Mechatronics Design, Fall 2022
Project Final Report
Team Members: Samuel Ang, Carlos Karve, Jingzhou Song, Manraj Gill

Opportunity

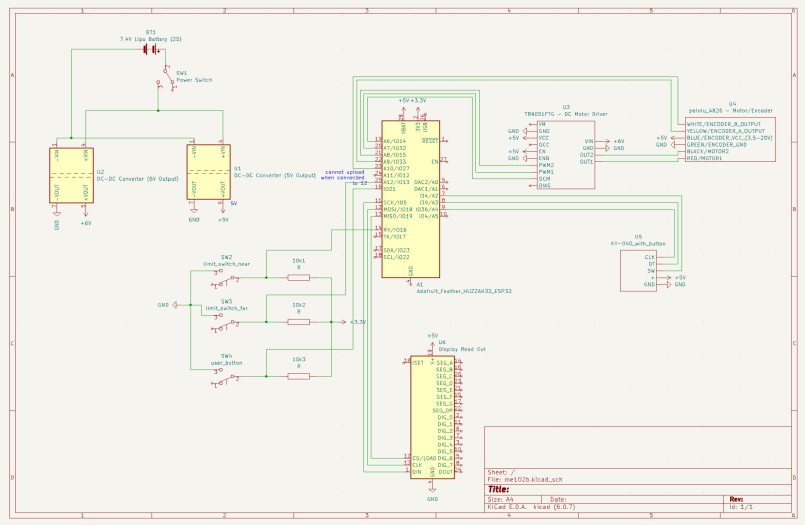
According to the World Health Organization (WHO), a third of the global population was short-sighted in 2020. Routine eye examinations may cost upwards of $200 in the US and access to reliable testing remains sparse in developing countries. With this in mind, our team sought out to create a portable, low cost myopia testing device in hopes of enabling universal testing without geographical constraint.

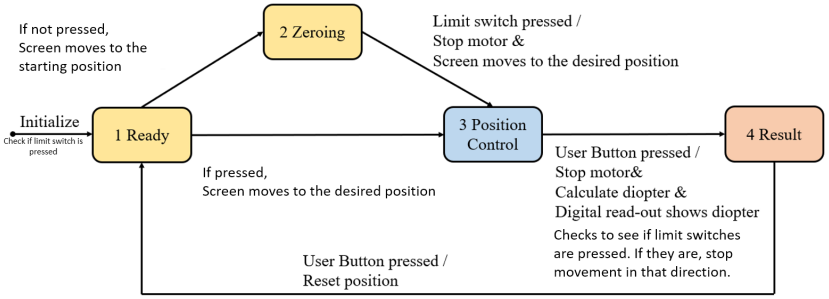
High Level Strategy and Achieved Specifications Comparison

When turned on, the device will enter ‘zeroing’ mode, where the sliding carriage is translated until it hits the far limit switch. This translation is done by means of a ball screw attached to a brushed DC motor. Once the far limit switch is engaged, the device enters ‘position control’. The user will twist the knob to translate the sliding carriage while looking through the lens until the image reaches maximum clarity, and then depress the knob. The device then enters ‘result’ mode, where the user’s diopter is displayed on a 7-segment display. Depressing the knob again returns the device to position control mode.

Our device generally functioned as intended. However there were some inaccuracies in our diopter reading as we were unable to control for the distance between the user’s eye and the lens. Therefore a user with a prescription of -3.00 may have obtained a reading of -4.50.

Circuit Diagram & State Diagram

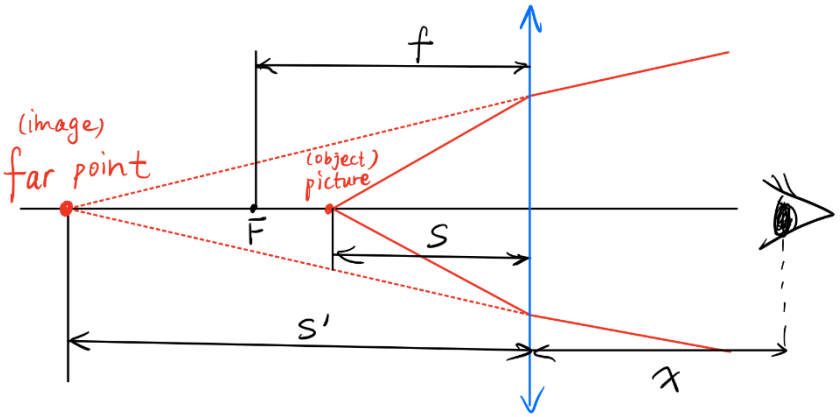




Function Critical Decisions

- Changing DC Brushed Motor from Pololu 4826 to Pololu 2215 to increase speed of translation, while maintaining sufficient torque
- Using a ball screw instead of lead screw
- Using 80-20 Aluminium T-slots for frame rigidity
- Using a fixed lens with known focal length and translating the image (see diopter calculations)

