



School of Computer Science and Engineering

Final Year Project

Final Report

SCSE21-0384: ICM Buddy – Zoom/Focus Assist Tool

Supervisor: A/P Chia Liang Tien

Examiner: Ast/P Lin Shang-Wei

Submitted by: Aleem Siddique Bin Abdul Jamal

Matriculation Number: U1821548E

NANYANG TECHNOLOGICAL
UNIVERSITY

**SCSE21-0384: ICM Buddy – Zoom/Focus
Assist Tool**

Submitted in Partial Fulfilment of the Requirements for the Degree of
Bachelor of Computer Engineering of the Nanyang Technological
University

By

ALEEM SIDDIQUE ABDUL JAMAL

Abstract

Long exposure photography is a technique of capturing an image for extended periods of time to get unique moments, in moving lights photography. There are few commercialised kits related to long exposure photography sold in the market. And most of these kits are not easily customisable, or any available automated pre-sets pre-installed. Long exposure photography can be very tedious to set up since a lot of knowledge of moving the zoom and focus lens during the shoot will give varying result. Though long exposure shoots can be a fun experience available to all ages, it requires a lot of time and precision to get the image they want.

This project aims to improve and extend the functionality relating to long exposure photography that involves the movements of the zoom and focus lens during the exposure period. The focus of the project is to engineer a prototype capable of bringing automation to the DSLR cameras during the long exposure shoots by swivelling the zoom and focus lens utilising motors.

Long exposure photography can be achieved easily with the integration of hardware and software on Arduino used in this project.

Acknowledgement

I would like to thank my supervisor, A/P Chia Liang Tien for his guidance and support throughout the entire course of this final year project. His extreme knowledge in hardware and his advice has helped me progress through the project without much hiccups.

I would also like to thank the lab attendants from the Software Project Labs for their assistance in loaning equipment especially the soldering equipment during the soldering process of this project.

Lastly, I want to thank my friends for their support and suggestions to improve especially the software part of the project.

Table of Contents

Abstract	3
Acknowledgement	4
Table of Figures.....	7
List of Tables	9
List of Abbreviations and Symbols	10
1. Introduction.....	11
1.1 Background	11
1.2 Project Objective	12
1.3 Project Scope	12
2. Literature Review	13
2.1 Long Exposure Photography	13
2.1.1 Setting Up.....	13
2.1.2 The ‘Standard’ Procedure	13
2.1.3 Camera Settings	14
2.2 Follow Focus Gears	14
3. Project Planning	15
3.1 Material/Equipment Resources (Hardware)	15
3.2 Material/Equipment Resources (Software)	16
3.3 Project Process Model.....	17
3.4 Project Schedule	18
4. Design Documentation.....	20
4.1 Design	20
4.1.1 3D CAD components	22
4.1.1.1 T-Slot Base Mount	23
4.1.1.2 Gears.....	26
4.1.1.3 Motor Housing	27
4.2 Motors	29
4.2.1 Stepper Motor	30
4.2.1.1 NEMA 8 Motor	30
4.2.1.2 NEMA 17 Motor	30
4.2.1.3 Stepper Drivers.....	31
4.2.2 Servo Motor.....	32
4.3 Coding/ Implementation.....	33
4.3.1 Display GUI.....	35
4.3.1.1 Main Menu GUI.....	36

4.3.1.2 Menu UI.....	39
4.3.1.3 Sliders UI.....	40
4.3.1.4 Motors in Motion UI	41
4.3.2 Motor Control	42
4.3.2.1 Stepper vs Servo Motors.....	42
4.3.2.2 Moving singular Motors	43
4.3.2.3 Moving multiple Motors simultaneously.....	44
4.3.3 Interface Functions.....	45
4.3.3.1 Calibration	45
4.3.3.2 Setting a specific distance.....	45
4.3.3.3 Moving motors to a certain distance.....	46
4.3.4 Customise Patterns	47
4.3.4.1 Moving motors from a specified string.....	47
4.3.4.2 Setting custom patterns.....	48
4.4 Integrating with PCB	49
4.4.1 PCB Designing	49
4.4.2 PCB Assembly	49
4.4 Integrated Testing	50
4.4.1 Capturing of objects in motion	50
4.4.2 Capturing of static objects	52
5. Challenges and Issues	54
6. Conclusions.....	55
References.....	56
Appendix A.....	58
Circuit Diagram of Project (Stepper Motors).....	58
Appendix B.....	59
Circuit Diagram of Project (Servo Motors)	59
Appendix C.....	60
Different Layers of the PCB	60
Appendix D.....	62
Available Pre-sets.....	62

Table of Figures

Figure 1: Long exposure image of traffic [2]	11
Figure 2: Long exposure image of lights at night [3]	11
Figure 3: A setup of a long exposure shoot [7]	13
Figure 4: Fotga's DP500II2 follow focus unit [10]	14
Figure 5: Waterfall Model [12]	17
Figure 6: Initial Gantt Chart	18
Figure 7: Arduino Nano used for the project [13]	20
Figure 8: Setup using Stepper Motors	20
Figure 9: Hiwonder 25kg Servo Motor used in the servo-based prototype [14]	21
Figure 10: 3D CAD of the motor housing used for the servo motors	21
Figure 11: The gear rings used that were attached to the zoom & focus part of the lens	22
Figure 12: Sketches of the 3D components with measurements	22
Figure 13: The motor housing secured to the T-Slot base	23
Figure 14: Standardised/Traditional T-Slot rail [16]	23
Figure 15: 3D render of the final iteration of the base mount	24
Figure 16: Code snippet of creating a T-Slot	24
Figure 17: Different iterations of the base mount	25
Figure 18: Gear library used with an intuitive customizer	26
Figure 19: NEMA 17 motor mount for OpenBeam [17]	27
Figure 20: A complete 3D printed setup; the motor housing, gear & base mount	27
Figure 21: Code Snippet of creating the servo motor mount	28
Figure 22: Small NEMA 8 stepper motor	30
Figure 23: Standard size of NEMA 17 [19]	30
Figure 24: NEMA 17 motor used for the project [19]	30
Figure 25: A4988 Driver	31
Figure 26: TMC2208 Driver	31
Figure 27: Servo motor used for the project [14]	32
Figure 28: The parts of the camera lens [21]	32
Figure 29: Flowchart of the main loop	33
Figure 30: Main Menu display	35
Figure 31: Camera settings is highlighted and selected	36
Figure 32: Different settings options	36
Figure 33: A slider to set the shutter time (the same for motor)	36
Figure 34: Motor calibration is selected	37
Figure 35: Different options in calibration menu	37
Figure 36: Calibrate minimum zoom angle	37
Figure 37: Calibrate maximum zoom angle	37
Figure 38: POV calibration is selected	38
Figure 39: Adjusting the zoom lens to the desired image	38
Figure 40: Customise patterns selected	38
Figure 41: No saved patterns found	38
Figure 42: Saved pattern available to be executed	38
Figure 43: Code snippet of the menu function	39
Figure 44: Code snippet of hotbar/slider function	40
Figure 45: An example slider screen	40
Figure 46: Countdown screen	41
Figure 47: Moving to the location	41
Figure 48: Moving back to default	41

Figure 49: Code snippet of RPM and acceleration calculations.....	42
Figure 50: Code snippet of the requirement of a delay.....	42
Figure 51: Code snippet of the moveMotor function	43
Figure 52: Code snippet of the direction decider	43
Figure 53: Code snippet of selecting the correct motor to move	44
Figure 54: Code snippet on moving multiple motors concurrently	44
Figure 55: Code snippet on setting the correct values to the correct motors	44
Figure 56: Before setCurrentPos method.....	45
Figure 57: After setCurrentPos method	45
Figure 58: A camera remote shutter release cord for Nikon DSLR [22]	46
Figure 59: Code snippet of the shutter control function	46
Figure 60: Code snippet of the buzzer.....	46
Figure 61: A code snippet of how the instruction set was read and executed.....	47
Figure 62: A flowchart to create a pattern	48
Figure 63: A pattern was successfully saved and can be executed.....	48
Figure 64: Auto routing tool available on EasyEDA.....	49
Figure 65: A fully assembled PCB board	49
Figure 66: Circular RGB light used in testing	50
Figure 67: The result of using 'zoom to max' [23]	50
Figure 68: The result of using 'zoom to min & focus to max' [24]	51
Figure 69: The Old Supreme Court Building taken without long exposure [25]	52
Figure 70: Old Supreme Building with colourful scaffoldings [26].....	52
Figure 71: Old Supreme Court Building painted pink [26]	53
Figure 72: Circuit Diagram of Project (Stepper Motors).....	58
Figure 73: Circuit Diagram of project (Servo Motors).....	59
Figure 74: Top Layer of PCB	60
Figure 75: Bottom Layer of PCB	60
Figure 76: Merged Layer of PCB.....	61
Figure 77: 3D rendered of PCB with components	61
Figure 78: Focus Movements is selected.....	62
Figure 79: The different patterns available	62
Figure 80: Zoom Movements is selected.....	62
Figure 81: The different patterns available	62
Figure 82: ZoomFocus Movements is selected.....	62
Figure 83: The different patterns available	62
Figure 84: Fixed Patterns is selected	63
Figure 85: The different patterns available	63

List of Tables

Table 1: Relationship between shutter speed and photo clarity	14
Table 2: Hardware Resources	15
Table 3: Software Resources.....	16
Table 4: Project's Updated Timeline	19
Table 5: Relationship of the size of gear	26
Table 6: Comparing features between stepper and servo motors.....	29
Table 7: Comparison between the two drivers.....	31
Table 8: Functionalities of each file	34
Table 9: An example instruction set of custom patterns.....	47

List of Abbreviations and Symbols

Abbreviation/Symbols	Description
DSLR	Digital Single Lens Reflex
ISO	International Organization for Standardization
GUI	Graphical User Interface
UI	User Interface
PCB	Printed Circuit Board
TFT	Thin Film Transistor
NEMA	National Electrical Manufacturers Association
3D	Three-Dimensional
LCD	Liquid Crystal Display
CAD	Computer-Aided Design
POV	Point of View
mm	Millimetre, unit measurement for lengths
Ω	Ohm, unit measurement for resistor
μF	Micro farad, unit of measurement for capacitor
$^{\circ}$	Degree, unit of measurement for angle

1. Introduction

1.1 Background

Long exposure photography, also known as slow exposure, is one of the many types of DSLR photography. Long Exposure photography is usually use by photographers to capture objects in motion or static objects with a moving background for long periods of time [1].



Figure 1: Long exposure image of traffic [2]



Figure 2: Long exposure image of lights at night [3]

Both process of photography requires time in scouting the area, setting up the equipment as well as multiple tries to get the desired image. A lot of calculations are also necessary in terms of the camera settings such as the shutter speed, the ISO, and others [4]. Furthermore, humans are inconsistent, and this will result in extra time needed to get a consistent result that photographers are satisfied [5]. The experimentation required during a long exposure shoot is extremely tedious and there is no straightforward approach available to get pleasurable photos. This applies to not only experienced photographers but even new aspiring photographers who wants to learn about long exposure photography will stay away due to the exhausting process and inconsistent results.

1.2 Project Objective

This project aims to improve the consistency of the results obtained from long exposure photography and present new ways to take slow exposure photos by automating the movements of the focus and zoom lens during the shoot. With the pre-sets available in the project, photographers will be able to control the type of resulting images that they can get. The projective objective is to improve the user experience for experienced and new photographers by providing automation to the camera during the long exposure shoot.

1.3 Project Scope

The followings are the scope of the project:

1. Research on long exposure photography, for objects in motion as well as static objects.
2. Research on available motor types to turn the camera lens.
3. Build a prototype (hardware) to control the lens through Arduino
 - a. Decide on the motor & Arduino to be used for the project
 - b. Have a display & joystick to efficiently control the different lens, zoom & focus lens.
 - c. Code (software) on the Arduino to integrate the hardware together
4. Implement functionalities for the Arduino
 - a. Create different pre-sets or ways to control the movement of the lenses.
 - b. Effortlessly allow users to get a long exposure image with a click of a button and a simple GUI.

2. Literature Review

2.1 Long Exposure Photography

Long exposure photography is a type of photography art in capturing an image where stationary elements are sharp while moving elements are smeared. Photographers normally use exposure photography to capture a certain view like the stars. The main settings for long exposure shots are the shutter speed, ISO, and the aperture of the DSLR camera [6].

2.1.1 Setting Up

The set-up for a long exposure shoot is simple. Other than the camera settings that requires to be experimented, a lot of scouting is required in finding a great area to work in. Furthermore, the use of a tripod is important for a good exposure photo as it keeps the DSLR camera from moving around like from strong winds, during the shoot.



Figure 3: A setup of a long exposure shoot [7]

After applying the necessary camera settings, the photographer then starts the slow exposure photograph by pressing the shutter. After the shutter speed set, the DSLR camera will close the shutter and a long-exposed image will be shown.

2.1.2 The ‘Standard’ Procedure

Ordinarily, long exposure photography involves the DSLR camera being still, and the lens should not move during the exposure. The lens is normally related to the sharpness of the image, and disrupting the setup during an exposure shoot, will affect it. However, there are instances where moving the lens during an exposure will get a particular type of effect. Quoted by Joe McNally, “Moving the lens during the shoot may seem like a dangerous thing but it can be kind of a fun thing to do.” [8] By rotating the lens during an exposure, many different types of outcomes can occur which will be further explored in this project.

2.1.3 Camera Settings

The main camera setting that affect long exposure photography are mainly shutter speed. Normally, during a long exposure photography session, only the shutter speed setting is tweaked while the ISO and aperture remains untouched.

