uOttawa

Tracking System
&
Wireless Transmission

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

February 17, 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 The project is delayed due to the late arrival of the MEMs mirror, which for the time being have been replaced by a temporary mirror system.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

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

1.2.1 Primary Goal

Tracking a target that rotates along the surface of a sphere within the range of $\pm 30°$ 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

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

The project consists of two main portion. One portion is the tracking system, which consists of tracking the target through a camera and process the information into usable data to drive other components onto the right location. The second portion is the wireless communication system. This system consists of capturing the image that is to be send, and process it into binary data through a laser. The the receiver side will capture the data from the laser and process the information back to image and display the result.

The most important element in the tracking system is the tracking algorithm. There are several ways to track an object such as Haar Cascade, MeanShift and CAMshift. MeanShift algorithm calculates the center point of the current tracking window and also the center point of the circular area where the highest sample density lies and move the current window to the new location with maximum density. The detail of the MeanShift can be found in Appendix [1] Section 2.1.2.2. The CAMshift stands for "Continuously Adaptive MeanShift", which gives more flexibility to the standard MeanShift. The standard MeanShift consists of fixed size of frames, which can possibly cause many issues. The CAMshift takes the size, shape and orientation into account while tracking, which should be more reliable then MeanShift. The detail of CAMshift can be found in Appendix [1] Section 2.1.2.3. The Haar Cascade tracking algorithm uses the OpenCV library to track and is cross-compatible across various platform and programming languages such as C++, Java and Python. This tracking method uses the samples of target from a pre-calculated database to cross match with the target. The database consists of the target's positive and negative samples under various condition, which will define what the target "looks" like to track it. The detail of Haar Cascade can be found in Section 3.6 and in Appendix [1] Section 2.1.2.4.

Another important element is our system is the wireless transmission system, where we are transmitting data through a laser. The data will be encode with Manchester Coding Method. The Manchester encoding method uses the one cycle of clock pulse to process one bit of data, hence will double the bit size. The method process the data with clock through the logic "XOR". The advantage of using Manchester to encode and decode our data before transmitting is that the signal synchronizes itself by being directly proportional to the clock rate; hence will helps clock recovery on the receiver and minimizes the error rate. Since the encoded signal synchronizes with the clock rate, it will be easier to decode by setting the same clock rate. The disadvantages of Manchester Coding is that the signal requires double of bits than the original signal to transmit; hence also double the original signal's bandwidth. The detail of Manchester Coding can be found in Appendix [1] Section 2.2.

1.5 System Block Diagram

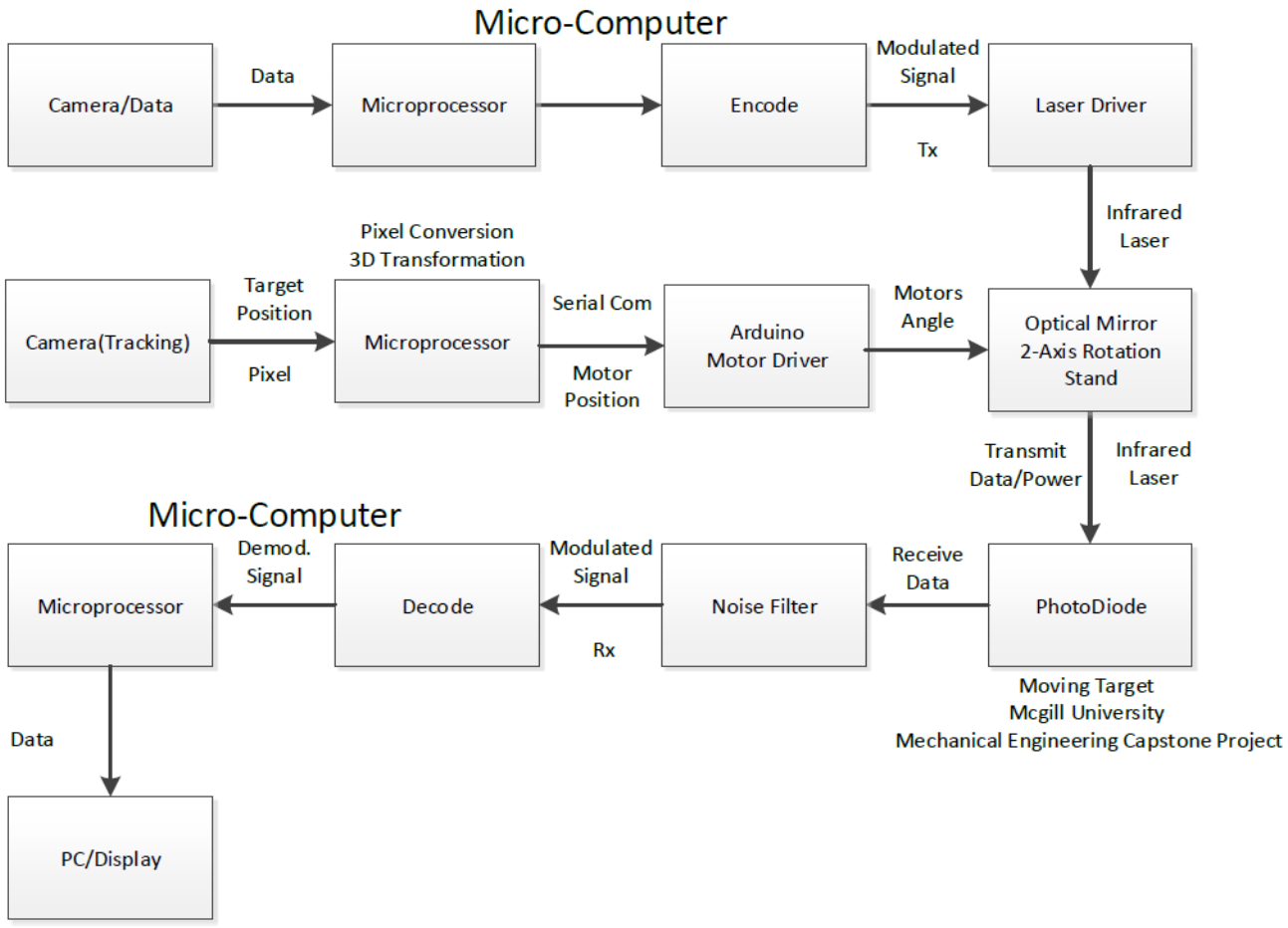


Figure 1: System Block Diagram

From the system block diagram in Figure 1, 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 transmit through a laser driver that ensured the power emitted from the 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 [2][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 Design

2.1 System Design

2.1.1 Physical Design

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 1 below will be used as a guideline for demonstrating the end project.

