steered by a MEMs deformable mirror will be transmitted onto the photodiode that will convert the laser beam into digital signals. The signal will be ltered and sent to a decoder to decode the data into pixels

to be displayed by the use of a microcontroller. A tracking system will also be developed to ensure that the laser driver will send the beam to the photodiode with minimal reflections. This will be done by the use of a camera to capture the movement of the target. By capturing the movement of the target, information can be sent to a microcontroller to ensure that the MEMs mirror will always reect the beam accurately. There are many dierent aspects to the project that will need to be considered and studied carefully. Research methods and detailed designs will be following in subsequent sections.

1.4 Literature Review

1.4.1 The Argus II (Noor)

The Argus Retinal Prosthesis System is the first approved retinal implant to restore partial functional vision to people with blindness, created by a company called Second Sight [3]. Second Sight developed their first retinal implant which was successfully implanted in 6 people [2]. The very first implant, the Argus I, contained 16 electrodes, and was later developed into the Argus II which now contains 60 electrodes [3]. The process used by the Argus II system is outlined in the steps below [3].

1. A video camera captures the scene
2. Video is transmitted to a computer which is worn by the patient
3. The processed data is sent back to the eyeglasses to be transmitted to the implant
4. The implant receives the data by the use of the attached antenna
5. The data is sent to the electrode array which converts the data into electrical pulses
6. The electric pulses are used to stimulate the cells in the retina

The Argus Retinal Prosthesis System is currently the only available solution to people with blindness due to retinal degenerative diseases. During the trial of the Argus II, all 30 patients were able to perceive light during the stimulation [5]. However, it is not an ideal system, due to associated disadvantages.

1.4.1.1 Disadvantages of the Argus II

Although the Argus II is the only available solution for people with retinal degenerative diseases, there are many drawbacks associated. Many patients had adverse effects to the implant, and although they were successfully repaired, these should be noted. According to [5], 10% of patients developed conjunctival erosion over the implant, two patients had retinal detachments after the surgery, and one had to have the implant removed. This can also be due to the fact that the Argus is placed by the use of a circumferential vitreous band that surrounds the human eye as shown in Figure 1 below. This can also be the cause of the lengthy procedure to place the implant.

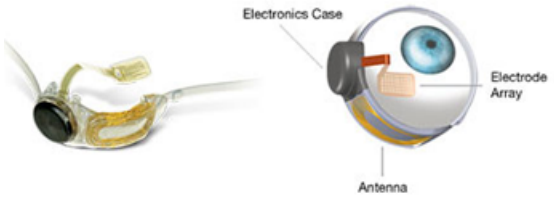


Figure 1: The Argus II Implant [3]

The procedure that patients receiving the Argus II have to go through is a long, complicated, and costly procedure. The procedure is complicated and lengthy due to the necessary placement of the implant and all the

wires that are connected to it. The implanted wires also increase the risk of bacterial infections that may be caused during the implant procedure. The procedure can take up to several hours due to the complication of the implant. The length of surgery, increases the cost of the overall Argus II Implant system, making it unaffordable for most people. It is currently estimated to cost $150,000 [6], excluding the cost of surgery. Therefore, although it is the only solution that people with blindness have, the Argus II has many drawbacks as discussed above.

1.4.2 OpenCV (Anas)

Is a multiplatform API - Application program interface that contains hundreds of computer vision algorithms. It is compatible with multiple coding languages such as C++, Java and Python which makes it very convenient if its used in conjunction with a computer-board such as a BeagleBone or a raspberry pi, which means that it supports different operating systems such as MacOS, Windows and Linux which simulates the environment on the mini - computers.

OpenCV has a many different static and public libraries and packages that can be embedded and utilized. These packages allow users to take advantage of previously implemented functions that can be manipulated for different usages. These packages include [**?**]:

- **Imgproc**: an image processing package that includes filtering, geometrical image transformations (resize, affine and perspective warping), histograms, and many more. It allows easy editing and manipulation of images, which means that users do not have to use external resources to do so.
- **Video**: a package that handles motion estimation, background subtraction, and object tracking algorithms.
- **highgui**: an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.
- **gpu**: stands for *Graphics Processing Unit* and it is simply a group of accelerated algorithms for a faster use.

It also contains method for automatic memory allocation and management, in addition to many functions that make it fixable and user friendly. It allows the ability to configure -since its open source- all the different aspects of the CAMShift algorithm such as the window size.

Another choice to make is whether to implement using Python or C++, since both the mini-computer and OpenCV support both, which one is better to use? Next we list a few pros and cons of each and make a decision based on them [**?**].

Table 1: Programming Language Pros and Cons

	Pros	**Cons**
C++	• Huge optimized library • Big community • Platforms and devices	• Weak documentation • Small machine learning library • Visualization and debugging

Python	• Ease of use • Python has become the language of scientific computing • Visualization and debugging • Building web backend	• Weak Documentation • Lack of Support • Slower run time • OpenCV is written in $C/C++$

1.4.3 Manchester Encoding (Zac)

Manchester encoding is a form of digital encoding in which data bits are represented by transitions from one logical state to other [13]. Where the typical encoding method is when the data is 0, the encoded bits become 01 and when the data is 1 the encoded bits become 10. This encoding method was developed by G.E. Thomas. The relation can be shown in Table 2.

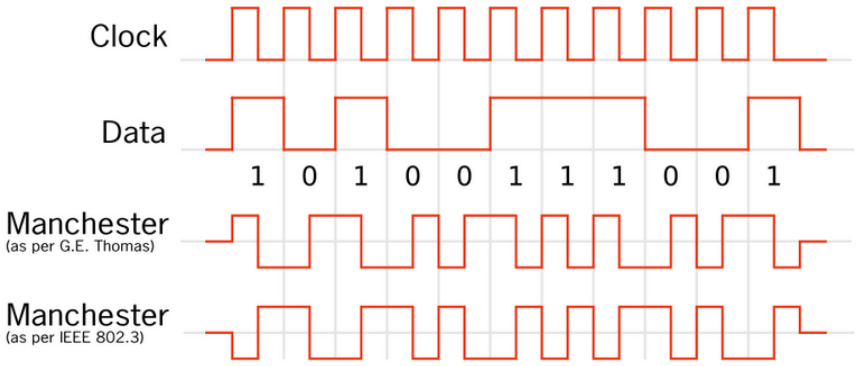


Figure 2: Manchester Coding

Table 2: Mancheser Coding Logic

Data	Clock		Manchester
0	0	XOR	0
	1		1
1	0		1
	1		0

From Table 2 we can see that the advantage of using Manchester Coding is that the signal synchronizes itself by being directly proportional to the clock rate [15]; which helps clock recovery. So the encoded values are also equal to the data bits XOR with clock. By having the signal synchronizing with the clock minimizes the error rate and optimizes the reliability, but the major disadvantage is that Manchester encoded signal requires more bits than the original signal to transmit, i.e. transmission rate is twice the original, [13], which also increase to twice the original bandwidth.

1.4.3.1 Encoding and Decoding Method

The encoding for Manchester can be simplify by translating 0 and 1 data bit to 01 and 10 encoded bits. From Marty [14] sync the encoded bits with the clock he defined. The total packets he send contains 8 bits of initial clock, 8 bits of sync patterns 2 start bits and final the encode data bits. He detects the Most Significant Bit (MSB) every positive trigger of the clock, then encode with every positive and negative trigger of the clock; this way he is synced with the clock and the encoded bits have twice the frequency respect to the original signal. After detecting the MSB he shifts the position of MSB left and detects the next bit.

Marty's decoding method is to detect the transition of encoded bits. He look for whether there is no bit change for a cycle of clock, the program decide there is a bit change in data bit, if not, there will no data bit change; hence the same of previous detect data bit. After detecting every 2 encoded bits, the program shifts left, hold the decoded data bits. After 16 encoded bits or a byte of decoded bits, the program save the values from hold [14]. Before the decoding is proceed, he first check if the received data is synchronized by setting upper and lower threshold. If there is no transition for the received data, the program waits until there is transition. Once a transition is detected, the program check if the bit time is within the upper and lower bit threshold time. If it is then the received data is synced with the clock, otherwise they are not synced. If the data is not synchronized, the program waits for another transition to happen. Once the first transition of the received data is detected, the decoding program start its clock for decoding [14].

1.4.4 Pinhole Camera Model [18] (Tony)

Camera captures the real world object and project it on the film.

Transform the 3-D real world to a 2-D image

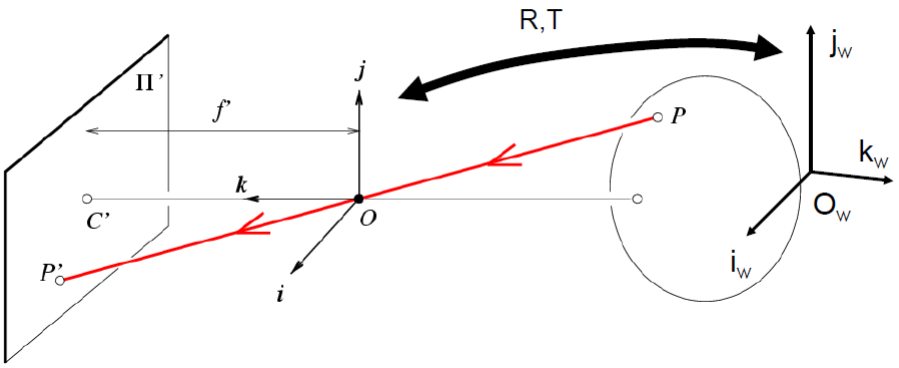


Figure 3: Illustration of Pinhole Model[18]

The light pass through a hole and project onto the film.

It will be shown in this section that how to project a 3D world point (X,Y,Z) onto the image plane at coordinate (u_1, u_2). And convert the coordinate into pixel with index a_{ij} where it is the i^{th} row and j^{th} column of the m $*$ n matrix (m $*$ n pixels).

Camera model transformation:

$$u = K \begin{bmatrix} R & t \end{bmatrix} X \tag{1}$$

Where:

- u: Image coordinate $(u_1, u_2, 1)^T$

- K: Intrinsic Matrix which shows the properties of the camera
- R: Rotation Matrix represent the rotation
- t: Translation Matrix which shows the position of camera
- X: World coordinate $(x, y, z, 1)^T$

Approximation from 2D to 3D.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix} + R^{-1}K^{-1} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} \tag{2}$$

Real world coordinate is in millimetre.

2 Design

2.1 System Block Diagram (Zac)

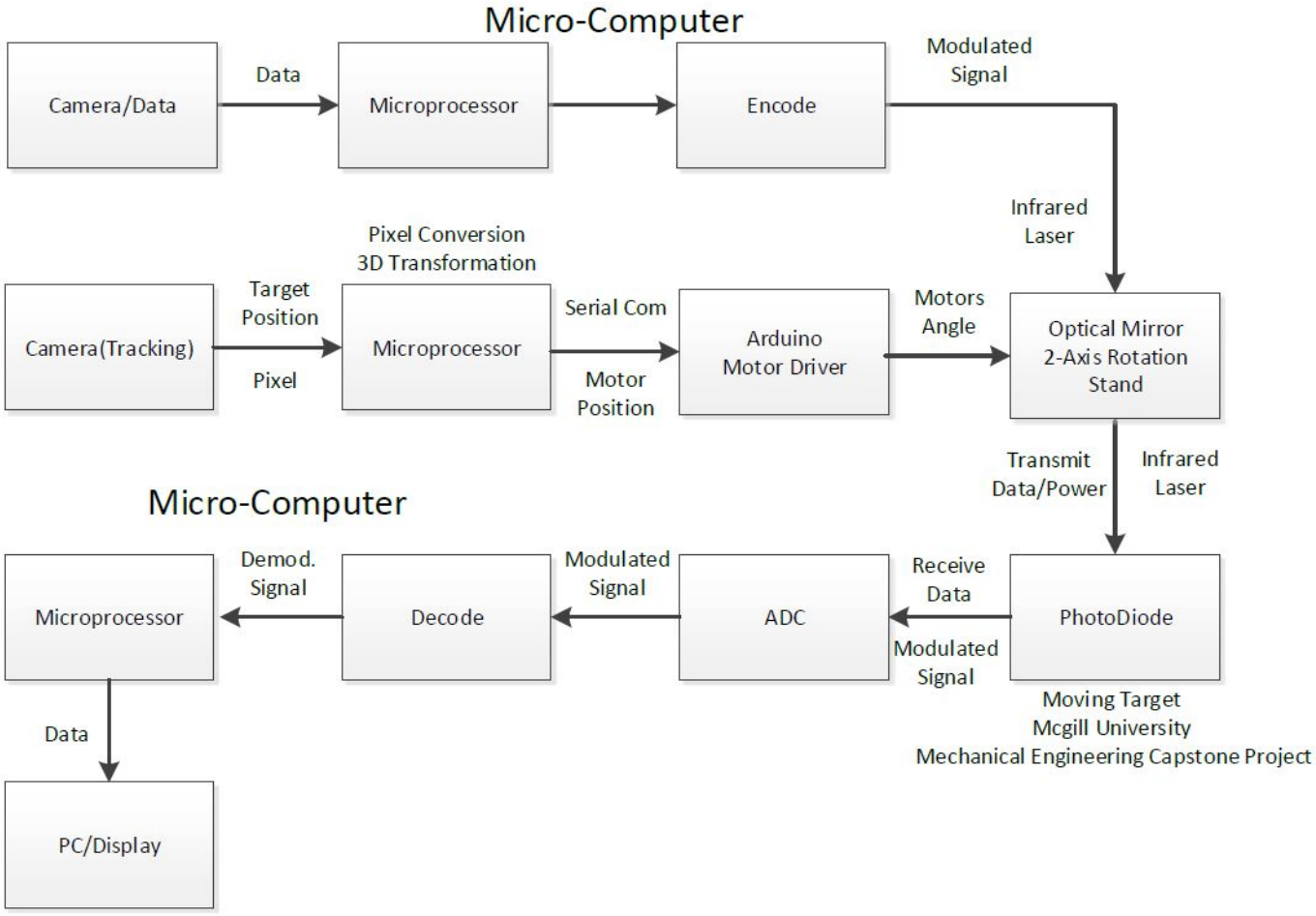


Figure 4: System Block Diagram

From the system block diagram in Figure 4, there are two cameras. The first camera is use to capture the image into data for transmission, the second camera is use for tracking the movement of the target. The image captured from the first camera will be processed and encoded by a Raspberry Pi 3. The processed data will be sent to a laser diode. Once the laser is send to the optical mirror, it will depends on the tracking system to tell the mirror where to reflect the laser. Hence the second camera will be use for tracking. The second camera will capture the pixelized target position, hence we need to convert the pixel into position relative to the optical mirror in 3D space and communicate to the servo motors on the optical stand. The reflected laser will be send onto a photodiode to receive the data transmitted. The photodiode will be mounted onto a Dynamic System [1][1] that simulate the movement of the target. The Dynamic System is created and built by Mechanical Engineers from Mcgill University for their capstone project. Once the photodiode receive the data, it will go through a filter and then pass to another Raspberry Pi 3 to decode and display the result.

[1]Materials related to Dynamic System must be treated as confidential

2.2 System Design

2.2.1 Physical Design (Noor)

This section of the report highlights the experimental design of the project, specifically the physical setup of the integration of the individual components. Due to the nature of the project, tracking the movement of a target as well as wirelessly transmitting data, the components are kept individually and integrated by wire connections. By connecting the components via wired connections as a prototype, a proof of concept is performed. The connections shown in Figure 5 below will be used as a guideline for demonstrating the end project.

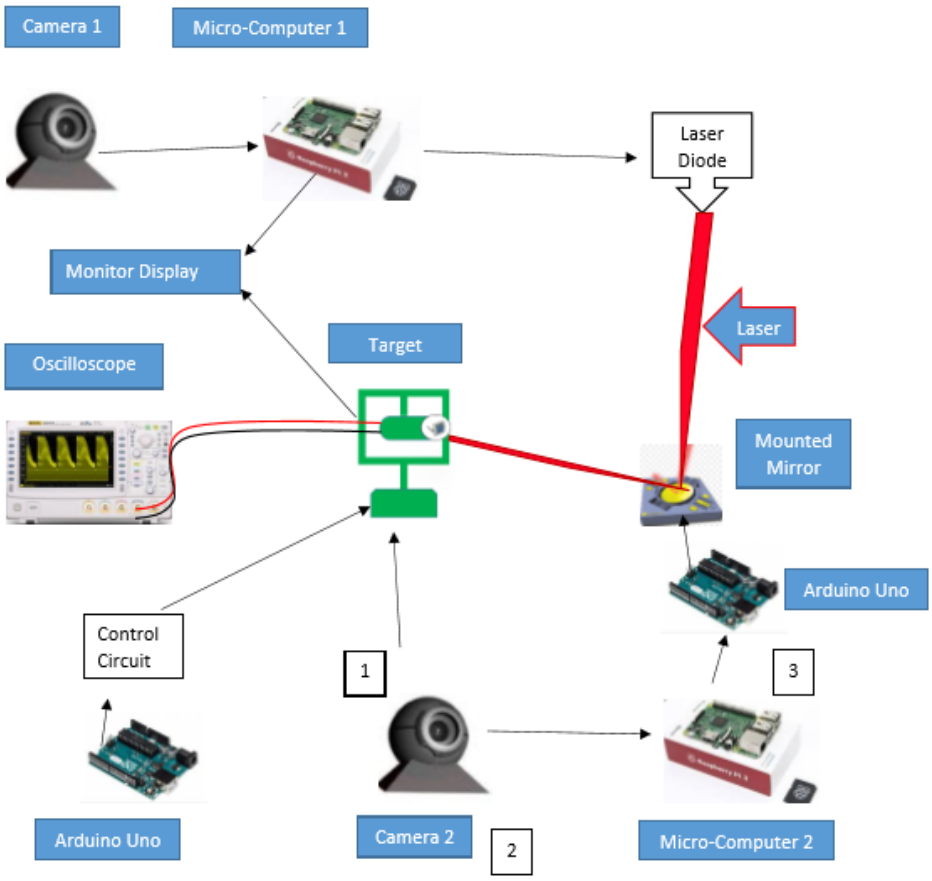


Figure 5: Physical Design for Project Demonstration

As shown in Figure 5, Camera 1 is designated to capture the data that represent the target's view, which is the data to be transmitted to the target. The data is then processed by Micro-computer 1 and used to power the laser diode. The laser beam will then deliver both the data as well as the power supply to the target. The laser beam will not be directly pointed towards the target, but will be reflected off of a mirror to provide a wider angle of coverage and easier control during the tracking. The target, a photo diode, will convert the received laser beam into an electrical signal. The target is in turn connected to the oscilloscope as well as a monitor to display the received and converted signal.

Since the target will be rotating $\pm 30^{\circ}$ in any direction from its initial position controlled by and Arduino Uno, tracking of the target is necessary to ensure that the laser beam reflected off of the mirror will be aimed appropriately. To track the target, Camera 2 will be positioned directly across the target, to continuously capture

images of the target's position. The images will be processed by Micro-computer 2 and depending on the location of the target, signals will be sent to the Arduino Uno which control the motors that control the mirror to accordingly adjust the mirror's rotation angle. Using the coordinates obtained from the target's image, the angle of the mirror will be adjusted according to the coordinate system designed for the physical setup.

As a result of the limited time frame of the project, the integration of the system into an ideal prototype as that shown in Figure 6, is not possible. Figure 6 illustrates the design of the ideal system, integrated onto a pair of glasses, powered by an external battery pack. Using very similar components to those shown in Figure 5, the two cameras, the laser diode, MEMs mirror, and microprocessors, the design shown in Figure 6 should yield similar results to those achieved by the system shown in Figure 5. The figure below shows different angles of a well-integrated system that has the same functionality as that shown in Figure 5, but with a sleeker look and a more practical design. Designed to be a mobile system, the prototype shown in Figure 6 is more practical as it is very compact.

Figure 6: The Physical Design of the Ideal System [17]

2.2.2 Tracking System Design (Zac)

2.2.2.1 Circuit Design

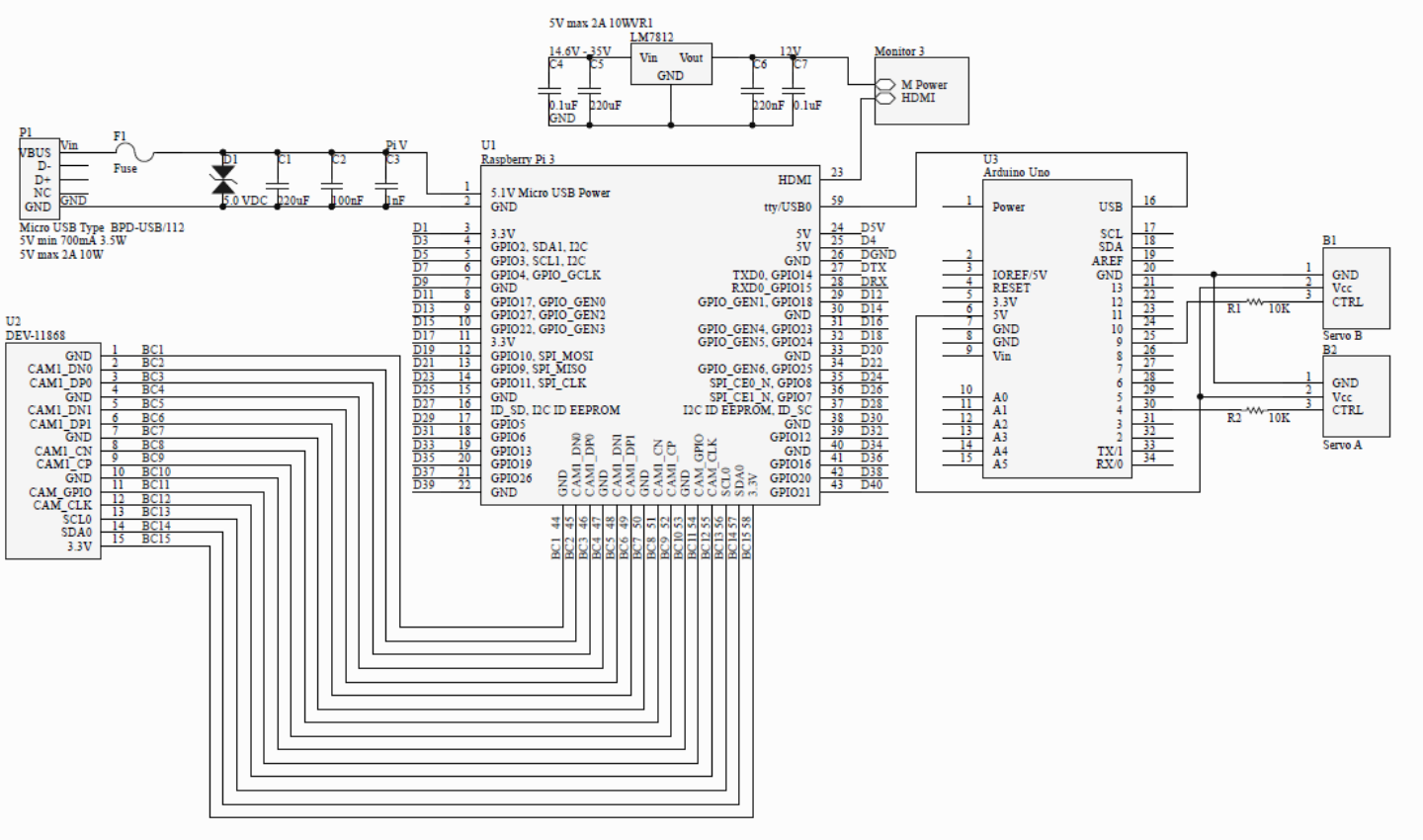


Figure 7: Tracking System Circuit Design

In the replacement of MEMs mirror, we are using two servo motors to rotate an optical mirror for 2 axis of rotation. The motors are labelled as Servo A and Servo B in Figure 7. The two servo motors will be driven by an Arduino Uno with PWM signal, hence the Arduino Uno serves as a motor drive in this circuit. The Arduino receive the position of where the motors should rotation and power directly from the USB port of a Raspberry Pi 3. The Raspberry Pi 3 will capture the target's position and in pixel and covert into physically unit and then convert into physical angles of rotation in reference of the motors. All these calculation will be done on the Raspberry Pi 3.

The tracking system can be divided into 3 parts. The first part is the MEMs system, which will be replaced by optical mirrors, servo motors and Arduino (Section 3.3). The second part is the tracking algorithm development, which will be running on the Raspberry Pi 3 along with the Raspicam. The third part is the Camera Calibration or the camera which will be done by running through the algorithm on MATLAB with pictures taken from the Raspicam .

2.2.3 Transmission Design (Zac)

2.2.3.1 Transmitter Circuit Design

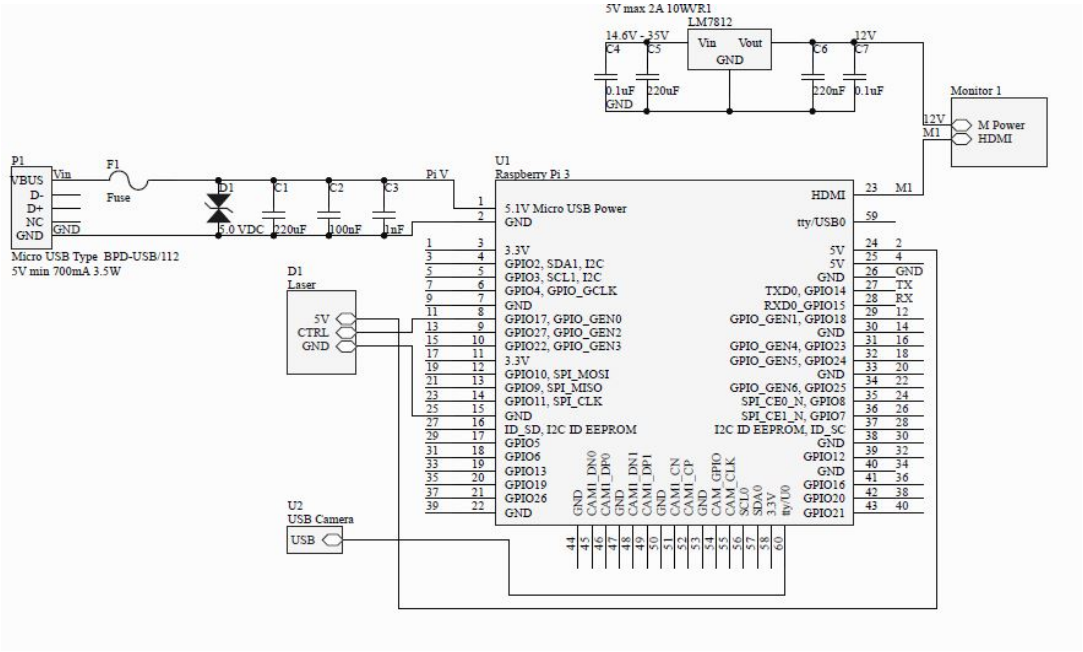


Figure 8: Circuit Design of Transmitter

The design of the transmitter consists of one laser and one webcam. The transmitter Raspberry Pi will take the image from the webcam, pixelize it through image processing and output into the laser. The monitor will be use to observing the original webcam frame and also the processed frame with its binary stream.