Shutter Speed

Shutter speed refers to the time for the camera shutter to stay open and expose the DSLR camera sensor to light [9]. For long exposure photography, this setting is normally set to a low shutter speed to capture more details and changes to the light.

Shutter Speed	Photo Clarity
Fast	Darker picture and less blur
Slow	Brighter picture but more blur

Table 1: Relationship between shutter speed and photo clarity

2.2 Follow Focus Gears

The follow focus gears were created to have finer controls over the DSLR focus adjustments. This prevents the photographer from focusing to the wrong subject. Follow focus is normally used in filmmaking with film cameras as it helps the operator to be more efficient and precise.



Figure 4: Fotga's DP500II2 follow focus unit [10]

Though their usage was meant to be used in videos for accurate focus control, the possibility of rotating the lens can be applied elsewhere, in the example of long exposure photography. By rotating both the zoom and the focus lens, photographers can get a whole new dimension in long exposure photography.

3. Project Planning

3.1 Material/Equipment Resources (Hardware)

The hardware resources used in this project are shown in Table 1.

Name	Quantity	Description
Arduino Nano	1	Small ATmega328 microcontroller
TMC2208 Stepper Driver	2	Ultra-quiet two-phase stepper motor drive chip, to drive the stepper motors
NEMA 17 Stepper Motor	2	High torque motors to provide precision movements of the lens
Buzzer	1	Sound indicator for users to know before and after a certain sequence is executed
1.8" TFT Display	1	Menu selector to control the lens
Optocoupler P627	1	Isolates the circuit from the camera shutter. Prevents damaging the circuit by high current when the camera shutter is triggered.
2.5mm Jack	1	Connector for DSLR Camera's shutter
5 Pin Joystick	1	Menu movements to select the different options in the menu
220 Ω Resistor	5	Limit the current to the TFT display
1k Ω Resistor	1	Limit the current to the Optocoupler
100 μ F Capacitor	1	Prevents accidental damage to the drivers in case of current surge
11.1V Lipo Battery	1	Power supply for the entire hardware
DSLR Camera	1	For capturing the long exposure images
Arca-Type compatible DSLR rail	1	Motors can be mounted along the rail
0.8 Mod Gear Rings	2	Rings attached to the focus & zoom lens
KW12-3 Limit Switch	4	[Optional] For further control to the lens

Table 2: Hardware Resources

The connection guide for the hardware can be seen in Appendix A.

3.2 Material/Equipment Resources (Software)

Name	Description
Arduino IDE	Official IDE for Arduino microcontroller software development
AccelStepper Library	Arduino Library by Mike McCauley maintained by Patrick Wasp to control the motor drivers
TFT Library	Arduino Library that enables communication with the Arduino TFT LCD screen
SPI Library	Arduino Library that enables communication with SPI devices
Program Space Utilities	Arduino Library for interfacing with the flash memory of the Arduino
EEPROM Library	Arduino Library for interfacing with the memory of the Arduino
OpenSCAD	A 3D CAD software to create 3D objects for 3D printing
EasyEDA	Software used for PCB designing and circuit diagram creation

Table 3: Software Resources

3.3 Project Process Model

This project follows the linear-sequential life cycle model or approach. The project is separated into small parts such as motor movements phase, GUI phase, assessed before integrating together during the testing phase of the model. This ensures tasks are easily arranged and rigorously evaluated before proceeding to the next phase [11]. Overall, ensuring the project development to be completed on time and well documented.

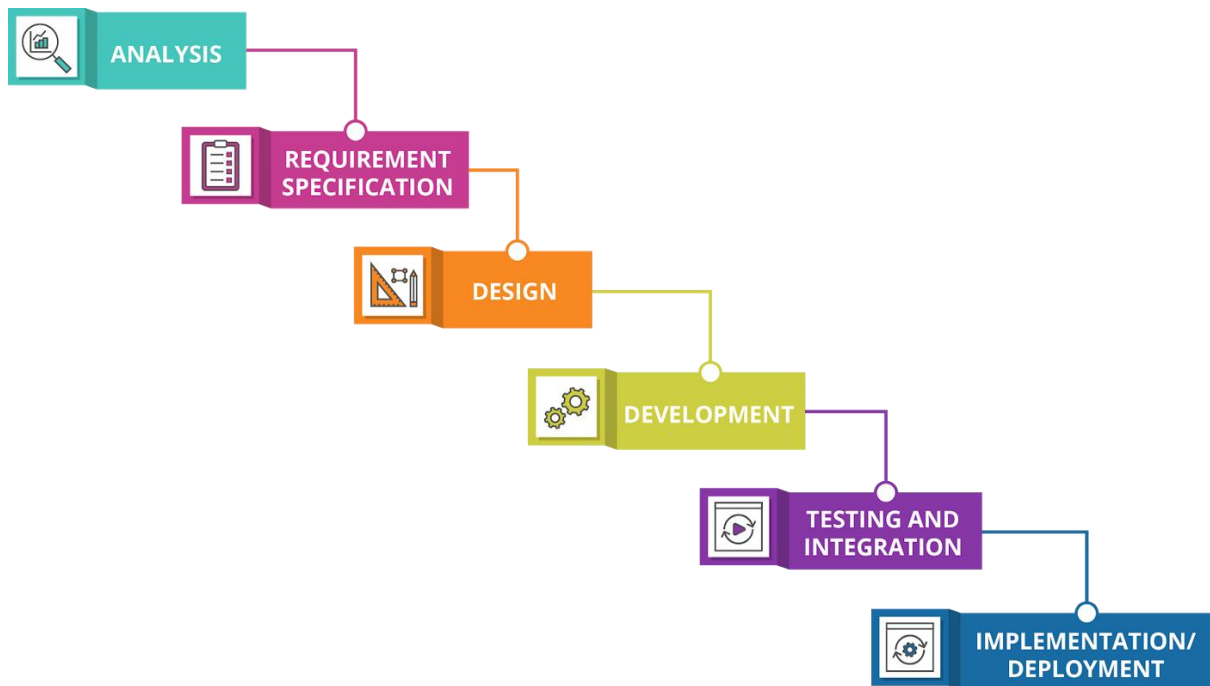


Figure 5: Waterfall Model [12]

3.4 Project Schedule

The initial project's timeline is shown below.

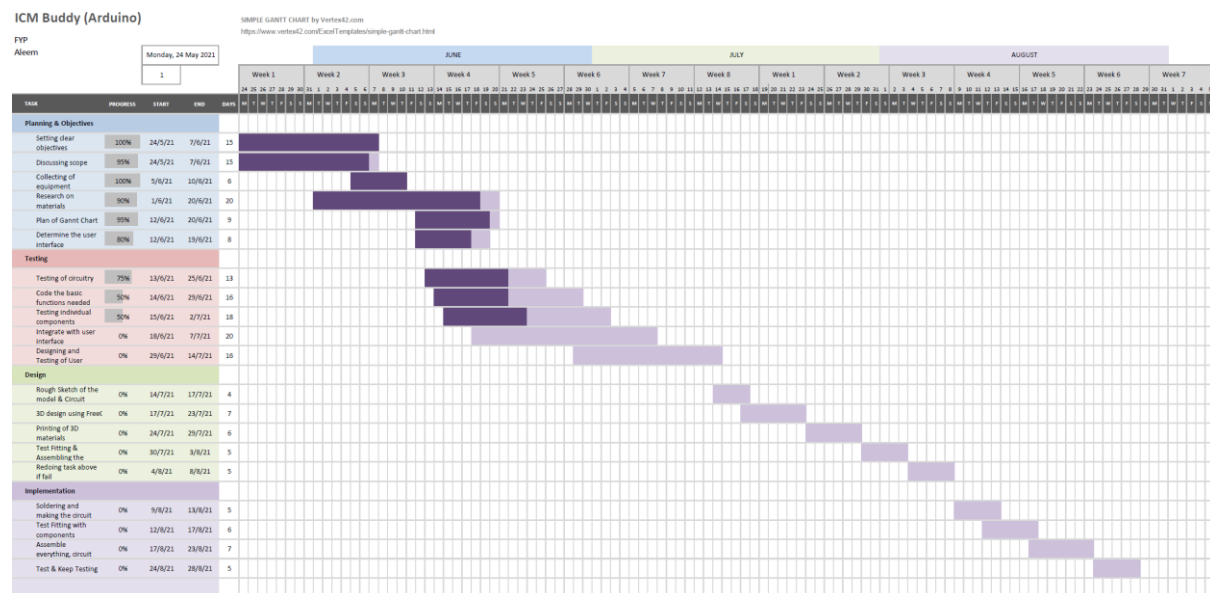


Figure 6: Initial Gantt Chart

Using the model mentioned in 3.3 Project Process Model, this was the initial deadlines for the separate phases from the Waterfall Model approach. However, due to some errors and insufficient research done, the project deviated from the original deadline. This will be further discussed in (insert heading). The project's updated timeline is shown in Table 3.

Task Description	Start Date	End Date	Duration (Days)
Discussing scope	24/5/2021	7/6/2021	15
Research on long-exposure photography	1/6/2021	20/6/2021	20
Research on different motors	1/6/2021	20/6/2021	20
Collecting of equipment	5/6/2021	10/6/2021	6
Testing of individual components (Circuitry)	21/6/2021	10/7/2021	19
Testing of individual components (Software)	11/7/2021	25/7/2021	14
3D Design of components required (gears, bases, case, etc.)	26/7/2021	31/7/2021	5
Printing of 3D components	31/7/2021	3/8/2021	3
Integrating the individual components together	4/8/2021	10/8/2021	7
Test fitting and assembling the circuitry	11/8/2021	13/8/2021	3

Testing of the integrated module	14/8/2021	18/8/2021	5
Testing failed which results in repeating the tasks again as there is a change in the motor used	19/8/2021	7/9/2021	20
Integrating and evaluating the integrated module again	8/9/2021	16/9/2021	8
Testing failed again, which results in more research needed and a change of hardware again	17/9/2021	27/9/2021	10
Revision and Examination	28/9/2021	9/10/2021	12
Update & print 3D designs to combat the previous failed testing	10/10/2021	23/10/2021	13
Redo the circuitry (hardware) components & testing of integrated module	24/10/2021	13/11/2021	21
Revision and Examination	16/12/2021	5/12/2021	15
Testing of integrated module	7/12/2021	21/12/2021	14
Improvements to GUI	22/12/2021	6/1/2022	16
Improvements to code stability	7/1/2022	21/1/2022	14
Documentation of code	22/1/2022	25/1/2022	4
PCB designing	26/1/2022	30/1/2022	5
Improvements to 3D designed components to improve overall stability	1/2/2022	10/2/2022	11
Tidying of code	11/2/2022	26/2/2022	15
Revision and Examination	1/3/2022	12/3/2022	12
PCB printing & soldering of components to PCB	7/3/2022	14/3/2022	7
Final Year Project Report	5/3/2022	21/3/2022	17
Revision and Examination	19/4/2022	7/4/2022	15
Oral Presentation	10/5/2022	10/5/2022	1

Table 4: Project's Updated Timeline

4. Design Documentation

4.1 Design

Two separate designs with the same functionality were prototyped for this project. Both circuits use the same microcontroller, the “Arduino Nano” supporting the basic functionalities of Arduino SOC.

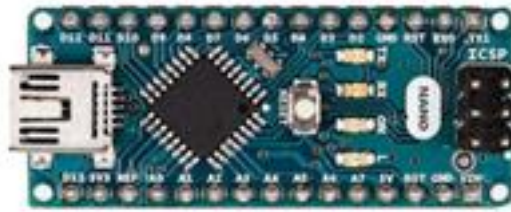


Figure 7: Arduino Nano used for the project [13]

The core components of the two designs were the 3D Printed components such as gears & the motor housing and the type of motor used. The main prototype that this report will be discussing is the prototype that used stepper motors. The circuit diagram for the main prototype can be found in Appendix A. Figure 8 highlighted the whole setup excluding the DSLR camera that was to be mounted on the Arca-Type compatible DSLR rail.

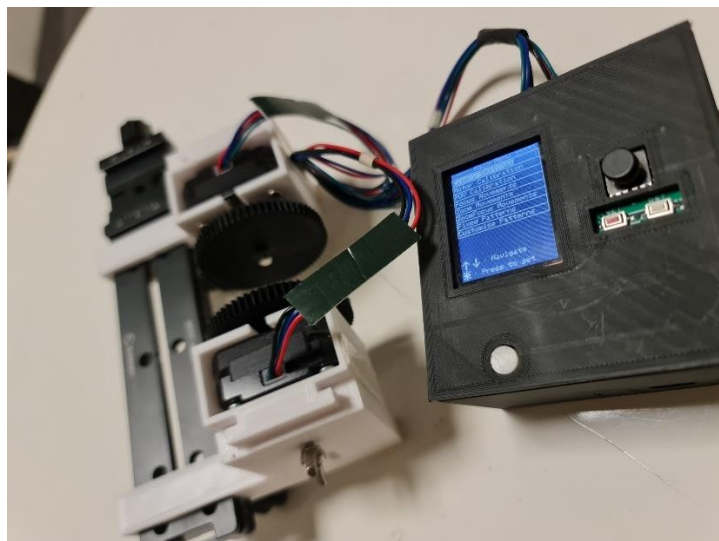


Figure 8: Setup using Stepper Motors

The prototype that used servo motors was an additional add-on to experiment whether different motors would vary the movement of the lens. The circuit diagram for the servo motor prototype can be found in Appendix B.