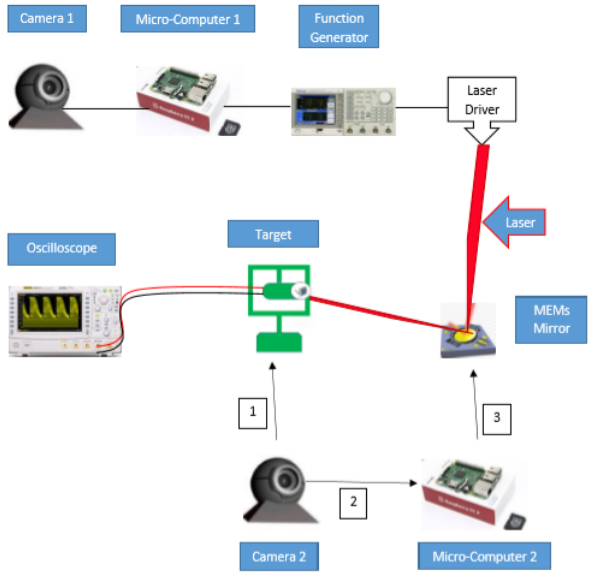


Figure 2: Physical Design for Project Demonstration

As shown in Figure 2, Camera 1 is designated to capture the images and videos 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 drive the function generator which powers 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 MEMs 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 to display the received and converted signal.

Since the target will be rotating $\pm$ 30° in any direction from its initial position, tracking of the target is necessary to ensure that the laser beam reflected off of the MEMs 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 MEMs mirror to accordingly adjust the mirror's rotation angle. Using the coordinates obtained from the target's image, which are to be converted to 3-D coordinates using the camera calibration (Section 3.4), the angle of the MEMs mirror will be adjusted according to the coordinate system (Section 3.1) 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 3, is not possible. Figure 3 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 2, the two cameras, the laser diode, MEMs mirror, and microprocessors, the design shown in Figure 3 should yield similar results to those achieved by the system shown in Figure 2. The figure below shows different angles of a well-integrated system that has the same functionality as that shown in Figure 3, but with a sleeker look and a more practical design. Designed to be a mobile system, the prototype shown in Figure 3 is more practical as it is very compact.

Figure 3: The Physical Design of the Ideal System [7]

2.1.2 Tracking System Design

2.1.2.1 Circuit Design

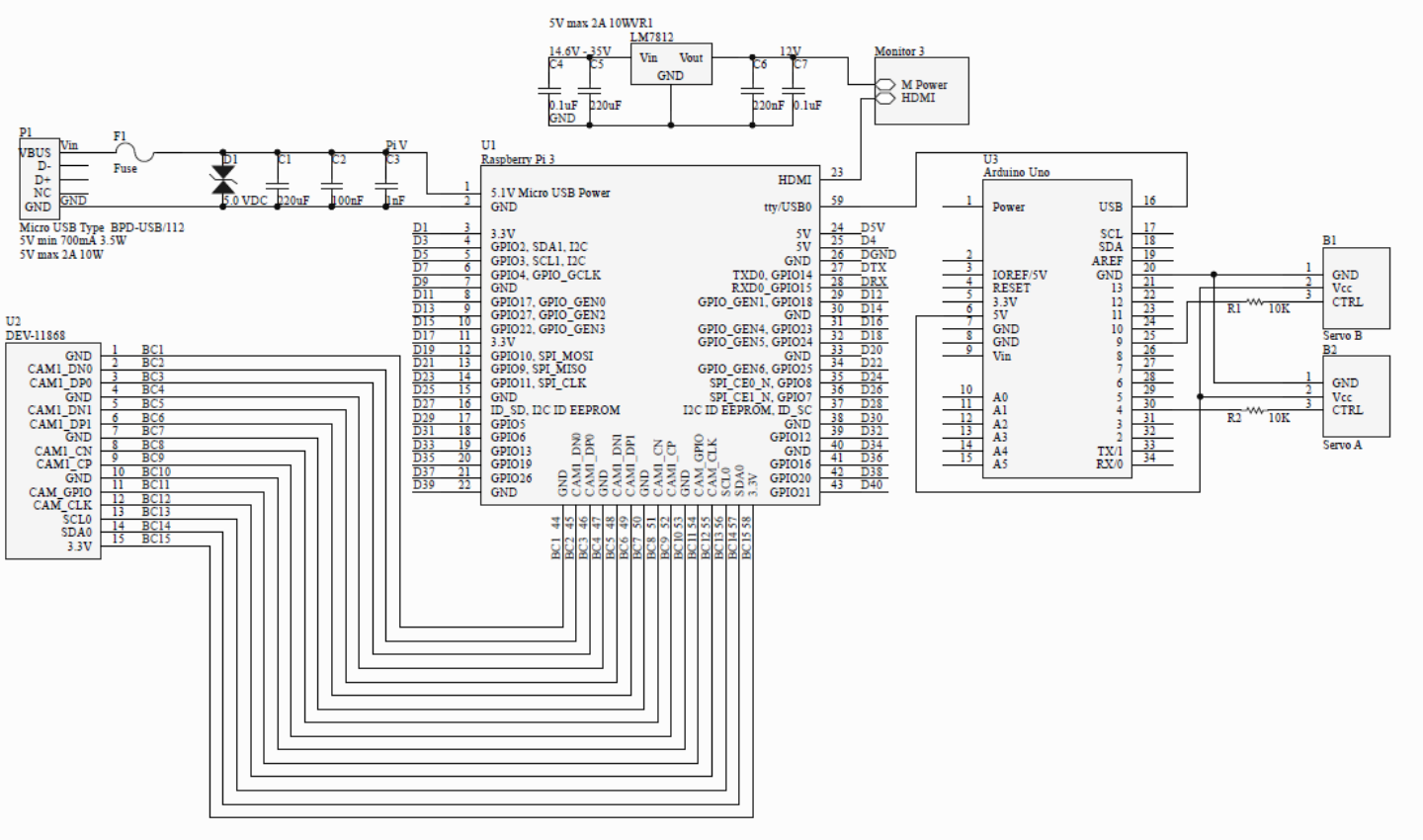


Figure 4: Tracking System Circuit Design

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 4. 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 2.1.2.2). The second part is the tracking algorithm development, which will be running on the Raspberry Pi 3 along with the Raspicam (Section 3.5). 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 (Section 3.4).