2.2.3.2 Receiver Circuit Design

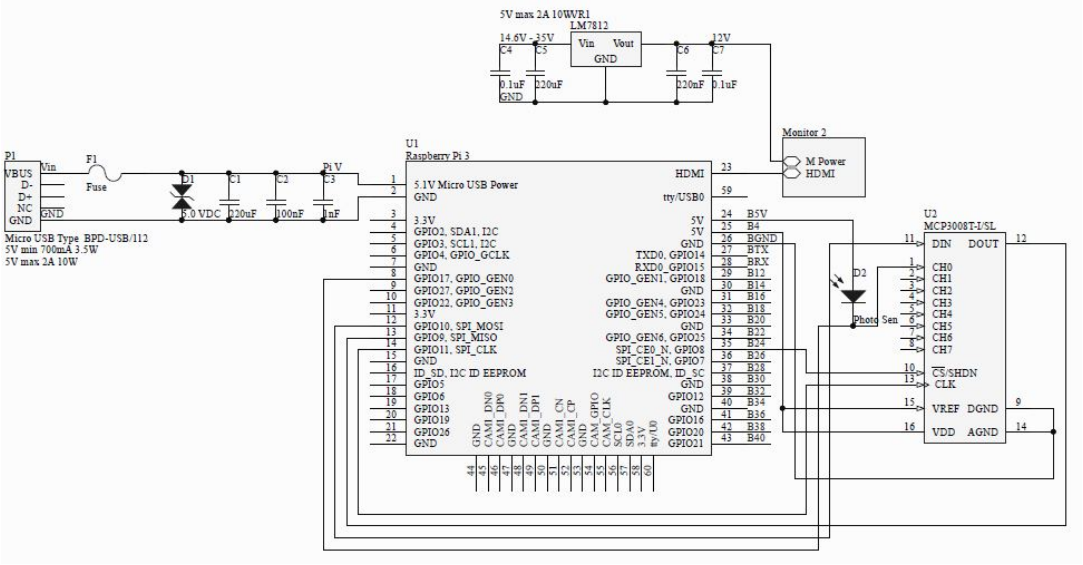


Figure 9: Circuit Design of Receiver

Since the receiving signal will not always have 0V or 5V for logic 0 or 1, and we will not have a stable connection between the transmitter and receiver, we cannot use the TX and RX of the Raspberry Pi. Since it will not always reach 5V when receiving the signal, the digital signal reader on the Raspberry Pi, due to unknown proper threshold voltage. So the only option is to use ADC and make logic decision in the software, but Raspberry Pi itself does not have ADC, so we have to use an external ADC which is the MCP3008 ADC chip from Microchip. [8]

The Vdd is the supplying voltage, while the Vref is the reference voltage for the ADC; hence both pins are connected to the 5V of the Raspberry Pi. Since the analog and digital share the same ground, both of them are also connected to the GND of the Raspberry Pi. The CLK is the sampling clock for the ADC, while Dout is Digital Signal output and the Din is the Digital Signal Input.

2.2.4 Dynamic System Design[1] (Zac)

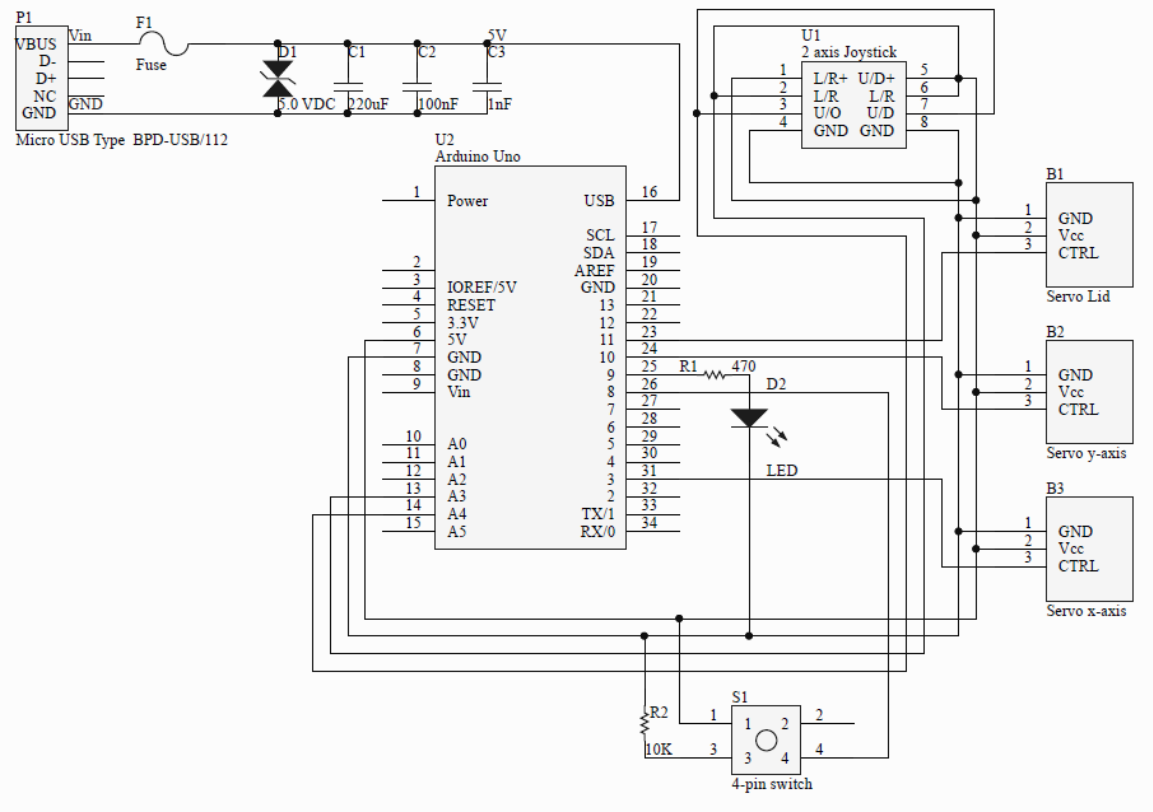


Figure 10: Dynamic System Circuit Design [1][2]

The system is done by Mechanical Students from Mcgill University for their Capstone Project. The Mcgill Students are also working with iBionics for the development [2]. This system will approximate and simulate an actual eye, hence this system will give us a bottom line of how well our system needs to perform. This system includes a 3D model made with respect to actual eye's dimension.

Our target, which is the Photodiode, will be mounted onto this system and our tracking system will need to actively track its location and adjust the mirror's angle. This system allow us to manually move the target with the joystick or make it move randomly. We can adjust their system to limit its speed of rotation and degrees of rotation to test our system's performance.

[2]Materials related to Dynamic System must be treated as confidential

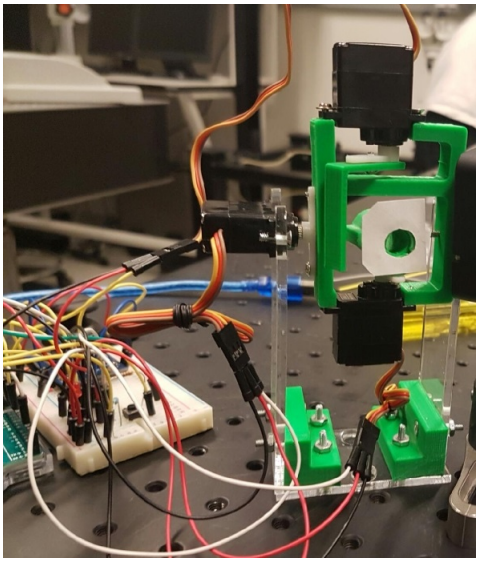


Figure 11: Dynamic System Model[1]

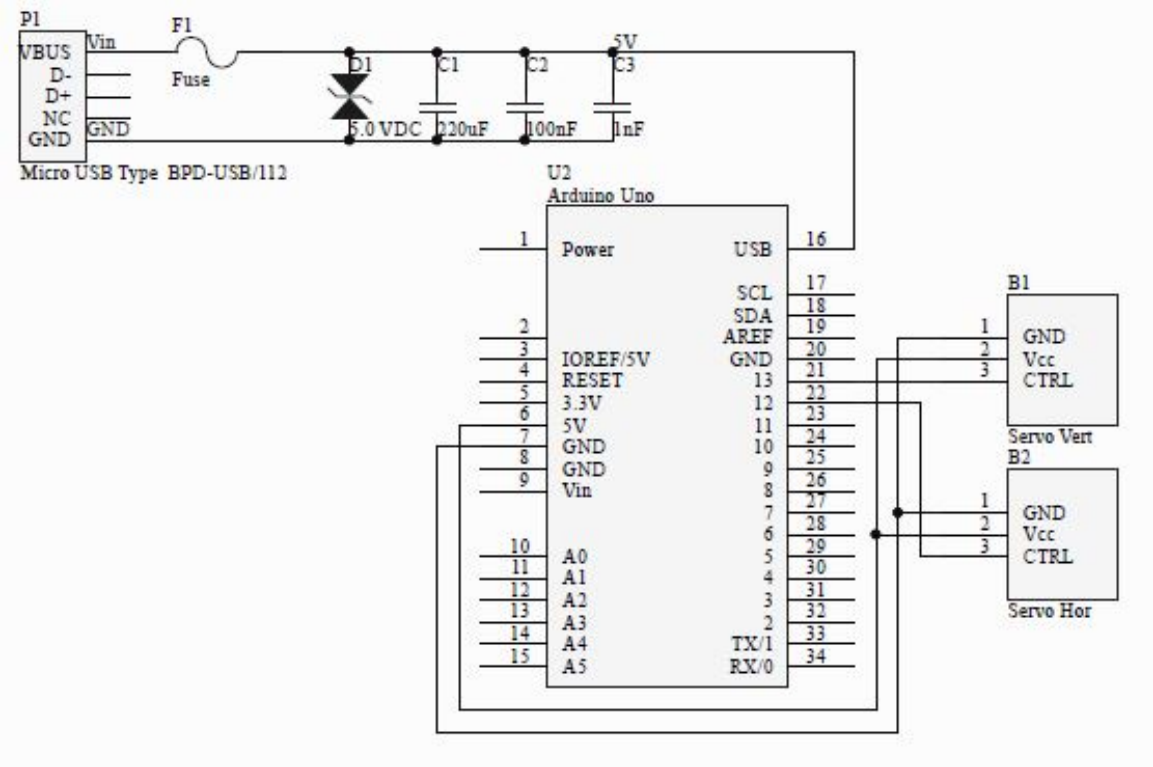


Figure 12: Adjusted Dynamic System Circuit Design

The detail reason and method for adjusting the Dynamic System will be mention in Section 3.2.1.

For confidentially reason, we cannot provide the algorithm or code of the Dynamic System.

3 Real Implementation

3.1 Experimental Setup (Noor)

3.1.1 Design of Coordinate System to Determine the Mirror's Angle of Rotation

Before implementing the physical setup of the experiment, a coordinate system is designed to ensure all the components are aligned appropriately. A coordinate system shown in Figure 13 below is designed to determine the rotation angle of the mirror based on the location of the related components in the experimental setup. In particular, the location of the laser diode, the target, as well as the location of the camera are used to calculate the necessary rotation angles of the mirror. Since the system is actively tracking a target by the use of a camera, the images produced by the camera will show the location of the target tracked in the image. The coordinates produced by the image will relate to the coordinates of the target in the real world by implementing a camera calibration pinhole model. Hence, when the image is produced, the camera pinhole model will convert the coordinates of the target produced by the image to the real coordinates of the target providing the X, Y, and Z components of the location of the target.

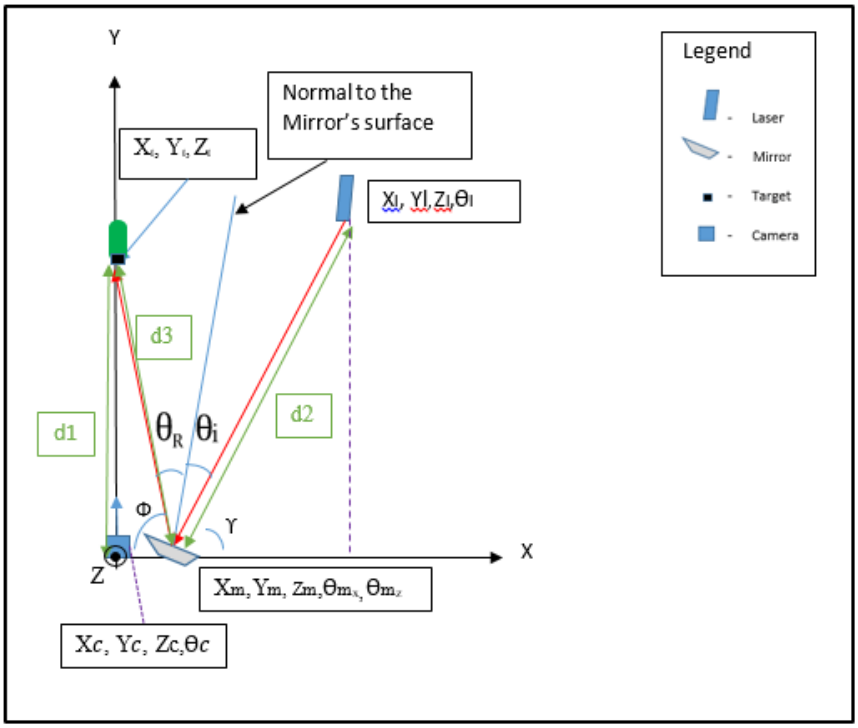


Figure 13: Coordinate System to Determine the Mirror's Angle of Rotation

Based on the design shown in Figure 13, the rotation angles of the mirror was computed to be

$$\theta_{mz} = 90^\circ + \frac{1}{2}cos^{-1}[\frac{2(x_l - x_m)^2 + y_l - y_l^2}{2(x_l - x_m)\sqrt{(x_l - x_m)^2 + y_l}}] - \frac{1}{2}[tan^{-1}(\frac{y_p}{x_m - x_p})] \tag{3}$$

$$\theta_{mx} = 90^\circ + \frac{1}{2}cos^{-1}[\frac{2(z_l - z_m)^2 + y_l - y_l^2}{2(z_l - z_m)\sqrt{(z_l - z_m)^2 + y_l}}] - \frac{1}{2}[tan^{-1}(\frac{y_p}{z_m - z_p})] \tag{4}$$

In the above equations, it is assumed that the camera designated for tracking the target will have the coordinates X_c, Y_c, Z_c, θ_c, and it will be fixed in position as it represents the origin of the coordinate system. The camera will be fixed at coordinates (0, 0, 0, 90°). The laser will be represented by the coordinates X_l, Y_l, Z_l,

θ_l. The target's location in the above equations are represented by (X_t, Y_t, and Z_t). As can be seen in Figure 13, the mirror's location in the coordinate system can be given by the coordinates Xm, Y_m, Z_m, θ_{mz}, and θ_{mx}. The location of the mirror will be constant, meaning that X_m, Y_m, and Z_m will remain constant, however, the angles of rotation θ_{mz} and θ_{mx}, will vary. θ_{mz} is the angle which represent the mirror's rotation angle around the z-axis, while θ_{mx} represents the MEMs mirror's rotation angle around the x-axis. When the mirror is rotated, the angle of incidence will vary. The angle of incidence which represents the angle between the incident laser beam and the normal to the mirror surface, θ_i, is shown in Figure 13. Also shown in Figure 13 is the reflected angle, θ_r, which represents the angle between the reflected laser beam and the normal of the mirror's surface. When the mirror rotates, this will cause the incident angle to vary which in turn varies the reflected angle (θ_r), since both the incident and reflected angles will always be equal in value, according to the Law of Reflection [7].

In order to implement the above design, certain components need to be kept at constant location while others are at a constant location but are free to rotate. All of the listed components above remain constant except for the target and the mirror which vary in position as well as rotation angles.

3.1.2 Implementation of the Experimental Setup

Aside from the design shown above in Figure 13, there are other factors to consider when implementing the physical setup of the experiment. For example, the first time the experimental setup was implemented, the components were placed in such a way where the target was in a position which maximized the amount of power received from the laser. Measurements of the power delivered by the laser diode were obtained from various locations. Initially the location of all the components relied heavily on the highest power recorded by the optical power meter. Since the power measurements which were obtained were affected by the light noise in the room, noise measurements were obtained and found to be **13.31** μW. Before the integration of the system, a limit of 5 V was set to the applied source, and a current of **0.2 A** was applied to the laser diode. The voltage delivered by the source to the laser diode as shown on the DC power supply showed a value of **1.867 Volts**. However, when measuring the actual applied voltage using a voltmeter, it was found that the applied voltage is **1.82 Volts**. This yields to a total applied electric power of **0.364 Watts** to the laser diode, considering the voltage applied using the voltmeter. However, after measuring the power of the laser beam directly with the power meter, with a 1 cm distance in between, the power was measured to be **90 mW**. This reflects an efficiency of approximately 25%. It is important to note the power received directly by the power meter before the laser beam is reflected off of the MEMs mirror to be able to account for the optical power lost during the transmission of the beam from the laser diode to the actual target. After placing the power meter in the target's location, 10 power measurements were obtained to give an average delivered power of **65.05 mW**, including noise power. Based on obtaining the highest optical power, the equipment was placed in a way that d1 from Figure 13 was measured to be 9.5cm, while d3 was measured to be 10.5 cm, and d2 was measured to be 8.5 cm. In addition, the distance from the centre of the camera to the centre of the MEMs mirror was measured to be 4.8 cm, making X_m to be a value of 4.8 cm. All the measurements obtained are tabulated below in Table 3.

Table 3: Physical Implementation Results

Optical Power Measured:		
	Measured Light Noise Power (μW)	**Measured Reflected Light Power (mW)**
1	14.3	65.5
2	14.8	68.3
3	16.3	67.4

4	12.7	67.8
5	10.5	67.7
6	12.6	64.6
7	13.5	61.7
8	14.1	62.7
9	11.8	61.9
10	12.5	62.9
Average	**13.31**	**65.05**
Source Specifications:		
Applied Current (DC Power Supply)		0.2 A
Applied Voltage (DC Power supply)		1.867 V
Measured Applied Voltage		1.82 V
Computed Electrical Input Power		(0.2 A)×(1.82V) = 0.364 Watts
Measured Output Optical Power (Laser beam)		90 mW
Efficiency of Laser Diode		(90 mW)/(0.364 W)× 100% ≈ 25%
Measured Distance (Pertaining to Figure 13)		
d_1	9.5 cm	
d_2	8 cm	
d_3	11 cm	
X_m	5.5 cm	

3.2 Target Integration (Noor)

3.2.1 Controlling a Target

The target to be used was initially supposed to be a custom made PV chip. However, as the project progressed, we were provided with photodiodes and quadrant detectors to use as our final targets. After conducting some tests, the quadrant detector was not selected as the final target. This is due to the fact that the quadrant detector first of all did not fit into the base which was provided to hold the target, as the circumference of the quadrant detector was larger than that of the target holder. In addition, while trying to track the quadrant detector, there was very large reflectivity of light due to the front glass, which led to inaccurate results. Due to these issues which were caused by the use of the quadrant detector, it was decided that the photodiode is the optimal target, due to its small size as well as non-reflective surface. To attach the photodiode onto the target holder, a round piece of cardboard was glued onto the face of the target holder. Two holes were punctured through the cardboard piece to accommodate the two legs of the photodiode, which were soldered onto two wires, to easily display the output of the target.

The base which holds the target in place was 3D printed and assembled at McGill University, provided to the group by iBIONICS. The target base consisted of three servo motors which were connected to a circuit that was controlled by a joystick as seen in Figure 14 below and the schematic Figure 10. Two of the servo motors were responsible for rotating the target along the horizontal and vertical axis, while the third servo motor moved to displace a cover piece which was to act as an eyelid on the target. Due to the fact that the project did not consist of an eyelid in the design, the eyelid controlling servo motor was disconnected and used to replace the faulty servo motor that was attached to the rotating mirror stand. When trying to test the tracking, the target had to be manually rotated using the joystick provided with the connected circuit, however the speed and the range of rotation could not be well controlled using the joystick. In addition, the movement of the target was very jittery during rotations.

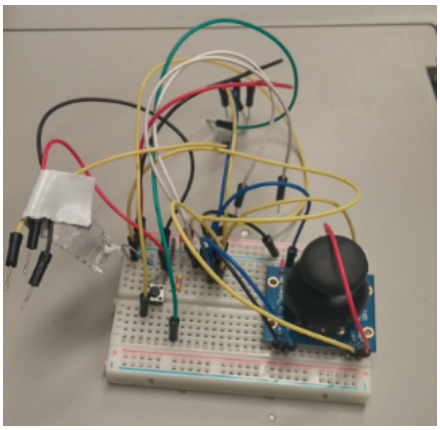


Figure 14: Joystick Controlled Circuit

To overcome the challenges posed by the integration of the target base, a new design approach was considered. A new circuit was designed and built to control the servo motors by attaching them to an Arduino Uno microcontroller and programmed to automatically control the motors. The code which was written to control the target allowed the control of increments that the target rotated at as well as how long each position that the target moved to was held for. To accommodate the servo motors which controlled the rotation of the mirrors and the fact that they only allow for tracking of the target when it is found on orbits which are in 10° increments from the origin, the target was chosen to rotate at increments of 10° as well. In addition to the increments of rotation, a delay of 3 seconds was introduced after every position that the target moved to, this was necessary to allow for processing delay. When the target is tracked, the coordinates need to be sent to the microcontroller which controls the servo motors that move the mirror. This demonstrates why there is a delay between moving the target and moving the mirror to reflect the laser beam. In order to improve the accuracy of the tracking, it is necessary to introduce a delay which holds the target in place for 3 seconds at each location it moves to, to allow time for the mirror to correctly reflect the laser beam onto the target's new location. The movement of the target was still jittery after automatically controlling it and completely changing the circuit it was attached to. Resistors were added to reduce the jittery movement, but no improvement was seen. After replacing all the wires one by one, it was found that the wire which connected the power supply to one of the motors was not very stable and had to be replaced. This led to a target moving consistently without jittering, by increments of 10° with a delay of 3 seconds at each new position. The target, target base, and the new circuit it was attached to can be seen below in Figure 15 and new schematic is in Figure 12 , while the code used to control the rotation can be found in Appendix A.1.

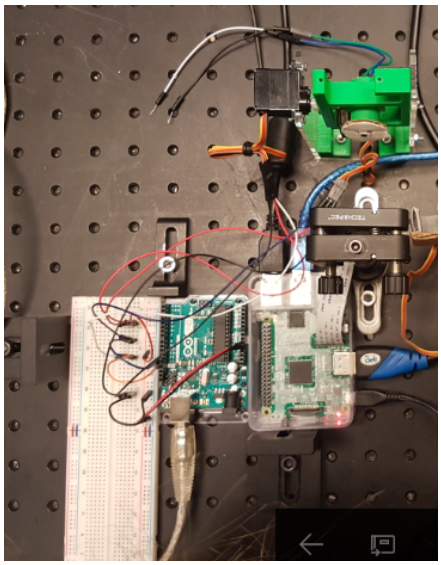


Figure 15: Automatically Controlled Circuit Circuit and Target

3.2.2 Rotation of Target: A Limitation on Tracking

After the target has been integrated into the system and the tracking has been in place and is correctly following the target, tests were carried out to observe the limitations of the tracking system. It was found that although the coordinates of the target were correctly tracked, meaning the tracking portion was accurate, the laser beam was not always reflected accurately onto the target. This is due to the fact that there are certain directions and degrees of rotation on the target that will make it impossible for the reflected beam to reach the target. Rotating the target to certain locations to a maximum of $\pm 30^\circ$ to measure the limitations of the system, the voltage measured at the output was recorded for each incremental rotation. The measurements obtained were recorded and can be seen in the diagrams drawn below in Figure 16 and Figure 17. As seen in Figure 16 and Figure 17 respectively, measurements of the output were obtained from -30° to $+30^\circ$ in increments of 10°, along both the horizontal as well as the vertical axes. As can be seen in both figures, when the target is at either extreme on both axes, so at $\pm 30^\circ$ on both the horizontal as well as the vertical axes, the measured output voltage is very low. This can be seen as the threshold voltage for a signal has been set to be 5 volts, as the target's output is at 5.71 V in the centre of both axes. The light noise in the room yields an output of 210 mV from the target as was shown on the oscilloscope. Therefore, when the output on the extremes of either figure is in the low volts range and is not above 5 volts, one can see that the laser beam is not being correctly reflected onto the target, since the yielded output was not acceptable as it was not above the threshold.

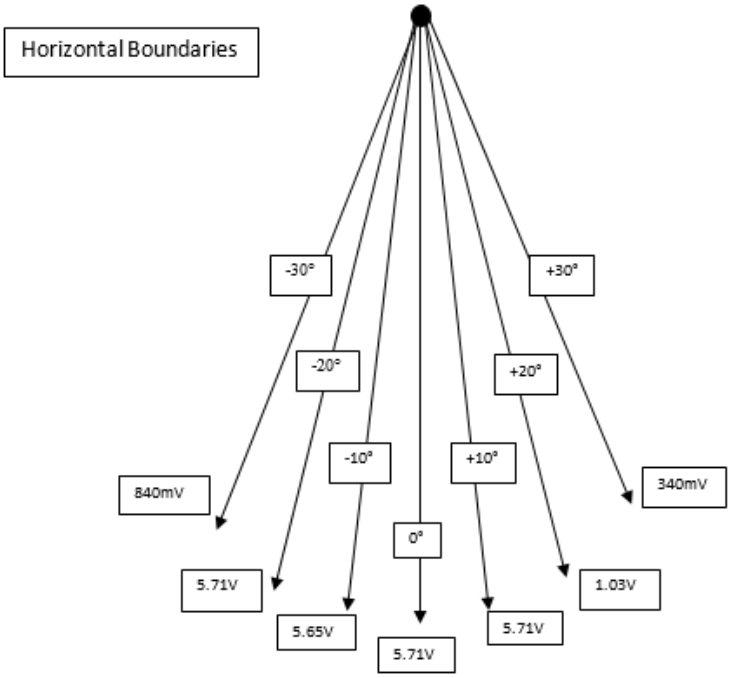


Figure 16: Measurements of Output Voltage as Target Rotates Horizontally

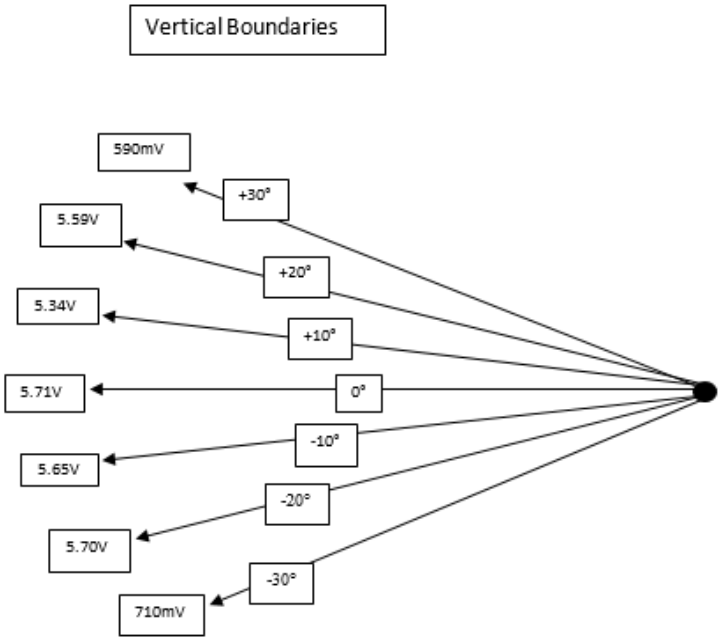


Figure 17: Measurements of Output Voltage as Target Rotates Vertically

In addition to the measurements which were conducted to find the blind spots of the tracking system, which are the spots that could not be reached by the reflected laser beam, measurements were conducted to

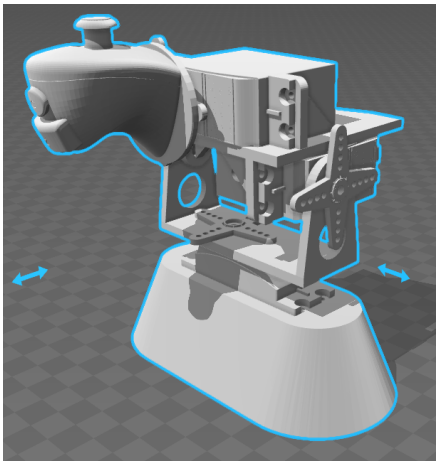
estimate the number of times that the target was not reached by the reflected laser beam during integration. These measurements were obtained by randomly moving the target using the Arduino code, and trying to reflect the laser beam onto the target's new position. The threshold was again set to 5 volts, and anything below the set threshold was not seen as an appropriate output. Many measurements were taken to highlight the accuracy of tracking the target and correctly reflecting the laser beam onto the tracked target. Some of these measured randomly obtained outputs are outlined in Table 4 below, which are a sample of the data collected to give an approximate accuracy of tracking the target and correctly reflecting the laser beam onto the target's new position to being approximately 67%. As seen in Table 4, 20 out of the 30 measured outputs were above a threshold of 5 V, while the other 10 output measurements were below the required threshold. Therefore, it can be approximated that the accuracy of the system tracking a target and correctly reflecting the laser beam onto the target's new position is approximately 67%.