Figure 9: Hiwonder 25kg Servo Motor used in the servo-based prototype [14]

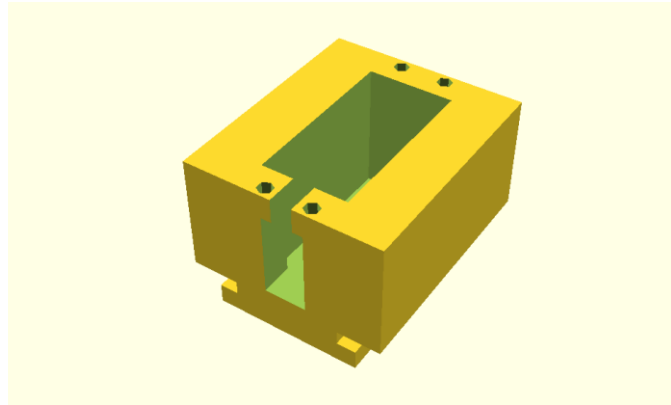


Figure 10: 3D CAD of the motor housing used for the servo motors

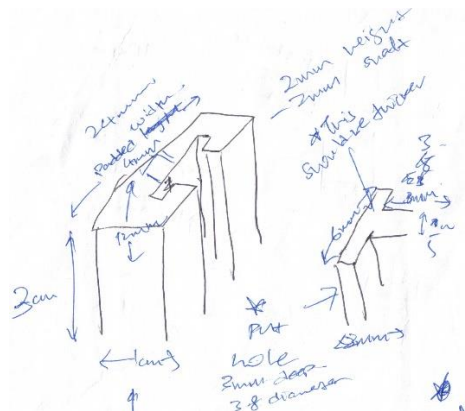
With the different motors used for both prototypes, this resulted various changes to the 3D components used since there is a change of dimension for the motor housings & gears.

Other than hardware components, an important research factor that I had to keep in mind during the designing process was torque. Torque is “a force that causes something to rotate,” as quoted from Cambridge Dictionary [15]. In this project, a certain amount of torque is required for the motor to be able to rotate the lens. The variables that determining factors for torque in this project were the size of gears, the type & size of motor used and the rigidity of the 3D printed motor mounts.

The 3D printed components were one of the key components for the project. The measurements had to be precise, especially the incisions for the screw holes, the T-Slot track, down to less than a millimetre in precision. The main 3D components that were necessary for the prototype to function were the T-Slot base, the gears as well as the motor housing. No 3D printed part for the lens were necessary as there were already plenty of aftermarket choices. These gear rings will only be needed if the lens lacks a 0.8 module gear to rotate it.



Due to the nature of 3D designing, a lot of pre-sketching of the designs were required to have a gist of how the 3D object would look like. In addition, 3D printing the components took plenty of time, combined with the availability of 1 3D printer, courtesy of my Professor, Prof Chia. Hence, some of the 3D printings were printed by external vendors.



The main software that I used during the 3D CAD objects was OpenSCAD, a script-based CAD software.

4.1.1.1 T-Slot Base Mount

The design ideology behind the requirement of a T-Slot was the need of an adjustable height requirement for the motor due to the assorted sizes of available DSLR lens. The T-Slot functioned as a means of secure for the height of the motor housing to be adjusted.

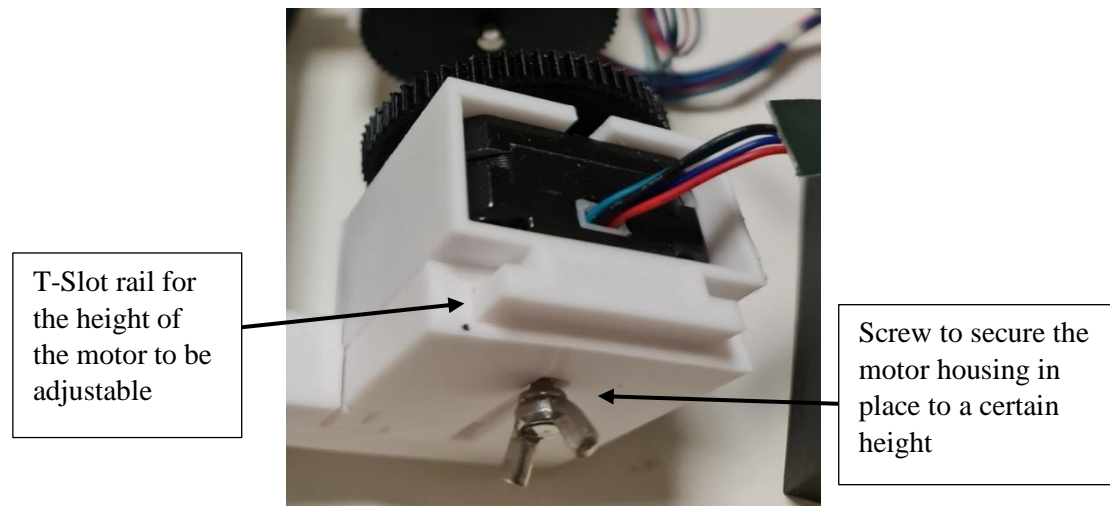


Figure 13: The motor housing secured to the T-Slot base

However, standardised, T-Slots had a very much complex design and to integrate it to the project, it would be too tedious and time consuming. Therefore, a much simpler design was needed to provide the same functionality as the traditional T-Slots, the rail like movement as well as the means of securement. Unlike traditional T-Slot rails, also called OpenBeam, where it has 4 T-slots available, the design only required one, and a simpler T-shape is needed rather than the complex trapezium shape from most standardised rails.



Figure 14: Standardised/Traditional T-Slot rail [16]

As for the base of the T-Slot, it was designed as an Arca-Type compatible mount, where it can slide onto any slides that are Arca-Type compatible. Depending on the length of the lens, different length of the slides is required for the base mount to be fitly secured to the gear rings on the lens. A lot of iterations of the base mount was required due to the inexperience of 3D modelling in addition to the strength testing requirement to keep the motor in place when it was rotating.

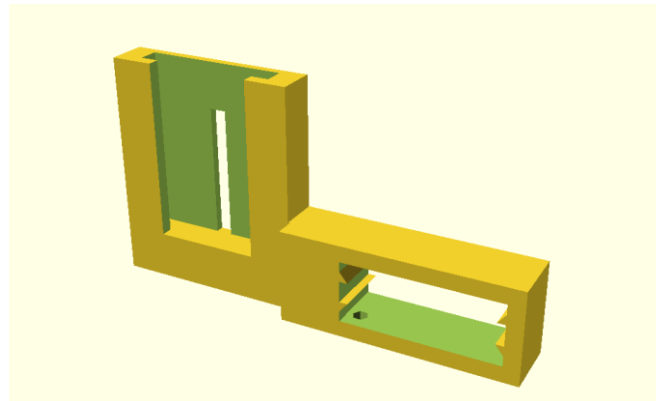


Figure 15: 3D render of the final iteration of the base mount

Since OpenSCAD is a script-based software, the 3D design of the base mount was not as tiresome and not including any complex shapes made the process simpler. A code snippet can be seen below.

```
module t_slot_beam(t_length, t_thickness) {
    slot_length = t_length;
    slot_thickness = t_thickness;
    padded_width = mount_width+thickness*2;

    difference() {
        // stuff to build up
        union() {
            cube([padded_width, slot_thickness, slot_length]);
        }
        // things to cut out (The T-Slot shaped)
        t_slot_length = 3;
        t_slot_width = 15+20;
        t_slot_thickness = 6+20;

        translate([padded_width/2-(t_slot_thickness/2), 0, 0])
        cube([t_slot_thickness, t_slot_length, slot_length]);

        translate([padded_width/2-(t_slot_width/2), t_slot_length, 0])
        cube([t_slot_width, thickness+3, slot_length]);

        translate([padded_width/2-(4/2), t_slot_length+thickness+3, slot_length/2-padded_width/2])
        cube([4, 2, slot_length-10.5]);
    }
}
```

Figure 16: Code snippet of creating a T-Slot

As seen from the code snippet, to render the 3D object, it was with a simple function call. The length and thickness of the T-Slot can be adjusted easily by changing the required parameters.



Figure 17: Different iterations of the base mount

Even then, a lot of effort was required when evaluating the bases to keep the motor in place during rotation along with being sturdy. Changes to the thickness, the T-Slot size were required to get the ideal amount of rigidity and sturdiness to prevent the motor from shaking during the lens rotation.

4.1.1.2 Gears

Gears were also the main components of the entire system. The gears were used to turn the lens of the DSLR camera. Through research, the size of the gears was important as they determine the overall torque of the motors, as mentioned previously. The 3D models for the gears were made with the use of a library created by Thingiverse user eleotlecam. It had a simple GUI to generate & render the gears.

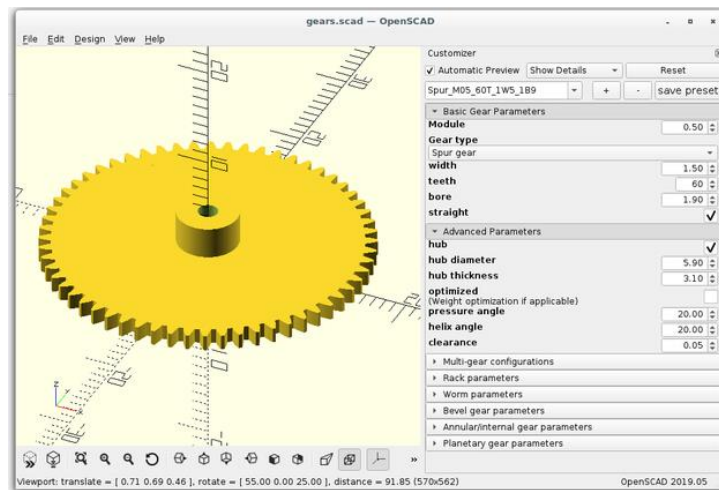


Figure 18: Gear library used with an intuitive customizer

The type of gear used was the basic 0.8 module spur gears; most DSLR gear rings are 0.8 module as described then. The only variable changed was the number of teeth of the gear (also corresponds to the size of the gear). The relationship between the number of teeth of a gear and the torque of the motor as well as the maximum degree of rotation of the lens.

Number of teeth of gear	Torque of motor	Degree of rotation of lens
More (Bigger gear)	Greatly reduced	Increased
Less (Smaller gear)	Increased	Reduced

Table 5: Relationship of the size of gear

Therefore, consideration towards the size of gear was considered to balance between the torque and the degree of rotation. Most importantly, the size of the gear had to be slightly bigger than the size of the motor housing to be able to rotate freely.

4.1.1.3 Motor Housing

The design for the motor housing was simple as the components for the housing were just the motor encasing with a T-Slot attach to it. With the T-Slot, the motor housing would be able to slide on the T-Slot rail on the base mount. Furthermore, the libraries made by 3D modellers were available especially for stepper motors. The library that I used was called “Customizable Stepper Motor Mount for OpenBeam,” which was made by ccox from Thingiverse.com. The default motor mount was made for OpenBeam rails as shown below.



Figure 19: NEMA 17 motor mount for OpenBeam [17]

The author provided adjustable mounts from NEMA 8 to NEMA 42 motors, two of which were used for this project: NEMA 8 & NEMA 17 motors. I had to make a few slight changes to the model, such as covered up all the sides of the motor as well as attached the T-Slot to the back.



Figure 20: A complete 3D printed setup; the motor housing, gear & base mount

For the servo motors, the online libraries available either were using the smaller type of servo motors or they did not fit to the design I envisioned. Therefore, I created the servo motor mount from scratch on OpenSCAD. The requirements for the mount: the same T-Slot as the stepper motor mounts for hot-swappable using the same base mount & as thin in the width as possible.

```

difference() {
  union() {
    // the base as well as the flaps
    translate([0,-(padded_motor_width/4),0])
    cube([padded_length,padded_motor_width,servo_height]);
    translate([servo_length+thickness,-(padded_motor_width/4),0])
    cube([protuding_length,padded_motor_width,servo_height]);
    translate([-protuding_length+thickness,-(padded_motor_width/4),0])
    cube([protuding_length,padded_motor_width,servo_height]);

    //t slot
    xtra = 11.5;
    translate([-xtra/2,(padded_width-t_slot_thickness)/2,-
(t_slot_length)])
    cube([padded_length+xtra,t_slot_thickness,t_slot_length]);

    translate([-xtra/2,(padded_width-t_slot_width)/2,-
(t_slot_length+thickness+2+0.7)])
    cube([padded_length+xtra,t_slot_width,thickness+2+0.7]);
  }

  // **** stuff to be removed ****
  // servo motor
  translate([thickness,thickness,thickness-5])
  cube([servo_length,servo_width,servo_height+5]);
  ...
}

```

The basic cubes required for the servo motor mount

The same T-Slot used for stepper motor mount

Cutting a cuboid for the servo motor to reside in

Figure 21: Code Snippet of creating the servo motor mount

There were few changes required for the motor mounts as they stayed the same throughout the project, other than the need to change the T-Slot when the 3D components required more rigidity to hold the torque of the motors.

4.2 Motors

For this project, two types of motors were experimented to rotate the lens, the stepper motor & the servo motor. There were advantages and disadvantages of using either motor. Additionally, each motor had their own considerations to follow. A list of compiled comparison between the two types of motor can be seen in Table 6.

	Stepper Motor	Servo Motor
Precision	Higher Precision – The chances of the stepper motor missing are low.	Lower Precision – The chances of the servo motor missing are higher under load.
Position Feedback	No Feedback – The stepper motor has no feature for it to know it is current position.	Feature present – Since it uses a sensor for position feedback [18], the servo motor could correct itself if it missed a step. However, this can be a problem for the project as the motor may rotate unexpectedly to correct itself in a middle of a shoot, affecting the image taken.
Torque	4.5kg.cm	25kg.cm
Speed of rotation	Slower	Faster
Degree of rotation	Infinite	180° (In the case of the servo motors used for the project)
Requires a driver?	Yes	No

Table 6: Comparing features between stepper and servo motors

As seen from the table above, both motors have their strengths and weaknesses. However, the main motor that was used for the project with its own custom PCB was the stepper motors.