2.1.2.2 2-Axis Rotation Stand with Optical Mirror

Table 1: List of Servo Motors and Specifications

Model	Speed	Rotational Range	Accepted PWM Frequency
Futaba S148	0.22 sec/60°	180°	50Hz
Tower Pro^{TM} Micro Servo SG90	0.1 sec/60°	180°	50Hz
GOTECK GS-9025MG	0.12 sec/60°	180°	50Hz

In this part of the system, we will be using the Futaba S148 as the base (y-axis) rotation motor and Tower Pro^{TM} SG90 as the x-axis rotation motor. Since the servo motors are standard controlled motor, we can control them with the same algorithm and method. The stand of the motors and optical mirror will be 3D printed and mention in Section 3

In the Arduino there is a library (Servo.h) that specifically controls the standard servo motors with 50Hz PWM signal. The user needs to define a name for the servo and assign a pin on Arduino for sending control signal. Then the user will need to calculate and specify the angle of rotation for each motor. As the angle gets higher, it sends out PWM with wider high time, hence output more current which will drive the motor to higher degrees. Similarly, if angle of rotation gets smaller, the high time of the PWM becomes narrower and output less current, which will drive the motor to smaller degrees. Theoretically, since all three motors have different speed, one will have to ensure that all motors are in position for the next command; but since we are doing active tracking and the delays we make will cause accumulation of errors, we do not need to wait for all motors to be in position for the next command. In other words, drive the motors to desired position regardless its current location.

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

(a) Servo.h library and defining Servos' name

```
A.attach(4);  //Motor A at pin 4
B.attach(9);  //MOtor B at pin 9
```

(b) Assigning pins to control each Servo

```
A.write(Apos2); //rotate motor
B.write(Bpos2);
```

(c) Drive the Servos into position

Figure 5: Using Servo.h in Arduino

Once the Motor Driver Algorithm is complete, we need to let the Arduino accept data from the Raspberry Pi 3 through Serial Port. Since 180 is smaller than 255, the motor position can be send in one byte per motor. So the Serial Communication Algorithm need to at least accept 2 bytes in an array. So Arduino needs wait for 2 bytes in the buffer, then save the data into an array of size 2. After successfully accepting the data, user need to extract the data into usable data, hence separate the the array of bytes according the to order of motors and convert the bytes into decimal motor angles.

```
Serial.begin(9600); //start serial com
```

(a) To start a Serial Communication at 9600 Baud Rate

```
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);
```

(b) Accepting 2 bytes at a time into an array and convert into decimal

Figure 6: Using Serial Communication in Arduino

To make the code more robust and easier to use, we create a function that can convert binary (byte) into decimal which is shown in Figure 7a.

```
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;
}
```

(a) Function to Convert Binary to Decimal

```
#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;
}
```

(b) SerialTest.cpp

Figure 7: Serial Communication Algorithm

A complete Arduino Code is in Appendix A.2.1.

The last part of the portion of the tracking system is sending data from the Raspberry Pi 3 through USB Serial Port. Library done by danjperrron [3] gives the ability to send data with an array at once. Once we include the library mySerial.h and mySerial.cpp in Section A.2.2 we can send data through serial communication. Before sending the data, we need to check which device the Arduino is currently on. To do so we can simply run "ls /dev/" on the Raspyberry Pi terminal and see which USB device is active. It can be either tty/USB0 or ttyUSB1 and change the device name of the SerialTest.cpp. Since in the previous part, we set the Arduino's Serial Baud Rate at 9600, here we need to set a matching Baud Rate of 9600 in order to communication properly.

In Figure 7b, the number 180 and 90 corresponds to 180 degrees for Servo A and 90 degrees for Servo B. Once the codes are settled, we can generate executable file by running "sudo g++ -o SerialTest SerialTest.cpp mySerial.cpp" in the Raspberry Pi terminal, which will generate executable file name "SerialTest". Then we can run the file by "./SerialTest". Here we did not clear the serial buffer, because it is be done in the Arduino Code using "Serial.flush()" shown in Appendix A.2.1.

2.1.3 Dynamic System Design [2]

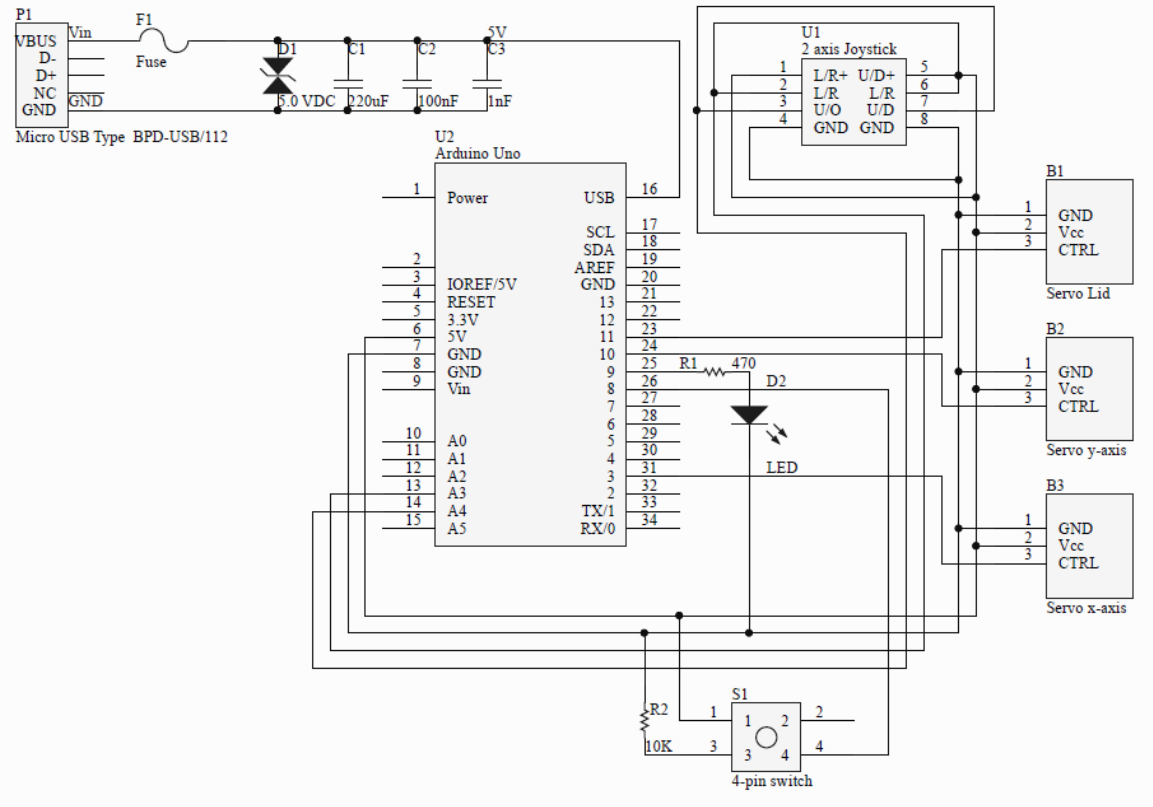


Figure 8: Dynamic System Circuit Design [2][2]

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.

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

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

3 Implementation

3.1 Experimental Setup Coordinates to Determine the MEMs Mirror's Angle of Rotation

To correctly implement the physical setup of the project, the components need to be aligned a certain way. A coordinate system shown in Figure 9, is designed to determine the rotation angle of the MEMs (Microelectromechanical systems) mirror from the location of the other components used in the experimental setup. In particular, the location of the laser diode, the target, as well as the camera are used to calculate the rotation angles of the MEMs mirror. When tracking the target, images will be produced by the camera that will show the location of the target in the image. The coordinates produced by this image will relate to the coordinates of the target in the real world. Hence, when the image is produced showing the coordinates of the target, the real coordinates of the target will be computed to provide the X, Y, and Z components of the location of the target.