Table 4: Measurements for Tracking and Reflection Accuracy

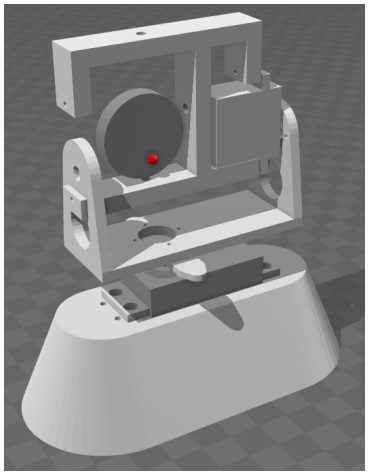
Trial Number	Measured Output (V)	Trial Number	Measured Output (V)
Trial 1	5.71	Trial 16	1.46
Trial 2	5.46	Trial 17	0.96
Trial 3	5.59	Trial 18	5.59
Trial 4	0.71	Trial 19	5.65
Trial 5	0.9	Trial 20	5.71
Trial 6	0.84	Trial 21	5.46
Trial 7	5.65	Trial 22	5.65
Trial 8	5.71	Trial 23	0.84
Trial 9	0.55	Trial 24	5.65
Trial 10	5.71	Trial 25	2.78
Trial 11	5.71	Trial 26	5.65
Trial 12	1.40	Trial 27	1.03
Trial 13	5.65	Trial 28	5.71
Trial 14	5.71	Trial 29	5.65
Trial 15	5.65	Trial 30	5.71

3.3 Replacement of MEMs Mirror (Zac)

The 3D model of the 2 axis rotational stand is modified with s1a2d3's model [16]. His original model, shown in Figure 18a has 3 axis of rotation, so we extract only the parts with y and x axis of rotation. The model is designed to use motor of size like Futaba S148.



(a) s1a2d3 Parallax (Futaba) servo motor 3-axis mount [16]



(b) Current Deign

Figure 18: 3D Model of Optical Mirror's Stand

Since we required more space for the optical mirror to fit in, we replace the center piece as shown is Figure 18b and the resulting center of rotation is indicated with red dot. The resulting stand is shown in Figure 19.

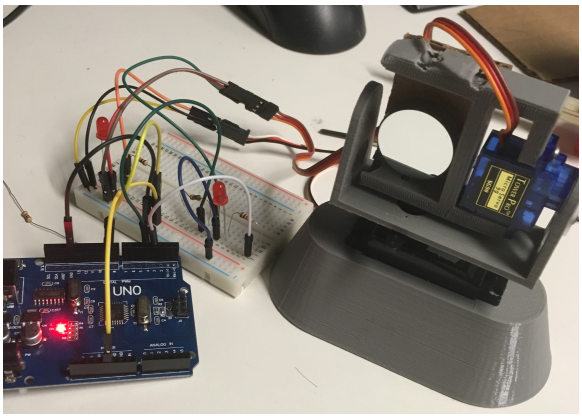


Figure 19: Printed Design with Mounted Optical Mirror

3.4 Tacking (Anas)

3.4.1 Background

Tracking the target is a critical part of this project; it tracks the movement of the object to ensure correct and flawless transmission of data. This means keeping track of the object's position and orientation at any time. One way to achieve that is by having the target report its position to other components through back connections or feedback. However, in this system in particular, targets do not have feedback and therefore unable to communicate their location or position to other components in the system.

As it is commonly known, feedback is essential to every system as without it the system's components are blind to each other and cannot function as a unit. Having feedback in systems allows different components to communicate with each other and report information about reactions from products to components. Which ensures that control component can take corrective measures to rectify error in the final result.

Since the target in this project is lacking in feedback, it was not possible to use conventional position feedback methods such as 3D accelerometers or gyroscopes. Which gave birth to the idea of detecting and tracking using a camera pointed directly in front of the target through computer vision algorithms.

3.4.2 Tracking and detection through computer vision algorithms [21]

In order to determine the most suitable method to achieve this purpose, many different tracking methods were explored and tested. Each method has advantages but also introduced some complications. Some were computationally costly but produced reliable results while others required less processing time but gave poor results.

One of the methods explored was tracking using HSV -Hue, Saturation, and Value- also known as HSL. It simply tracks objects of specific color on a background of a distinct color. It requires users to calibrate the color of the target using the minimum and maximum values of HSV. Then applying the Morphology Transformations and Dilation algorithms to obtain a consistent shape of the target.

This method performed very well and consumed very little computational capacity. However, it forces many constraints on the object as well as the background. It also required constantly making adjustments to the parameters when there is a change in the background or lighting.

With the previous method proving to be fast but impractical, another method was explored. Machine learning and neural networks, this method entails training the machine with hundreds of samples both positive (of the target) and negative (of the background and objects that look like the target but we wish not to detect). Machine learning proved to give the best results however it is computationally heavy. It demands most of the Raspberry Pi's capability, which cannot be afforded. Which led us to seek a third method that satisfies the best trade-off between computational cost and quality, which led us to use Haar cascades classification and tracking.

3.4.3 Detection and Tracking Via Haar Cascades Classifiers and Sliding Window Algorithm [20]

With Haar Cascades, users face the choice of working with a pre-implemented cascade (Also known as a classifier) or designing their own custom one. This approach was most suitable for this system as it allows multiple degrees of freedom in terms of detection and tracking. Since the method relies on capturing video streams and applying the cascade, it gives users the freedom of choosing the video resolution and quality in addition to the number of samples obtained per second.

After thorough testing this method proved to be very effective. It was chosen due to its very high response adaptability as well its reliability and good results. Sometimes some false detectives were triggered but that can easily be solved. False detections can be eliminated by designing a custom Haar Cascade specific to the target. But since the project is still in its initial phases, and the shape of the target is not yet finalized, creating a custom Haar cascade is still premature. The figure below shows how well the pupil Haar cascade preforms when tracking an actual pupil Figure 20(left image), and how it is used in this project to track the target showing also a single case of false detection Figure 20(right image).

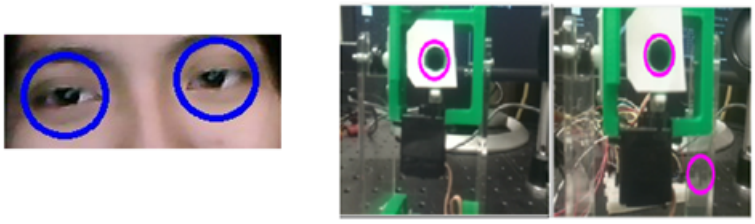


Figure 20: Pupil Haar Cascade detecting an actual pupil vs Haar Cascade detecting a Target

3.4.4 Active Feedback Tracking

Now that the target's location can be accurately estimated at any moment in time fairly quickly. It is possible to actively track the target to ensure the laser beam always follows. To do so, a servo-controlled mirror was designed. Such mirror, is controlled by two servo motors mounted on horizontal and vertical axes of the mirror. Which allows it to rotate in two axes which in turns achieves a two-dimensional tracking on the surface of the target which is suitable in this case as the distance between camera and the target never changes.

The set of images below show the vertical and horizontal range that the servo motors are able to cover. It is worth noting that the blue point in the middle does not represent the center. However, it is shown as a reference to show the transition of the laser (red dot). The top row of the images show the horizontal range of tracking (Left to right). While the below row shows the vertical tracking range (bottom to top). Updating each axis separately and using a combination of the two, it is possible to track in a 2D fashion. The figure shows that the servo mirror covers a wide range which allows us to completely follow the target and keeping track of it's movements.

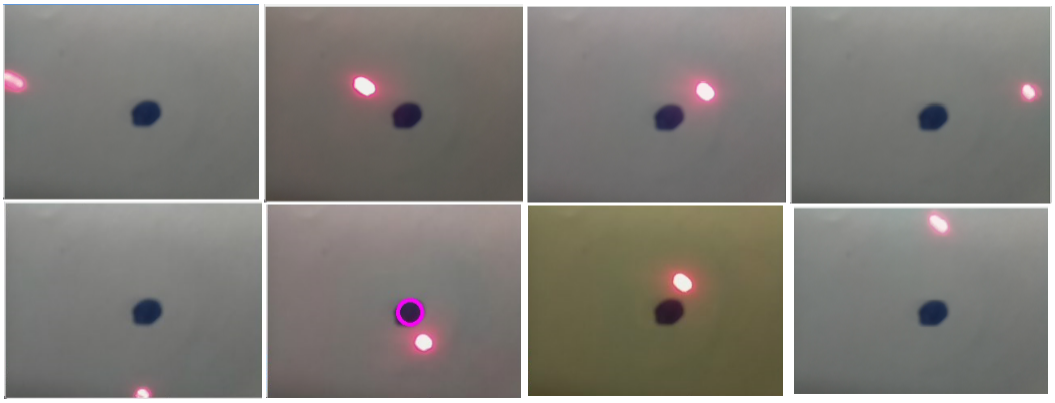


Figure 21: Servos' Range

3.4.5 Tracking algorithms

To fully understand the tracking algorithm, readers must be familiar with the setup of the system. Mainly, the active feedback tracking system contains the servo-controlled mirror unit and the target detection and image processing unit. A single Raspberry Pi responsible for controlling the servos as well as image processing and target detection. A flow diagram of the algorithm is shown below.

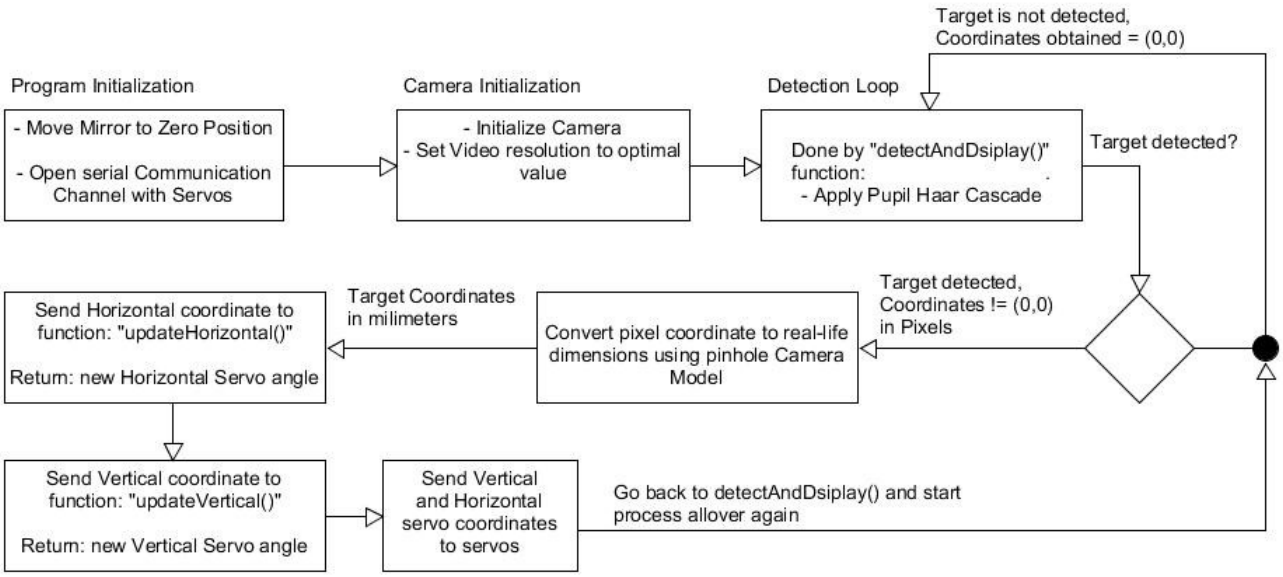


Figure 22: Tracking Algorithms

The algorithm starts by setting the system to zero initial conditions, which means that the coordinates are all set to zero. More precisely, the target is going to be initialized to the zero coordinates while the mirror is pointing directly on it. This is a necessary step in this implementation as there are many variables in the system and coordinates are constantly changing. Updating the vertical and horizontal coordinates relies on the initial conditions.
After the system is initialized, the detecting function captures a frame from a video stream and applies the Haar cascade to it. Obtaining the coordinates of the center of the target. The targets are then mapped using the pinhole camera model and using the following two equations the angles of the servo are calculated.

$$\theta_{mz} = 90^\circ + \frac{1}{2}cos^{-1}[\frac{2(x_l - x_m)^2 + y_l - y_l^2}{2(x_l - x_m)\sqrt{(x_l - x_m)^2 + y_l}}] - \frac{1}{2}[tan^{-1}(\frac{y_p}{x_m - x_p})] \tag{5}$$

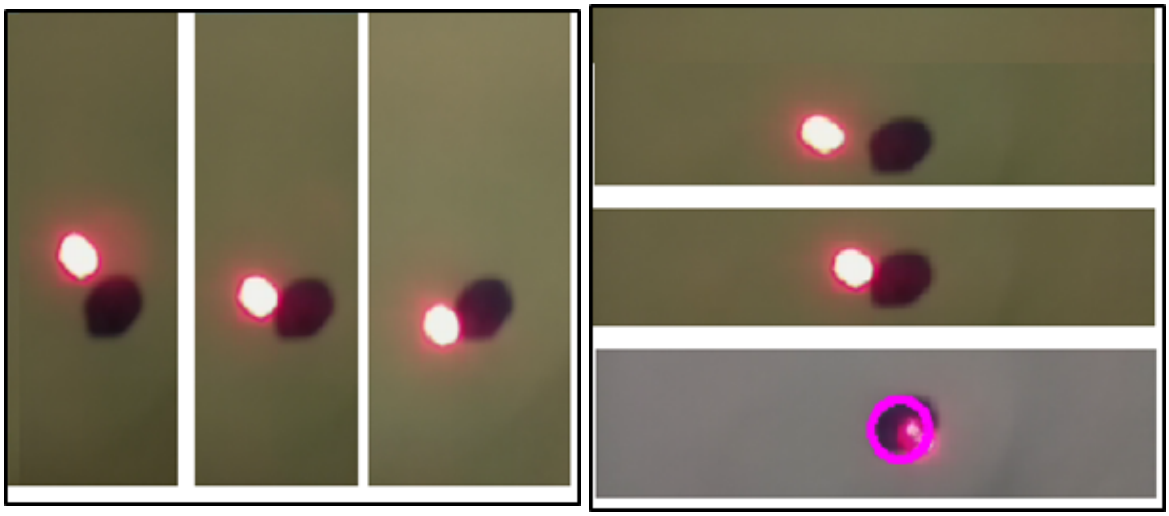
$$\theta_{mx} = 90^\circ + \frac{1}{2}cos^{-1}[\frac{2(z_l - z_m)^2 + y_l - y_l^2}{2(z_l - z_m)\sqrt{(z_l - z_m)^2 + y_l}}] - \frac{1}{2}[tan^{-1}(\frac{y_p}{z_m - z_p})] \tag{6}$$

After the horizontal and vertical angles are calculated, the angles are sent to the servos and the loop is restarted all over again. Regardless of some delays that were introduced in the code for consistency purposes, the system is responding quickly. A rough test of the speed versus the resolution was done in the midterm report.

3.4.6 Implementation Problems

3.4.6.1 Inaccuracy of Servos

Due to the nature of the servos, an error is always involved in the transitions. That was a predicted issue as servos are usually avoided for such precise and fine applications. Servos were able to perform well and redirect the beam when the change in the target is significant. However, when the change is fine the servos were not able to move in such small increments. The following images show the smallest step size the servos could move in both vertical and horizontal directions. In the left figure, the middle image shows the original position. To the left is an image that shows a transition of 1° degree up and the right one shows a transition of 1° down. The right figure shows the transition on the horizontal axis. Similarly, the middle image shows the original position. While the left one shows a left transition and the right one shows a transition of 1° to the right.



(a) Vertical Motion Step Size

(b) Horizontal Motion Step Size

Figure 23: Motion Step Size

As illustrated by Figure 23, the servos were not capable of keeping track of changes smaller than 10° on each axis. This obviously affects the accuracy of the data transmission as it leads to loss of packets.

To significantly reduce the effects of this issue, better servos can be used. Brushless servo motors, deliver more accurate results and even faster response which also improves response time. The full metal gears and dual ball bearings allow for greater accuracy which the standard low-end servos lack. However, a slight error will always be associated with servos thats why a better solution would be the MEMs mirror which will be discussed in later parts of the report in Section 5.1.1.

3.4.6.2 Blind Spots

A Blind spot occurs when the target reaches an orientation where it is impossible for the laser to hit the target regardless of the accuracy of the tracking. In this case, the active feedback camera is able to pick up the coordinates of the target, and the angles of the servos are calculated. However, the orientation of the target is simply unreachable as it is looking *away* from the mirror. This problem only occurs on one side, more precisely, the left side of the viewer (which is the right side of target). As when the target is oriented to the right hand-side it is facing the laser and therefore this issue ceases to exist. The figure below shows a few orientations where the target encounters blind spots. A possible solution will be discussed in later parts of the report in Section 5.1.3.

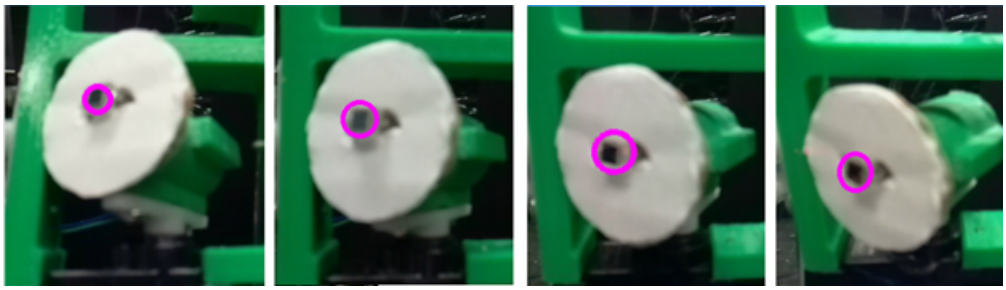


Figure 24: Blind Spots

3.4.6.3 Instability of Mirror Stand

One of the reasons that affects the repeatability of the servos and hence the accuracy of the tracking, is the instability of the servo-mirror stand. Parts of the stand were 3D printed out of plastic material then assembled. The structure proved to be somewhat *wobbly* when the servos are in motion, which affected the consistency of the results. Instability of the structure might not contribute when the transitions are minimal. However, when the servo-mirror moves a large angle (45° for example), the servos' rapid movement and the sudden stop causes structure to shift. Namely, the system would not reach the same position if it had moved to the same destination from a smaller transition. Another problem that was encountered was the slippage of one of the servos, due to the nature of the mirror stand said servo could not be screwed tight to the structure.

To avoid this issue, an aluminium stand can be specially designed to secure the two servos while allowing the two degrees of freedom that is needed for the design. The robust structure will eliminate any slippage and will give more reliable results, which will significantly improve the system.

3.4.6.4 False Detections and Autofocus

As seen by the previous screenshots, an occasional false detection is always triggered which can be regarded to how the Haar cascade function. In this project, a Haar Cascade of a pupil was used which is essentially trained to detect irises in pupils. More precisely, a dark region surrounded by a white surrounding or background (colored irises are converted to greyscale which corresponds maps them to dark colors). Since a photodiode is being tracked and not a pupil, and due to the low quality of video stream which was reduced to improve response time, it is natural to observe an occasional false detection. Auto focus plays an important role in triggering false positives as well.

Unfortunately, since a Ribbon Raspi Camera is being used for the tracking, one is faced with two choices; To either turn off autofocus which leads to worse results, or changing the focus manually which requires physically altering the lens of the Camera. As for the Haar cascade, the issue can be reduced and mostly eliminated by using a custom Haar cascade. Training a Haar cascade to only detect the photodiode -Or the final target- while reducing the complexity of the background by cropping unnecessary portions of the video stream not only improves the response time but eliminates this issue almost entirely.

3.5 Camera Calibration (Tony)

We can correct the camera distortion by using calibration and some remapping. Furthermore, with calibration, you can also determine the relation between the camera's natural units (pixels) and the real world units (for example, millimetres or inches). Camera calibration is an important step towards getting a highly accurate representation of the real world in the captured images. Camera calibration can actually refer to two things: geometric calibration and color calibration. What we need at this point is geometric calibration.

2D plane (chessboard pattern) based calibration

We can correct the camera distortion by using calibration and some remapping. Furthermore, with calibration, you can also determine the relation between the camera's natural units (pixels) and the real world units (for example, millimetres or inches). Camera calibration is an important step towards getting a highly accurate representation of the real world in the captured images. Camera calibration can actually refer to two things: geometric calibration and color calibration. What we need at this point is geometric calibration.

The Camera Calibration will provide the approximation of intrinsic matrix K. It is done through MATLAB camera calibration tool.

Camera parameters include intrinsics, extrinsics, and distortion coefficients. To estimate the camera pa-

rameters, we need to have 3-D world points and their corresponding 2-D image points. We can get these correspondences using multiple images of a calibration pattern, such as a checkerboard. Using the correspondences, we will solve for the camera parameters. After calibration for the camera, to evaluate the accuracy of the estimated parameters, we need to do the following:

- Plot the relative locations of the camera and the calibration pattern
- Calculate the re-projection errors.
- Calculate the parameter estimation errors.
- Use the Camera Calibrator to perform camera calibration and evaluate the accuracy of the estimated parameters.

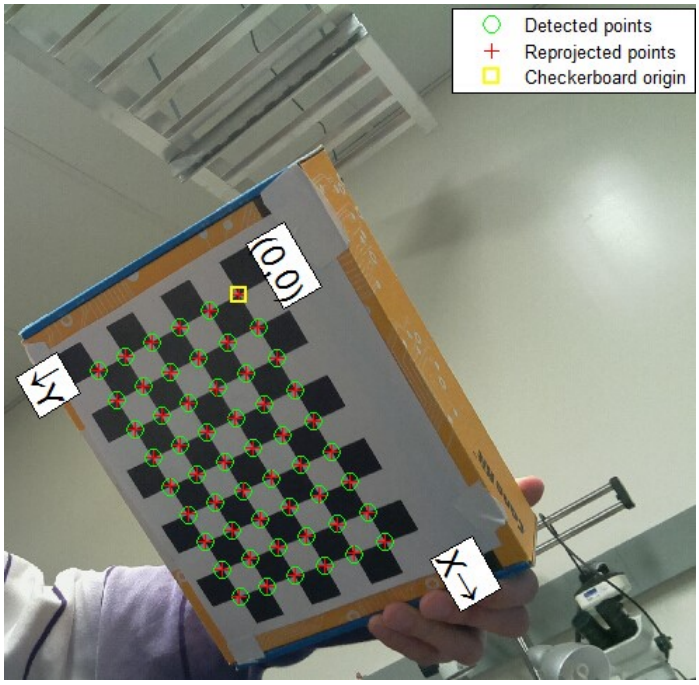


Figure 25: Detection and reprojection on the chessboard pattern

Seventeen chessboard-pattern pictures are taken from the Raspberry Pi Camera module with different orientation. The pictures are imported to MATLAB camera calibration tool box (The MATLAB code for camera calibration can be found in Appendix A.5. The observed position and the re-project position of the chessboard cross points are close on the picture shown above. The detailed quantitative analysis will be shown in the next section.

MAT LAB code for calibration is shown as following:

```
% Detect checkerboards in images
[imagePoints, boardSize, imagesUsed] = detectCheckerboardPoints(imageFileNames);
imageFileNames = imageFileNames(imagesUsed);

% Generate world coordinates of the corners of the squares
squareSize = 1.964000e+01;  % in units of 'mm'
worldPoints = generateCheckerboardPoints(boardSize, squareSize);

% Calibrate the camera
cameraParams = estimateCameraParameters(imagePoints, worldPoints, ...
    'EstimateSkew', false, 'EstimateTangentialDistortion', false, ...
    'NumRadialDistortionCoefficients', 2, 'WorldUnits', 'mm');
```

```
% View reprojection errors
h1=figure; showReprojectionErrors(cameraParams, 'BarGraph');

% Visualize pattern locations
h2=figure; showExtrinsics(cameraParams, 'CameraCentric');

% For example, you can use the calibration data to remove effects of lens distortion.
originalImage = imread(imageFileNames{1});
undistortedImage = undistortImage(originalImage, cameraParams);
```

The program will provide the approximation of the internal and external parameters of the camera that will be used for re-projection of the 3-D coordinate.

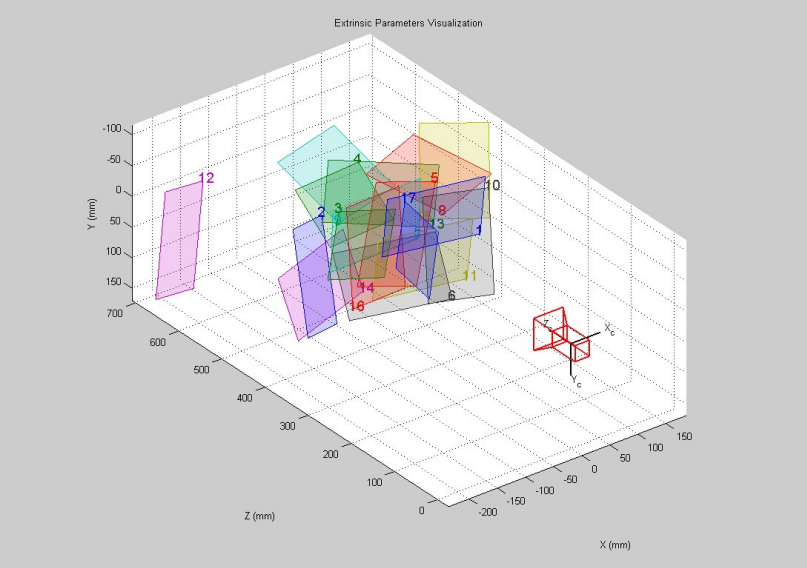


Figure 26: Camera Re-projection of the positions of the chessboard

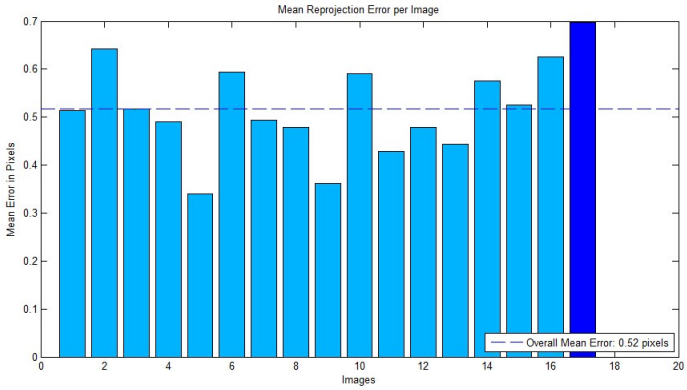


Figure 27: Mean Reprojection Error per Image

The mean error on the re-projection is 0.52 pixels per image which is an acceptable result.

3.5.1 Discussion

The target, which is a photovoltaic receiver, moves on a surface of a sphere within a range of ± 10° degrees. Using one camera to track the target will introduce an error due to the non-linear motion of the receiver. At this point, we approximate the movement of the target is on the tangential plane of the sphere because it only rotates 10° degree off its axis. If the error is found too large to transfer the data properly from laser, two alternative ways could be used to mitigate this issue.

The first way is that to have two cameras to capture the position of the target. Therefore, a more precise measurement in 3-D position can be achieved by employ two 2-D image from different view. The second approach is to implement a feedback system. The controller will adjust the position of mirror based on the error between the data from measurement of feedback system and the camera measured one.

3.6 Pixelization (Zac)

The main function of this portion of the project is to translate the frame capture by the camera into binary stream for transmission. The processed image will be down resolution and convert into grayscale. Typically when a camera is taking a picture or video, it is already pixelizing the real-life image into pixels. In our application, since we are required to transmit low resolution frame, we will be developing solution towards transmitting one signal pixel in grayscale. So three major function the transmitter has to have are, capable of taking a stream of frame, down resolute the frame and convert RGB-coloured frame in to grayscale frame.

3.6.1 Magick++

Magick++ is the object-oriented C++ API that is an image processing library. It is a light-weighted and has friendly towards beginner in image processing.

3.6.1.1 Testing Magick++

Since Magick++ is very easy to use, user can simply open and read the desired image is to be processed then simply do the necessary image process. Below is an example code that reads an image called “tmp.jpg” with 128x128 resolution, then it resize and convert into 64x64 resolution grayscale called “grey.jpg”. Then the last line of code will inquire the resulting pixels information for later use.

```
InitializeMagick(*argv);
Image image;

image.read("tmp.jpg");
image.resize("64x64");
image.type( GrayscaleType );
image.write("grey.jpg");
int w=image.columns();
int h=image.rows();

Magick::Quantum* pixels = image.getPixels(0,0,w,h);
```

Figure 28: Example Code for using Magick++

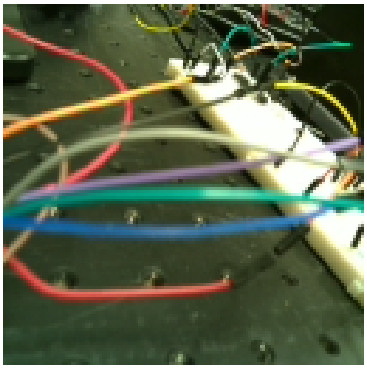
Since Magick++ can only process one picture at a time instead of a stream of frame, we need an alternative solution to can take pictures fast enough to be a stream.