4.2.1 Stepper Motor

Two models of the stepper motors, NEMA 8 and NEMA 17 stepper motors were experimented and assessed. I started off with the smaller NEMA 8 motor before progressing to the much larger NEMA 17 motor.

4.2.1.1 NEMA 8 Motor

NEMA 8 motors were chosen during the pre-testing phase due to the sheer small size of the motor. Since the components had to fit the small area underneath or beside the camera.

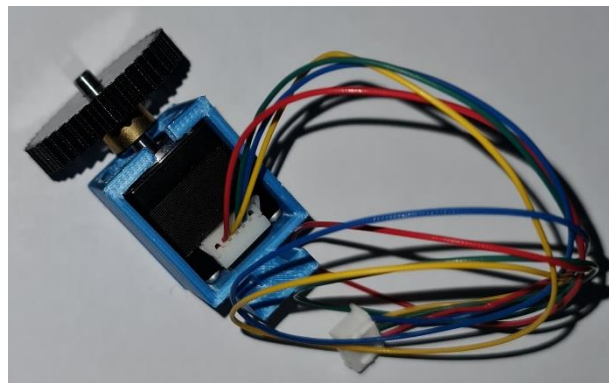


Figure 22: Small NEMA 8 stepper motor

However, due to the sheer size of the stepper motor, the torque of the motor was not enough to rotate the DSLR lens, and a different motor had to be considered.

4.2.1.2 NEMA 17 Motor

Stepper motors have different lengths available for the same NEMA typed motor. In the case of NEMA 17, there were an abundance of lengths to choose from. The “standard” length of a NEMA 17 motor is 39mm in length. Nonetheless, this length was too big and too heavy to be test fitted on the prototype. Therefore, a thinner NEMA 17 motor was selected and used.



Figure 23: Standard size of NEMA 17 [19]



Figure 24: NEMA 17 motor used for the project [19]

4.2.1.3 Stepper Drivers

To drive the stepper motors, a stepper driver was required. A stepper motor driver can manage continuous rotation with precise position control [20]. Paired with a microcontroller, the driver can rotate the stepper motors with a step and direction input. More of this will be covered in The order of the display is as follow:

1. A countdown screen is displayed before the start of a sequence followed by a beep of the buzzer.
2. A display of information will be displayed to the user about the current motor status.
3. If the motor must go back to its previous location, the user will be informed by the display.
4. At the end of the sequence, a beep will be heard, signalling the end of the motor movements.

4.3.2 Motor Control.

Two types of stepper motor drivers were evaluated, the more common A4988 stepper driver as well as the TMC2208 driver. A comparison between the two drivers can be seen in the table below


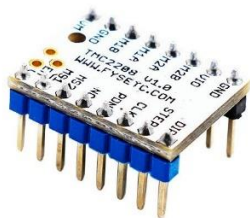
	 Figure 25: A4988 Driver	 Figure 26: TMC2208 Driver
Driver chip:	A4988	TMC2208/9
Pin Configuration:	Both the drivers have the same pin layout which made the drivers easily interchangeable with one another during testing	
Noise:	Very Noisy	Quiet
Heat Generation:	With the small heatsink provided, less heat was produced	Requires a larger heatsink to combat the large amount of heat generated
Torque:	Lesser torque	Higher torque

Table 7: Comparison between the two drivers

Comparing the two drivers, the TMC2208 stepper motor driver was chosen as the final driver to be used for the project. The quiet noise and the higher torque offsets the negligible heat generated from the driver. A standard NEMA 17 motor requires 200 steps to rotate one round. The TMC driver defaults to 1/8 micro stepping which suggests that a total of $8 * 200 = 1600$ steps is required for 1 360° rotation.

4.2.2 Servo Motor

The servo motor used had 180° maximum rotation.



Figure 27: Servo motor used for the project [14]

Having a fixed maximum rotation suggested that I had to integrate the different components together first because I do not have any idea on the maximum rotation for the lens. After assessing the rotation, I concluded that the maximum rotation for any lens will always be less than 180°.



Figure 28: The parts of the camera lens [21]

However, this was true for the zoom ring as they have a maximum and minimum rotation. For the focus ring, some lenses do not have any maximum rotation. This issue unfortunately could be resolved during the calibration process of the lens.

4.3 Coding/ Implementation

The programming model that was used for this project could follow procedural programming as the software contains procedures or functions. A simple flowchart of the main loop can be seen in the figure below.

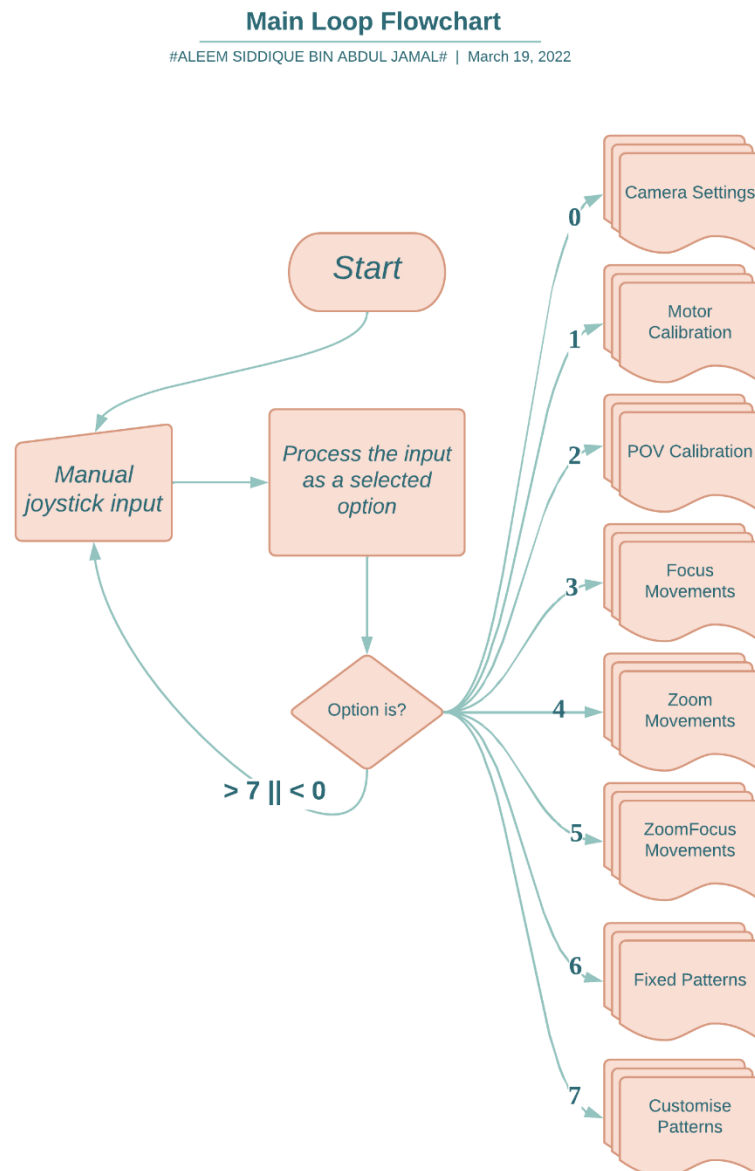


Figure 29: Flowchart of the main loop

The programming style follows a top-down approach, but I did add some Object-Oriented principles in the code, which was to ensure that each function in the “separate” file had only one job to do. Only the interface functions were functions to integrate the distinct functions from the different files together. A table below to describe the functionalities of each file.

File Name	Task/Role
Motor_ICM.ino (Stepper) Servo_ICM.ino (Servo)	The main Arduino file where all the variables were declared, the pins used, and the setup and loop function.
a_DISPLAY.ino	Functions targeted primarily to the display. The primary functions in this file were the menu and slider function.
b_MOTORCONTROL.ino	Functions involving the stepper driver and stepper motors (for stepper motor setup) or the servo motor themselves. The primary functions in this file were the moving motor and the moving multiple motor function.
c_JOYSTICK.ino	Functions requiring the current user input. Any changes to the joystick movements would update the relevant variables required.
d_INTERFACE.ino	The main integrator. Functions in this file integrated all the three above modules. The functions were customised based on a relevant procedure or movement logic of the motor. If the functions here are insufficient to move a motor in a certain manner, it can easily be added in this file.

Table 8: Functionalities of each file

In a way, the three “module” files function as a library and their files were normally untouched once done and evaluated. By implementing the code this way, for expendabilities purposes, only the main Arduino file will need to be touched if other patterns are required. Though, I also added a way for users to create and customise their own patterns which will be further discussed in the report.

Some of the global variables are kept in storage to prevent users for redoing some of the procedures again such as motor calibration or POV calibration.

4.3.1 Display GUI

When a user starts the program, they will be greeted with the main menu screen. The GUI was coded as intuitive as possible, and instructions were also given for a user to navigate around the menu. Both prototypes with different motors have the same menu since the overall functionality remained the same.

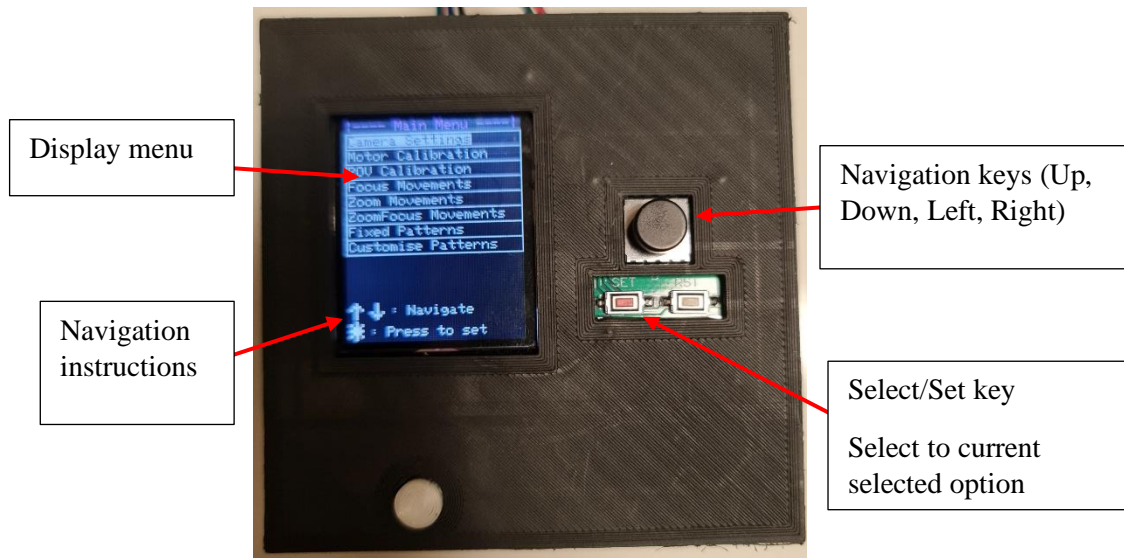


Figure 30: Main Menu display

There would be a total of eight options in the main menu, of which five of them related to the motor movements. User would be able to navigate the different options by up & down and press the select key to select the option chosen. Each option and the functionalities will be covered. For the implementation of how the UI was done through Arduino, this will be discussed after explaining the basic GUI that the user will experience.

4.3.1.1 Main Menu GUI

Camera Settings

This was the first option in the main menu. Here, users can swap motor positions. Not all lenses come with the same Focus and Zoom ring positions. Therefore, a swap motor position here is necessary for users with different lenses without the need to physically change the hardware motor wiring.



Figure 31: Camera settings is highlighted and selected



Figure 32: Different settings options



Figure 33: A slider to set the shutter time (the same for motor)

The variables used for the settings was displayed in Figure 32 for users to know the current set camera settings. In Figure 33, the slider UI was used, and users can set the value by navigating left and right using the navigation keys. This menu was also the same for the motor movement time but with a different text.

Camera shutter time refers to the shutter time of the DSLR camera. This does not change the time of on the camera but must be manually sync to the camera shutter time.

Motor movement time refers to the time needed to execute a sequence. This by default would always set to -1 to 2 seconds from the shutter time for the DSLR camera to capture the current scene before the motor movements can happen.

Motor Calibration

Calibration of the motors was needed as the minimum and maximum angle of rotation for the focus and zoom rings are important during the motor movements.

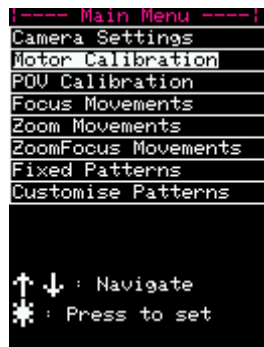


Figure 34: Motor calibration is selected

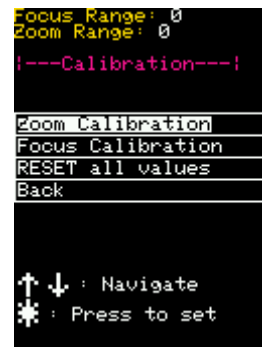


Figure 35: Different options in calibration menu

Three options were available to be selected in the calibration menu. Zoom and Focus calibration and reset all values. Zoom and Focus calibration were straightforward, set the minimum angle, followed by the maximum angle and this returns the maximum range that the motor can rotate for the lens. The values here are stored in memory for the Arduino microcontroller. Users can restart the application/prototype without the need for a recalibration. However, recalibration will wipe out the values stored, and the new values will be saved instead.