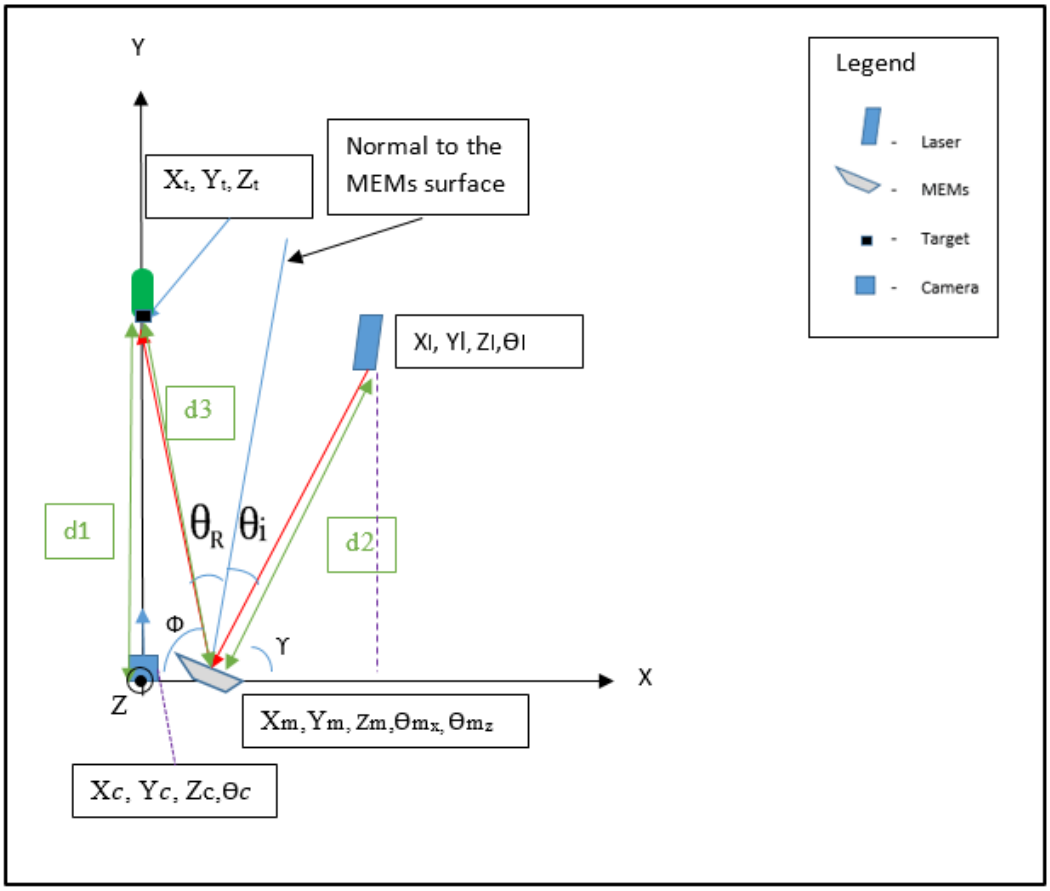


Figure 9: Coordinate System to Determine the MEMs Mirror's Angle of Rotation

Based on the design shown in Figure 9, the rotation angle of the MEMs mirror was computed to be:

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

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

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 9, the MEMs location in the coordinate system can be given by the coordinates $X_m, Y_m, Z_m, \theta_{mz}$, and θ_{mx}. The location of the MEMs mirror will be constant, meaning that X_m, Y_m, and Z_m will remain constant, however, the angles of rotation mz and θ_{mx}, will vary. θmz is the angle which represent the MEMs 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 MEMs mirror surface, θ_i, is shown in Figure 9. Also shown in Figure 9 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 [6].

To compute the necessary angles, certain components in the experimental setup are kept at a fixed position, while the target and the MEMs mirror vary in position and rotation angle. To decide on the placement of all the necessary components of the setup, a measurement was taken of the power delivered by the laser diode from various location. To optimize the design of the system, the location of the laser diode, the MEMs mirror, as well as the target was chosen based on the highest power recorded by the optical power meter. The power meter was placed in the target's position to record the power delivered to that position by the laser beam. Trying to keep the distance between the components small to mimic the ideal design of the system, but at the same time ensuring a functioning system, the location of the components was carefully chosen to yield the highest power measured.

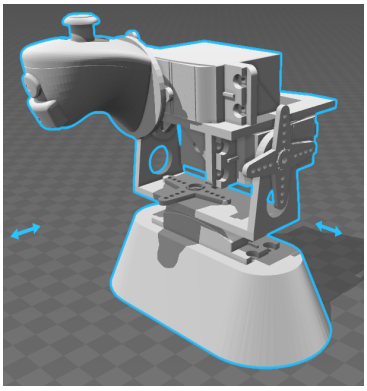
Due to the presence of other light available in the room, various measurements were taken of the light noise power in the room. Those measurements are tabulated in Table 2 below. The average of the light noise power in the room was found to be **13.31 μW**. Before discussing the average received power transmitted by the laser diode, it is important to note the electric power received by the laser diode. To keep the project as safe as possible, 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 d_1 from Figure 9 was measured to be 9.5 cm, while d_3 was measured to be 10.5 cm, and d_2 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 2.

Table 2: Physical Implementation Results

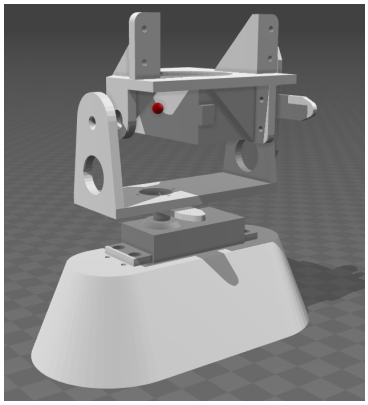
Optical Power Measured:		
	Measured Light Noise Power (μW)	**Measured Reflected Light Power (mW)**
1	14.3	65.5
2	14.8	68.3
3	16.3	67.4
4	12.7	67.8
5	10.5	67.7
6	12.6	64.6
7	13.5	61.7
8	14.1	62.7
9	11.8	61.9
10	12.5	62.9
Average	**13.31**	**65.05**
Source Specifications:		
Applied Current (DC Power Supply)		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 9)		
d_1	9.5 cm	
d_2	8.5 cm	
d_3	10.5 cm	
X_m	4.8 cm	

3.2 Replacement of MEMs Mirror/2-Axis Rotation Stand with Optical Mirror

The 3D model of the 2 axis rotational stand is modified with s1a2d3's model [4]. His original model, shown in Figure 10a 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.