3.6.1.2 RaspiFastCamD

RaspiFastCamD is Linux Bash Scripts that are written by Userland [10] that are capable to take 10 pictures per-second. From the "start_camd.sh" user can modify the output picture's title and also the resolution of the picture simply by change the code line **output_file=${1-/home/pi/Desktop/send/tmp.jpg}** and the line **./raspifastcamd -w 128 -h 128 -o $output_file &** . The Bash code can be found in Appendix A.4.2. The way to call the Bash Scripts directly from C++ is using **system("./xxxx.sh")**.

3.6.1.3 Advantages and Disadvantages

Combining Magick++ and RaspiFastCamD, we are able to capture pictures from the webcam, process and read into desired binary data. The advantages of using such a combination is that this method is very light-weighted; hence are worth to further develop a more reliable program. The downside of such a method is that RaspiFastCamD are only fast when we are saving all the pictures in different name by doing **output_file=${1-/home/pi/temp_%d.jpg }**, this way the picture title will increment every time it runs. But in our application we are best to have a single output data frame rather than multiple of them. As we give a constant picture title, it will replace the old picture with the newly captured picture, but it takes approximately 1 second time for the system to save and replace the picture, otherwise Magick++ cannot read the picture. Hence the combination of Magick++ and RaspiFastCamD are not suitable in our application.



(a) Original Picture Capture by RaspiFastCamd

(b) Processed Image by Magick++

Figure 29: Result of Magick++ with RaspiFastCamd, 128x128 colored picture into 64x64 grayscale

3.6.2 OpenCV

OpenCV is a very well-known image processing which is cross-platform supportive, such as python, C and C++. The type of library of OpenCV we will be using in this project will be C and C++. C and C++ library behaves and function similar but have slight difference. One of the most noticeable difference is that in C++ it reads frame into Mat form, which is a matrix form that consist of RGB parameters in each matrix element, but in C the frame is read into an array with pointers that specify RGB. So way to access information of a frame is different. Hence the way of displaying windows are also slightly different. Even though C++ library is more popular, but in our application, the use of simply array is more suitable; hence we are using C library of OpenCV.

3.6.2.1 Testing OpenCV

Before outputting the data stream of pixels, we want to see the effect of pixelization and grayscale conversion. By request from iBionics, we are pixelizing frames into circle pixels. The pictures below shows pixelized grayscale

frame with various resolution along with its binary data stream. The binary data stream will be further discuss in Section 3.7.

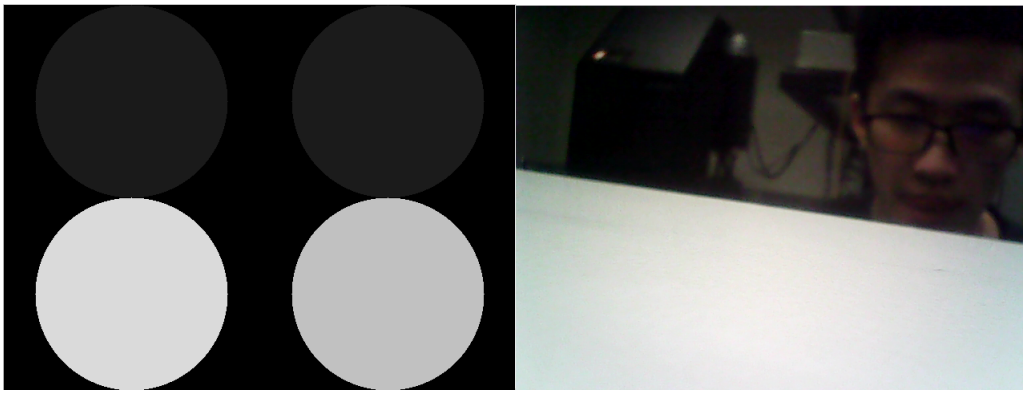


Figure 30: Original Frame and Processed Frame, 2x2 Resolution

Figure 31: Original Frame and Processed Frame, 10x10 Resolution

The algorithm is simply read the image stream and save the data into an array. Then we divide the whole frame into "pixelblocks". For example if we desire to have a 2x2 resolution, we divide the width and length of the frame into 2, and divide with 10 if we want 100 pixels. Then we calculate the brightness value **x** in grayscale by averaging the Red, Blue and Green parameters in each "pixelblocks" then re-project with circles with Red, Blue and Green parameters all equal to **x**, since in grayscale the Red, Blue and Green values are the same. Then loop for every pixelblock.

3.6.2.2 Advantages and Disadvantages

As the previous section mentioned, OpenCV is a very powerful and well-known image processing library. It provides reliable and robust solution for image processing, but the downside of using OpenCV is that, since it is very powerful, it uses a lot of processing resource from the Raspberry Pi. To capture a constant stream, process the frame into pixels and displaying both windows, the CPU of the Raspberry Pi can go up to 80%, which may result in overheating. So extra heatsink is added to prevent from overheating. Since the software is so resource consuming, the output windows can be relatively delayed. So to prevent from further lagging the output GPIO for transmission, we minimize the pixels into 1 pixel and disabled the Original Webcam Frame.

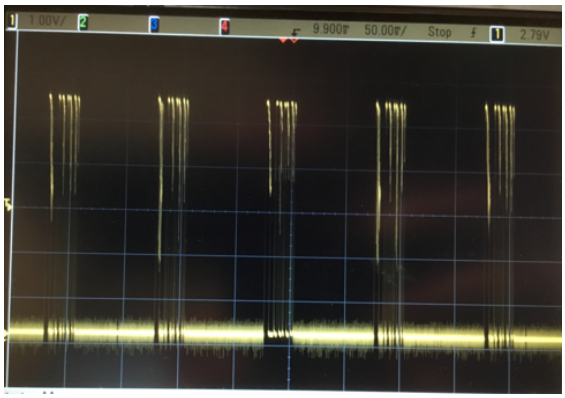


Figure 32: Scope of Transmission with 1 Pixel

Our Current system take approximately 70 ms to process a single pixel, which is equivalent to 280 ms for 4 pixels and 7 seconds for 100 pixels. The impact of this can be seen in Figure 32. A fast solution to this is to simply have faster processor, lower the resolution of the image stream as mentioned above.

3.7 Laser Transmission (Zac)

This section of the project will be dedicated to transmit and receive the binary data stream of a square wave, binary data of a byte counting from 0 to 255 and the resulting data stream from Section 3.6. The wireless link will be made by a laser and a photodiode.

3.7.1 External ADC for Raspberry Pi 3

During the beginning phase of developing the transmission system, we realize that there is no ADC on the Raspberry Pi, hence we have to make use of an external ADC from Microchip MCP3008 [8]. We will be using the Serial Peripheral Interface (SPI) on the Raspberry Pi to control the ADC [11]. The Raspberry Pi can establish master and slave devices using the SPI, Master In Slave Out (MISO), Master Out Slave In (MOSI) and Serial Clock (SCLK). The detailed control using SPI can be found in Reference [11] and the schematic for connecting Raspberry Pi and ADC can be found in Section 9. The following software are used to access and control the ADC.

```
int adcread(float vth){
    int a2dVal = 0;
    int a2dChannel = 0;
    unsigned char data[3];
    mcp3008Spi a2d("/dev/spidev0.0", SPI_MODE_0, 1000000, 8);
    data[0] = 1;  //  first byte transmitted (start bit)
    data[1] = 0b10000000 |( ((a2dChannel & 7) << 4)); // second byte transmitted -> (SGL/DIF = 1, D2=D1=D0=0)
    data[2] = 0; // third byte transmitted(don't care)
    a2d.spiWriteRead(data, sizeof(data) );

    a2dVal = 0;
    a2dVal = (data[1]<< 8) & 0b1100000000; //merge data[1] & data[2] to get result
    a2dVal |=  (data[2] & 0xff);

    if (a2dVal <= vth){
    tempbit=0;
    }
    else{
    tempbit=1;
    }
    //delayMicroseconds(delaytime);
    return tempbit;
}
```

Figure 33: Software to control ADC which output logic 1 or 0

The user can specify the sampling rate of the ADC by modifying the third parameters in **a2d** upto the maximum sampling rate of 200k samples per seconds, and can further control the number of bits per word by changing the **8** in **a2d** function and select the ADC channel by modify **data[1]**. The resulting ADC value is then store in **a2dVal**. So we can further take this code and develop of function that can return a logic 0 or 1 every time it is called to act as a digital signal reader.

3.7.2 Non-Encoded Transmission & Sampling

Initially, we simply sent out the data stream directly out of the GPIO into the laser and tried to extract the receiving signal by sampling at the same bit rate as the transmitter. The first thing that a receiver should do is try to distinguish packets correctly. In order to do this, we need to send a unique sequence for the receiver to pickup. Here we choose "11110000" as the unique sequence which will also be use as initialization sequence and select the transmission rate of 1kbps. Since we do not know when we will have a stable connect, we must send the initialize sequence for every packet of data. Hence the receiver should pickup the initialize sequence, synchronize and extract the byte of data. The algorithm for receiver to pick out the initialize sequence is make use of timer to time the time of every high bits follow by low bits. If the timing of high bits or low bits does not match 4 msec $\pm 10\%$, the receive classify it as not initialize sequence and loop back until the condition of initialize sequence is met. Then we sample the rest of the sequence to extract the data.

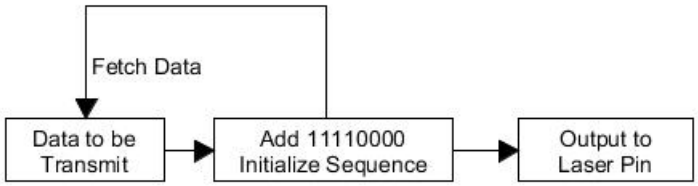


Figure 34: First version of transmitter algorithm

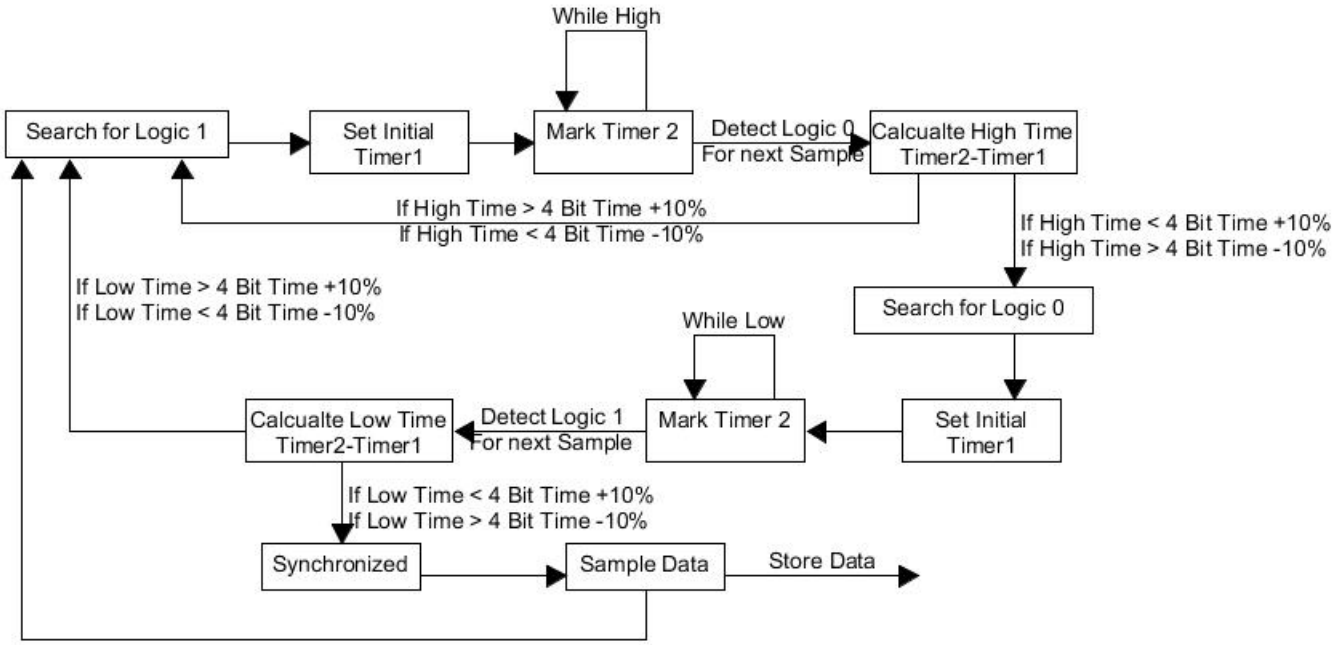


Figure 35: First version of receiver algorithm

3.7.2.1 Testing Result

As this method is straight forward and simple to implement, there are a lot of problems within. We were seeking to sampling the signal from the laser at the ideal sampling rate, which is the Bit rate, and should extract the data perfectly. But in fact the sampling rate is effected by the ADC delay. The measured ADC delay is approximately 50 ns, but the delay is not a constant time. As time goes on, the error between the ideal sampling rate and actual sampling rate cumulates and eventually result in extracting the incorrect data. The illustration of this can be seen below in Figure 36.

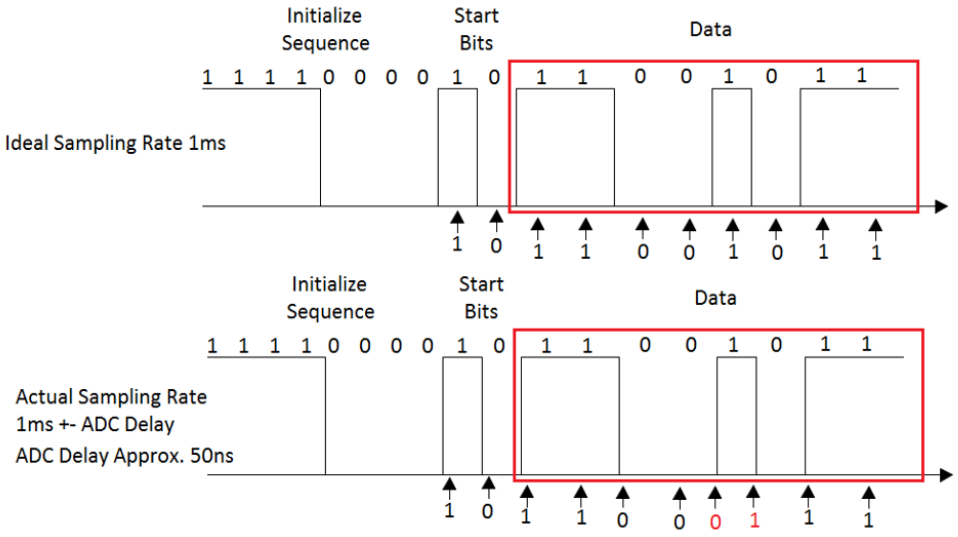


Figure 36: Illustration of Extracting Incorrect Data

We tested on average up to what bits the receiver will start extracting the wrong bits by sending a pure square wave and see when will the receiver starts picking up more then 1 same bits.

Table 5: Testing of Sampling

Test Number	1	2	3	4	5
Incorrect Bit	21	26	28	25	25
Average	25				

From Table 5 we can see that on average the receiver starts picking up wrong data starting on the 25^{th} bit, which is equivalent to the second packets of data. This is a relatively bad performance; hence we need to find another method that can solve the sampling problem.

Another problem with this method is that the data is not encoded, so the sequence of the data may be the same as the initialize sequence. This will cause the receiver to have a hard time to distinguish between the packets.

3.7.3 Manchester Encoding/Decoding & Edge Detection

In Manchester Coding, there are many different version. Here we are using the version that when signal is low it encoding into 01 and 10 when the signal is high. An illustration can be seen in Figure 37. By encoding the data before transmitting, the data can no longer have the same sequence as the initialize sequence; by using edge detection, we no longer need to concern about sampling.

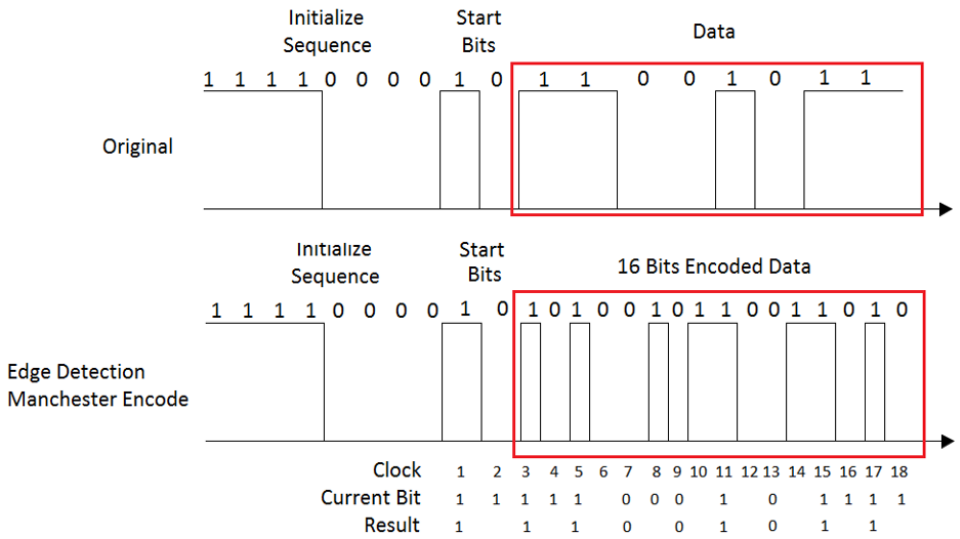


Figure 37: Illustration of Manchester Coding & Edge Detection

The usual method for receiving and decoding a Manchester Encode signal is to sample the encoded data and perform a logic XOR with the synchronize clock. The synchronize clock is done by the timing the initialize sequence and start the clock at Start Bits. If we use this method, then we will still have the same problem with ADC delays. Since our wireless link are not stable, even if we are able to sample at the ideal sampling rate, this method will perform very slow. The reason is that this method may be slow is that the receiver first needs to pick out the packets and sample all the data while mark down the clock at the same time, and **after** all

the data is received, perform XOR with the clock then extract every other bits. So the usual method is not suitable.

Hence the alternative algorithm for receiver is simply detect when there is a transition from low to high or high to low by using edge detection. By using this method, we are not concern about when it is high or low that is transmitted, but we want to know how much time is between the transition [12]. When there are 2 consecutive bits detected, the current bit is flipped. By using this method we can extract the data bits by saving the current bit for every other clock as the data is being detected. The result of using the combination of Manchester encoding and Edge detection can be seen in Figure 37.

3.7.4 Transmitter Algorithm

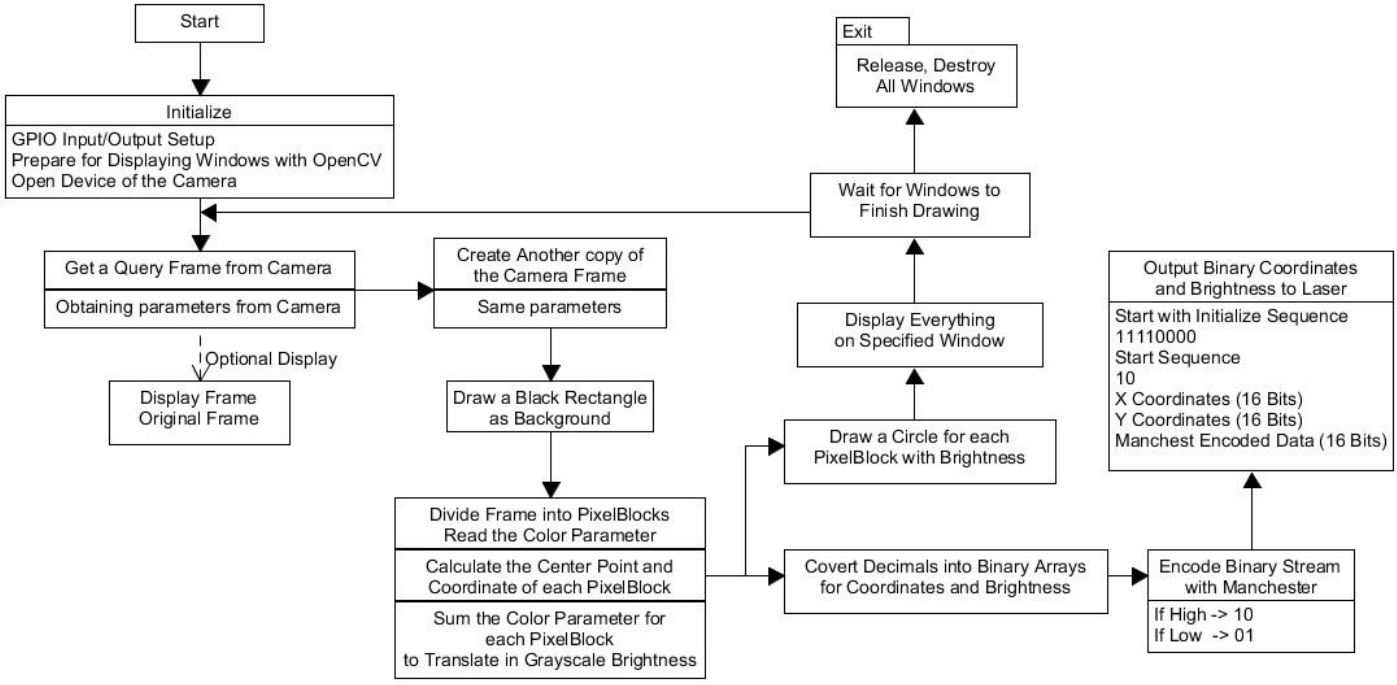


Figure 38: Transmitter Algorithm with Manchester Encoding

The algorithm start by capturing the image stream and load the stream into frames. Once a frame is obtained, start dividing into pixelblocks. Then perform grayscale conversion, while save the coordinates of each pixelblocks. The effect of this can be seen in Section 3.6.2.1. Once the pixelblocks are determined, display on a window. At the same time encode the value of x coordinates, y coordinates and data with Manchester then output to the laser. The resulting length of a packets which contains the coordinates and brightness of one pixelblock is 58 bits.

3.7.5 Receiver Algorithm

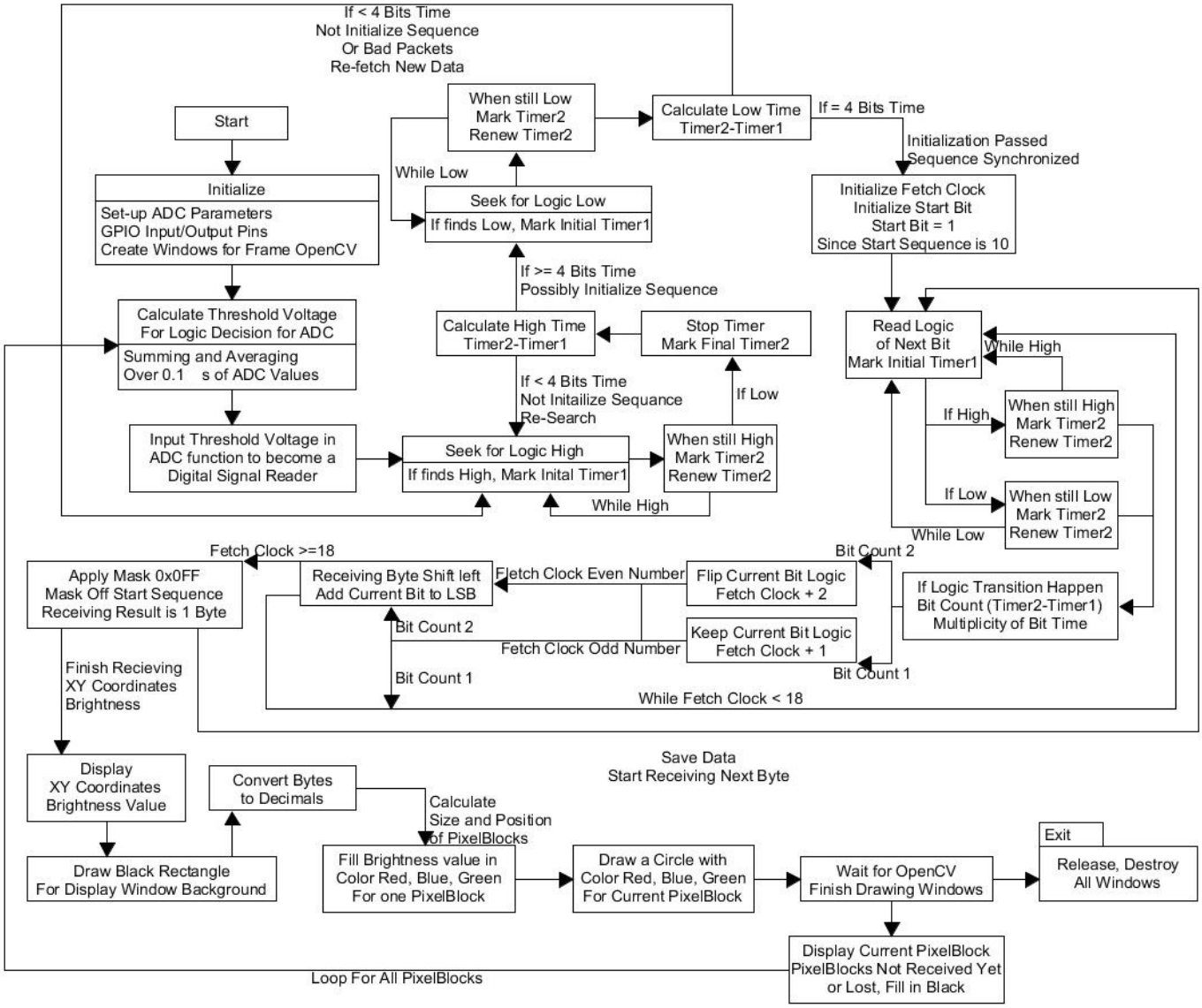


Figure 39: Receiver Algorithm with Manchester Decoding & Edge Detection

The algorithm start by calculating the threshold of logic decision and picking out the initialize sequence amongst the binary data stream. Once an initialize sequence is detected, start the clock for fetching the Manchester encoded data and the "current bit" start with 1. If the time between transition is 1 ms, the current bit stays the same and the clock increment. If the time between transition is 2 ms, the current bit is flipped and the clock increment by 2. Then save the current bit value for every other clock, and loop for 18 clock cycle. Since the data at current stage contains the start bits, we apply a mask to eliminate it. Then loop back to receive the rest of the 2 bytes. Once the 3 bytes are received, convert the binary data into decimal, then project the pixelblock onto a window with according x,y coordinates and brightness. Then finally seek for another initial sequence for another pixelblock.

3.7.5.1 Advantages and Disadvantages

The advantage of using such a combination of method is that no sampling is needed and we can extract the data as the encoded sequence flows in. We do not need to wait for the whole sequence is fetched to start extracting the data. By having initialize sequence, we can display pixelblock by pixelblock and not need to wait for the whole frame to arrive. This also solve the problem when the laser is not on the target, since it will keep looping to search for an initialize sequence. By having a moving logic threshold, we also took care of when the laser is partially on the target. So even if we are only receiving logic level up to 1V, the receiver can still decode the sequence accordingly.

The disadvantage of this is that since Manchester doubles the number of bit of the data, which means it cost twice the time to fetch the data sequence. The transmitter sends out a pixel for approximately 70 ms and the receiver process a pixel for approximately 58 ms and 100 ms to calculate the current logic threshold. Since the receiver and transmitter are not synchronous, and the different times for each of them to process a set of the data, many data will be lost. The receiver are able to process a pixel for approximately 158 ms, which is 632 ms for 4 pixels and 15.8 second for 100 pixels. So the speed is also a problem.

3.7.6 Testing Wireless Transmission

Once the algorithm of the wireless link is complete, the algorithm needs to be tested to see its performance. Below are three methods we used to test out the performance.

3.7.6.1 Pure Binary Square Wave

In this test, we send out data with 101010... and so on, and see how fast, how accurate and how stable the wireless link is. Even though we name this testing sequence as “square wave”, we are sending the encoded 1 and 0 along with its initialize sequence and start bits. For example “1111000001010” or “111100001001”. The receiver will still do the necessary decoding algorithm to display the decoded data.