Calculations for diopter



-f: Focal length of the lens
-s: Distance between the picture (object) and the lens
-s': Distance between the far point (image) and the lens
-x: Distance between the eyes and the lens

When you check your eyesight, you need to move the picture from far to close slowly until you can see it clearly and stop immediately. At this time, the position of the image is the furthest distance you can see, that is, the far point. Our encoder will measure s, and given the focal length of the lens f, we can calculate the position s' of the image using the lens imaging formula:

$$\frac{1}{s} + \frac{1}{s'} = \frac{1}{f}$$

$$s' = \frac{1}{\frac{1}{f} - \frac{1}{s}}$$

We want the image to be on the same side as the object, so s' has to be less than 0:

$$s' = \frac{1}{\frac{1}{f} - \frac{1}{s}} < 0$$

So, we get:

$$0 < s \leq f$$

The diopter is the inverse of the distance between the far point and the eyes, so that:

$$diopter = \frac{1}{s' - x} = \frac{1}{\frac{1}{\frac{1}{f} - \frac{1}{s}} - x} < 0$$

Health eyes' diopter is 0, so that:

$$\left(s' - x\right) \to -\infty,\ s' \to -\infty,\ s = f$$

In our project design, the given parameters are:

$$f = 0.15m,\ x = 0.02m$$

With the measurement of s, we can get the diopter. Some particular diopter with several s:

s(mm)	150	140	130	120	110	100	90	80	70	60	50
diopter	0	-0.47	-1.00	-1.61	-2.31	-3.125	-4.08	-5.22	-6.61	-8.33	-10.52

Reflection and future process/ design improvements

To measure a person's diopter more accurately, we could have built a chin-rest to keep the user's eye steady. This would also control for differences in head placement among different users. That said, we believe that we achieved our aim of developing a simple, low-cost and easy-to-use device to measure the diopter of someone with myopia. With some refinement, this product could actually prove useful in rural communities, given that it does not require an optometry professional to operate and has very simple user controls.

Appendix A: BOM

QTY	Part #	Description	Assembly	Vendor	Total Price (USD)	Remarks
Off-the-shelf components						
1	4826	25-d-metal-gearmotor-34-47-encoder	Transmission	Pololu	$45.21	
1	2676	25-d-mm-metal-gearmotor-bracket	Transmission	Pololu	$7.95	
1	765154263632	300MM Ball Screw SFU1204 RM1204 12mm with Metal Deflector Ballscrew nut BK/BF10 End Supports Ball NUT HOUSINGS 1pcs Coupler: includes SFU 1204 Ball Screw Nut, 3899_BF_10 (Floated End), 3894_BK_10 (Support Integrated With Motor Bracket), BSS1204-300 (Precision Ball Screw, Standard Nut), dsg 12 (Nut Housing)	Transmission	Amazon	$32.99	
1	-	Clamping Precision Flexible Shaft Couplings - 4mm to 8mm	Transmission	Amazon	$10.99	
				Subtotal	$97.14	
Fabricated components						
1	5537T514	T-Slotted Framing, Single Four Slot Rail, Silver, 20 mm Square, Solid, 10’ total length (Saw cut to length)	Supporting Frame	McMaster	$41.17	
4	-	Image holder brackets	Sliding image holder	-	-	3D Printed
1	-	7.5 mm Base Plate (Pololu Bracket Spacer)	Supporting Frame	-	-	3D Printed