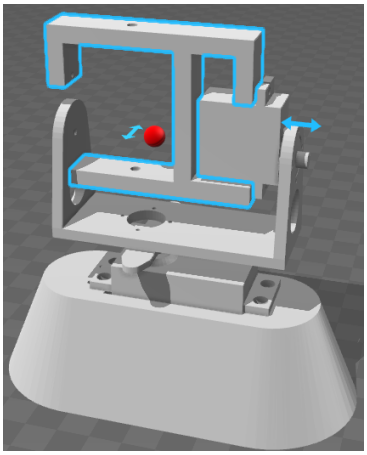


(a) s1a2d3 Parallax (Futaba) servo motor 3-axis mount [4]

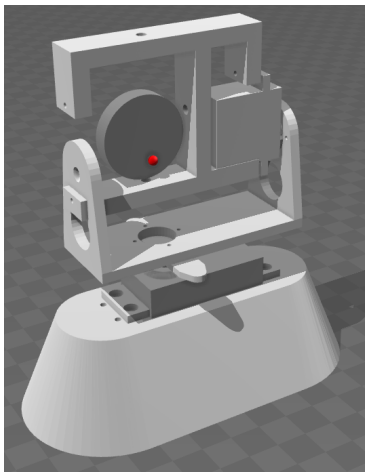


(b) Extracted from s1a2d3's model [4]

Figure 10: Extraction for 2 axis rotation stand with center of rotation



(a) Current Design for the 2 axis optical stand



(b) Current Design with Optical Mirror

Figure 11: Current Design

Since the optical mirror requires to be place at the center of rotation and this model from Figure 10a, its center of rotation is blocked by the x-axis motor so there are not enough space for the mirror, so we have to change out the center piece. The center of rotation is shown as the red dot in Figure 10b. The center piece is replaced by the center piece of the Dynamic System [2], since the Dynamic system uses smaller motors, which will allow more space for the optical mirror. The replacement and the new center of axis is shown in Figure 11a. The excess part highlighted in blue will be cut off physically.

Since the thickness of the center piece is measure to be approximately 11.5 mm, which means the center of rotation is at around 5.75 mm. The Optical mirror we are using has 1 inch diameter and 5.5 mm thickness. So if we take into account of the thickness of the rubber cement, the center of rotation should be at the surface of the mirror. The illustration of the design with the mirror is shown in Figure 11b The resulting stand is shown in Figure 12.

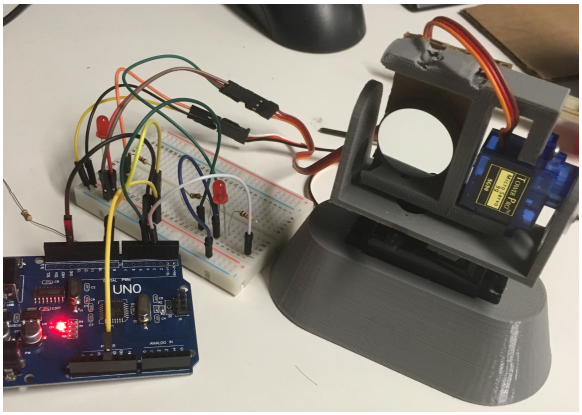


Figure 12: Printed Design with Mounted Optical Mirror

3.2.1 Simulation and Test

In order to control the motors, we are using the "Servo.h" library to generate 50 Hz PWM signal and control the motors' angle with the width of the logic high portion of the signal.

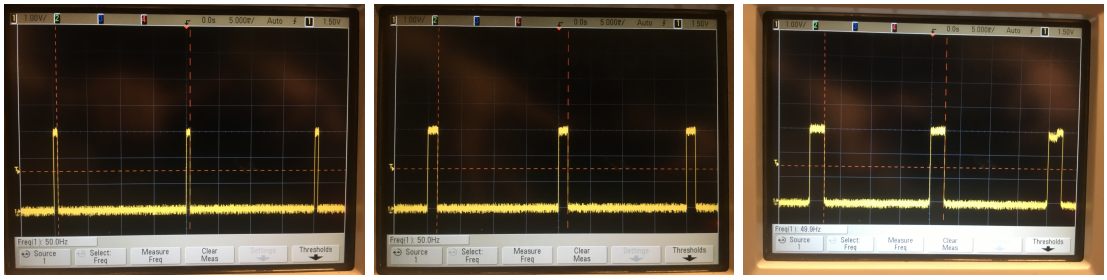


Figure 13: PWM Signals for controlling Motors to 0, 90, 180 degrees

Before we integral this system into the rest of the system, we need to test if this system is liable but running tests and calibrate the motors' position. First we want to know if these motors work properly, is there any broken gear? Hence we want to move the motors to every possible angles by running the following code on Arduino:

Secondly, we would like to know what is the actual speed of the motors. To achieve that, we tell the motors to go 0 to 180 and 180 to 0 degrees in a loop with a certain delay. If the motors turn before reaching at 0 or 180 degrees that means we need to set the delay longer. So we try to obtain the minimum time delay for the motor to reach 0 to 180 or 180 to 0 degrees. The test can be done by input the following code into Arduino and change the delay number.

```
//Motor test
for(pos = 0; pos <= 180; pos += 1)
{
  A.write(pos);
  B.write(pos);
  delay(15);
}
for(pos = 180; pos>=0; pos-=1)
{
  A.write(pos);
  B.write(pos);
  delay(15);
}
```

(a) Motor Test

```
//SPeed test
if( pos == 0){
 pos=180;
 pos2=180;
  A.write(pos);
  B.write(pos2);
  delay(800);
}
if (pos == 180){
  pos=0;
  pos2=0;
  A.write(pos);
  B.write(pos2);
  delay(800);
}
```

(b) Motor Speed Test

The result is that all motors works properly, no broken gear.

Table 3: Motor Speed Test

Motor	Delays Needed (msec)	Speed (msec/60°)
Motor A	430	143.33
Motor B	800	266.67

Then we want to calibrate the motors to obtain what is the required degrees so that the motors are centered. We can achieve this by inputting several positions to both motors around the center of 90 degrees.

Table 4: Calibration of Motors

Motor	Input Angle	Output Angle	Mismatch Angle
A	90	85	5
A	95	90	5
B	90	90	0

So we need to account for the mismatch angle of motor A in the future algorithm.