Figure 40: Data stream of Transmitter and Receiver for pure square wave

For the transmitter, it cost approximately 5 seconds to send 415 0s and 1s. So it is approximately 12.05 ms per packet of data, which is also equivalent to 1.004 ms per bits. So this shows that the transmitter are working well and to our expectation.

For the receiver, it cost approximately 5 seconds to receive and decode 40 0s and 1s. So it takes approximately 125 ms to process one packet. The number is reasonable, since we estimate it takes around 12 ms + 100 ms = 112 ms to process a packet, but the performance is a bit lower than expected.

Over the wireless link work as expected without any data or image processing. We did not see any drift happening like what happened during Section 3.7.2. The code use for this section is in Appendix A.4.4.

3.7.6.2 Number Counting

In this test we will start sending data counting for 0 to 255 in binary, rest to 0 and loop. This will test how the system perform with 1 byte of data. Now the transmitter is send 8 + 2 + 16 =26 bits per pixels. So we are expecting the transmitter to send 1 packets every 26 ms.

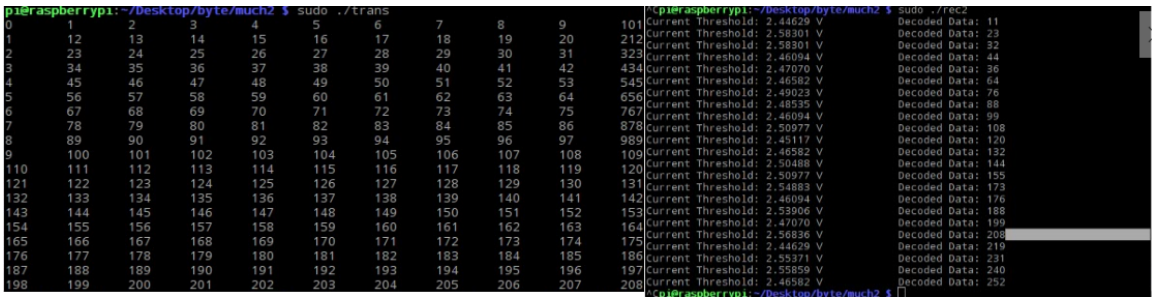


Figure 41: Data stream of Transmitter and Receiver for 1 byte of unencoded data

The transmitter finished counting from 0 to 255 for approximately 6.72 seconds, which is equivalent to 26.25 ms per packets and 1.0096 ms per bits. So as the same as previous, the transmitter is meeting our expectation.

For the receiver, it takes around 7 seconds before it receive the next cycle of data. The number of data received out of the 256 data is 23 from Figure 41, which means it takes 304 ms to fetch and process a data where it should be 126 ms. We were expecting the receiver to pick up data for every 126/26=4.85 $\approx$ 5 numbers since it takes long for the receiver to process per data than the transmitter. This may be cause by the receiver to be able to find the initialize sequence, resulting searching loop. So the receiver lost on average of 10 packets of data instead of 10 where our expectation were.

Overall the transmitter still perform well and expect, since there are less computation required, but the receiver perform worse than we expect. Despite the lost packets, the calculation of logic threshold and the decoding data seem to be perform as expected. As Figure 41, the decoded data is also counting up. The test code for this section can be found in Appendix A.4.5.

3.7.6.3 One Pixel

In this test, we will attempt to send data of one pixel and see how the wireless link perform. We expect the transmitter to have a delay of approximately 70 ms for every pixel send, and will require approximately double for the receiver.

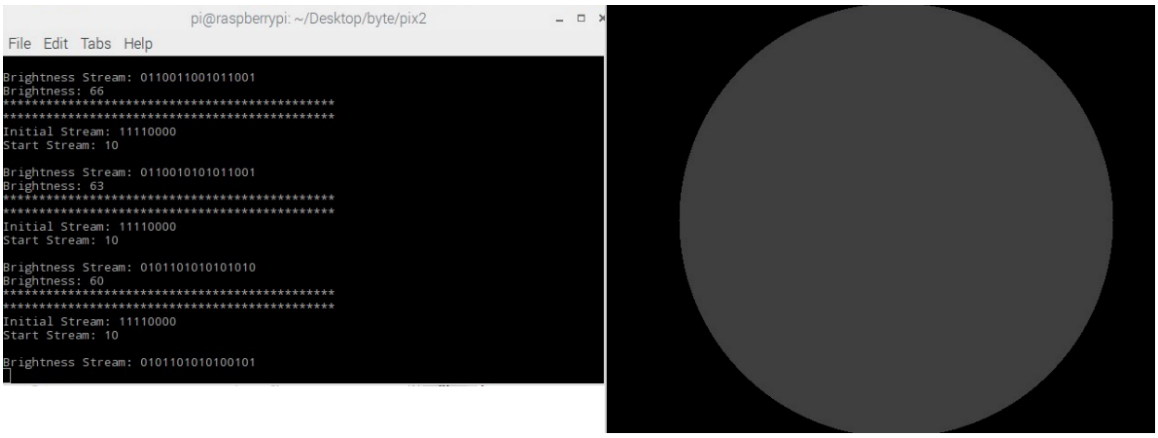


Figure 42: Data stream of Transmitter for one pixel of data

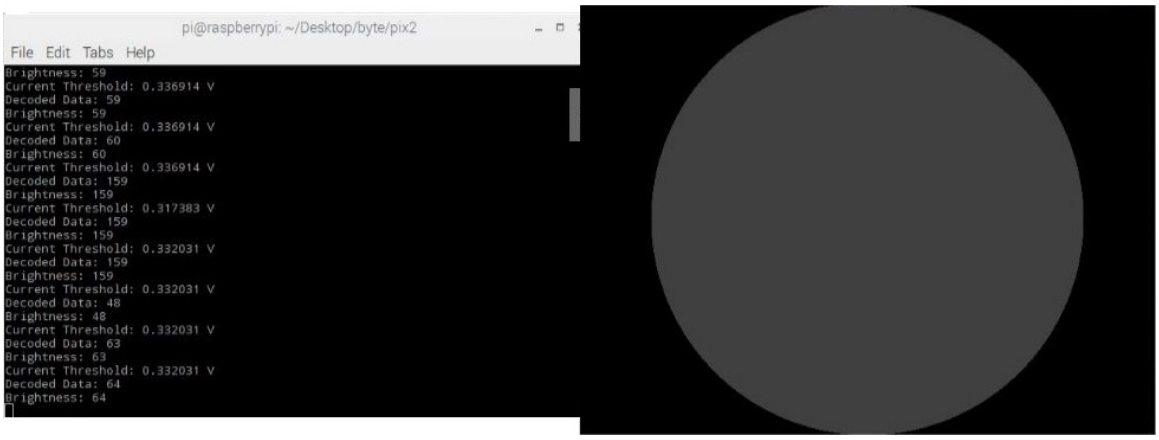


Figure 43: Data stream of Receiver for one pixel of data

The transmitter performed as expect to have approximately 70 ms delay between sending a pixel of data, which is equivalent to have 70 + 58 = 128 ms for every pixel. The receiver cost approximately 6 seconds to receive 10 pixels, which is 600 ms per pixel. Taking into when the receiver were not able to pick out the initialize sequence as mentioned in Section 3.7.6.2 and the delay from the transmitter, this number of reasonable and expect although it is relatively slow. We also notice that there some wrong decoded data. This is most likely cause by the long delay between each data, and the calculation for logic threshold take those into account too resulting in low threshold voltage, so small noise can cause bit error as we can see in Figure 43, as the threshold voltage in Section 3.7.6.2 are much higher and reasonable. By breaking the decoded data into binary such as 159 = 10011111, we are able to roughly estimate the Bit Error Rate. Out of the 10 pixels receiver, there are approximately 93 bits that are decoded incorrectly, so the Bit Error Rate is approximately 93/580=16.03%. The test code for this can be found in Appendix A.4.6.

3.8 Integration

3.8.1 Integration of the Experimental Setup (Noor)

After integrating the software with the hardware for the tracking component of the project, there it was quickly realised that the physical setup must be changed. The distances between components had to be decreased to allow for a wider range of rotation on the mirror to cover the target. This meant that components had to be recalibrated to ensure power was still being received by the target in the new location. The new measured

distances which were used for the demonstration of the project are presented below in Table 6.

Table 6: Physical Implementation Results Pertaining to Figure 13

d1	5 cm
d2	13 cm
d3	7.5
X_m	5 cm
Camera Height off of table	10.2 cm
Target Height off of table	9.7 cm
Mirror Height off of table	10.2 cm
Laser Height off of table	10 cm

In addition to the integration of the software and hardware for the tracking portion of the project influencing the physical set up, the integration of the transmission and tracking portions also affected the physical set up. When the transmission system was integrated with the tracking system, the laser diode wasn't functioning as brightly as it was before the integration. Due to the added components, the laser wasn't receiving enough power and was therefore not sending a good signal to the target. To overcome this issue, the laser diode was replaced with a more powerful laser. The new laser diode which was used to demonstrate the project is a red diode laser with a wavelength of **635 nm**. This type of laser is classified as a Type 3A laser diode. To ensure safety during the implementation of the project, the maximum voltage supplied was **5 V**, and the current supplied to the laser was measured to be **30 mA**. This led to the input electric power to the laser diode to be **0.15 Watts**. The optical power out of the laser was measured to be **9.53 mW**, which leads to a **6.35%** efficiency of the laser diode. The optical power of the laser measured at the target was **8.62 mW**, which led to an approximate **9.55%** loss.

One of the main issues faced during integration was the delay in the arrival of the MEMs mirror, which in turn delayed the physical setup by a few weeks. After it was decided that the temporary mirror setup was to replace the MEMs mirror, the decision was to place all the components and carry on with the project to prepare for the demonstration. After all the components were in place, the tracking part of the project was still not accurate enough. One of the motors was not moving accordingly as it had only one arm attached to the mirror stand. The motor was replaced with a second bigger motor with two arms connected to the base. After both motors now functioning similarly, the tracking was still not accurate enough, and the data obtained from the testing was not consistent. It was then realized that the optical stand which held the main motor in place, which in turn held the mirror in place was slipping through the stand when the mirror rotated to wide angles. To overcome this issue in the little time that was left before the demonstration, the motor was glued onto the optical mount which held it in place. To further secure the motor, a thin metal wire was wrapped around it to hold it in place. After running some tests to make sure that certain outputs were consistent, the physical setup of the project was completed.

3.8.2 Integration of the Transmission System (Zac)

Initially while the tracking system and transmission system were progressing separately, the laser was only centimetres away from the photodiode, but the actual system is tens of centimetres away including the reflection from optical mirrors. So the initial laser was not bright enough for the integration. Hence in terms of the hardware component of the transmission system, the laser we were working with while the tracking and transmission system were progressing separately was switch out to a brighter laser.

During the initial integration phase, the transmission algorithm does not account for the movement of the laser. The algorithm developed saw a pure transmission of data with a stable constant laser. Hence we have to come up with the idea of the moving logic threshold to take into account when the laser is on the target, partially on the target or even not on the target at all. By having this extra algorithm, the transmission system still performed well individually. When integrate with the tracking system again, the system performed as expected well. But when the optical servo mirror is in transaction of moving one point to another, the Bit Error Rate went up, and when the optical servo mirror stopped moving, the Bit Error Rate dropped back to normal again. The detailed analysis can be found in Section 3.6 and 3.7.

3.8.3 Integration of the Tracking System (Anas)

During integration, minor unexpected issues were faced. Although each component was developed separately, but integration was considered all along. Namely, since the start of the project, tracking was implemented with the existence of a laser. Although said laser beam did not transmit any data at that time, it was only introduced to study the effects of reflection on tracking.

From the tracking point of view, introducing the transmitter only affected the tracking by changing the laser form a constant beam to a blinking one. Which essentially does not affect the tracking process as its independent of laser mode. The blinking bream caused some reflections onto the feedback camera but that did not affect tracking in anyway. Hence, we can say that the integration was instantaneous and produces no complexities.

Several meetings between tracking team and transmission team were conducted to update expectations, give feedback, and raise any concerns any team has about other teams work. This ensured a smooth transition into integrating different components of the project. Communication between the two teams and constant reflecting of expectations led to having practically no issues in the integration process.

4 Project Management (Noor)

Task 1 - Experimental Setup (3 weeks, once all parts have arrived) - **Noor**:

1. Connect the holder mounts to the workbench
2. Connect the laser diode to the power source
3. Connect the mirror to mirror mount and place in front of laser
4. Measure the power of the laser to ensure the experiment is being conducted under safe requirements
5. Once laser and mirror are aligned as per experimental setup design, place the target across the mirror
6. Connect target to oscilloscope to obtain output
7. Measure output and record placement of laser, mirror and target
8. Move the laser and the mirror to optimize the power received by the target, recording the distance and location with every movement.
9. Once an optimal location for all the components has been found, place the camera used for pupil detection across from the target, ensuring that the laser beam is not blocked
10. Change distance between camera and target to optimize pupil detection
11. Ensure all components are wired properly and are functioning within required specifications

Outcome of task: The physical setup of the experiment will be complete

Task 2 - Raspberry Pi 3 setup (2 week)) - **Zac and Anas**:

1. Start-up Raspberry Pi3 and download libraries necessary for later configurations

Outcome of task: The Raspberry Pi will be ready for use to implement tracking software

Task 3 - Pupil Tracking (4 Weeks) - **Anas**:

1. Connect the Raspberry Pi 3 camera to a Raspberry Pi 3
2. Measure and run detection software various times, testing for an optimal distance between camera and pupil that will optimize the output location.
3. Download and import OpenCV
4. Run test software; Implement Haar Cascade (if needed)
5. Feed the system with negative and positive samples for various detection environment
6. Measure the speed and accuracy
7. Try different tracking algorithm if possible

Outcome of task: Tracking the target and generate its coordinates as an output

Task 4 - Dynamic System Integration (3 Weeks) - **Noor**:

1. Set up the physical system
2. Learn how the system moves and target rotates
3. Add the system into experimental setup
4. Connect the target (photodiode) to the dynamic system (solder to extend wires to breadboard)
5. Ensure that the dynamic system is well integrated into the physical setup (adjust height, angle, etc.)
6. Create a circuit to connect the system to the Arduino board to be able to control it automatically by the use of implemented code

7. Create a code to automatically move the dynamic system for the purpose of the experiment (maximum rotating angles, speed, etc.)

Outcome of task: Integration of the dynamic system into the physical setup

Task 5 - Camera Calibration (3 Weeks) - **Tony**:

1. Connect the camera to the Raspberry Pi 3 to obtain images
2. Obtain camera parameters using the image
3. Projection of the image into 3-D coordinates using matrix
4. Implement results into a C++ Algorithm
5. Error analysis

Outcome of task: Camera Calibration

Task 6 - Temporary mirror Setup (3 Weeks) - **Noor and Anas**

1. Setup and Test the temporary mirror and understand its performance and limits
2. Design an algorithm to control the mirror based on design
3. Run a few test cases of certain locations, and the resulting mirror movement, to ensure that the beam is actually centered on the target
4. Once the mirror has been configured and working appropriately, placed at an appropriate distance and experimental setup is completely optimized, use the output data from the pupil detection software as an input argument to mirrors control of motors
5. Run test cases to ensure the two software algorithms are compatible and deliver the required output (Reflecting the laser beam onto the target, depending on the targets new location)

Outcome of task: Integrate the temporary mirror into physical setup and tracking algorithm

Task 7 - Data Modulation/Demodulation (3 Weeks) - **Zac**

1. Generate square wave to simulate the data signal
2. Modulate the square wave
3. Obtain output from the target
4. Demodulate output and obtain the information on monitor
5. Send a complete data
6. Manchester encode and decode pixelated frame, send and receive
7. Adjust the algorithm if necessary

Outcome of task: Able to send data through the laser and display received data by target

Task 8 - Optimization/Integration (6 weeks) - **Tony, Noor, Zac and Anas**

1. Finding the optimal parameters for each component separately
2. Ensure after integrating components that system is functioning appropriately
3. Optimize performance (latency and accuracy) once system is integrated

Outcome of task: Obtain the best possible results

Task 9 - MEMs temporary set-up (3 weeks) - **Zac**:

1. Familiarize the specification of the servo motors (Required current, frequency, voltages)
2. Generate 50Hz PWM signal from the Arduino Uno

3. Control the motors with Arduino
4. Setup Serial Communication between Arduino and Raspberry Pi
5. 3D print the base and holder of the mirror
6. Mount the servo motors and mirrors onto the holder according to the center of rotation
7. Test overall algorithm
8. Integrate with other parts of the setup

Outcome of task: The temporary MEMs replacement are ready to use.

Task 10 - Test MEMs Mirror and Quad Detector for Future Improvement (2 weeks) - **Tony**

1. Test Accuracy, Response time of MEMs Mirror
2. For Quad Detector testing, perform four voltage output analysis, specs and response time
3. Plan out an algorithm for tracking using MEMs Mirror and Quad detector

Outcome of task: Algorithm for MEMs Mirror and Quad detector will be ready for future use.

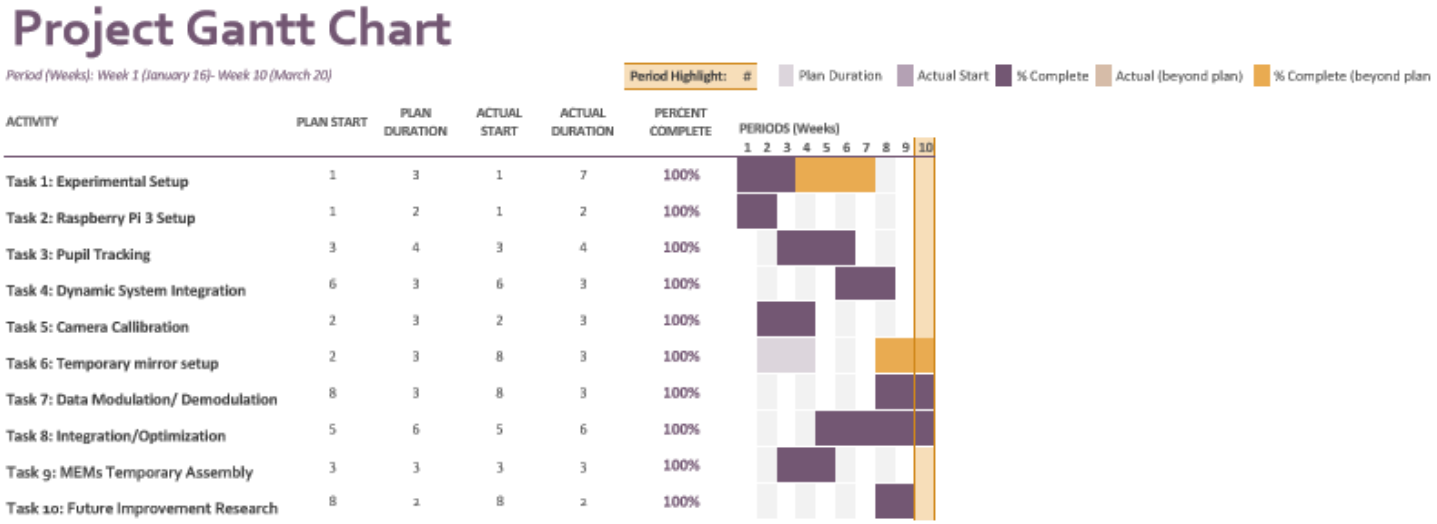


Figure 44: Project Gantt Chart Schedule

4.1 Member Contributions (Noor)

This section of the report highlights the contribution of group members to the defining tasks of the project. The details of each specific task is highlighted in the Project Plan section above. Some tasks are shared amongst group members, while other tasks are completed individually. The Optimization/Integration Task (Task 8) is an ongoing task which is carried out by the entire group to optimize any completed components. The detail of each task can be found in Section 4.

Table 7: Highlighting Members Contributions to Each Task

	Anas	Noor	Tony	Zac
Task 1: Experimental Setup	-	100%	-	-
Task 2: Raspberry Pi3 Setup	50%	-	-	50%

Task 3: Pupil Tracking	100%	-	-	-
Task 4: Dynamic System Integration	-	100%	-	-
Task5: Camera Calibration	-	-	100%	-
Task6: Temporary Mirror Setup	50%	50%	-	-
Task 7: Data Modulation/ Demodulation	-	-	-	100%
Task 8: Optimization/Integration	25%	25%	25%	25%
Task 9: MEMs Temporary Assembly	-	-	-	100%
Task 10: Future Improvement Research	-	-	100%	-

5 Conclusion and Future Work

5.1 Future Work (Tony)

5.1.1 MEMs Mirror[19]

The inaccuracy and delay of the servo motors are the most significant limitation to the performance of the current system.

The mems mirror has a positional repeatability better than 500 micro-degrees at room temperature. In the lab, we draw a small black dot with 1 mm diameters and program the mirror to draw a circle around the black dot. The figure below is captured by camera.

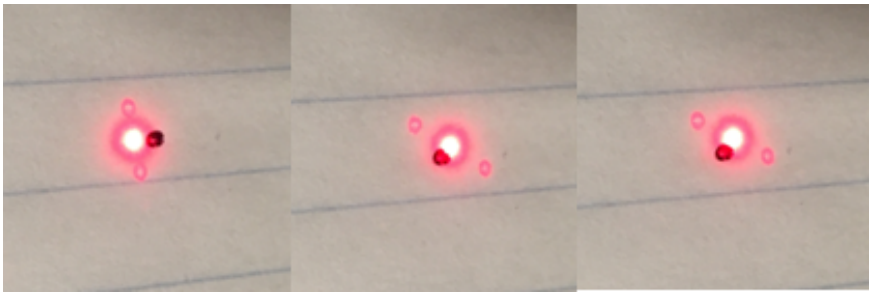


Figure 45: Image to show the accuracy of MEMs Mirror

As one can see the laser is redirected by the mirror and moved just on the edge of the target where the mirror is positioned 200 mm away from the target. On the other hand, the servo motor currently using in system is not able to move the mirror accurately enough and the error is 3 mm where the mirror is positioned 40 mm away from the target. Therefore, we conclude that the mems mirror is able to solve the inaccuracy problem we currently encountered.

The other issue with the servo motor is delay. It takes approximately 0.5 second to redirect laser in order to track the target. The delay mainly contains two components: controller processing delay, motor response delay. For the delay due to motor response, mems mirror has a step response delay less than 1 millisecond and it is able to rotate at 500 rad/s. The figure below shows a good demonstration of the fast response of the mirror.

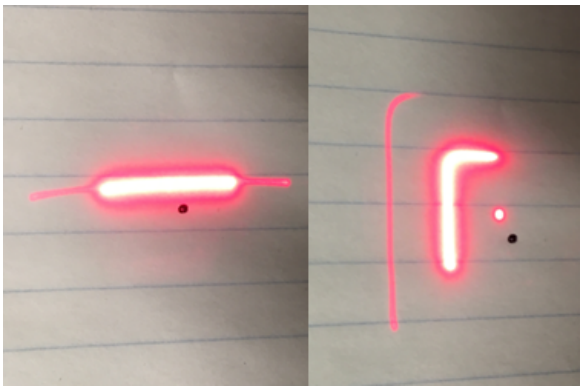


Figure 46: Image to show speed of mirror

The mirror is programed to redirect a modulated laser to draw a square with side 15 mm on the paper. The picture is taken by a camera with 1/60 second exposure time. It is able to display one side of the square

within the frame of the camera. This test illustrates the quick response of the mems mirror.

5.1.2 Quadrant Detector

As discussed before, feedback is essential to every control system but due to the nature of this project it was not an option. However, if wireless feedback is to be integrated on the target that will improve the system greatly. Several additions to the tracking system can be introduced which eliminate a lot of the errors and inconstancies encountered before.

The quadrant detector will improve the performance in terms of delay and accuracy. The quadrant detector contains four photo diodes in parallel. Each diode would provide a voltage whose value based on the intensity of the light falls into the quadrant. Each quadrant outputs a different output which provides greater flexibility in determining the change in laser by studying the differential output between the four panels. Once the laser is on the quadrant detector, the four voltages will feedback to the driver of mems mirror to redirect laser so that it is balanced on the target. In this case, the control algorithm is simpler than the one with camera tracking because the camera tracking algorithm involves video taking, image processing and coordinate transformation. A large portion of process time could be reduced by using quadrant detector. In this case, even the target is continuously moving, it is still can be caught by laser beam.

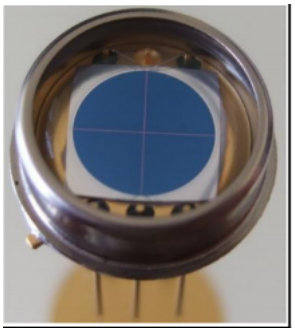


Figure 47: Quadrant Detector

By replacing the photodiode with a quad-detector, the reliance on the camera feed is reduced significantly. Camera tracking will contain error due to many sources such as calibration errors, quantization errors, distortions, and errors due to linear approximation. With the help with quadrant detector, the camera tracking will be executed only when the laser is completely off from the quadrant detector and bring it back onto the target. In this case, there is more error tolerance for the camera tracking because it does not need to point the laser exactly on the center of the target.

Assuming the laser at the center of the Quad-detector, if the laser moves slightly up on the vertical axis, the output of the top two panels will increase while decreasing the output on the bottom two. By studying the differential output of the two sets of panels, the system knows that the laser should be shifted upwards until the output is somewhat the same or to a certain threshold. As soon as the quadrant detector receives the laser signal, the feedback from the quadrant detector will be used for tracking.

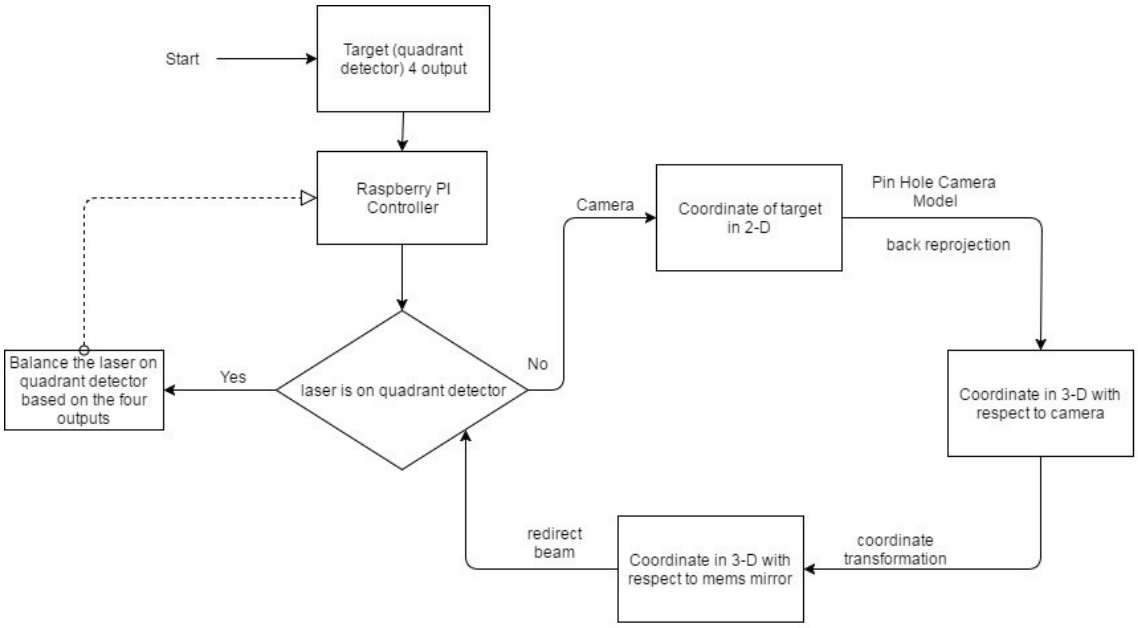


Figure 48: Tracking Algorithm based on Camera and Quadrant Detector

The flow chart above shows the combined tracking algorithm with camera and quadrant detector. The advantage of this new combined algorithm is that it reduces the inaccuracy and the response time significantly.

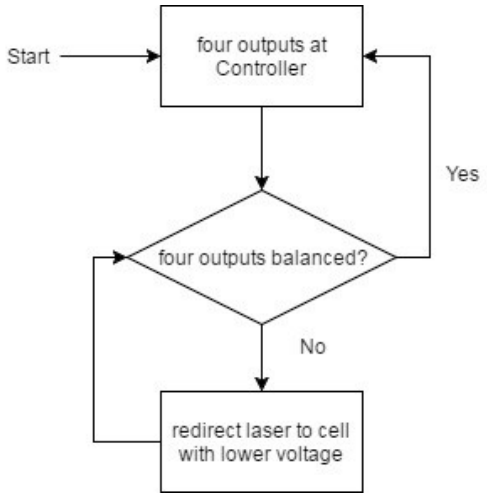


Figure 49: Tracking algorithm based on camera and quadrant detector when laser is on target.