Figure 36: Calibrate minimum zoom angle



Figure 37: Calibrate maximum zoom angle

Reset the values could be used when changing lens, a quick way to reset the values rather than recalibrating again.

POV calibration

POV refers to the image that the user wants to capture. A user would want to capture a certain image by adjusting the zoom and focus of the lens. The calibration is required for the motor to know the location that it wants to go to or to return after a motor movement sequence is executed. This ensured that the image was always kept in view at any point of time during the shoot. The same slider as motor calibration to adjust the appropriate zoom or focus rings.

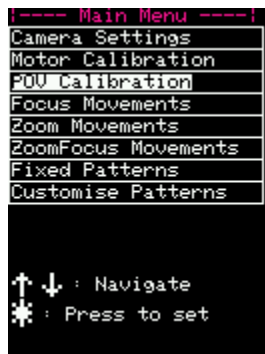


Figure 38: POV calibration is selected

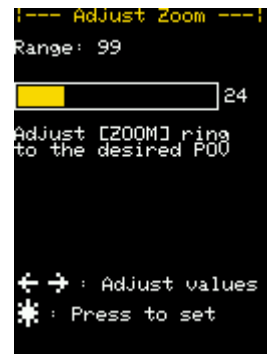


Figure 39: Adjusting the zoom lens to the desired image

Motor Movements

Focus movements, zoom movements, zoomfocus movements, fixed patterns and customise patterns all belong to the same hierarchy that was to rotate the motor in a specific way. The different default patterns can be viewed in Appendix D. For customise patterns however, there were some differences as it requires users to make their own patterns.

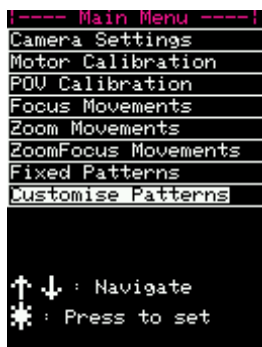


Figure 40: Customise patterns selected



Figure 41: No saved patterns found



Figure 42: Saved pattern available to be executed

Only one pattern could be saved at any point of time and the pattern can have a maximum of eight sequences. If the pattern is available, user can execute the pattern. Any new pattern created afterwards will override the previous pattern.

4.3.1.2 Menu UI

The two main UI used for the software is the menu and sliders UI. Regarding the menu UI, it was split to three parts, a header, a body, and a footer, each were customisable based on the parameters feed in the function. The code documentation in Figure 43 that was done easily explained how the menu UI functions.

```
/**
 * ----- MENU -----
 * Creates a menu using a string array
 * Display which option is being selected
 *
 * [Top of screen]                {Configurable}
 * Shutter Speed: __
 * Front Motor: ____
 * Rear Motor: _____
 *
 * [Middle of screen]
 * |----< Menu Name >----|
 * | Option 1
 * | Option 2
 * | ...
 *
 * [Bottom of screen]
 * < > Move the lens              {Will be dependent on the menu}
 * ^v Select the options
 * o Select current option
 *
 * @param array_size              Maximum choices that the menu should have.
 * @param string_table[]         Character array of strings to be displayed.
 * @param option_selected         Current selected option.
 * @param header                 Values from -2 to 4, to display varying
 *                               headers.
 * @param footer                 Values from 0 to 2, to display varying footers.
 * @param color                  uint16_t colors can be used for display.
 * @return int                   The maximum choices of the menu.
 */
int menu(int array_size, const char *const string_table[],
         int option_selected, int header=0, int footer=2,
         uint16_t color=DEEPPINK)
```

Figure 43: Code snippet of the menu function

Different parameters can be passed to get desired menu on the screen. `string_table[]` is a character array which requires the title followed by the different option texts: option one, option two... An example of a menu can be seen in the figure below.

```
max_option = menu(8, main_menu, option, 0);
```

An eight-option menu was created with the option selected was highlighted in white. If the option changes, the menu would update and reflect properly. Users will then know which option is currently selected visually.

4.3.1.3 Sliders UI

The sliders UI, similarly, to the menu UI also have different customisation available. The slider function parameters and what each parameter corresponds to can be seen in the code snippet below.

```

**
* --- Hotbar Screen ---
* Creates a template for a bar type screen
*
* @param title           Title of the hotbar/screen.
* @param current         Current value.
* @param max_range       Maximum range allowed. (Always start from 0)
* @param current_option  Current option selected. (Only used if `haveBack`
is true)
* @param haveBack        Insert a back button if needed.
* @param header          Values from -2 to 4, to display varying headers.
* @param footer          Values from 0 to 2, to display varying footers.
* @param color           uint16_t colors can be used for display.
* @param updateBar       Only updates certain areas of the hotbar. (Save
Time)
*/
void hotbar(char title[], int current, int max_range,
            int current_option=0, bool haveBack=false, int header=0,
            int footer=3, uint16_t color=WHITE, bool updateBar=false)

```

Figure 44: Code snippet of hotbar/slider function

The parameter `haveBack=false` referred to an optional back button for user to navigate back to the previous screen. What parameters correspond to what can be seen in the figure below.

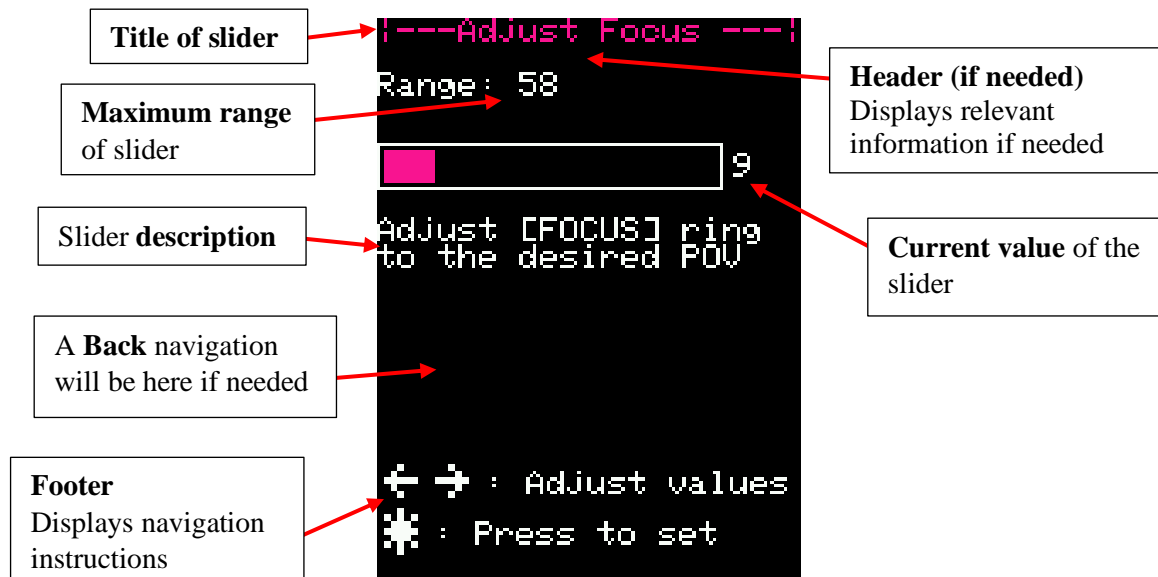


Figure 45: An example slider screen

The most important parameter for the function is `updateBar`. If the parameter is set to true, only the current value and the bar of the slider refreshes. As TFT display were slow to refresh, this was a countermeasure to that.

4.3.1.4 Motors in Motion UI

Motors in motion UI are a set of screens that will be displayed to the user during a motor sequence. This was to inform the user of the current task. Paired with a buzzer, the user will be notified of the start and the end of a motor rotation sequence.



Figure 46: Countdown screen



Figure 47: Moving to the location

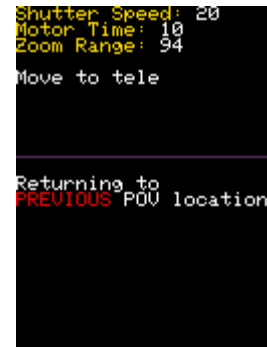


Figure 48: Moving back to default

The order of the display is as follow:

5. A countdown screen is displayed before the start of a sequence followed by a beep of the buzzer.
6. A display of information will be displayed to the user about the current motor status.
7. If the motor must go back to its previous location, the user will be informed by the display.
8. At the end of the sequence, a beep will be heard, signalling the end of the motor movements.

4.3.2 Motor Control

Motor control forms the backbone of the software as the controlling of the motors must be done in these functions. Since the two prototypes using different motors share similar code, the functions here are different due to the nature of the libraries used.

4.3.2.1 Stepper vs Servo Motors

For the project, I used the “AccelStepper” library to control the drivers and motors for the stepper motor setup. There were extensive functions available and an easy way for the motors to remember their current positions. Due to the nature of stepper motors having a lower torque as compared to servo motors, there was a need to have acceleration to the motor for a smooth transition. This would prevent the motor from losing additional torque during the rotation process.

```
float calcRPM(int steps, float seconds) {  
    return (float) (steps * MS_STEP) / seconds;  
}  
  
float calcAccel(int steps, float seconds) {  
    float max_speed = calcRPM(steps, seconds);  
    return (float) max_speed / seconds;  
}
```

Figure 49: Code snippet of RPM and acceleration calculations

Also, since there was a timing for the motor to move called ‘*motor_time*’, the speed calculation was critical in regulating the time for the motor to complete a fixed rotation. The speed and acceleration were calculated as shown in Figure 49.

I used the “Servo” library to control the servo motors. Unlike stepper motors, servo motors remember their current position due to the sensors attached (refer to 4.2 Motors). For stepper motors, it was based on position coordinates and the library calculates if any negative or positive movements was required. Furthermore, since the speed for servo motors cannot be regulated, I had to use delays between each degree angle of rotation.

```
if (steps_to_move > 0) { // +ve  
    for (int pos=pos_current; pos<=pos_desired; pos+=1) {  
        if ((type == 1 && orientation == 0) || (type == 0 && orientation ==  
1)) { // if rear motor  
            motor->write(MOTOR_STEPS - (pos + min_pos));  
        } else {  
            motor->write(pos + min_pos);  
        }  
        → delay(toMS((float) shutter_spd / abs(steps_to_move)));  
    }  
}
```

Figure 50: Code snippet of the requirement of a delay

4.3.2.2 Moving singular Motors

Stepper motor

Moving a singular stepper motor was straightforward. From the library, there were only two functions required to move a motor. Set the position for the stepper motor to move to followed by the run command in a while loop.

```
stepper->moveTo(pos_desired * MS_STEP);

//blocking statement
while (stepper->distanceToGo() != 0) {
    stepper->run();
    if(shutter_spd != 0) {
        delay(toMS((float)shutter_spd/abs(steps_to_move)));
    }
}
```

Figure 51: Code snippet of the moveMotor function

As mentioned earlier, to move the stepper motor to the desired location in a certain time, the speed of the motor had to change. This was set before the stepper motor was allowed to run.

Servo Motor

Servo motor implementation of moving a motor was less complicated from the library examples. However, the need to do an extra calculation to decide if the servo motor needed to go the positive or negative direction made the implementation a little complicated.

```
int steps_to_move = (pos_desired - pos_current);
// if +ve, move clockwise
// else -ve, move anti-clockwise
if (shutter_spd != 0) {
    if (steps_to_move > 0) { // +ve
        for (int pos=pos_current; pos<=pos_desired; pos+=1) {
            if ((type == 1 && orientation == 0) ||
                (type == 0 && orientation == 1)) { // if rear motor
                motor->write(MOTOR_STEPS - (pos + min_pos));
            } else {
                motor->write(pos + min_pos);
            }
            delay(toMS((float)shutter_spd/abs(steps_to_move)));
        }
    }
    if (steps_to_move < 0) { // -ve
```

Check if the steps to move to desired location is positive or negative

Figure 52: Code snippet of the direction decider

Selecting the correct motor

In the 4.3.1.1 Main Menu GUI, it mentioned that users can select which motors (the rear or the front motors) are used for rotating the zoom or focus gears. The implementation on how to select the correct motor was by using an object pointer and points it to the correct motor that requires to rotate.

```
AccelStepper *stepper;
if (type) {
    stepper = orientation ? &front_motor : &rear_motor;
} else {
    stepper = orientation ? &rear_motor : &front_motor;
}
```

An object pointer is declared

The pointer **points** to the correct motor based on set orientation

Figure 53: Code snippet of selecting the correct motor to move

4.3.2.3 Moving multiple Motors simultaneously

Moving of multiple motors at the same time was straightforward as it was the same implementation as singular motors except the need for another run function in the while loop.

```
while (rear_motor.distanceToGo() != 0 || front_motor.distanceToGo() != 0) {
    rear_motor.run();
    front_motor.run();
    if (shutter_speed != 0) {
        delay(toMS((float)shutter_spd/average_steps));
    }
}
```

Condition: waits for both motors to complete their movements

Both motors to run. If a motor is **done**, this function does nothing

Delay: Uses average steps to have an ideal delay between the two motors

Figure 54: Code snippet on moving multiple motors concurrently

Feeding the right position to go for each motor was simpler as there was not a need for an object pointer. The values were fed to each individual motor instead depending on the orientation set.

```
if (zoom_value == -1) {
    rear_position = orientation ? focus_value : zoom_current;
    front_position = orientation ? zoom_current : focus_value;
} else if (focus_value == -1) {
    rear_position = orientation ? focus_current : zoom_value;
    front_position = orientation ? zoom_value : focus_current;
} else if (focus_value == -1 && zoom_value == -1) {
    rear_position = orientation ? focus_current : zoom_current;
    front_position = orientation ? zoom_current : focus_current;
} else {
    rear_position = orientation ? zoom_value : focus_value;
    front_position = orientation ? focus_value : zoom_value;
}
```

The rear and front motors were feed with the required movements depending on the orientation of the motors

Figure 55: Code snippet on setting the correct values to the correct motors

4.3.3 Interface Functions

The interface functions combine the functions from separate file, remarkably like the integration phase of the waterfall model.