3.3 Dynamic System

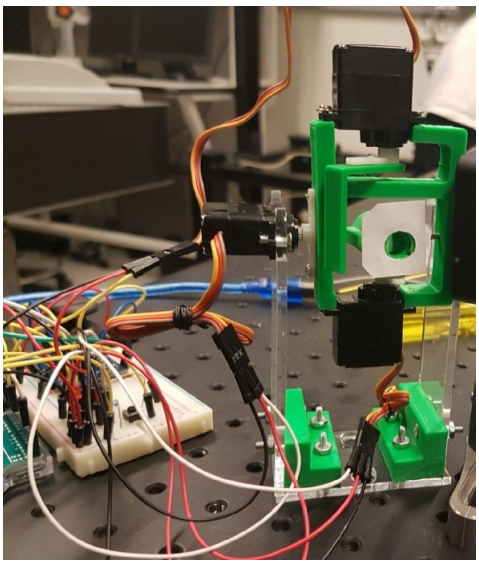


Figure 15: Dynamic System Model[2]

Since this model is already be tested by Mcgill University Mechanical Students, we do not want to repeat the same tests. We only want to test its speed. Since the motors have higher torque than the SG90, the speed is actually higher than SG90. The result of the testing is 350msec needed for 180 degrees, hence its speed is 116msec/60°.

3.4 Camera 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.

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 parameters, 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.

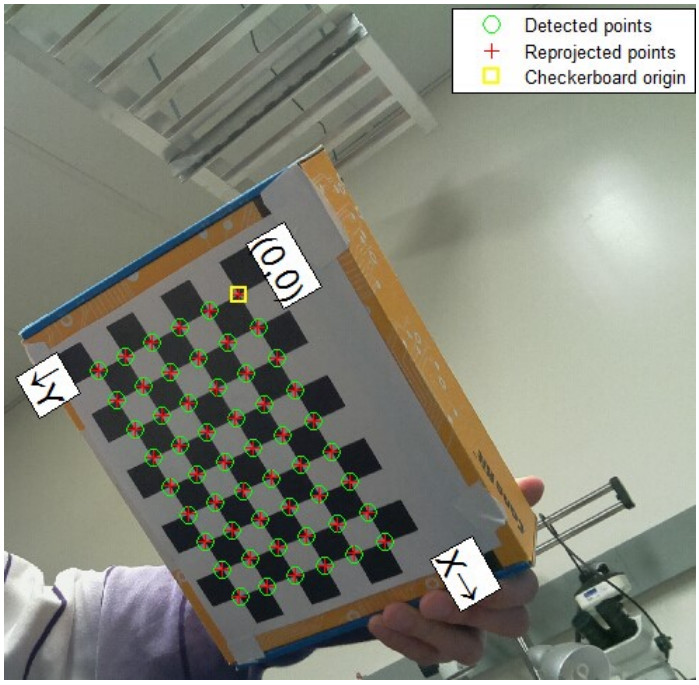


Figure 16: Detection and reprojection on the chessboard pattern

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.1. 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.

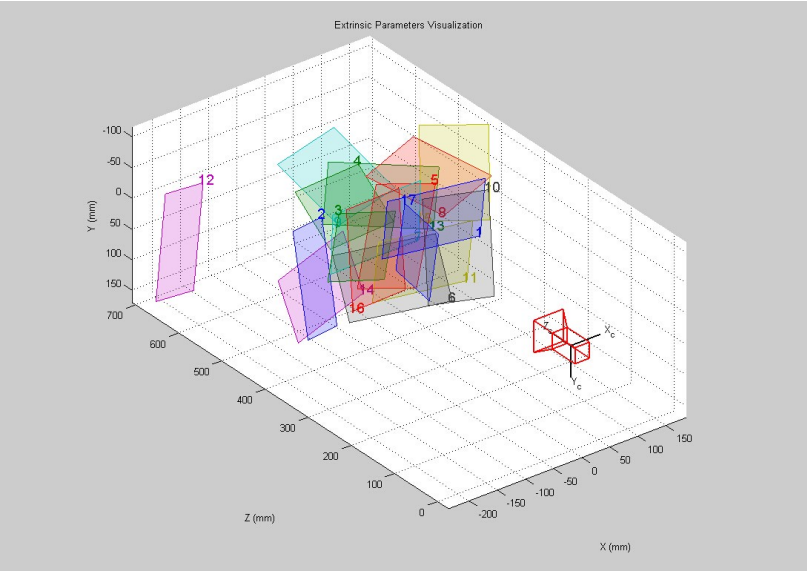


Figure 17: Camera Re-projection of the positions of the chessboard

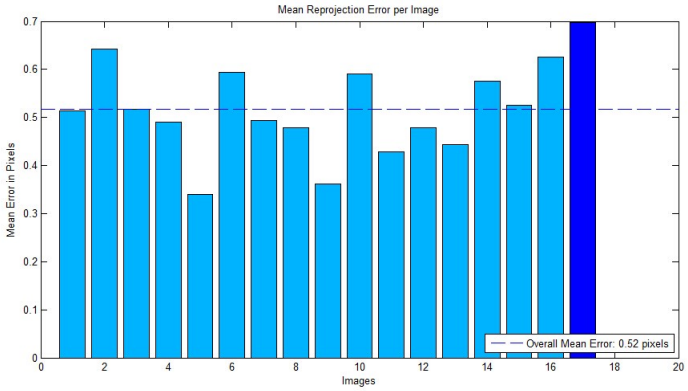


Figure 18: Mean Reprojection Error per Image

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

3.5 Tracking Via Color Filtering and Saturation

After evaluating multiple options of tracking we decided to implement the tracking using Haar cascades. Initially, doubts of the cascades not detecting the objects arose after we learnt that the target provided is not spherically shaped. Those concerns were not completely unjustified, as the flexibility of the cascades was not known. Since we had no prior experience with Haar cascades and target detection, there was a concern that the slightest change in parameters might affect the detection process. Factors such as; the curvature of the target, color contrast, lighting and many more. Therefore, the need to search for alternative methods became essential.

Many different techniques were explored taking into consideration the properties of the new target. So, we decided to track using the color saturation of the image. Initially, this approach seemed to solve the problem that we had in terms of the consistency of the target shape. Since it's solely dependent on saturation, it was only needed to obtain the numerical parameters of the color we're tracking.

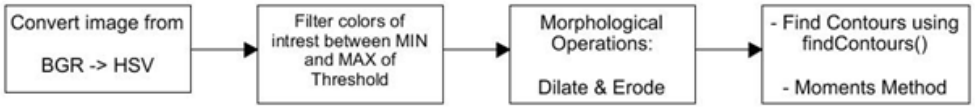


Figure 19: Dynamics of Tracking via Color HSV

The Red Green Blue parameters are obtained by adjusting color saturation in the image until the user is satisfied with the range of colors. This is made easy by using sliders as shown in the right image in the below figure, therefore by trial and error the parameters are easily obtained. The left most image shows the original target, the next image shows the saturated image. After manipulating the parameters, the black and white image starts to change into the desired region of tracking. In this case, the surrounding of the target will be tracked -the white region- because it moves simultaneously with the target at all times.