This method proved to be very effective when used for tracking the sun in solar panels. Its more responsive and better for real-time implementations as there exists no processing time what so ever. The response is immediate as only the differential output is used to make a decision. This approach however, does not completely replace camera, the camera is still used to determine the location of the target when the laser is not hitting the diode, namely, when there is no feedback from the Quad-detector. However, the camera can be switched off or put on hold while observing the differential output from the Quad-detectors.

5.1.3 Ellipsoidal Mirror

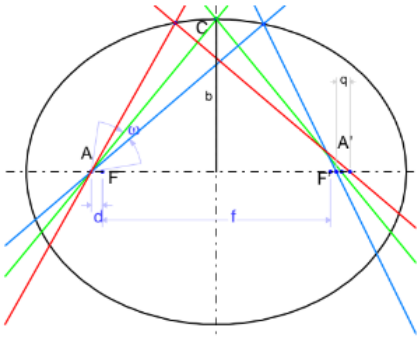


Figure 50: Ellipsoidal Mirror

According to the law of physics, if a laser beam travels through one of the focus of ellipsoid, the reflected beam from the surface will pass through the other focus. Based on this fact, the mems mirror is placed at one of the focus, and the target is moving on the surface of a sphere whose center is the other focus. The blind spot issue could be solved with such a configuration. Additionally, the laser beam could approximately normally incident on the target while the target is moving. Hence, the received power is higher than when the angle of incidence is large. As a result, the signal to noise ratio is increased and bit error rate is lower.

5.1.4 Camera Synchronization (Anas)

As shown by the system diagram, the system contains two cameras. The active feedback camera and the modulation camera. The output of the second camera is what is being modulated and send over laser to the receiver that demodulates it. Typically, it is desired to have the two cameras operate in conjunction which means that the target receives feed and data that corresponds to its current orientation. With the current setup of the system, this is simply not possible. A delay is always associated with single-board computers and data processing, especially when also having communication between different components in the system as well as image processing and transmission. It is possible however, if the improvements suggested above are implemented.

The current setup contains 3 Raspberry Pis and 2 Arduino Uno's which makes it very difficult to operate without bit errors and delays as said before. In addition to, showing outputs to the screen for demonstration purposes which is computationally consuming. Once a more optimal system is built with components that are designed for specific tasks, synchronizing the two cameras becomes not only possible but mandatory. Which means that this system can be used for more critical applications such as; health applications.

5.2 Conclusion (Noor)

To conclude this project, the implementation and integration stages were successfully completed. A successful demonstration of meeting the set goals was conducted to show the functionality of the implemented design. Throughout the project there were some risks and challenges which were mitigated by properly planning for alternative solutions. Although some issues were faced during the integration portion of the project, the fact that two distinct systems (tracking and transmission) were integrated together minimized the complexity of those issues. Only one prototype was developed by the end of the designated time. Although it was not an optimal prototype, it was sufficient to show a proof of concept and illustrate the implementation of the design. As discussed in the section above, there are certain components which could be integrated to optimize the current prototype for future releases.

A Software Code

A.1 Arduino Uno Target Integration Code (Noor)

```
//Written By: Noor Allami
//Last Updated: April 1, 2017

#include <Servo.h> //servo library allows arduino board to control servo motors

//defining Variables
  int X; //horizontal angle
  int Y; //vertical angle

//defining Objects
  Servo vertical; //creating object to control vertical motion servo
  Servo horizontal; // creating object to control horizontal motion servo


 int A_horizontal[]= {50, 60, 70, 80, 90, 100, 110};//creating global variable, array storing all the posit
 int B_vertical[]= {60, 70, 80, 90, 100, 110, 120};//creating global variable, array storing all the positi


void setup()
{

// connecting Servos to output pins
  vertical.attach(13); //attaching vertical servo to pin 13
  horizontal.attach(12); //attaching vertical servo to pin 12


  X= 80;//setting horizontal servo to centre position (0 coordinate)
  X=map(X, 50 ,110, 50, 110);//mapping the value obtained to ensure the values are in the range between 50
  Y= 90;//setting vertical servo to centre position (0 coordinate)
  Y=map(Y, 60, 120, 60, 120);  //mapping the value obtained to ensure the values are in the range between 6
   horizontal.write(X);//moving the horizontal servo to centre position
   vertical.write(Y);// moving the vertical servo to centre position

   delay (3000);//a delay of 3 seconds, before entering into the random loop

// initialize Serial
Serial.begin(9600); //sets the data rate in bits per second (baud) for serial data transmission
}

void loop()
{
// Moving target and waiting
  X= A_horizontal[random_variable(0,6)]; //assigning position to horizontal servo, a value from the horizon
  X=map(X, 50 ,110, 50, 110);//mapping the obtained position to values ranging between 50 and 110 degrees
  Y= B_vertical[random_variable (0,6)];//assigning position to vertical servo, a value from the vertical po
  Y=map(Y, 60, 120, 60, 120); //mapping the obtained position to values ranging between 60 and 120 degrees
horizontal.write(X);//moving the horizontal servo to assigned random position from the horizontal array
vertical.write(Y);//moving the vertical servo to assigned random position from the vertical array
   delay (3000);//a delay of 3 seconds, waiting in position for 3 seconds before next poition
}
//The below function obtains a minimum and a maximum value passed on by the calling function
//it returns a random number between the minimum and maximum numbers passed, saving it as an integer type
  int random_variable (int min, int max)//classifiying function type and defining the values it receives
  {int index;//defining variable index
  index= (int) random(min,max);//assigning the variable index an integer random value between the minimum a
  return index;}//returning the variable index (random integer value between passed min and max numbers)
```

A.2 2 Axis Rotation Mirror Stand (Zac)

A.2.1 Arduino Uno Servo Motor Driver

```
#include <Servo.h>
#include <math.h>
Servo A;
Servo B;

int Apos = 0;
int Apos2 = 95;
int Bpos = 0;
int Bpos2 = 90;
float da = 0;
float db =0;
float dt = 0;
byte number = 0;
unsigned char angbytes[2];

int pos = 0;

int binary2decimal(byte b){   //converts byte into decimal
 int dec = 0;
 int power = 1;
 byte mask;
 int weight;

 for (mask = 0x01; mask; mask <<= 1){
   if (b & mask){
     weight = 1;
   }
   else{
     weight = 0;
   }
   dec = dec + (power * weight);
   power = power * 2;
}
 return dec;
}
void setup()
{
  A.attach(4);  //Motor A at pin 4
  B.attach(9);  //MOtor B at pin 9
  Serial.begin(9600); //start serial com
  pinMode(2,OUTPUT);
  A.write(Apos2); //set motor initial position
  B.write(Bpos2);
}

void loop()
{
  while(Serial.available()<2){}  //wait for 2 bytes
  for(int k=0;k<2;k++){
    angbytes[k]=Serial.read();  //get 2 bytes
  }
  byte Anum=angbytes[0];
  byte Bnum=angbytes[1];
  int Apos2=binary2decimal(Anum); //convert byte into decimal angle
  int Bpos2=binary2decimal(Bnum);
```

```
  if (Apos2==0 && Bpos2==0){    //incase serial com fail
    Apos2=Apos;                 //stay at last position
    Bpos2=Bpos;
  }
  A.write(Apos2); //rotate motor
  B.write(Bpos2);

  Apos=Apos2;
  Bpos=Bpos2;
  Serial.flush();        //clear serial buffer
}
```

A.2.2 Serial Communication Raspberry Pi

A.2.2.1 mySerial.h

```
#ifndef SERIAL
#define SERIAL

#include <string>


class  mySerial
{

public:

  int handle;
  std::string  deviceName;
  int baud;

  mySerial(std::string deviceName, int baud);
  ~mySerial();

  bool Send( unsigned char  * data,int len);
  bool Send(unsigned char value);
  bool Send( std::string value);
  int Receive( unsigned char  * data, int len);
  bool IsOpen(void);
  void Close(void);
  bool Open(std::string deviceName, int baud);
  bool NumberByteRcv(int &bytelen);
};

#endif
```

A.2.2.2 mySerial.cpp

```
extern "C" {
#include <asm/termbits.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <fcntl.h>
}
#include <iostream>
using namespace std;

#include "mySerial.h"
```

```
mySerial::mySerial(string deviceName, int baud)
{
   handle=-1;
   Open(deviceName,baud);
}

mySerial::~mySerial()
{
  if(handle >=0)
      Close();
}

void mySerial::Close(void)
{
   if(handle >=0)
      close(handle);
   handle = -1;
}


bool mySerial::Open(string deviceName , int baud)
{
    struct termios tio;
    struct termios2 tio2;
    this->deviceName=deviceName;
    this->baud=baud;
    handle  = open(this->deviceName.c_str(),O_RDWR | O_NOCTTY /* | O_NONBLOCK */);

    if(handle <0)
       return false;
    tio.c_cflag =  CS8 | CLOCAL | CREAD;
    tio.c_oflag = 0;
    tio.c_lflag = 0;       //ICANON;
    tio.c_cc[VMIN]=0;
    tio.c_cc[VTIME]=1;     // time out every .1 sec
    ioctl(handle,TCSETS,&tio);

    ioctl(handle,TCGETS2,&tio2);
    tio2.c_cflag &= ~CBAUD;
    tio2.c_cflag |= BOTHER;
    tio2.c_ispeed = baud;
    tio2.c_ospeed = baud;
    ioctl(handle,TCSETS2,&tio2);

//   flush buffer
    ioctl(handle,TCFLSH,TCIOFLUSH);

    return true;
}

bool mySerial::IsOpen(void)
{
   return( handle >=0);
}

bool mySerial::Send( unsigned char  * data,int len)
{
   if(!IsOpen()) return false;
   int rlen= write(handle,data,len);
   return(rlen == len);
}
```

```
bool mySerial::Send( unsigned char value)
{
   if(!IsOpen()) return false;
   int rlen= write(handle,&value,1);
   return(rlen == 1);
}

bool mySerial::Send(std::string value)
{
   if(!IsOpen()) return false;
   int rlen= write(handle,value.c_str(),value.size());
   return(rlen == value.size());
}


int  mySerial::Receive( unsigned char  * data, int len)
{
   if(!IsOpen()) return -1;

   // this is a blocking receives
   int lenRCV=0;
   while(lenRCV < len)
     {
       int rlen = read(handle,&data[lenRCV],len - lenRCV);
       lenRCV+=rlen;
     }
   return  lenRCV;
}

bool mySerial::NumberByteRcv(int &bytelen)
{
   if(!IsOpen()) return false;
   ioctl(handle, FIONREAD, &bytelen);
   return true;
}
```

A.2.2.3 SerialTest.cpp

```
#include "mySerial.h"
#include <iostream>
using namespace std;


int  main(void)
{

    mySerial serial("/dev/ttyUSB0",9600);
    while(1){

    unsigned char  dataArray[] = {180,90};
    serial.Send(dataArray,sizeof(dataArray));

    }
    return 0;
}
```

A.3 Tracking (Anas)

A.3.1 HSV Tracking

```
//objectTrackingTutorial.cpp

//Written by  Kyle Hounslow 2013

//Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associa
//, to deal in the Software without restriction, including without limitation the rights to use, copy, modi
//and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, sub

//The above copyright notice and this permission notice shall be included in all copies or substantial port

//THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIM
//FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
//LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WIT
//IN THE SOFTWARE.

#include <sstream>
#include <string>
#include <iostream>
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/objdetect.hpp"
#include "opencv2/videoio.hpp"
#include "opencv/cv.h"



using namespace cv;
using namespace std;
//initial min and max HSV filter values.
//these will be changed using trackbars
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;
//default capture width and height
const int FRAME_WIDTH = 80;
const int FRAME_HEIGHT = 60;
//max number of objects to be detected in frame
const int MAX_NUM_OBJECTS=50;
//minimum and maximum object area
const int MIN_OBJECT_AREA = 20*20;
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;
//names that will appear at the top of each window
const string windowName = "Original Image";
const string windowName1 = "HSV Image";
const string windowName2 = "Thresholded Image";
const string windowName3 = "After Morphological Operations";
const string trackbarWindowName = "Trackbars";
void on_trackbar( int, void* )
{//This function gets called whenever a
    // trackbar position is changed




}
string intToString(int number){
```

```
    std::stringstream ss;
    ss << number;
    return ss.str();
}
void createTrackbars(){
    //create window for trackbars


    namedWindow(trackbarWindowName,0);
    //create memory to store trackbar name on window
    char TrackbarName[50];
    sprintf( TrackbarName, "H_MIN", H_MIN);
    sprintf( TrackbarName, "H_MAX", H_MAX);
    sprintf( TrackbarName, "S_MIN", S_MIN);
    sprintf( TrackbarName, "S_MAX", S_MAX);
    sprintf( TrackbarName, "V_MIN", V_MIN);
    sprintf( TrackbarName, "V_MAX", V_MAX);
    //create trackbars and insert them into window
    //3 parameters are: the address of the variable that is changing when the trackbar is moved(eg.H_LOW),
    //the max value the trackbar can move (eg. H_HIGH),
    //and the function that is called whenever the trackbar is moved(eg. on_trackbar)
    //                                  ---->    ---->     ---->
    createTrackbar( "H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar );
    createTrackbar( "H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar );
    createTrackbar( "S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar );
    createTrackbar( "S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar );
    createTrackbar( "V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar );
    createTrackbar( "V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar );


}
void drawObject(int x, int y,Mat &frame){

    //use some of the openCV drawing functions to draw crosshairs
    //on your tracked image!

    //UPDATE:JUNE 18TH, 2013
    //added 'if' and 'else' statements to prevent
    //memory errors from writing off the screen (ie. 160 x 120 ratio: 1.33 (-25,-25) is not within the wind

    circle(frame,Point(x,y),20,Scalar(0,255,0),2);
    if(y-25>0)
    line(frame,Point(x,y),Point(x,y-25),Scalar(0,255,0),2);
    else line(frame,Point(x,y),Point(x,0),Scalar(0,255,0),2);
    if(y+25<FRAME_HEIGHT)
    line(frame,Point(x,y),Point(x,y+25),Scalar(0,255,0),2);
    else line(frame,Point(x,y),Point(x,FRAME_HEIGHT),Scalar(0,255,0),2);
    if(x-25>0)
    line(frame,Point(x,y),Point(x-25,y),Scalar(0,255,0),2);
    else line(frame,Point(x,y),Point(0,y),Scalar(0,255,0),2);
    if(x+25<FRAME_WIDTH)
    line(frame,Point(x,y),Point(x+25,y),Scalar(0,255,0),2);
    else line(frame,Point(x,y),Point(FRAME_WIDTH,y),Scalar(0,255,0),2);

    putText(frame,intToString(x)+","+intToString(y),Point(x,y+30),1,1,Scalar(0,255,0),2);

}
void morphOps(Mat &thresh){

    //create structuring element that will be used to "dilate" and "erode" image.
    //the element chosen here is a 3px by 3px rectangle
```

```
    Mat erodeElement = getStructuringElement( MORPH_RECT,Size(3,3));
    //dilate with larger element so make sure object is nicely visible
    Mat dilateElement = getStructuringElement( MORPH_RECT,Size(8,8));

    erode(thresh,thresh,erodeElement);
    erode(thresh,thresh,erodeElement);


    dilate(thresh,thresh,dilateElement);
    dilate(thresh,thresh,dilateElement);



}
void trackFilteredObject(int &x, int &y, Mat threshold, Mat &cameraFeed){

    Mat temp;
    threshold.copyTo(temp);
    //these two vectors needed for output of findContours
    vector< vector<Point> > contours;
    vector<Vec4i> hierarchy;
    //find contours of filtered image using openCV findContours function
    findContours(temp,contours,hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPLE );
    //use moments method to find our filtered object
    double refArea = 0;
    bool objectFound = false;
    if (hierarchy.size() > 0) {
        int numObjects = hierarchy.size();
        //if number of objects greater than MAX_NUM_OBJECTS we have a noisy filter
        if(numObjects<MAX_NUM_OBJECTS){
            for (int index = 0; index >= 0; index = hierarchy[index][0]) {

                Moments moment = moments((cv::Mat)contours[index]);
                double area = moment.m00;

                //if the area is less than 20 px by 20px then it is probably just noise
                //if the area is the same as the 3/2 of the image size, probably just a bad filter
                //we only want the object with the largest area so we safe a reference area each
                //iteration and compare it to the area in the next iteration.
                if(area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA && area>refArea){
                    x = moment.m10/area;
                    y = moment.m01/area;
                    objectFound = true;
                    refArea = area;
                }else objectFound = false;


            }
            //let user know you found an object
            if(objectFound ==true){
                putText(cameraFeed,"Tracking Object",Point(0,50),2,1,Scalar(0,255,0),2);
                //draw object location on screen
                drawObject(x,y,cameraFeed);}

        }else putText(cameraFeed,"TOO MUCH NOISE! ADJUST FILTER",Point(0,50),1,2,Scalar(0,0,255),2);
    }
}
int main(int argc, char* argv[])
{
    //some boolean variables for different functionality within this
    //program
```

```
    bool trackObjects = true;
    bool useMorphOps = true;
    //Matrix to store each frame of the webcam feed
    Mat cameraFeed;
    //matrix storage for HSV image
    Mat HSV;
    //matrix storage for binary threshold image
    Mat threshold;
    //x and y values for the location of the object
    int x=0, y=0;const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;
    //create slider bars for HSV filtering
    createTrackbars();
    //video capture object to acquire webcam feed
    VideoCapture capture;
    //open capture object at location zero (default location for webcam)
    capture.open(0);
    //set height and width of capture frame
    capture.set(CV_CAP_PROP_FRAME_WIDTH,FRAME_WIDTH);
    capture.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);
    //start an infinite loop where webcam feed is copied to cameraFeed matrix
    //all of our operations will be performed within this loop
    while(1){
        //store image to matrix
        capture.read(cameraFeed);
        //convert frame from BGR to HSV colorspace
        cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
        //filter HSV image between values and store filtered image to
        //threshold matrix
        inRange(HSV,Scalar(H_MIN,S_MIN,V_MIN),Scalar(H_MAX,S_MAX,V_MAX),threshold);
        //perform morphological operations on thresholded image to eliminate noise
        //and emphasize the filtered object(s)
        if(useMorphOps)
        morphOps(threshold);
        //pass in thresholded frame to our object tracking function
        //this function will return the x and y coordinates of the
        //filtered object
        if(trackObjects)
            trackFilteredObject(x,y,threshold,cameraFeed);

        //show frames
        imshow(windowName2,threshold);
        imshow(windowName,cameraFeed);
        imshow(windowName1,HSV);
        cout << x << "," << y << endl;

        //delay 30ms so that screen can refresh.
        //image will not appear without this waitKey() command
        waitKey(30);
    }




    return 0;
}
```

A.3.2 Haar Cascades

A.3.2.1 Tracker.h

```
/***


#include "opencv2/objdetect.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;




class Tracker
{
    public:
        static int* Track ( );
    protected:
    private:
};



// Function Headers

//void detectAndDisplay( Mat frame, VideoCapture capture );
void detectAndDisplay( Mat frame );
// Global variables
String eye_cascade_name = "/home/pi/Desktop/pupil_detection/haarcascade_eye.xml";
CascadeClassifier eye_cascade;
//CascadeClassifier eyes_cascade;
String window_name = "Capture - Face detection";

// @function Tracker

int* Tracker::Track( void )
{
    VideoCapture capture;
    Mat frame;

    //-- 1. Load the cascades
    if( !eye_cascade.load( eye_cascade_name ) ){ printf("--(!)Error loading eye cascade\n"); return -1; };

    //-- 2. Read the video stream
    capture.open( -1 );
    if ( ! capture.isOpened() ) { printf("--(!)Error opening video capture\n"); return -1; }

    // Set Resolution of frame
    capture.set(CV_CAP_PROP_FRAME_WIDTH, 320);
    capture.set(CV_CAP_PROP_FRAME_HEIGHT, 240);
```

```
    // Error checking
        if ( capture.read(frame) != 1)
        {
            cout << "--(!) Frame was not initilized, Frame was not read!" << endl;
        }

        if( frame.empty() )
        {
            printf(" --(!) No captured frame -- Break!");
            break;
        }


        //-- 3. Apply the classifier to the frame
        detectAndDisplay( frame);

        //detectAndDisplay( frame, capture );
        char c = (char)waitKey(10);
        if( c == 27 ) { break; } // escape


    return 0;
}



// @function detectAndDisplay
//void detectAndDisplay( Mat frame, VideoCapture capture  )
void detectAndDisplay( Mat frame)
{
    std::vector<Rect> eyes;
    Mat frame_gray;
    //double frameRate = capture.get(CAP_PROP_FRAME_COUNT );

/home/pi/Desktop/project_files/Tracker.h:33:24: error: variable or field  detectAndDisplay  declared vo

    cvtColor( frame, frame_gray, COLOR_BGR2GRAY );
    equalizeHist( frame_gray, frame_gray );

    //-- Detect eyes
    eye_cascade.detectMultiScale( frame_gray, eyes, 1.1, 2, 0|CASCADE_SCALE_IMAGE, Size(30, 30) );

    for ( size_t i = 0; i < eyes.size(); i++ )
    {
        Point center( eyes[i].x + eyes[i].width/2, eyes[i].y + eyes[i].height/2 );
        ellipse( frame, center, Size( eyes[i].width/2, eyes[i].height/2 ), 0, 0, 360, Scalar( 255, 0, 255 )

        cout << eyes[i].x + eyes[i].width/2 <<", "<< eyes[i].y + eyes[i].height/2<< endl;



    }
    //-- Show what you got
    imshow( window_name, frame );


    // int fillarr(int* arr)
    // int foo [5] = { 16, 2, 77, 40, 12071 };
```

```
    int coordinates [2] = ;


}

***/
```

A.3.2.2 updateServos.h

```
#include "mySerial.h"
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <iostream>

using namespace std;


class updateServos
{
    public:
        static void sendToServos (unsigned char motor1, unsigned char motor2, mySerial serial );
    protected:
    private:
};


void updateServos::sendToServos (unsigned char motor1, unsigned char motor2, mySerial serial )
{
    unsigned char dataArray[2]={};

    dataArray[0] = motor1;
    dataArray[1] = motor2;

    serial.Send(dataArray,sizeof(dataArray));

}
```

A.3.2.3 main.cpp

```
/**
 * This is the main program interface where the identifying and the
 * detection of the object happens. Then a decision on how to move the
 * servos is made!
 *
 *
 * This program assumes that the coordinates of the center of the object
 * is known. As well as, the inital position of the servos. This is made
 * by desogn as it allows for greater flixability and a more accurate
 * results.
 *
 */



#include <stdio.h>
#include <iostream>
```

```
#include <stdlib.h>
#include <iostream>
#include <chrono>
#include <thread>
#include <unistd.h>
#include <vector>



#include "opencv2/objdetect.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"



#include "mySerial.h"
#include "updateServos.h"
#include "Tracker.h"


// Importing names spaces: Standard and computer Vision
using namespace std;
using namespace cv;




/** Global variables */
String eye_cascade_name = "/home/pi/Desktop/project_files/haarcascade_eye.xml";

CascadeClassifier eye_cascade; // used to detect the object

unsigned char dataArray[2]={};



/** Thresholds and limits for image processing */

int insigChange = 5;
int level1Change = 10;
int level2Change = 20;

int changeDir_Hor = 5;
int changeDir_Ver = 5;







//CascadeClassifier eyes_cascade;
String window_name = "Capture - Face detection";


/* Initilize Coordinates and Servos
Upon starting the application, users have to define the inital value
the two servos and the inital location of the center of the object
 */

int default_servo_ver = 96;
```

```
int default_servo_hor = 99;

int default_center_x = 162;
int default_center_y = 147;


/** Function Headers */
vector<int> detectAndDisplay( Mat frame );

// moving the mirror function
void moveMirror (int,  int, mySerial);

/**
 * updating horizental coordinate, takes in the currunt horezintal
 * coordiante, then processes it and makes a decision on wether or not
 * servos should be moved.
 * Returns: new horizenrtal angle degree integrers
*/

int updateHorizental(int);
int updateVertical(int);



int main()
{

    // initializing serial channel
    // for sending data through USB port to the servos

    mySerial serial("/dev/ttyUSB0",9600);
    usleep(2000000);


    /** Initilizing system. Moving servos to default position
     * Defailt position: Which is position zero of the target.
     *
     * */

    // Load default coordinates
    dataArray[0]=default_servo_ver;
    dataArray[1]=default_servo_hor;

    // Move servos to default position, initilize
    serial.Send(dataArray,sizeof(dataArray));

    // Notify user of Servos location
    cout << "Servos: " << default_servo_ver <<", "<< default_servo_hor << endl;


    // Camera instance and frame
    // A frame will be captured from the Video Stream and saved onto
    // frame

    VideoCapture capture;
    Mat frame;

    // vector of size 2 that hold x coordinate of center in
    // coord[0] and y coordinate in coord[1]
    vector<int> coord;
```

```
// Load the cascades
if( !eye_cascade.load( eye_cascade_name ) ){ printf("--(!)Error loading eye cascade\n"); return -1; };

// Read the video stream
capture.open( -1 );
if ( ! capture.isOpened() ) { printf("--(!)Error opening video capture\n"); return -1; }


/** Set Resolution of frame\
 * This resolution was chosen after several trial and error itterations
 * a high resolution means more data to process which leads to more
 * latency but better detection. A low resolution leads to mucg faster
 * processing however, less reliable information
*/

capture.set(CV_CAP_PROP_FRAME_WIDTH, 320);
capture.set(CV_CAP_PROP_FRAME_HEIGHT, 240);



/** Detection and Tracking Loop */

    while (true)
    {

        /** Error detection statements and Error Checking*/

        // VideoCapture capture
        if ( capture.read(frame) != 1)
            {cout << "--(!) Frame was not initilized, Frame was not read!" << endl;}

        if( frame.empty() )
            {printf(" --(!) No captured frame -- Break!");}


         /** Detection */
        // Obtain currunt coordinates of center
        coord = detectAndDisplay( frame );

        // Load new coordinates
        int currunt_hor_coord = coord[0] ;
        int currunt_ver_coord = coord[1] ;


        int servo_hor_angle = updateHorizental(currunt_hor_coord);
        int servo_ver_angle = updateVertical(currunt_ver_coord);

        // notify user of new angles
        cout<< "Center in pixels: " << coord[0] <<", "<< coord[1]<< endl;


        if (currunt_hor_coord!= 0 && currunt_ver_coord != 0 )
        {
            // Load new coordinates
            dataArray[0]=servo_ver_angle;
            dataArray[1]=servo_hor_angle;


            // send new angles to servos and wait for 0.5 seconds to
            // allow time for the servos to move
            serial.Send(dataArray,sizeof(dataArray));
```

```
                //usleep(500000);
            }



               }


    // Close serial channel when done looping
    serial.Close();
    return 0;
}



/** Move servos to new specified location */
void moveMirror (int motor_1,  int motor_2, mySerial serial)
{

    unsigned char dataArray[2]={};

    dataArray[0] = motor_1;
    dataArray[1] = motor_2 ;

    serial.Send(dataArray,sizeof(dataArray));
    usleep(1000000);


}



vector<int> detectAndDisplay( Mat frame  )
{

        std::vector<Rect> eyes;
        Mat frame_gray;
        vector<int> coord (2);

        cvtColor( frame, frame_gray, COLOR_BGR2GRAY );
        equalizeHist( frame_gray, frame_gray );

        //-- Detect eyes
        eye_cascade.detectMultiScale( frame_gray, eyes, 1.1, 2, 0|CASCADE_SCALE_IMAGE, Size(30, 30) );


        for ( size_t i = 0; i < eyes.size(); i++ )
        {
            Point center( eyes[i].x + eyes[i].width/2, eyes[i].y + eyes[i].height/2 );
            ellipse( frame, center, Size( eyes[i].width/4, eyes[i].height/4 ), 0, 0, 360, Scalar( 255, 0, 2

            //cout << eyes[i].x + eyes[i].width/2 <<", "<< eyes[i].y + eyes[i].height/2<< endl;

        coord [0] = eyes[i].x + eyes[i].width/2;
        coord [1] = eyes[i].y + eyes[i].height/2;
        }
```

```
        //-- Show what you got
        imshow( window_name, frame );


        char c = (char)waitKey(10);
        //if( c == 27 ) { break; } // escape


        return coord;

}



 int updateHorizental(int current_x_coor)
 {
    int updatedCoord;

    // direction = 1 means move to the left. Direction = -1 means move to the right
    int direction = 1;
    int key = default_center_x - current_x_coor ;


    // Following statement determines if motors move to the left or right
    // based on the value direction
    if (key < -1*changeDir_Hor)
    {direction = -1;
    }


    // Due to the inaccurecy of the servos. A linear Mapping had to be
    // used instead of the original approached we planned.


    // Linear mapping of certain regions of the camera plane to certain
    // angles of the servos.

    // When using MEMs, this mapping is to be replaced by pinhole Camera
    // Model. Angles are then calculated using equations obtained
    // in last term's report.

    // Mapping is done using if statements but can easily be changed
    // to switch case

    if (key > -insigChange && key < insigChange)
        updatedCoord = default_servo_hor ;
    else if (key < level1Change || key > -1*level1Change)

        {
            // In this case servo undershoots to the right side so
            // the factor had to be doubled
            if (direction == -1)
            {
                updatedCoord = default_servo_hor +  2*direction;
        }
        else {updatedCoord = default_servo_hor +  1*direction;}
        }
    else if (key < level2Change || key > -1*level2Change)
        updatedCoord = default_servo_hor +  2*direction;
    else
        updatedCoord = default_servo_hor +  3*direction;
```

```
    cout << "New Horizontal Angle: " <<     updatedCoord << endl;
    return updatedCoord;

 }



//  This method is identical to the previous method. However its used \
//  to update the vertical axis

 int updateVertical(int current_y_coor)
 {
    int updatedCoord;

    // direction = 1 means move to the Down. Direction = -1 means move to the Up
    int direction = -1;
    int key = default_center_y - current_y_coor ;



    // Following statement determines if motors move to the Up or Down
    // based on the value direction
    if (key < -1*changeDir_Ver)
    {direction = 1;
    }


    if (key > -insigChange && key < insigChange)
        updatedCoord = default_servo_ver ;
    else if (key < level1Change || key > -1*level1Change)

        {
            if (direction == -1)
            {
                updatedCoord = default_servo_ver +  3*direction;
        }
        else {updatedCoord = default_servo_ver +  2*direction;}
        }



    else if (key < level2Change || key > -1*level2Change)
        updatedCoord = default_servo_ver +  2*direction;
    else
        updatedCoord = default_servo_ver +  3*direction;


    cout << "New Vertical Angle: " <<   updatedCoord<< endl;
    cout << "****************************************"<<   "\n" <<endl;

    return updatedCoord;

 }
```