4.3.3.1 Calibration

The implementation of calibration for the motors was to get the minimum angle, followed by the maximum angle and get the range.

$$\text{zoom range} = \text{maximum zoom} - \text{minimum zoom}$$

After getting the 3 values required, the minimum angle becomes the absolute minimum position. For stepper motors, there was a function called *setCurrentPos* which was able to set a position relative to the current stepper motor position.

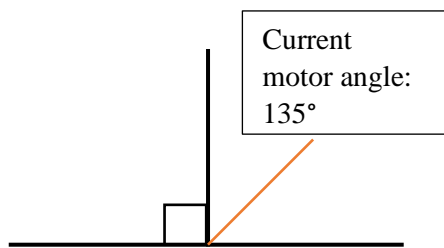


Figure 56: Before setCurrentPos method

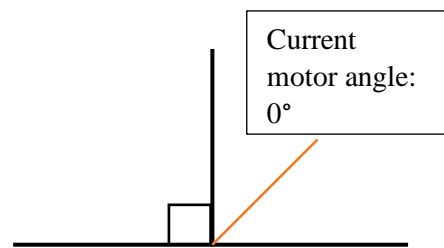


Figure 57: After setCurrentPos method

In the example above, the stepper motor sets its current position relative to whatever was supplied in this case, '*setCurrentPos(0)*'. This suggested that the minimum angle set for the stepper motor will always be 0.

However, for servo motor, the implementation is slightly different. Since there was not any method like the ones used in stepper motor, the minimum angle of the servo motor must be saved. The angle of movements acquired the additional minimum angle during movements.

$$\text{total angle to move} = \text{angle to move} + \text{minimum angle}$$

The 'total angle to move' was then supplied to the motor to rotate accordingly.

4.3.3.2 Setting a specific distance

There was also a function that allows the user to set a specific distance, this was usually necessary for custom movements other than the pre-sets available. The backbone of the method functions similarly to the calibrate function though only 2 values were required, which was the position to go to. The values can then be supplied to the motor to rotate.

4.3.3.3 Moving motors to a certain distance

The interface function for this integrates the 4.3.1.4 Motors in Motion UI and 4.3.2 Motor Control. The control of the shutter as well as the buzzer was set here. The sequence to move the motor to a certain distance are as follow:

1. Users press a certain pattern/sequence to execute.
2. Shutter of the DSLR camera opens and the buzzer starts beeping.
3. The screen starts to display the relevant information to the user while the motor starts to rotate.
4. Upon the end of the motor movements, shutter of the DSLR camera closes, capturing the image and the buzzer beeps signalling the end of a sequence.

Controlling the shutter

Shutter control was done simply by using the camera remote shutter release cord connected to the 2.5mm jack on the board.



Figure 58: A camera remote shutter release cord for Nikon DSLR [22]

Software side, a short pulse of LOW to HIGH was required to toggle the shutter.

```
void nikonTime() { // Controls the shutter of a Nikon camera
  digitalWrite(SHUTTER, LOW); // close shutter (fully pressed)
  delay(20);
  digitalWrite(SHUTTER, HIGH);
  delay(20);
}
```

Figure 59: Code snippet of the shutter control function

Buzzer

Controlling the buzzer was done using the inbuilt Arduino library in running any buzzer. The code can be seen below. Beeps the buzzer for 500ms by default.

```
void buzz(int delay_ms=500, int freq=1000) {
  tone(BUZZER, freq); // Send freq(Hz) sound signal...
  delay(delay_ms); // ...for delay_ms
  noTone(BUZZER); // Stop sound...
}
```

Figure 60: Code snippet of the buzzer

4.3.4 Customise Patterns

The implementation of allowing users to create their own patterns was something that I spent lots of time and was immensely proud of the process.

4.3.2.1 Moving motors from a specified string

The core of the customisation comes from the motor movements. To create custom patterns, I decided to use strings to store the sequence. Strings can easily be stored in Arduino storage (more in 4.3.4.2 Setting custom patterns) therefore the string had to be read in a certain format and execute the instructions embedded in it. Each sequence consisted of 4 aspects with up to 6 characters. An example instruction set can be seen in the table below.

Character position [6]	[0]	[1]	[2]	[3++]
Character option	<i>Z or F</i>	<i>T or F</i>	<i>T or F</i>	<i>Int (0-999)</i>
Representation	Either zoom or focus motor	If the motor needed to go back?	If this is the last sequence in the procedure?	The position of the motor to move to
A full example	ZFF80, FFF40, ZTF18, FTT7			

Table 9: An example instruction set of custom patterns

A simple split function was used to separate the string by the comma. Each sequence was then passed to another function to execute the instruction of the sequence.

```
void setMotor(char* data, char title[], int custom_itemcount) {
    bool goBack = false;
    if ((data[1] == 'T' || data[1] == 't')) {
        goBack = true;
    }

    bool lastSequence = false;
    if ((data[2] == 'T' || data[2] == 't')) {
        lastSequence = true;
    }

    if ((data[0] == 'F') || (data[0] == 'f')) {
        int steps = strtol(data+3, NULL, 10);
        // set focus motor steps to steps
        goDist(FOCUS, title, steps, SNOW, goBack, motor_time/custom_itemcount, lastSequence);
    }

    if ((data[0] == 'Z') || (data[0] == 'z')) {
        int steps = strtol(data+3, NULL, 10);
        // set zoom motor steps to steps
        goDist(ZOOM, title, steps, AZURE, goBack, motor_time/custom_itemcount, lastSequence);
    }
}
```

Checks if the motor has to go back based on the instruction set

Checks if this was the last sequence

The position to move was passed to the interface function goDist

Figure 61: A code snippet of how the instruction set was read and executed

4.3.4.2 Setting custom patterns

To set a custom pattern, the method is simple for users to use. This was briefly shown in the Motor Movements section. A flowchart to show the user interaction when creating a new pattern can be seen below.

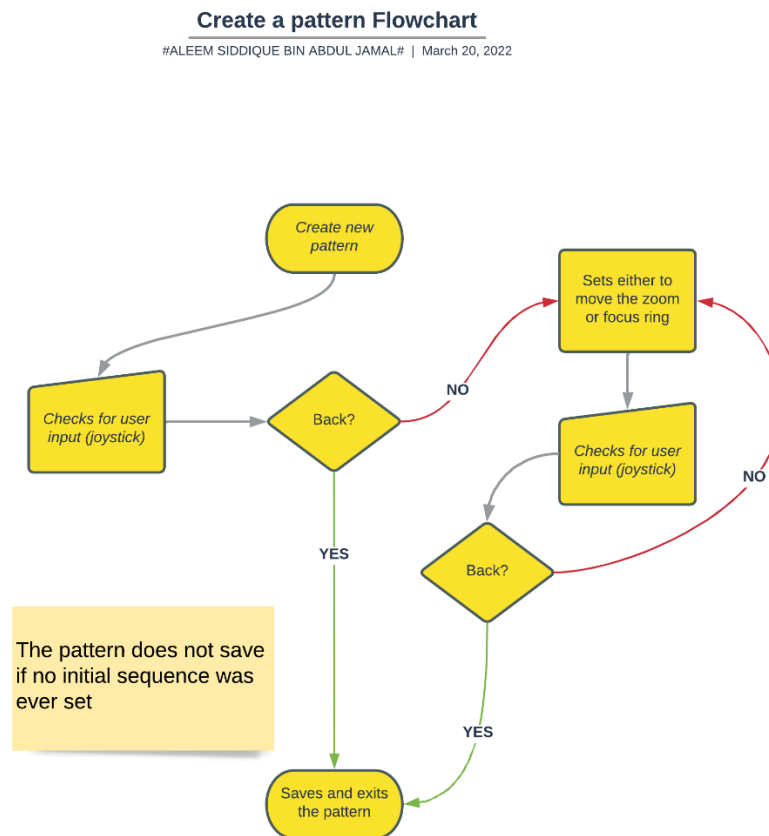


Figure 62: A flowchart to create a pattern

The pattern will be saved when the user clicks back and executing the pattern/procedure as set would be possible.



Figure 63: A pattern was successfully saved and can be executed

Creating a new pattern will override the current saved pattern since the microcontroller only has enough storage to only save 1 array of sequences.

4.4 Integrating with PCB

The project started off on a breadboard before progressing to a PCB board after the hardware was confirmed. Using EasyEDA online designer tool, I was able to design a PCB and it was sent to printing. The full PCB board can be viewed in Appendix C.

4.4.1 PCB Designing

The designing process of the PCB started off from the circuit diagram before using the designer built-in tool to generate a PCB. Thanks to most of the community, most of the hardware used were available and they were just a drag and paste to the circuit board.



Figure 64: Auto routing tool available on EasyEDA

The PCB wire connection could also be done thanks to the auto routing function of the tool. However, some tweaks still needed to be done such as thickening the wires and rewire certain connections.

4.4.2 PCB Assembly

The assembly process of the PCB was done with the help of my professor, Professor Chia. The hardware required for the PCB was soldered to the PCB and a 3D case was printed to encase the PCB board along.

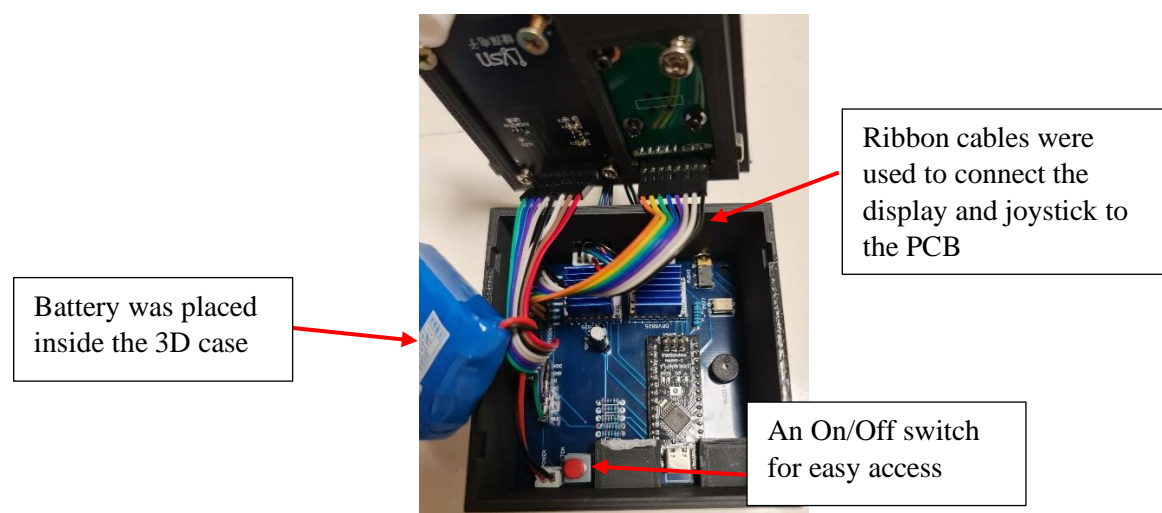


Figure 65: A fully assembled PCB board

In the end, the prototype was ready to be extensively evaluated in different kind of conditions for the long exposure shoot.

4.4 Integrated Testing

4.4.1 Capturing of objects in motion

For this test, the prototype was setup in a completely pitched black environment other than a circular RGB light. This RGB light changes as it travels around the loop. The prototype was evaluated with the help of my FYP mentor, Prof Chia. The type of light used is seen below.

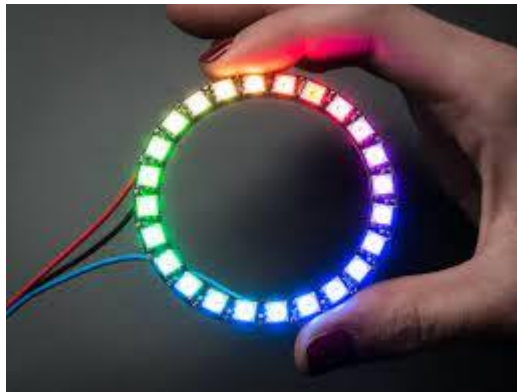


Figure 66: Circular RGB light used in testing

The 1st sequence used was the zooming to the maximum sequence (see Zoom Movements in Appendix D). The sequence was to rotate the zoom lens of the DSLR from the current POV to the maximum range that was set. The result of the photo can be seen below.

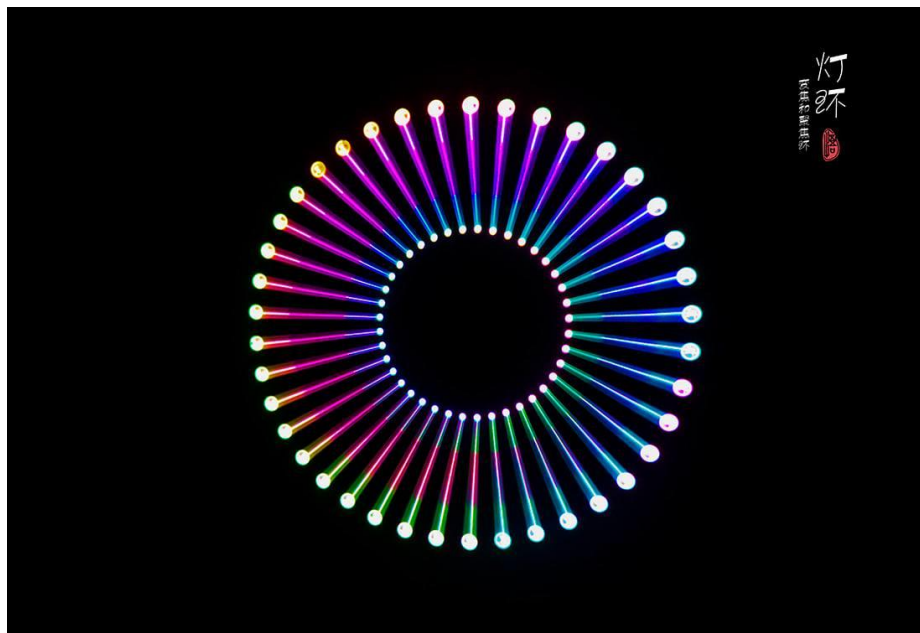


Figure 67: The result of using 'zoom to max' [23]

The resulting image was as expected as the lens zoomed in to the RGB light. However, as seen from the long-exposed image, there are bits of lights with no colours, just white, which suggests an over exposure.