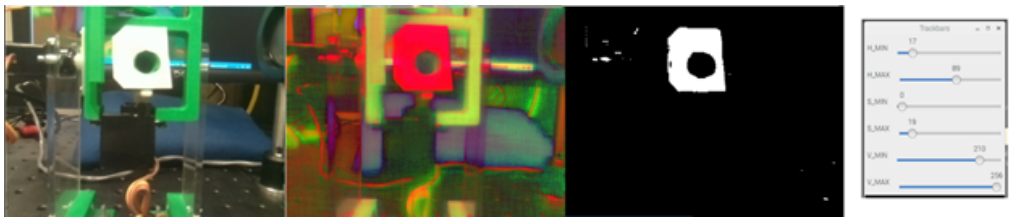


Figure 20: Tracking via Color HSV

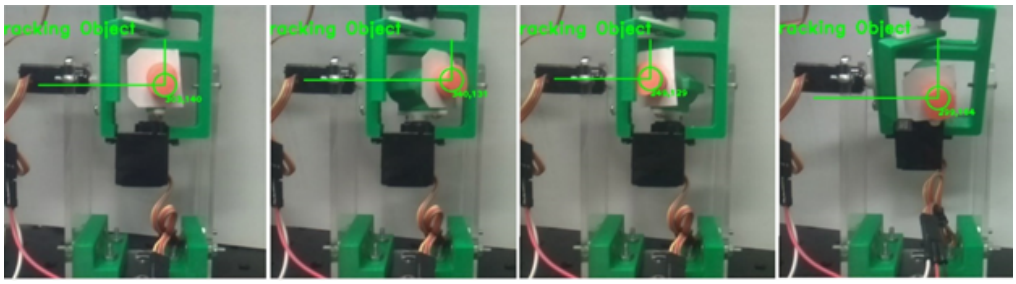


Figure 21: Testing of Tracking via Color HSV- Detection in different Orientations

Upon testing this approach, it proved rather inconsistent and impractical. It forces many constraints on what the color of the object and the surrounding should be. Therefore, if this type of tracking is to be employed, an ideal environment must be guaranteed to obtain useful results. It calls for constant calibration and readjusting the color parameters sometimes even for every iteration. Even for the purpose of this test, an object of unique and distinct color - orange in this case- had to be introduced to the target to ensure consistent tracking. Hence was the decision to abandon this approach. Many different variables are involved in this project. Variables that proved to be hard to control such as the complexity of the background and light intensity. The software code can be found in Appendix A.3.2.

3.6 Cascades

An initial doubt of the effectiveness of this approach was present, as to whether a suitable cascade will be found or not. Cascades are trained using hundreds if not thousands of positive samples of objects -Target- and others of negative samples -backgrounds-, hence they're designed to be rather rigid than fixable.

For the purpose of this project, we used*haarcascade_eye.xml* which is a pre-implemented OpenCV cascade used for detection of eyes and pupils. Said cascade should be sufficient to detect a pupil-shaped target. Knowing the nature of the cascades and how they operate, the shape of the target had to be altered to ensure consistent detection and tracking. Fortunately, this meant that a custom cascade is no longer necessary and a ready cascade can be utilized for this implementation.

Upon comparing the two detection algorithms, it was determined that cascades bare a higher computational cost but produce more consistent results. Despite the computational cost, detection through Haar Cascades is chosen as the method of tracking for this project due to their consistent and accurate results. However, adopting this approach meant somehow reducing the cost on the Raspberry Pi that will process the video stream and apply the cascades. One way to do that is by reducing the resolution of the video stream or images. Lower resolutions mean lower cost but less accurate tracking. The software code can be found in Appendix A.3.1.

3.6.1 Testing Haar Cascades

To determine the optimal tracking resolution, a series of multiple tests were conducted. The resolution is changed then the number of iterations taken in 10 seconds is recorded. An iteration represents a single time the Raspberry Pi extracts an image from a video stream, applies the cascade then determine the location. It is important to note that an iteration does not necessarily mean a successful detection. The previous test is a pure test of speed and raw computational power of the Raspberry Pi versus the resolutions of processed images. Variables like displaying the image, outputting text to the terminal also contribute to computational cost. These variables will be eliminated at the final implementation stage however for now, they are essential for troubleshooting and debugging. Three standard resolutions are chosen for this test 640x480, 320x240, and 160x120. As higher and lower resolutions are not practical for our purposes. This test helped determine the optimal resolution which is 320x240. The following results were observed and recorded:

Table 5: Test Results of Cascade

Resolution /Iteration	First Iteration	Second Iteration	Third Iteration	Notes and Conclusion
640x480	42	38	45	• Slow Speed • Accurate Detections
320x240	132	136	129	• Relatively fast detections • Acceptable Accuracy
160x210	305	317	300	• Very Fast Detections • Bad quality of detections

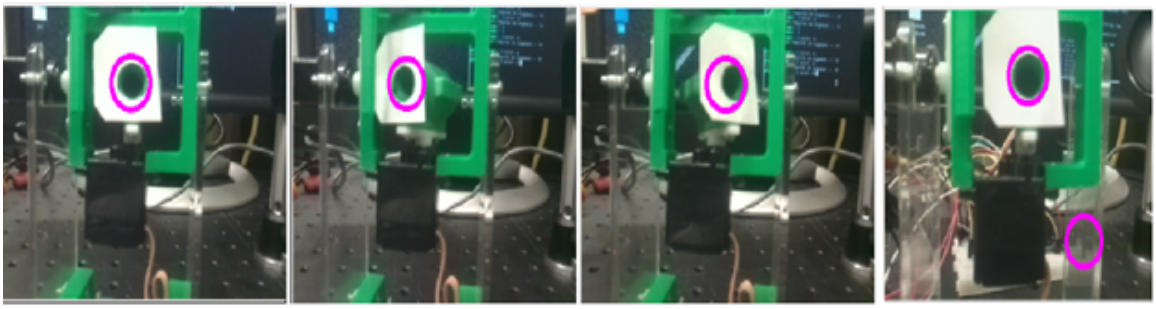


Figure 22: Testing of Tracking via Haar Cascades - Detection in different Orientations

3.6.1.1 Advantages and Disadvantages

As shown in the figure 22 in the far right image, Haar Cascades can sometimes falsely detect objects that are closely similar to the desired object. Despite that, it still detects desired object with very good accuracy. This can easily be fixed by implementing a custom cascade. The down side to that is the custom cascade is going to be very specific to the target and might not be fixable to change. As for the HSV detection, it proved reliable but hard to implement. Result of detection were satisfying but the initial process of setting up the detection is rather difficult. After many iterations, it was found that results of this detection are unrepeatable and hard to

reproduce under slightly different condition. The previous analysis justifies the choice of Haar Cascade over HSV color detection. There is room for improvement on the cascades to make them more reliable and faster, which will be the main focus in the near future.