A.3.3 Servo Mirrors Testing and Calibration

```
/**
 * This is the main program interface where the identifying and the
 * detection of the object happens. Then a decision on how to move the
 * servos is made!
```

```
 *
 *
 * This program assumes that the coordinates of the center of the object
 * is known. As well as, the inital position of the servos. This is made
 * by desogn as it allows for greater flixability and a more accurate
 * results.
 *
 */



#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <iostream>
#include <chrono>
#include <thread>
#include <unistd.h>
#include <vector>



#include "opencv2/objdetect.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"



#include "mySerial.h"
#include "updateServos.h"
#include "Tracker.h"


// Importing names spaces: Standard and computer Vision
using namespace std;
using namespace cv;



/** Global variables */
String eye_cascade_name = "/home/pi/Desktop/project_files/haarcascade_eye.xml";

CascadeClassifier eye_cascade; // used to detect the object

unsigned char dataArray[2]={};



/** Thresholds and limits for image processing */

int insigChange = 5;
int level1Change = 10;
int level2Change = 20;

int changeDir_Hor = 5;
int changeDir_Ver = 5;
```

```
//CascadeClassifier eyes_cascade;
String window_name = "Capture - Face detection";


/* Initilize Coordinates and Servos
Upon starting the application, users have to define the inital value
the two servos and the inital location of the center of the object
 */

int default_servo_ver = 96;
int default_servo_hor = 99;

int default_center_x = 162;
int default_center_y = 147;


/** Function Headers */
vector<int> detectAndDisplay( Mat frame );

// moving the mirror function
void moveMirror (int,  int, mySerial);

/**
 * updating horizental coordinate, takes in the currunt horezintal
 * coordiante, then processes it and makes a decision on wether or not
 * servos should be moved.
 * Returns: new horizenrtal angle degree integrers
*/

int updateHorizental(int);
int updateVertical(int);



int main()
{

    // initializing serial channel
    // for sending data through USB port to the servos

    mySerial serial("/dev/ttyUSB0",9600);
    usleep(2000000);


    /** Initilizing system. Moving servos to default position
     * Defailt position: Which is position zero of the target.
     *
     * */

    // Load default coordinates
    dataArray[0]=default_servo_ver;
    dataArray[1]=default_servo_hor;

    // Move servos to default position, initilize
    serial.Send(dataArray,sizeof(dataArray));

    // Notify user of Servos location
    cout << "Servos: " << default_servo_ver <<", "<< default_servo_hor << endl;
```

```
// Camera instance and frame
// A frame will be captured from the Video Stream and saved onto
// frame

VideoCapture capture;
Mat frame;

// vector of size 2 that hold x coordinate of center in
// coord[0] and y coordinate in coord[1]
vector<int> coord;



// Load the cascades
if( !eye_cascade.load( eye_cascade_name ) ){ printf("--(!)Error loading eye cascade\n"); return -1; };

// Read the video stream
capture.open( -1 );
if ( ! capture.isOpened() ) { printf("--(!)Error opening video capture\n"); return -1; }


/** Set Resolution of frame\
 * This resolution was chosen after several trial and error itterations
 * a high resolution means more data to process which leads to more
 * latency but better detection. A low resolution leads to mucg faster
 * processing however, less reliable information
*/

capture.set(CV_CAP_PROP_FRAME_WIDTH, 320);
capture.set(CV_CAP_PROP_FRAME_HEIGHT, 240);



/** Detection and Tracking Loop */

    while (true)
    {

        /** Error detection statements and Error Checking*/

        // VideoCapture capture
        if ( capture.read(frame) != 1)
            {cout << "--(!) Frame was not initilized, Frame was not read!" << endl;}

        if( frame.empty() )
            {printf(" --(!) No captured frame -- Break!");}


         /** Detection */
        // Obtain currunt coordinates of center
        coord = detectAndDisplay( frame );

        // Load new coordinates
        int currunt_hor_coord = coord[0] ;
        int currunt_ver_coord = coord[1] ;


        int servo_hor_angle = updateHorizental(currunt_hor_coord);
        int servo_ver_angle = updateVertical(currunt_ver_coord);
```

```
            // notify user of new angles
            cout<< "Center in pixels: " << coord[0] <<", "<< coord[1]<< endl;


            if (currunt_hor_coord!= 0 && currunt_ver_coord != 0 )
            {
                // Load new coordinates
                dataArray[0]=servo_ver_angle;
                dataArray[1]=servo_hor_angle;


                // send new angles to servos and wait for 0.5 seconds to
                // allow time for the servos to move
                serial.Send(dataArray,sizeof(dataArray));
                //usleep(500000);
            }



           }


    // Close serial channel when done looping
    serial.Close();
    return 0;
}



/** Move servos to new specified location */
void moveMirror (int motor_1,  int motor_2, mySerial serial)
{

    unsigned char dataArray[2]={};

    dataArray[0] = motor_1;
    dataArray[1] = motor_2 ;

    serial.Send(dataArray,sizeof(dataArray));
    usleep(1000000);


}



vector<int> detectAndDisplay( Mat frame  )
{

        std::vector<Rect> eyes;
        Mat frame_gray;
        vector<int> coord (2);

        cvtColor( frame, frame_gray, COLOR_BGR2GRAY );
        equalizeHist( frame_gray, frame_gray );

        //-- Detect eyes
        eye_cascade.detectMultiScale( frame_gray, eyes, 1.1, 2, 0|CASCADE_SCALE_IMAGE, Size(30, 30) );
```

```
        for ( size_t i = 0; i < eyes.size(); i++ )
        {
            Point center( eyes[i].x + eyes[i].width/2, eyes[i].y + eyes[i].height/2 );
            ellipse( frame, center, Size( eyes[i].width/4, eyes[i].height/4 ), 0, 0, 360, Scalar( 255, 0, 2

            //cout << eyes[i].x + eyes[i].width/2 <<", "<< eyes[i].y + eyes[i].height/2<< endl;

        coord [0] = eyes[i].x + eyes[i].width/2;
        coord [1] = eyes[i].y + eyes[i].height/2;
        }




        //-- Show what you got
        imshow( window_name, frame );


        char c = (char)waitKey(10);
        //if( c == 27 ) { break; } // escape


        return coord;

}



 int updateHorizental(int current_x_coor)
 {
    int updatedCoord;

    // direction = 1 means move to the left. Direction = -1 means move to the right
    int direction = 1;
    int key = default_center_x - current_x_coor ;


    // Following statement determines if motors move to the left or right
    // based on the value direction
    if (key < -1*changeDir_Hor)
    {direction = -1;
    }


    // Due to the inaccurecy of the servos. A linear Mapping had to be
    // used instead of the original approached we planned.


    // Linear mapping of certain regions of the camera plane to certain
    // angles of the servos.

    // When using MEMs, this mapping is to be replaced by pinhole Camera
    // Model. Angles are then calculated using equations obtained
    // in last term's report.

    // Mapping is done using if statements but can easily be changed
    // to switch case

    if (key > -insigChange && key < insigChange)
        updatedCoord = default_servo_hor ;
    else if (key < level1Change || key > -1*level1Change)
```

```
        {
            // In this case servo undershoots to the right side so
            // the factor had to be doubled
            if (direction == -1)
            {
                updatedCoord = default_servo_hor +  2*direction;
        }
        else {updatedCoord = default_servo_hor +  1*direction;}
        }
    else if (key < level2Change || key > -1*level2Change)
        updatedCoord = default_servo_hor +  2*direction;
    else
        updatedCoord = default_servo_hor +  3*direction;

    cout << "New Horizontal Angle: " <<     updatedCoord << endl;
    return updatedCoord;

 }



//  This method is identical to the previous method. However its used \
//  to update the vertical axis

 int updateVertical(int current_y_coor)
 {
    int updatedCoord;

    // direction = 1 means move to the Down. Direction = -1 means move to the Up
    int direction = -1;
    int key = default_center_y - current_y_coor ;



    // Following statement determines if motors move to the Up or Down
    // based on the value direction
    if (key < -1*changeDir_Ver)
    {direction = 1;
    }


    if (key > -insigChange && key < insigChange)
        updatedCoord = default_servo_ver ;
    else if (key < level1Change || key > -1*level1Change)

        {
            if (direction == -1)
            {
                updatedCoord = default_servo_ver +  3*direction;
        }
        else {updatedCoord = default_servo_ver +  2*direction;}
        }



    else if (key < level2Change || key > -1*level2Change)
        updatedCoord = default_servo_ver +  2*direction;
    else
        updatedCoord = default_servo_ver +  3*direction;
```

```
    cout << "New Vertical Angle: " <<   updatedCoord<< endl;
    cout << "****************************************"<<   "\n" <<endl;

    return updatedCoord;

 }
```

A.4 Pixelization and Wireless Communication (Zac)

A.4.1 Magick++ Code

```
#include <Magick++.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
using namespace Magick;
int main(int argc,char **argv){
    system("./stop_camd.sh");
    system("./start_camd.sh");
    usleep(1000000);
while(1){
    system("./do_caputure.sh");
    usleep(1000000);
    InitializeMagick(*argv);
    Image image;

    image.read("tmp.jpg");
    image.resize("64x64");
    image.type( GrayscaleType );
    image.write("grey.jpg");
    int w=image.columns();
    int h=image.rows();

    Magick::Quantum* pixels = image.getPixels(0,0,w,h);

    cout << w << endl;
    cout << h << endl;
    cout << pixels[20] << endl;

Pixels view(image);
}
    return 0;
}
```

A.4.2 RaspiFastCamD

do_capture.sh

```
#!/bin/bash
#Helper script to capture an image with raspifastcamd

pid_file=/tmp/raspifastcamd.pid

if [ ! -e $pid_file ]; then
    echo "Error: The pid file $pid_file does no exist, looks like raspifastcamd is not running."
    exit 1
fi
```

```
kill -USR1 $(cat $pid_file)
```

start_camd.sh

```
#!/bin/bash
#Helper script to start RaspiFastCamD

#We would like to write this to /var/run, but this need root privileges...
pid_file=/tmp/raspifastcamd.pid

if [ -e $pid_file ]; then
    echo "Error: The pid file $pid_file exists, looks like raspifastcamd is already running."
    echo "If this is not the case delete the file."
    exit 1
fi

output_file=${1-/home/pi/Desktop/send/tmp.jpg}

echo "Output will be written to $output_file"

#This will make pictures of 200x200px feel free to change.
./raspifastcamd -w 128 -h 128 -o $output_file &
pid=$!

echo "Pid of raspifastcamd is $pid"

echo $pid > $pid_file

exit 0
```

stop_camd.sh

```
#!/bin/bash
#Helper script to start RaspiFastCamD

pid_file=/tmp/raspifastcamd.pid

if [ ! -e $pid_file ]; then
    echo "Error: The pid file $pid_file does not exist, looks like raspifastcamd is not running."
    exit 1
fi

pid=$(cat $pid_file)

echo "Pid of raspifastcamd was $pid"

kill $pid

echo "Killed raspifastcamd"

rm $pid_file

exit 0
```

A.4.3 ADC Library - MCP3008

mcp3008Spi.h

```
/*************************************************************************
 * This header file contains the mcp3008Spi class definition.
 * Its main purpose is to communicate with the MCP3008 chip using
 * the userspace spidev facility.
 * The class contains four variables:
 * mode        -> defines the SPI mode used. In our case it is SPI_MODE_0.
 * bitsPerWord -> defines the bit width of the data transmitted.
 *        This is normally 8. Experimentation with other values
 *        didn't work for me
 * speed       -> Bus speed or SPI clock frequency. According to
 *                https://projects.drogon.net/understanding-spi-on-the-raspberry-pi/
 *            It can be only 0.5, 1, 2, 4, 8, 16, 32 MHz.
 *                Will use 1MHz for now and test it further.
 * spifd       -> file descriptor for the SPI device
 *
 * The class contains two constructors that initialize the above
 * variables and then open the appropriate spidev device using spiOpen().
 * The class contains one destructor that automatically closes the spidev
 * device when object is destroyed by calling spiClose().
 * The spiWriteRead() function sends the data "data" of length "length"
 * to the spidevice and at the same time receives data of the same length.
 * Resulting data is stored in the "data" variable after the function call.
 * ***********************************************************************/
#ifndef MCP3008SPI_H
    #define MCP3008SPI_H

#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/spi/spidev.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string>
#include <iostream>


class mcp3008Spi{

public:
    mcp3008Spi();
    mcp3008Spi(std::string devspi, unsigned char spiMode, unsigned int spiSpeed, unsigned char spibitsPerWo
    ~mcp3008Spi();
    int spiWriteRead( unsigned char *data, int length);

private:
    unsigned char mode;
    unsigned char bitsPerWord;
    unsigned int speed;
    int spifd;

    int spiOpen(std::string devspi);
    int spiClose();

};

#endif
```

mcp3008Spi.cpp

```
#include "mcp3008Spi.h"
using namespace std;
/*************************************************************
 * spiOpen() :function is called by the constructor.
 * It is responsible for opening the spidev device
 * "devspi" and then setting up the spidev interface.
 * private member variables are used to configure spidev.
 * They must be set appropriately by constructor before calling
 * this function.
 * ***********************************************************/
int mcp3008Spi::spiOpen(std::string devspi){
    int statusVal = -1;
    this->spifd = open(devspi.c_str(), O_RDWR);
    if(this->spifd < 0){
        perror("could not open SPI device");
        exit(1);
    }

    statusVal = ioctl (this->spifd, SPI_IOC_WR_MODE, &(this->mode));
    if(statusVal < 0){
        perror("Could not set SPIMode (WR)...ioctl fail");
        exit(1);
    }

    statusVal = ioctl (this->spifd, SPI_IOC_RD_MODE, &(this->mode));
    if(statusVal < 0) {
      perror("Could not set SPIMode (RD)...ioctl fail");
      exit(1);
    }

    statusVal = ioctl (this->spifd, SPI_IOC_WR_BITS_PER_WORD, &(this->bitsPerWord));
    if(statusVal < 0) {
      perror("Could not set SPI bitsPerWord (WR)...ioctl fail");
      exit(1);
    }

    statusVal = ioctl (this->spifd, SPI_IOC_RD_BITS_PER_WORD, &(this->bitsPerWord));
    if(statusVal < 0) {
      perror("Could not set SPI bitsPerWord(RD)...ioctl fail");
      exit(1);
    }

    statusVal = ioctl (this->spifd, SPI_IOC_WR_MAX_SPEED_HZ, &(this->speed));
    if(statusVal < 0) {
      perror("Could not set SPI speed (WR)...ioctl fail");
      exit(1);
    }

    statusVal = ioctl (this->spifd, SPI_IOC_RD_MAX_SPEED_HZ, &(this->speed));
    if(statusVal < 0) {
      perror("Could not set SPI speed (RD)...ioctl fail");
      exit(1);
    }
    return statusVal;
}

/*************************************************************
 * spiClose(): Responsible for closing the spidev interface.
 * Called in destructor
 * ***********************************************************/

int mcp3008Spi::spiClose(){
```

```
    int statusVal = -1;
    statusVal = close(this->spifd);
        if(statusVal < 0) {
      perror("Could not close SPI device");
      exit(1);
    }
    return statusVal;
}

/*****************************************************************
 * This function writes data "data" of length "length" to the spidev
 * device. Data shifted in from the spidev device is saved back into
 * "data".
 * ***************************************************************/
int mcp3008Spi::spiWriteRead( unsigned char *data, int length){

  struct spi_ioc_transfer spi[length];
  int i = 0;
  int retVal = -1;
  bzero(spi, sizeof spi); // ioctl struct must be zeroed

// one spi transfer for each byte

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

    spi[i].tx_buf        = (unsigned long)(data + i); // transmit from "data"
    spi[i].rx_buf        = (unsigned long)(data + i) ; // receive into "data"
    spi[i].len           = sizeof(*(data + i)) ;
    spi[i].delay_usecs   = 0 ;
    spi[i].speed_hz      = this->speed ;
    spi[i].bits_per_word = this->bitsPerWord ;
    spi[i].cs_change = 0;
}

 retVal = ioctl (this->spifd, SPI_IOC_MESSAGE(length), &spi) ;

 if(retVal < 0){
    perror("Problem transmitting spi data..ioctl");
    exit(1);
 }

return retVal;

}

/*************************************************
 * Default constructor. Set member variables to
 * default values and then call spiOpen()
 * ***********************************************/

mcp3008Spi::mcp3008Spi(){
    this->mode = SPI_MODE_0 ;
    this->bitsPerWord = 8;
    this->speed = 1000000;
    this->spifd = -1;

    this->spiOpen(std::string("/dev/spidev0.0"));

    }

/*************************************************
 * overloaded constructor. let user set member variables to
```

```
 * and then call spiOpen()
 * ******************************************/
mcp3008Spi::mcp3008Spi(std::string devspi, unsigned char spiMode, unsigned int spiSpeed, unsigned char spib
    this->mode = spiMode ;
    this->bitsPerWord = spibitsPerWord;
    this->speed = spiSpeed;
    this->spifd = -1;

    this->spiOpen(devspi);

}

/*****************************************
 * Destructor: calls spiClose()
 * ***************************************/
mcp3008Spi::~mcp3008Spi(){
    this->spiClose();
}
```

A.4.4 Wireless Link Test Code - Square wave

Transmitter

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#include <iostream>

//sudo g++ -o trans trans.cpp -lwiringPi
//sudo ./trans

using namespace std;

int tmpbin[8];  //store binary bits
int laspin=4;   //laser output pin
int flagpin=5;  //flag pin

//Min Period 7.5e-5 sec 75usec
//1 bit 37.5usec 26.67kbps
float bittime=1000-75;      //1msec
int syncd[8]={1,0};
int initd[8]={1,1,1,1,0,0,0,0};
//costum byte for testing
int test[8]={1,1,1,1,1,1,1,1};

//Convert decimal to binary (input decimal, number of bits, name of int output array)
void dec2bin(unsigned int in, int count, int* out)
{
    unsigned int mask = 1U << (count-1);
    int i;
    for (i = 0; i < count; i++) {
        out[i] = (in & mask) ? 1 : 0;
        in <<= 1;
    }
}
//output data with delay included
```

```
void high(){
    digitalWrite(laspin,HIGH);
    //cout << "1" << endl;
    delayMicroseconds(bittime);
}
void low(){
    digitalWrite(laspin,LOW);
    //cout << "0" << endl;
    delayMicroseconds(bittime);
}
//output a whole size 8 array
void outbyte(int A){
    //every byte start with initialize sequence
    for(int i=0; i<8; i++){
        if(initd[i]==1){
            high();
        }
        if(initd[i]==0){
            low();
        }
    }
    //sync sequence
    for(int n=0; n<2; n++){
        if(syncd[n]==1){
            high();
        }
        if(syncd[n]==0){
            low();
        }
    }

    //output bits
        if(A==1){
            high();
            low();
        }
        if(A==0){
            low();
            high();
        }
}

int main(void){
    wiringPiSetupGpio();
    pinMode(laspin,OUTPUT);
    int testbit=1;

while(1){
    //send out 1010101010101....
    outbyte(testbit);
    testbit=!testbit;
}
}
```

Receiver

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <wiringPi.h>
#include <wiringSerial.h>
```

```
#include <iostream>
#include <algorithm>
#include <iterator>
#include <time.h>
#include <vector>
#include <math.h>
#include "mcp3008Spi.h"

//sudo g++ -Wall -o rec2 mcp3008Spi.cpp rec2.cpp -lwiringPi
//Need to enable SPI
//Vdd -- 5V
//Vref -- 5V
//AGND -- GND
//CLK -- SCLK
//Dout -- MISO
//Din -- MISI
//CS/SHDN -- CSO (GPIO8)
//DGND -- GND
//Currently using CH0, change with data[1]


#define BASE 100
#define SPI_CHAN 0

using namespace std;

int val;
int recbit;
int tempbit;
int syncd[2]={1,0};
int initd[8]={1,1,1,1,0,0,0,0};
int bittime=1000;
int adcmint=50;
int vthtime=10000;
float vth;

bool transfound;
const int framelen=20;
char frame[framelen];

//adc min sample period 50us per bit
//adc read, output logic 1 or 0 with threshold vth
int adcread(float vth){
    int a2dVal = 0;
    int a2dChannel = 0;
    unsigned char data[3];
    mcp3008Spi a2d("/dev/spidev0.0", SPI_MODE_0, 1000000, 8);
    data[0] = 1;  //  first byte transmitted (start bit)
    data[1] = 0b10000000 |( ((a2dChannel & 7) << 4)); // second byte transmitted -> (SGL/DIF = 1, D2=D1=D0=
    data[2] = 0; // third byte transmitted(don't care)
    a2d.spiWriteRead(data, sizeof(data) );

    a2dVal = 0;
    a2dVal = (data[1]<< 8) & 0b1100000000; //merge data[1] & data[2] to get result
    a2dVal |=  (data[2] & 0xff);

    if (a2dVal <= vth){
    tempbit=0;
    }
    else{
    tempbit=1;
    }
```

```
    return tempbit;
}
//return adc read 0 to 1023
int adcreadth(void){
    int a2dVal = 0;
    int a2dChannel = 0;
    unsigned char data[3];
    mcp3008Spi a2d("/dev/spidev0.0", SPI_MODE_0, 1000000, 8);
    data[0] = 1;  //  first byte transmitted (start bit)
    data[1] = 0b10000000 |( ((a2dChannel & 7) << 4)); // second byte transmitted -> (SGL/DIF = 1, D2=D1=D0=
    data[2] = 0; // third byte transmitted(don't care)
    a2d.spiWriteRead(data, sizeof(data) );

    a2dVal = 0;
    a2dVal = (data[1]<< 8) & 0b1100000000; //merge data[1] & data[2] to get result
    a2dVal |=  (data[2] & 0xff);

    return a2dVal;
}
//calculate threshold voltage
float vthcalc(void){
    cout.setf(ios::showpoint);  //display float in cout
    cout << "Calculating Threshold Voltage" << endl;
    float vth=0;
    int vtotal=0;
    int counter=0;
    float tvth=clock();
    float tempvtht;
    while(tempvtht<vthtime){

        vtotal=vtotal+adcreadth();
        counter=counter+1;
        tempvtht=clock()-tvth;
        tempvtht=tempvtht/CLOCKS_PER_SEC;
        tempvtht=tempvtht/(1e-6);
        //cout << tempvtht << endl;
        //cout << counter << endl;
    }
    vth=vtotal/counter;
    float threshold=(vth*5)/1024;
    cout << "Calculation done. Threshold Voltage: " << threshold << " V"<< endl;
    //cout << vth << endl;
    return vth;
}
int main(void){
    //cout.setf(ios::showpoint);    //display float in cout
    wiringPiSetupGpio();        //GPIO setup
    vth=vthcalc();              //Calculate threshold voltage

    while(1){
        //start searching for initialize sequence
        bool start=false;
        while(!start){  //exit loop if number of high > 4 and number of low =4
            if(adcread(vth)==1){
                    float fetchinit=clock();
                    while(adcread(vth)==1){
                    }
                    float fetchtime=clock()-fetchinit;
                    fetchtime=fetchtime/CLOCKS_PER_SEC;
                    fetchtime=fetchtime/(1e-6);
                    int bitcounthigh=round(fetchtime/bittime);
                    if(bitcounthigh>=4){
```

```
                        float fetchinit=clock();
                        while(adcread(vth)==0){
                        }
                        float fetchtime=clock()-fetchinit;
                        fetchtime=fetchtime/CLOCKS_PER_SEC;
                        fetchtime=fetchtime/(1e-6);
                        int bitcountlow=round(fetchtime/bittime);
                        if(bitcountlow==4){
                        start=true;
                        }
                    }

                }
        }
        //found initialize sequence
        //start fetching and decode data

            float transtime=clock();
            int noclock=0;  //fetch clock
            int nowbit=1;   //start bit
            if(adcread(vth)==1){
                while(adcread(vth)==1){}
            }
            if(adcread(vth)==0){
                while(adcread(vth)==0){}
            }
            transtime=((clock()-transtime)/CLOCKS_PER_SEC)/(1e-6);
            int bitcount=round(transtime/bittime);
            if(!(bitcount%2)){  //transition happen
                nowbit=!nowbit;
                //noclock=noclock+2;
                cout << nowbit << endl;
            }
            else{
                //noclock=noclock+1;
                cout << nowbit << endl;
            }
    }
}
```

A.4.5 Wireless Link Test Code - Counting Numbers

Transmitter

```
#include <stdio.h>
#include <unistd.h>     //Used for UART
#include <fcntl.h>      //Used for UART
#include <termios.h>        //Used for UART
#include <wiringPi.h>
#include <wiringSerial.h>
#include <iostream>

//sudo g++ -o trans trans.cpp -lwiringPi
//sudo ./trans

using namespace std;

int tmpbin[8];  //store binary bits
int laspin=4;   //laser output pin
```

```
//Min Period 7.5e-5 sec 75usec
//1 bit 37.5usec 26.67kbps
float bittime=1000-75;      //1msec
int syncd[8]={1,0};
int initd[8]={1,1,1,1,0,0,0,0};
//costum byte for testing
int test[8]={1,0,1,0,1,0,1,0};

//Convert decimal to binary (input decimal, number of bits, name of int output array)
void dec2bin(unsigned int in, int count, int* out)
{
    unsigned int mask = 1U << (count-1);
    int i;
    for (i = 0; i < count; i++) {
        out[i] = (in & mask) ? 1 : 0;
        in <<= 1;
    }
}
void high(){
    digitalWrite(laspin,HIGH);
    delayMicroseconds(bittime);
}
void low(){
    digitalWrite(laspin,LOW);
    delayMicroseconds(bittime);
}
//encode and output size 8 array with initialize and start bits
void outbyte(int* A){

    //every byte start with initialize sequence
    for(int i=0; i<8; i++){
        if(initd[i]==1){
            high();
        }
        if(initd[i]==0){
            low();
        }
    }

    //sync sequence
    for(int n=0; n<2; n++){
        if(syncd[n]==1){
            high();
        }
        if(syncd[n]==0){
            low();
        }
    }

    //output bits
    for(int k=0; k<8; k++){
        if(A[k]==1){
            high();
            low();
        }
        if(A[k]==0){
            low();
            high();
        }
    }
}
```

```
int main(void){
    wiringPiSetupGpio();
    pinMode(laspin,OUTPUT);

while(1){
    //loop count from 0 to 255
    for(int count=0;count<256;count++){
        dec2bin(count,8,tmpbin);
        outbyte(tmpbin);
        cout << count << endl;
    }
    //outbyte(test);
}
}
```