The next sequence to be used was the 'zoom to min and focus to max' pattern (see ZoomFocus Movements). The sequence rotates both the zoom rings and focus rings at the same time in opposite direction. The image was to get zoomed out while also get blurry.

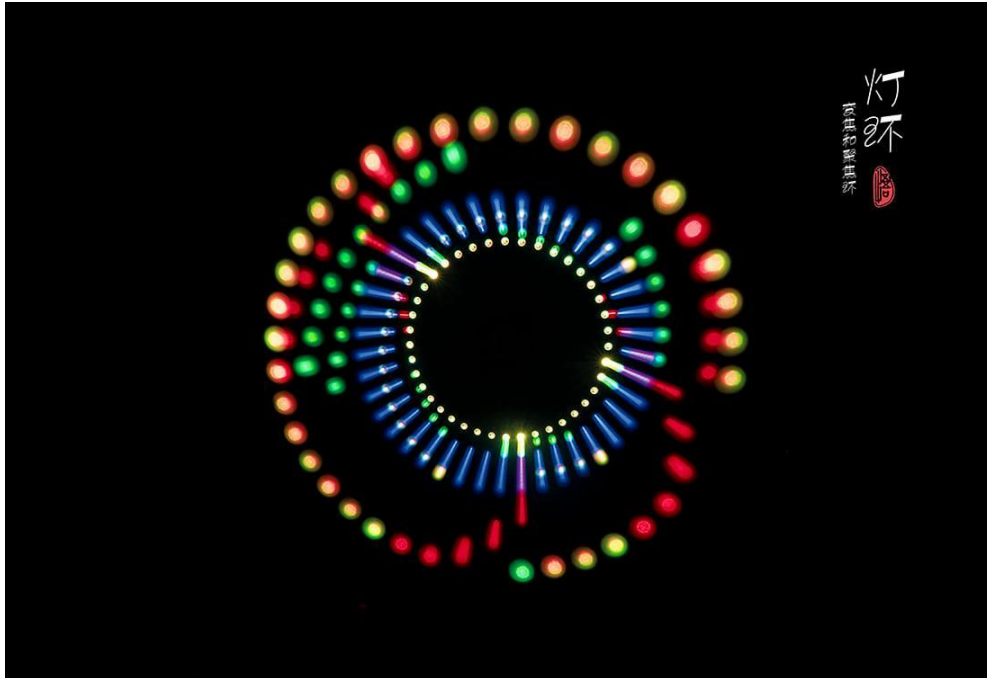


Figure 68: The result of using 'zoom to min & focus to max' [24]

The resulting long-exposed image looked stunning with the different colours and the blurry fade out of the image made it a beautiful art piece.

4.4.2 Capturing of static objects

The following testing were done on a building but with colourful lights painting the Old Singapore Supreme Court Building.



Figure 69: The Old Supreme Court Building taken without long exposure [25]

The 1st sequence used for the long exposure shoot was ‘Zoom and Focus to maximum’ (see ZoomFocus Movements).



Figure 70: Old Supreme Building with colourful scaffoldings [26]

The DSLR camera zooms in while also blurring the image resulted in the image above. The foreground was completely normal, but the background of the Old Supreme Building was distorted and blurred out.

The next sequence used was the opposite of the sequence used previous, where both the zoom and focus rings rotates to the minimum. The image gets zoomed out and the foreground gets blurry.



Figure 71: Old Supreme Court Building painted pink [26]

The images captured of static objects with moving background (in this case, the lights), were outstanding. Furthermore, all of this were done with a simple push of a button, which was to select the chosen pattern.

5. Challenges and Issues

Even though I was a Computer Engineering student, I had never had any experience with hardware. Therefore, a lot of hardware was lost due to short circuit or mishandling of the equipment. However, after reading and learning about the Dos and Don'ts, I was able to avoid destroying any more hardware.

Also, the lack of knowledge of motors was the main cause to a major setback to the timeline that I first had in mind. Understanding why the motor could not rotate the DSLR lens. With the help of online videos to grasp about torque, I was able to change the motor used as well as the driver resulting in a working motor that could now rotate the lens. It was definitely a huge success.

The main problem that I had due to the TFT display were the huge amount of program storage space used in the Arduino Microcontroller. I had many pertaining problems that involves insufficient storage, and I was required to shorten the code or made it as efficient as possible. More often than naught, I had to recode the entire portion to make the code function as dynamic as possible. A total of 3 recode to the entire project had to be done to remove any redundancies in the code and the approach assisted by following procedural programming.

6. Conclusions

To reiterate, this project was designed and implemented to produce a system capable in conducting a long exposure shoot with unique sequences to get the optimal long-exposure image results. Through the integration of hardware and software, high precision, and consistency of the DSLR lens rotation could be achieved. This unfolds an endless possibility for both experienced and aspiring photographers in long exposure photography.

From the confident results gathered during testing, this project can produce a consistent and repeatable long exposure image. Paired with the various patterns extended with the custom pattern that user can input, various beautiful long exposed images can be taken.

Despite the current standings of the project, there are many further optimisations that can be made. In the PCB, there are connectors for 4 limit switches that can be used to further improve the calibration of the motors enhancing the system. Furthermore, the project was coded in a way where future developments can be integrated and added in such as directly controller the DSLR camera settings itself.

References

- [1] F. Gola, "Long Exposure Photography: A step-by-step guide," Digital Photography School, [Online]. Available: <https://digital-photography-school.com>. [Accessed March 2022].
- [2] Wirestock, Artist, *Traffic with long exposure light trails of cars*. [Art]. Freepik.
- [3] tawatchai07, Artist, *Lotte world amusement park at night and cherry blossom*. [Art]. Freepik.
- [4] Masterclass, "8 Tips for Shooting Long Exposure Photography," 8 November 2020. [Online]. Available: <https://www.masterclass.com/>. [Accessed March 2022].
- [5] S. L. Y. & T. B. Kannappan, "Human consistency evaluation of static video summaries.," *Multimedia Tools and Applications*, vol. 78, no. 9, pp. 12281-12306, 2019.
- [6] Format Team, "Here's How To Master The Art Of Long Exposure Photography," Format Magazine, 21 February 2019. [Online]. Available: <https://www.format.com/magazine/resources/photography/long-exposure-photography>. [Accessed March 2022].
- [7] L. Cole, Artist, *Long exposure photography tips*. [Art].
- [8] *Moving Your Lens During Exposure: Photo on the Go with Joe McNally*. [Film]. United States of America: Adorama Photography TV, 2014.
- [9] B. O. Carroll, "A Comprehensive Beginner's Guide to Aperture, Shutter Speed, and ISO," PetaPixel, 25 July 2016. [Online]. Available: <https://petapixel.com/aperture-shutter-speed-iso-beginners-guide-photography/>. [Accessed March 2022].
- [10] S. March, Artist, *Seamless Follow Focus Gears*. [Art].
- [11] J. Waddel, Artist, *The Cascading Costs of Waterfall*. [Art]. 2019.
- [12] S. Lewis, "Waterfall Model," Tech Target, February 2019. [Online]. Available: <https://www.techtarget.com/>. [Accessed March 2022].
- [13] Arduino, Artist, *Arduino Nano*. [Art].
- [14] Hiwonder, Artist, *Hiwonder LD-25MG Strong Torque Digital Servo 25KG Supports Robotic Arm/RC Car/Single Shaft*. [Art].
- [15] Cambridge, *Torque*, Cambridge Dictionary, 1995.
- [16] E. Hup, Artist, *Eng Hup Aluminium Glass Trading*. [Art].
- [17] ccox, Artist, *Customizable Stepper Motor Mount*. [Art]. 2017.
- [18] D. Sawicz, "Hobby Servo Fundamentals," 2012.
- [19] StepperOnline, Artist, *Nema 17 Motors*. [Art].

- [20] Polulu, “Stepper Motor Drivers,” [Online]. Available: <https://www.pololu.com/category/120/stepper-motor-drivers>. [Accessed March 2022].
- [21] K. McEnaney, *Boost Your Photography: Learn Your DSLR*, Amazon Kindle, 2013.
- [22] Neewer, Artist, *Neewer 2.5mm-N1 Off Camera Remote Shutter Release Connecting Cord Cable for Nikon*. [Art].
- [23] A. C. L. Tien, Artist, *Circular Light Zooming to the Max*. [Art]. Nanyang Technological University, 2021.
- [24] A. C. L. Tien, Artist, *The finest lights*. [Art]. Nanyang Technological University, 2021.
- [25] S. Views, Artist, *Photo of City Hall Building*. [Art]. 2016.
- [26] A. C. L. Tien, Artist, [Art]. Nanyang Technological University, 2022.

Appendix A

Circuit Diagram of Project (Stepper Motors)

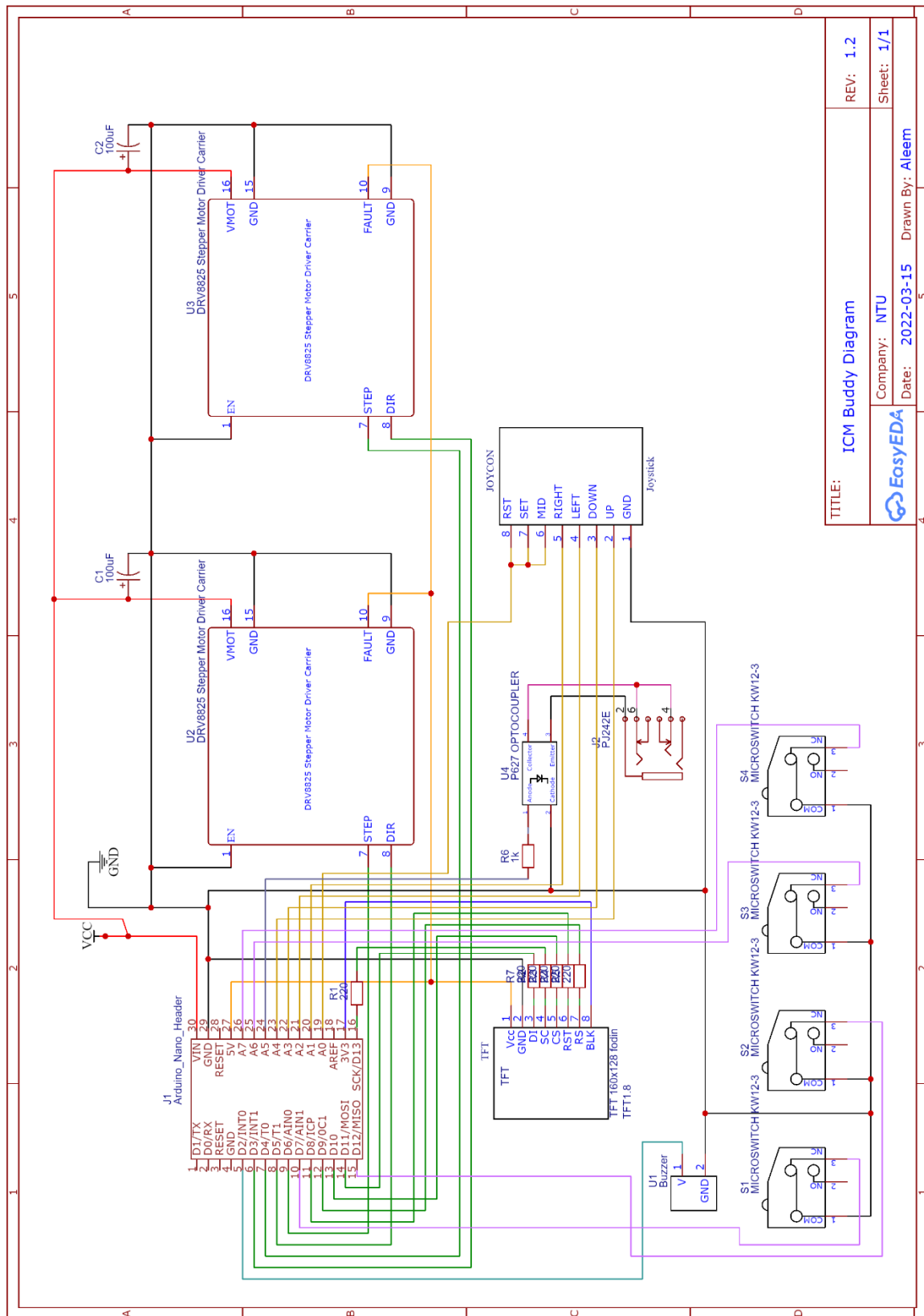


Figure 72: Circuit Diagram of Project (Stepper Motors)

Appendix B

Circuit Diagram of Project (Servo Motors)