4 Project Management

The project plan included in this section highlights the defining tasks of the project. Each task is clearly defined by the duties entailed, the duration of the task, the outcome upon completion of the task, as well as the group members responsible for the task. The Gantt Chart attached at the end of the project plan, shown in Figure 23, emphasizes the duration of each task as well as an estimated start and end date.

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 (2 Weeks) - **Noor and Zac**:

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. Adjust the given code for 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 - MEMs mirror Setup (3 Weeks) - **Noor and Anas**:

1. Setup and Test the MEMs mirror kit understand its performance and limits
2. Explore the software provided with the kit
3. Design an algorithm using the provided software to control the mirror based on design
4. Run a few test cases of certain locations, and the resulting mirror movement, to ensure that the beam is actually centered on the target
5. Once MEMs mirror have 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 the MEMs movement software
6. 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 MEMs mirror into physical setup and tracking algorithm

Task 7 - Data Modulation/Demodulation (2 Weeks) - **Tony, 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 oscilloscope
5. Replace square wave with text, camera and test if the algorithm will still work properly
6. Adjust the algorithm if necessary

Outcome of task: Able to send data through the laser driver and receive data by target

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

1. Finding the optimal parameters for each component separately
2. 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.

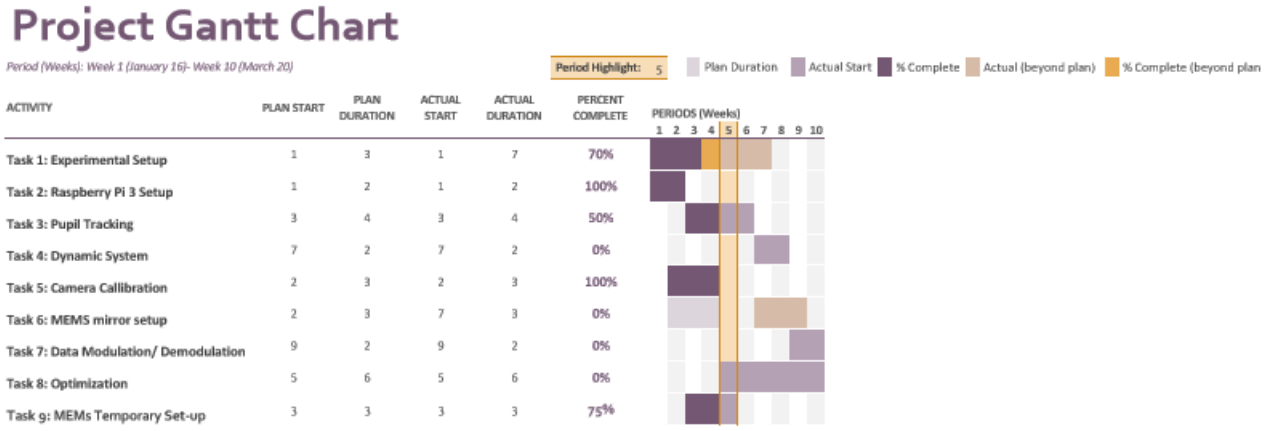


Figure 23: Project Gantt Chart Schedule

4.1 Risks, Challenges and Mitigation Plan

One of the challenges we will face once we integrate all the system together is that the tracking system will not be fast enough to track the target. In other words, the tracking system might not be fast enough to send out target's current location, but the location that is few msec ago. The second challenge is that the camera pixelized the target's location as images, which in other words quantized the target's location. When quantizing a continuous object, there always exists quantization error. In other words, if the target is at 2.5 mm, then the camera can only estimate to 2 or 3 mm. The third challenge is that the 2 axis mirror stand might not be fast enough to reproject the laser onto the target.

Since the Haar Cascade tracking algorithm is design for detailed tracking and recognition, so the algorithm requires higher processing speed hence decrease the tracking speed significantly. So an alternative is to change the tracking algorithm such as color detection. Color detection tracks the object based on its color, so this is enhance tracking speed significantly.

The mitigation plan for the second challenge can be increase the resolution of the camera, which is decrease the quantization error, but as we increase the resolution, it will slow down the tracking system; hence a optimal ratio of the resolution and speed is needed. The mitigation plan of the third challenges we might faces can be simply replace the motors with higher speed and torque motors, or best to bring back the MEMs mirror.

Another challenge that we might face is that, by 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.

4.2 Member Contributions

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 Task (Task 8) in Section 4 is an ongoing task which is carried out by the entire group to optimize any completed components.

Table 6: Highlighting Member's 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	-	50%	-	50%
Task5: Camera Calibration	-	-	100%	-
Task6: MEMs Mirror Setup	50%	50%	-	-
Task 7: Data Modulation/ Demodulation	-	-	50%	50%
Task 8: Optimization	25%	25%	25%	25%
Task 9: MEMs Temporary Setup	-	-	-	100%

5 Conclusion and Future Work

At this current stage of the project, most of the individual part of the tracking system are complete. Therefore, the next stage will be the integration of all the individual components together as one system. This entails the communication of different parts of the system amongst each other in a reliable fashion. After that, we must monitor the performance of the system. This allows us to further enhance the speed and accuracy, in addition to increasing the reliability of the system to avoid any errors results. Improvements will be conducted as discussed in Section 4.1.

Once the prototype of the tracking system is complete, the work on the wireless transmission starts. The initial goal is to transmit and receive a square wave. However, depending on the time we have, the focus will be to transmit 50-60 pixels per frame. Hence, our short-term goal is to transmit simple images, but later on the goal is to transmit a low-resolution video stream.

Once this system is complete, we will be spending the rest of our time optimizing our system, by trying out different solutions and manipulating parameters to maximize performance.

A Software Code

A.1 Camera Calibration

```
% 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.1.1 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
```

A.2 2 Axis Rotation Mirror Stand

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 Algorithm Code

A.3.1 Pupil Detection

```
#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;

/** 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 main */
int main( void )
{
    VideoCapture capture;
    Mat frame;

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

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



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



    //if (!capture.set(CAP_PROP_FRAME_COUNT, 60))
    //{
    //cout<< "Error: Frame Rate is was not set properly"<< endl;
    //}
    int ctr = 1;

    while ( capture.read(frame) )
    {
        if( frame.empty() )
        {
```

```
            printf(" --(!) No captured frame -- Break!");
            break;
        }

        //-- 3. Apply the classifier to the frame
        detectAndDisplay( frame);
        cout << ctr << endl;
        ctr = ctr +1 ;


    //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 );


    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 );
}
```

A.3.2 Object 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;
}
```

References

[1] N. Allami, S. Han, T.C. Liu, A. Zurkiyeh. *ELG4912F Electrical Engineering Design Project: Tracking System & Wireless Transmission*, University of Ottawa, 2016.

[2] 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`

[3] danjperron. *Sending serial data out (via USB) with C++*, Raspberry Pi Forum, 2016.
`https://www.raspberrypi.org/forums/viewtopic.php?f=33&t=131208`

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

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

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

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