Receiver

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#include <iostream>
#include <algorithm>
#include <iterator>
#include <time.h>
#include <vector>
#include <math.h>
#include "mcp3008Spi.h"

//sudo g++ -Wall -o rec2 mcp3008Spi.cpp rec2.cpp -lwiringPi
//Need to enable SPI
//Vdd -- 5V
//Vref -- 5V
//AGND -- GND
//CLK -- SCLK
//Dout -- MISO
//Din -- MISI
//CS/SHDN -- CSO (GPIO8)
//DGND -- GND
//Currently using CH0, change with data[1]


#define BASE 100
#define SPI_CHAN 0

using namespace std;

int val;
int recbit;
int tempbit;
int syncd[2]={1,0};
int initd[8]={1,1,1,1,0,0,0,0};
int bittime=1000;
int adcmint=50;
int vthtime=100000;
float vth;
int discount=0;


//adc min sample period 50us per bit
```

```
//adc read, return 0 or 1 with threshold vth
int adcread(float vth){
    int a2dVal = 0;
    int a2dChannel = 0;
    unsigned char data[3];
    mcp3008Spi a2d("/dev/spidev0.0", SPI_MODE_0, 1000000, 8);
    data[0] = 1;  //  first byte transmitted (start bit)
    data[1] = 0b10000000 |( ((a2dChannel & 7) << 4)); // second byte transmitted -> (SGL/DIF = 1, D2=D1=D0=
    data[2] = 0; // third byte transmitted(don't care)
    a2d.spiWriteRead(data, sizeof(data) );

    a2dVal = 0;
    a2dVal = (data[1]<< 8) & 0b1100000000; //merge data[1] & data[2] to get result
    a2dVal |=  (data[2] & 0xff);

    if (a2dVal <= vth){
    tempbit=0;
    }
    else{
    tempbit=1;
    }
    return tempbit;
}
//adc read, return 0 to 1023
int adcreadth(void){
    int a2dVal = 0;
    int a2dChannel = 0;
    unsigned char data[3];
    mcp3008Spi a2d("/dev/spidev0.0", SPI_MODE_0, 1000000, 8);
    data[0] = 1;  //  first byte transmitted (start bit)
    data[1] = 0b10000000 |( ((a2dChannel & 7) << 4)); // second byte transmitted -> (SGL/DIF = 1, D2=D1=D0=
    data[2] = 0; // third byte transmitted(don't care)
    a2d.spiWriteRead(data, sizeof(data) );

    a2dVal = 0;
    a2dVal = (data[1]<< 8) & 0b1100000000; //merge data[1] & data[2] to get result
    a2dVal |=  (data[2] & 0xff);

    return a2dVal;
}
//calculate vth
float vthcalc(void){
    cout.setf(ios::showpoint);      //display float in cout
    //cout << "Calculating Threshold Voltage" << endl;
    float vth=0;
    int vtotal=0;
    int counter=0;
    float tvth=clock();
    float tempvtht;
    while(tempvtht<vthtime){

        vtotal=vtotal+adcreadth();
        counter=counter+1;
        tempvtht=clock()-tvth;
        tempvtht=tempvtht/CLOCKS_PER_SEC;
        tempvtht=tempvtht/(1e-6);
        //cout << tempvtht << endl;
        //cout << counter << endl;
    }
    vth=vtotal/counter;
    float threshold=(vth*5)/1024;
    cout << "Current Threshold: " << threshold << " V       ";
```

```
    //cout << vth << endl;
    return vth;
}


int main(void){
    //cout.setf(ios::showpoint);    //display float in cout
    wiringPiSetupGpio();        //GPIO setup
    while(1){

        vth=vthcalc();              //Calculate threshold voltage
        bool start=false;
        while(!start){
            if(adcread(vth)==1){
                    float fetchinit=clock();
                    while(adcread(vth)==1){
                    }
                    float fetchtime=clock()-fetchinit;
                    fetchtime=fetchtime/CLOCKS_PER_SEC;
                    fetchtime=fetchtime/(1e-6);
                    int bitcounthigh=round(fetchtime/bittime);
                    //cout << fetch8time << endl;
                    //cout << "high bit number:" << bitcounthigh<<endl;
                    if(bitcounthigh>=4){
                        float fetchinit=clock();
                        while(adcread(vth)==0){
                        }
                        float fetchtime=clock()-fetchinit;
                        fetchtime=fetchtime/CLOCKS_PER_SEC;
                        fetchtime=fetchtime/(1e-6);
                        int bitcountlow=round(fetchtime/bittime);
                        //cout << fetch8time << endl;
                        //cout << "low bit number:" << bitcountlow<<endl;
                        if(bitcountlow==4){
                        start=true;
                        }
                    }

                }
        }
        //cout << "done" << endl;
        unsigned char rec;
        int nowbit=1;
        int noclock=0;
        do{
            float transtime=clock();
            if(adcread(vth)==1){
                while(adcread(vth)==1){}
                //cout << "high" << endl;
            }
            else{
                while(adcread(vth)==0){}
                //cout << "low" << endl;
            }
            transtime=((clock()-transtime)/CLOCKS_PER_SEC)/(1e-6);
            //cout << transtime << endl;
            int bitcount=round(transtime/(bittime));
            //cout << "bitcount" << bitcount << endl;
            if(!(bitcount%2)){  //transition happen
                nowbit=!nowbit;
                noclock=noclock+bitcount;
                //cout << nowbit << endl;
```

```
            }
            else{
                noclock=noclock+bitcount;
                //cout << nowbit << endl;
            }
            //cout << "noclock;" <<noclock << endl;
            if(!(noclock%2)){
                rec <<= 1;
                rec += nowbit;
                //cout << nowbit << endl;
            }
            else{
                if(bitcount==2){
                    rec <<= 1;
                    rec += nowbit;
                    //cout << nowbit << endl;
                }
            }

        }while(noclock<18);     //loop for the rest of the bits
        rec = rec & 0x0ff;      //apply mask
        std::cout << "Decoded Data: " << (int)rec << std::endl;

    }
}
```

A.4.6 Wireless Link Test Code - One Pixel

Transmitter

```
#include <iostream>
#include <wiringPi.h>
#include "opencv/cv.h"
#include "opencv/highgui.h"
using namespace std;

//g++ `pkg-config opencv --cflags` trans.cpp -o trans `pkg-config opencv --libs` -l wiringPi
//sudo ./trans

const int bittime=1000; //Microsecond pulse times

int sliderPosition =2; // Initial slider position
int divisions = 1;     // Initial number of divisions

int blockXSize;         // Horizontal block size
int blockYSize;         // Vertical block size

int pixelCount;         // The number of pixels in a block (blockXSize multiplied by blockYSize)

int width;              // The width  of the input stream
int height;             // The height of the input stream

int arraX[8]={};    //sets digital out pin
int arraY[8]={};    //sets digital out pin
int arraB[8]={};    //sets digital out pin

const int laspin = 4;   // Regular LED - Broadcom pin 24, P1 pin GPIO 12
const int inPin=24;     //input Pin Broadcom pin 18, gpio pin 24

int transmit=0;
```

```
int syncd[2]={1,0};
int initd[8]={1,1,1,1,0,0,0,0};
const char*  windown="Pixelized Display";
//Converts decimal Integers into 8 bit unsigned binary
void binConv8bit(int blck,int arr[])
{
memset(arr, 0, 8);
//cout <<   "   Decimal = "<< blck<< endl;
int X =blck;
int D = 1;
 for (int i=0 ;i<8; i++  )
    {

    arr[i]=X/(128/D);

    if (arr[i] ==1){X=X-(128/D);}
    D= D*2;
    }

//cout <<    arr[0] << arr[1] <<arr[2] <<arr[3] <<arr[4] <<arr[5] <<arr[6] <<arr[7] <<endl;
}
void high(){
    digitalWrite(laspin,HIGH);
    delayMicroseconds(bittime);
}
void low(){
    digitalWrite(laspin,LOW);
    delayMicroseconds(bittime);
}
//convert decimal to binary
void dec2bin(unsigned int in, int count, int* out)
{
    unsigned int mask = 1U << (count-1);
    int i;
    for (i = 0; i < count; i++) {
        out[i] = (in & mask) ? 1 : 0;
        in <<= 1;
    }
}

//output binary of from array
void outbyte(int* A){

    //every byte start with initialize sequence
    cout << "*****************************************" << endl;
    cout << "Initial Stream: ";
    for(int i=0; i<8; i++){

        if(initd[i]==1){
            high();
            cout << "1";
        }
        if(initd[i]==0){
            low();
            cout << "0";
        }
    }cout<< "" << endl;

    //sync sequence
    cout << "Start Stream: ";
    for(int n=0; n<2; n++){
        if(syncd[n]==1){
```

```
            high();
            cout << "1";
        }
        if(syncd[n]==0){
            low();
            cout << "0" << endl;
        }
    }cout<< "" << endl;

    //output bits
    cout << "Brightness Stream: ";
    for(int k=0; k<8; k++){
        if(A[k]==1){
            high();
            cout << "1";
            low();
            cout << "0";
        }
        if(A[k]==0){
            low();
            cout << "0";
            high();
            cout << "1";
        }
    }cout<< "" << endl;
    digitalWrite(laspin,LOW);   //done sending, then stay low, don't float
}

int main()
{
    wiringPiSetupGpio();
    pinMode(laspin, OUTPUT);
    // lock pin is input
    pinMode(inPin,INPUT);

    // Create two windows
    //cvNamedWindow("WebCam", CV_WINDOW_AUTOSIZE);
    cvNamedWindow(windown, CV_WINDOW_AUTOSIZE);

    //Full Screen setting
    //cvSetWindowProperty("Low Rez Stream",CV_WND_PROP_FULLSCREEN, CV_WINDOW_FULLSCREEN);

    //int maxSliderValue = 11;

    // Create the divisions slider lider
    //cvCreateTrackbar("Divisions", windown, &sliderPosition, maxSliderValue, onDivisionSlide);

    // Start capturing data from the web cam
    CvCapture* pCapture = cvCaptureFromCAM(-1);

    // Reduce framerate from default
    //cvSetCaptureProperty( pCapture, CV_CAP_PROP_FPS , 5 );

    //Set stream resolution
    //cvSetCaptureProperty( pCapture, CV_CAP_PROP_FRAME_WIDTH , 320);
    //cvSetCaptureProperty( pCapture, CV_CAP_PROP_FRAME_HEIGHT , 240 );

    // Get an initial frame so we know the size of things (cvQueryFrame is a combination of cvGrabFrame and
    IplImage* pFrame = NULL;
    pFrame = cvQueryFrame(pCapture);

    // Create an image the same size and colour-depth as our input stream
```

```
    IplImage* pLowRezFrame = cvCreateImage(cvSize(pFrame->width, pFrame->height), IPL_DEPTH_8U, 3);

    uchar *ptr; // Pointer to our pixel

    int red, green, blue; // Integers to hold our pixel values

    // Get the width and height of our webcam input stream
    int width  = pFrame->width;
    int height = pFrame->height;

    // Integers to hold our total colour values (used to find the average)
    int redSum     = 0;
    int greenSum   = 0;
    int blueSum    = 0;

    // Loop controling vars
    char keypress;
    bool quit = false;
    //Loop Output Coordinates
    int XX=0;
    int YY=0;
    while (quit == false)
    {

    //reads pixels if transmit pin is high
    //Jumper from 3.3V to start
    if (digitalRead(inPin)==1)
{
        // Grab a frame from the webcam
        pFrame = cvQueryFrame(pCapture);

        // Draw the original frame and low resolution version
        //cvShowImage("WebCam", pFrame);
        cvShowImage(windown, pLowRezFrame);

        // Calculate our blocksize per frame to cater for slider
        blockXSize = width  / divisions;
        blockYSize = height / divisions;

        pixelCount = blockXSize * blockYSize; // How many pixels we'll read per block - used to find the av

        // Draw a rectangle of Black

            cvRectangle(
                    pLowRezFrame,
                    cvPoint(0,0),
                    cvPoint(640,480),
                    CV_RGB(0, 0, 0),
                    CV_FILLED,
                    8,
                    0
                );

        // Loop through each block horizontally
        for (int xLoop = 0; xLoop < width; xLoop += blockXSize)
        {
            // Loop through each block vertically
            for (int yLoop = 0; yLoop < height; yLoop += blockYSize)
            {
                // Reset our colour counters for each block
                redSum   =greenSum  = blueSum = 0;
                // Read every pixel in the block and calculate the average colour
```

```
                for (int pixXLoop = 0; pixXLoop < blockXSize; pixXLoop++)
                {
                    for (int pixYLoop = 0; pixYLoop < blockYSize; pixYLoop++)
                    {
                        // Get the pixel colour from the webcam stream
                        ptr = cvPtr2D(pFrame, yLoop + pixYLoop, xLoop + pixXLoop, NULL);
                        // Add each component to its sum
                        redSum   += ((ptr[2]+ptr[1]+ptr[0])/3);

                    } // End of inner y pixel counting loop
                } // End of outer x pixel counter loop
                // Calculate the average brightness of the block
                red =green=blue =redSum   / pixelCount;
                //cout << redSum << endl;
                //cout << pixelCount << endl;
                //cout << red << endl;

        //convert pixelblocks into binary array
        binConv8bit(xLoop/blockXSize,arraX);
        binConv8bit(yLoop/blockYSize,arraY);
        binConv8bit(red,arraB);

        cout << "Brightness: " << red << endl;
        cout << "*****************************************" << endl;
        //output binary array
        outbyte(arraB);
        //draw circle
        cvCircle(
                pLowRezFrame,
        cvPoint(
                xLoop + ((blockXSize)/2), yLoop + ((blockYSize)/2)
                ),
            //Radius
            ((blockYSize)/2),
            //Colour
            CV_RGB(red, green, blue),
                    CV_FILLED,
                    8,    0
                );
            } // End of inner y loop
        } // End of outer x loop
} // end of transmit
        // Wait 1 millisecond
        keypress = cvWaitKey(1);
        // Set the flag to quit if the key pressed was escape
        if (keypress == 27)
        {
           quit = true;
        }
    } // End of while loop
    // Release our stream capture object
    cvReleaseCapture(&pCapture);
    // Release our images & destroy all windows
    cvReleaseImage(&pFrame);
    cvReleaseImage(&pLowRezFrame);
    cvDestroyAllWindows();
}
```

Receiver

```
#include <stdio.h>
#include <unistd.h>
```

```
#include <fcntl.h>
#include <termios.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#include <iostream>
#include <algorithm>
#include <iterator>
#include <time.h>
#include <vector>
#include <math.h>
#include "mcp3008Spi.h"
#include "opencv/cv.h"
#include "opencv/highgui.h"

//sudo g++ -Wall -o rec2 `pkg-config opencv --cflags` mcp3008Spi.cpp `pkg-config opencv --libs` rec2.cpp -l
//Need to enable SPI
//Vdd -- 5V
//Vref -- 5V
//AGND -- GND
//CLK -- SCLK
//Dout -- MISO
//Din -- MISI
//CS/SHDN -- CSO (GPIO8)
//DGND -- GND
//Currently using CH0, change with data[1]

#define BASE 100
#define SPI_CHAN 0

using namespace std;
using namespace cv;

int val;
int recbit;
int tempbit;
int syncd[2]={1,0};
int initd[8]={1,1,1,1,0,0,0,0};
int bittime=1000;
int adcmint=50;
int vthtime=100000;
float vth;
int discount=0;
const int length=3;
unsigned char frame[length];

//adc min sample period 50us per bit
//read adc value, return logic high low for threshold vth
int adcread(float vth){
    int a2dVal = 0;
    int a2dChannel = 0;
    unsigned char data[3];
    mcp3008Spi a2d("/dev/spidev0.0", SPI_MODE_0, 1000000, 8);
    data[0] = 1;  //  first byte transmitted (start bit)
    data[1] = 0b10000000 |( ((a2dChannel & 7) << 4)); // second byte transmitted -> (SGL/DIF = 1, D2=D1=D0=
    data[2] = 0; // third byte transmitted(don't care)
    a2d.spiWriteRead(data, sizeof(data) );

    a2dVal = 0;
    a2dVal = (data[1]<< 8) & 0b1100000000; //merge data[1] & data[2] to get result
    a2dVal |=  (data[2] & 0xff);

    if (a2dVal <= vth){
```

```
    tempbit=0;
    }
    else{
    tempbit=1;
    }
    return tempbit;
}
//read adc return adc value 0 to 1023
int adcreadth(void){
    int a2dVal = 0;
    int a2dChannel = 0;
    unsigned char data[3];
    mcp3008Spi a2d("/dev/spidev0.0", SPI_MODE_0, 1000000, 8);
    data[0] = 1;  //  first byte transmitted (start bit)
    data[1] = 0b10000000 |( ((a2dChannel & 7) << 4)); // second byte transmitted -> (SGL/DIF = 1, D2=D1=D0=
    data[2] = 0; // third byte transmitted(don't care)
    a2d.spiWriteRead(data, sizeof(data) );

    a2dVal = 0;
    a2dVal = (data[1]<< 8) & 0b1100000000; //merge data[1] & data[2] to get result
    a2dVal |=  (data[2] & 0xff);

    return a2dVal;
}
//calculate threshold voltage
float vthcalc(void){
    cout.setf(ios::showpoint);
    //cout << "Calculating Threshold Voltage" << endl;
    float vth=0;
    int vtotal=0;
    int counter=0;
    float tvth=clock();
    float tempvtht;
    while(tempvtht<vthtime){

        vtotal=vtotal+adcreadth();
        counter=counter+1;
        tempvtht=clock()-tvth;
        tempvtht=tempvtht/CLOCKS_PER_SEC;
        tempvtht=tempvtht/(1e-6);
        //cout << tempvtht << endl;
        //cout << counter << endl;
    }
    vth=vtotal/counter;
    float threshold=(vth*5)/1024;
    cout << "Current Threshold: " << threshold << " V" << endl;
    //cout << vth << endl;
    return vth;
}


int main(void){
    //cout.setf(ios::showpoint);    //display float in cout
    wiringPiSetupGpio();        //GPIO setup
    //create window to show image
    cvNamedWindow("Display", CV_WINDOW_AUTOSIZE);
    //create empty image of 640x480
    IplImage* image = cvCreateImage(cvSize(640,480), IPL_DEPTH_8U, 3);
    char keypress;
    bool quit = false;
    while(!quit){
    while(1){
```

```
//display current frame
cvShowImage("Display", image);
//wait to finish drawing
char c = (char)cvWaitKey(10);
if( c == 27 ) { break; } // escape
vth=vthcalc();              //Calculate threshold voltage
bool start=false;
//loop until initialize sequence found
while(!start){
    if(adcread(vth)==1){
            float fetchinit=clock();
            while(adcread(vth)==1){
            }
            float fetchtime=clock()-fetchinit;
            fetchtime=fetchtime/CLOCKS_PER_SEC;
            fetchtime=fetchtime/(1e-6);
            int bitcounthigh=round(fetchtime/bittime);
            //cout << fetch8time << endl;
            //cout << "high bit number:" << bitcounthigh<<endl;
            if(bitcounthigh>=4){
                float fetchinit=clock();
                while(adcread(vth)==0){
                }
                float fetchtime=clock()-fetchinit;
                fetchtime=fetchtime/CLOCKS_PER_SEC;
                fetchtime=fetchtime/(1e-6);
                int bitcountlow=round(fetchtime/bittime);
                //cout << fetch8time << endl;
                //cout << "low bit number:" << bitcountlow<<endl;
                if(bitcountlow==4){
                start=true; //found initialize sequence
                }
            }
        }
}
//cout << "done" << endl;
unsigned char rec;
int nowbit=1;
int noclock=0;
//define how many bytes the data has
//int framecount=0;
//do{
    do{
        float transtime=clock();
        if(adcread(vth)==1){
            while(adcread(vth)==1){}
            //cout << "high" << endl;
        }
        else{
            while(adcread(vth)==0){}
            //cout << "low" << endl;
        }
        transtime=((clock()-transtime)/CLOCKS_PER_SEC)/(1e-6);
        //cout << transtime << endl;
        int bitcount=round(transtime/(bittime));
        //cout << "bitcount" << bitcount << endl;
        if(!(bitcount%2)){  //transition happen
            nowbit=!nowbit;
            noclock=noclock+bitcount;
            //cout << nowbit << endl;
        }
        else{
```

```
                noclock=noclock+bitcount;
                //cout << nowbit << endl;
            }
            //cout << "noclock;" <<noclock << endl;
            if(!(noclock%2)){
                rec <<= 1;
                rec += nowbit;
                //cout << nowbit << endl;
            }
            else{
                if(bitcount==2){
                    rec <<= 1;
                    rec += nowbit;
                    //cout << nowbit << endl;
                }
            }

        }while(noclock<18);
        rec = rec & 0x0ff;
        std::cout << "Decoded Data: " << (int)rec <<std::endl;
        //frame[framecount]=rec;    //save current frame data
        //framecount=framecount+1;  //increment frame count
    //}while(framecount < length);  //loop until all bytes fetched
    //std::cout << ""<<std::endl;
    int width=640;
    int height=480;
    //draw a black rectangle for background
    cvRectangle(
                image,
                cvPoint(0,0),
                cvPoint(640,480),
                CV_RGB(0, 0, 0),
                CV_FILLED,
                8,
                0
            );
    int red, green, blue;
    int xLoop=0;
    int yLoop=0;
    int blockXSize=width;
    int blockYSize=height;
    int color;
    //turn char into int
    color=reinterpret_cast<unsigned char&>(rec);
    //grayscale red=blue=green
    red=color;
    green=color;
    blue=color;
    cout << "Brightness: " << color << endl;
    //draw circle with red=blue=green=brightness
    cvCircle(
            image,
            //Centre
            cvPoint(xLoop + ((blockXSize)/2), yLoop + ((blockYSize)/2)),
            //Radius
            ((blockYSize)/2),
            //Colour
            CV_RGB(red, green, blue),
            CV_FILLED,
            8,0);
}
//wait to finish drawing
```

```
    keypress = cvWaitKey(0);
    if (keypress == 27)
        {
           quit = true;
        }
    }
    //release the image array
    cvReleaseImage(&image);
    //close all windows
    cvDestroyAllWindows();
}
```

A.5 Camera Calibration (Tony)

```
% Define images to process
imageFileNames = {...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\1.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\2.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\3.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\5.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\6.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\9.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\10.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\11.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\12.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\16.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\19.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\20.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\21.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\23.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\25.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\29.jpg',...
    'C:\Tony\Data\U of O\Electrical Engineering\2016-2017winter\ELG4913\Matlab\Pic\30.jpg',...
    };

% Detect checkerboards in images
[imagePoints, boardSize, imagesUsed] = detectCheckerboardPoints(imageFileNames);
imageFileNames = imageFileNames(imagesUsed);

% Generate world coordinates of the corners of the squares
squareSize = 1.964000e+01;  % in units of 'mm'
worldPoints = generateCheckerboardPoints(boardSize, squareSize);

% Calibrate the camera
cameraParams = estimateCameraParameters(imagePoints, worldPoints, ...
    'EstimateSkew', false, 'EstimateTangentialDistortion', false, ...
    'NumRadialDistortionCoefficients', 2, 'WorldUnits', 'mm');

% View reprojection errors
h1=figure; showReprojectionErrors(cameraParams, 'BarGraph');

% Visualize pattern locations
h2=figure; showExtrinsics(cameraParams, 'CameraCentric');

% For example, you can use the calibration data to remove effects of lens distortion.
originalImage = imread(imageFileNames{1});
undistortedImage = undistortImage(originalImage, cameraParams);
```

A.5.1 Reprojection

```
clear all;close all;clc

%function r = reprojection(p1,p2)
k1=-5;
k2=-5;

%Intrinsic Matrix with unit mm
K_temp = [2.454484381983271e+03, 0, 0;-0.381085258365849,2.454270891896557e+03,0;...
    1.351743018547775e+03, 9.732412832576541e+02, 1];
K =K_temp';
B = (K')^(-1)*K^(-1);
lambda = B(3,3)-((B(1,3))^2-K(2,3)*(B(1,2)*B(1,3)-B(1,1)*B(2,3)))/B(1,1);

r = lambda*(K)^(-1)*[k1;k2;1]
%end
```

References

[1] Faculty of Engineering. *MECH 463D2 Design 3: Mechanical Engineering Project: Dynamic System*, Mcgill University, 2017.
https://www.mcgill.ca/study/2016−2017/courses/mech−463d1

[2] M.H. Maghami, A.M. Sodagar, A. Lashay, H. Riazi-Esfahani, M. Riazi-Esfahani. *Visual Prostheses: The Enabling Technology to Give Sight to the Blind. Journal of Ophthalmic & Vision Research*, 9(4): 494-505, 2014.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4329712/

[3] Second Sight. *Argus®II Retinal Prosthesis System*, 2016.
http://www.secondsight.com/argus-ii-rps-pr-en.html

[4] WHO. *Visual impairment and blindness*, 2014.
http://http://www.who.int/mediacentre/factsheets/fs282/en/

[5] S. Garg. *Retinal Prostheses Offer Hope to Blind Patients*, Review of Ophthalmology, 2013.
https://www.reviewofophthalmology.com/article/retinal-prostheses-offer-hope-to-blind-patients

[6] AT. Chuang, CE. Margo, PB. Greenberg. *Retinal implants: a systematic review*, The British Journal of Ophthalmology, 98 (7): 8526, 2014.

[7] S.O. Kasap. *Optoelectronics and Photonics: Principles and Practices*, Second Edition, Pearson, 2013.

[8] Microchip. *MCP3004/3008 2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface*, 2017.

[9] Magick++. *ImageMagick Magick++ API*, 2017.
http://www.imagemagick.org/Magick++/

[10] Userland. *RaspiFastCamD*. Bitbucket, 2017.
https://bitbucket.org/niklas_rother/rasperry-pi-userland/raw/master/host_applications/linux/apps/raspicam/raspifastcamd_scripts

[11] Hertaville. *Interfacing an SPI ADC (MCP3008) chip to the Raspberry Pi using C++ (spidev)* , Hertaville.com, 2017.
http://www.hertaville.com/interfacing-an-spi-adc-mcp3008-chip-to-the-raspberry-pi-using-c.html

[12] Marty. *The Pi & I: Manchester Encoding/Decoding Data Between Devices*, The Piandi, 2017.
http://thepiandi.blogspot.ca/2015/03/manchester-encodingdecoding-data.html

[13] M. Rouse. *Manchester encoding*, WhatIs.com, 2005.
http://searchnetworking.techtarget.com/definition/Manchester-encoding

[14] Marty. *Manchester Encoding/Decoding Data Between Devices*, 2015.
http://thepiandi.blogspot.ca/2015/03/manchester-encodingdecoding-data.html

[15] A. Tanenbaum. *Computer Networks*, 4th Edition, Prentice Hall, ISBN 0-13-066102-3, 2002.

[16] s1a2d3. *Parallax (Futaba) servo motor 3-axis mount*, Thingiverse, 2015.
http://www.thingiverse.com/thing:829556/#files

[17] iBIONICS. *The iBIONICS Diamond Eye*. Products, 2016.
http://ibionics.ca/products/

[18] Y. Morvan. *Pinhole camera model*, Expixea, 2015.
http://www.epixea.com/research/multi-view-coding-thesisse8.html

[19] mirrorcle Technologies, Inc. *MEMs Mirros*, mirrorcle Technologies, Inc, 2017.

http://mirrorcletech.com/devices.html

[20] OpenCV. *Cascade Classifier*, OpenCV 2.4.13.2 documentation, 2017.
http://docs.opencv.org/2.4/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html

[21] K. Hounslow. *objectTrackingTutorial.cpp*, github, 2013.
https://raw.githubusercontent.com/kylehounslow/opencv-tuts
/master/object-tracking-tut/objectTrackingTut.cpp

[22] S. Hameed. *haarcascade_eye.xml* , github, 2013.
https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_eye.xml