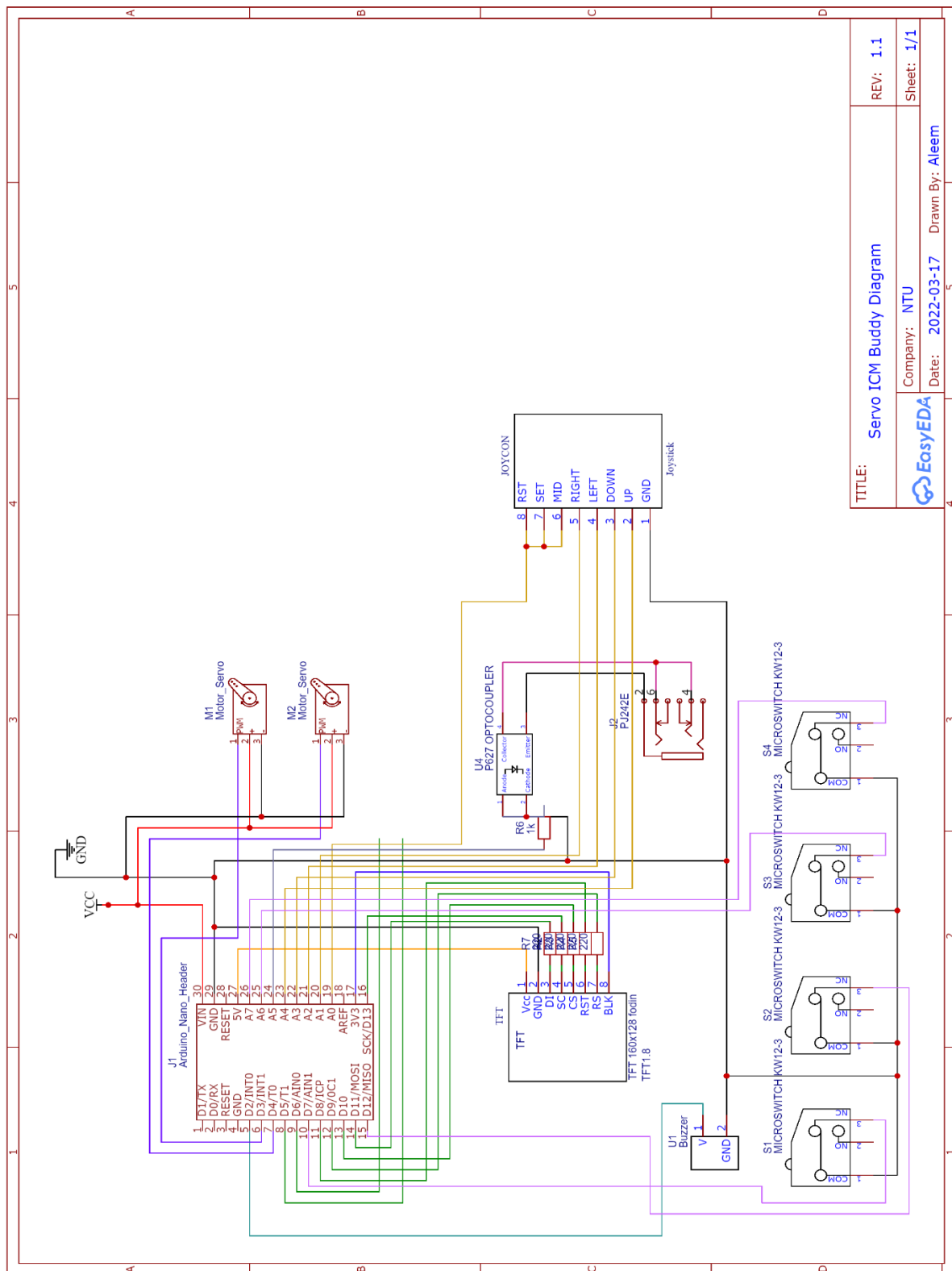


Figure 73: Circuit Diagram of project (Servo Motors)

Appendix C

Different Layers of the PCB

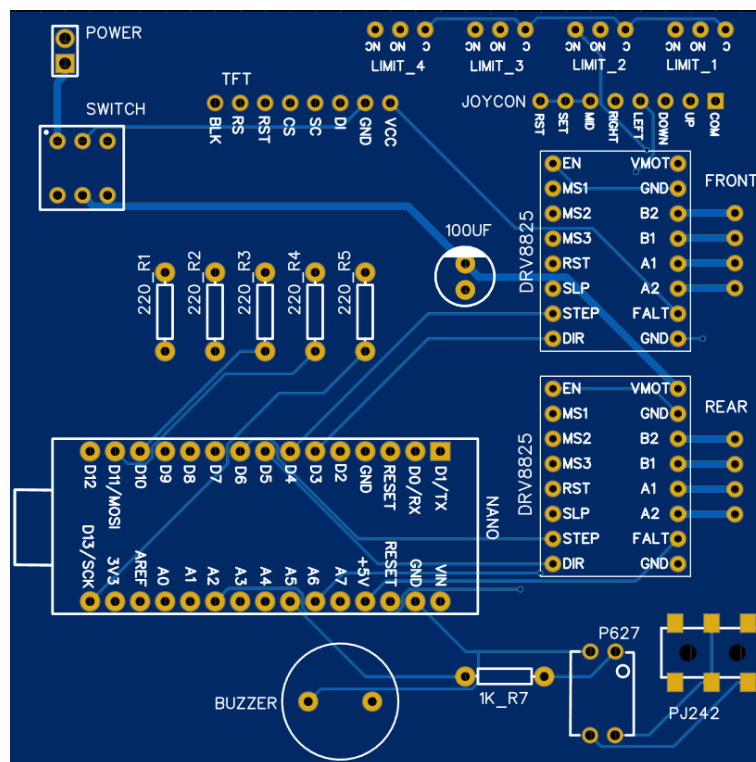


Figure 74: Top Layer of PCB

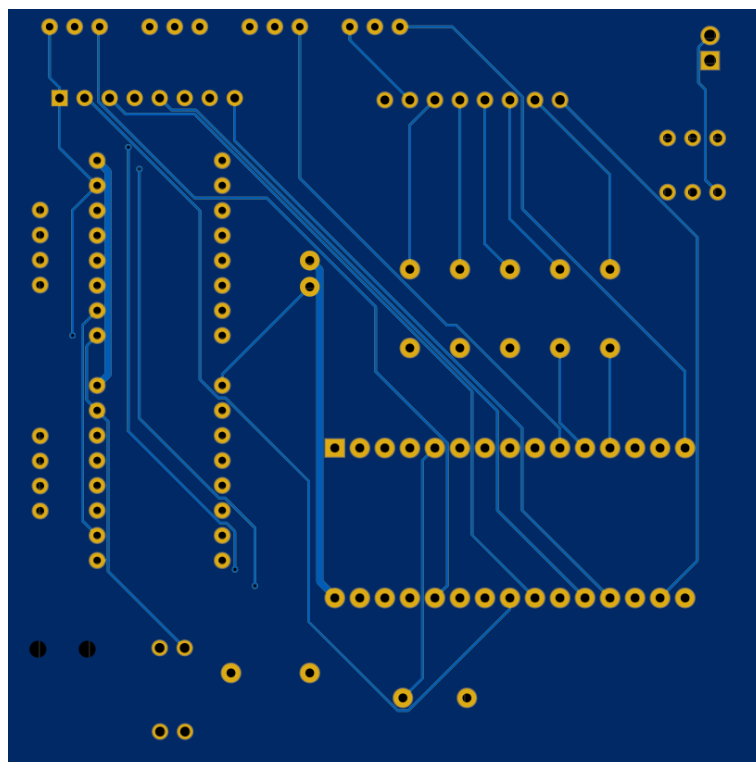


Figure 75: Bottom Layer of PCB

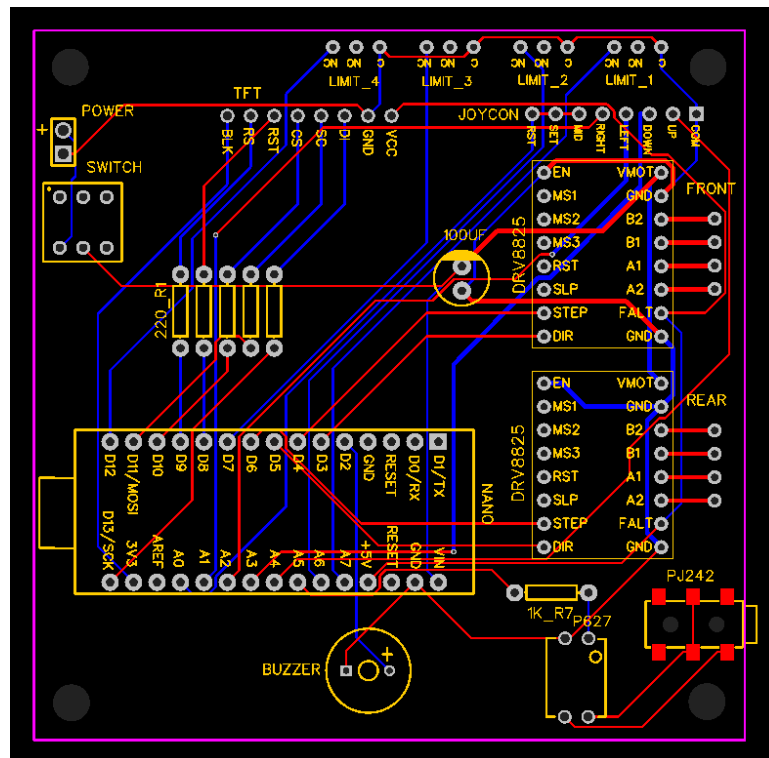


Figure 76: Merged Layer of PCB

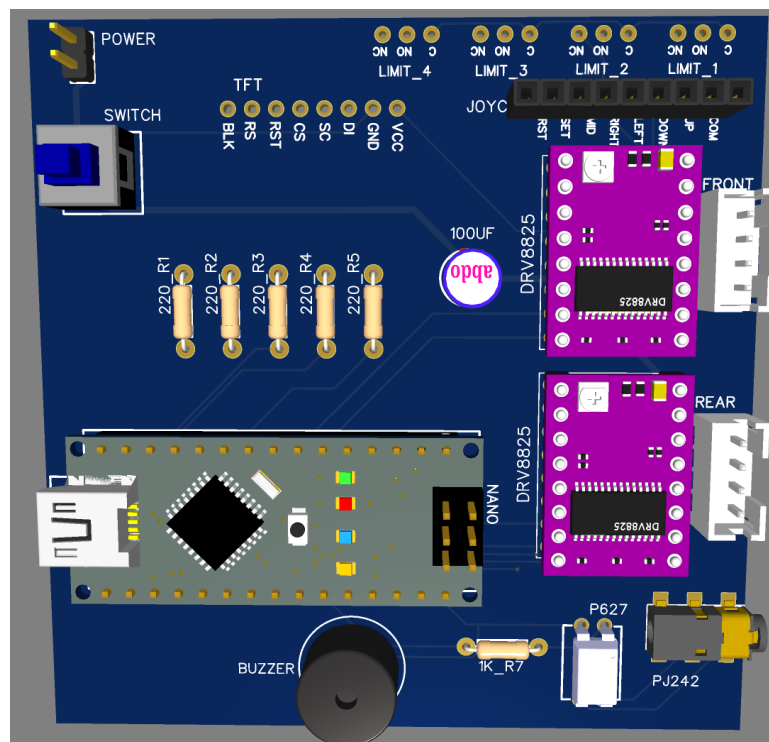


Figure 77: 3D rendered of PCB with components

Appendix D

Available Pre-sets

Focus Movements

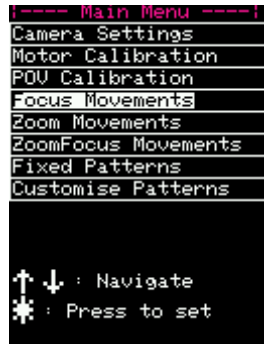


Figure 78: Focus Movements is selected



Figure 79: The different patterns available

Zoom Movements



Figure 80: Zoom Movements is selected



Figure 81: The different patterns available

ZoomFocus Movements

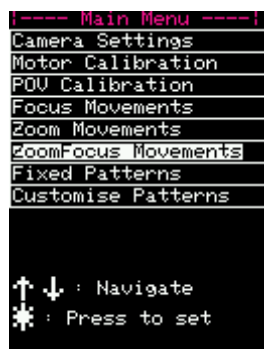


Figure 82: ZoomFocus Movements is selected

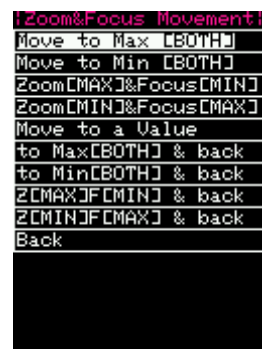


Figure 83: The different patterns available

Fixed Patterns

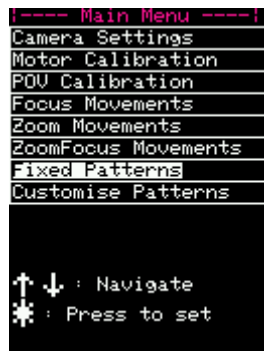


Figure 84: Fixed Patterns is selected

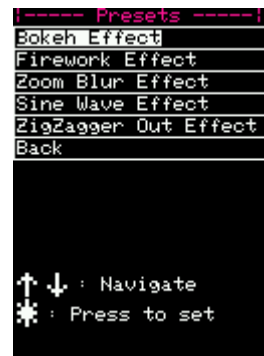


Figure 85: The different patterns available