4	9146T53	Multipurpose 6061 Aluminum Sheets and Bars: 40mm x 1ft	Box Frame	McMaster	$42.32	
1	-	Clear Acrylic - 1/4" x 16" x 32" × 1	Supporting Frame	Jacobs Hall Material Store	$38.85	
2	-	Plywood - 1/4" x 18" x 30" × 2	Box Frame	Jacobs Hall Material Store	$12.50	
				Subtotal	$151.67	
Fasteners, brackets, etc						
1(100)	-	M5x0.8 hex nut	Misc	Amazon	$8.99	
1(100)	-	M5 T-slot extrusion nut	Misc	Amazon	$13.99	
1(100)	-	M5 washer	Misc	Amazon	$7.99	
16	-	M5x0.8 16mm hex screw	Box Frame, supporting frame	Ace Hardware	$9.60*	
6	-	M5x0.8 25mm hex screw	Box Frame	Ace Hardware	$3.60*	
6	-	M5x0.8 40mm hex screw	Transmission	Ace Hardware	$3.60*	
6	-	M4x0.7 12mm hex screw	Transmission	Ace Hardware	$3.60*	

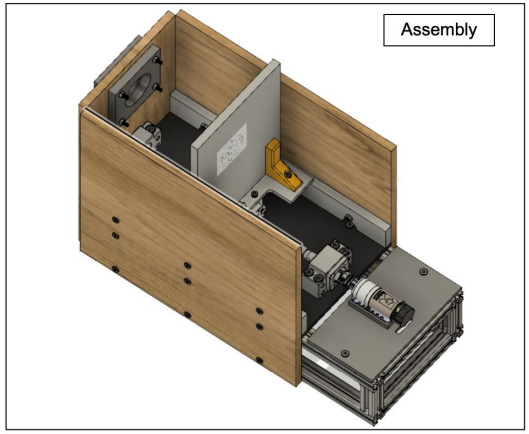
8	-	M5x0.8 12mm hex screw	Supporting frame	Ace Hardware	$4.80*	
2	-	M3x0.5 20mm hex screw	Supporting frame	Ace hardware	$1.20*	
2	-	M3x0.5 hex nut	Supporting frame	Ace hardware	$1.20*	
4	-	M4x0.7 35mm hex screw	Supporting frame (lens mount)	Ace hardware	$2.40*	
4	-	M4x0.7 20mm hex screw	Sliding image holder	Ace hardware	$2.40*	
4	-	M4x0.7 12mm hex screw	Sliding image holder	Ace hardware	$2.40*	
8	-	M4x0.7 hex nut	Sliding image holder	Ace hardware	$4.80*	
2(20)	-	Black Inside Corner Bracket Gusset with T-Slot Nuts & Screws	Supporting frame	Amazon	$30.60	
				Subtotal	$63.97	
Electronics						
QTY	**Part #**	**Description**	**Assembly**	**Vendor**		
2	SW767-ND	Limit Switches	Box Frame	Digikey	$3.00	
1	EG5617-ND	Power Button	Box Frame	Digikey	$0.67	
2	CKN4082	User Input Buttons	User Input	Digikey	$5.14	

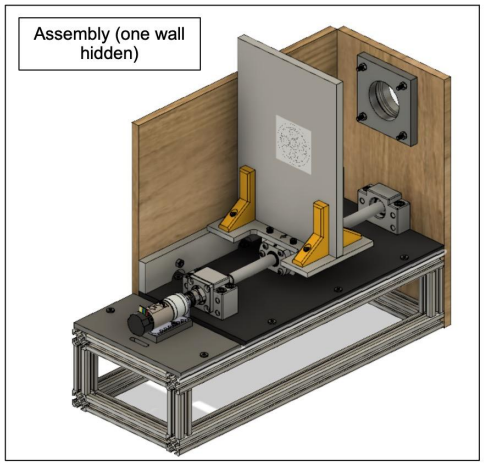
	-ND		Board			
1	118-PEC12R-4215F-N0024-ND	Rotary Encoder	User Input Board	Digikey Amazon	$1.25 $9.20	
1	MAX7219	8-Digit 7 Segment Display	Box Frame	Amazon	$11.92	
1	HiLetgo ESP-WROOM-32	ESP-32S Development Board	Electronics Board	Amazon	n/a	
1	5x7cm Bakelite DIY Prototype Board	Prototype Board	Electronics Board	Amazon	$6.99	
1	Header Connector Assortment Kit	Header Pin set	Electronics Board	Amazon	$8.99	
1		Battery				
1		DC-DC converter				
				Subtotal	$37.96	
				Grand total:	$350.74	

Appendix B: CAD Model & Diagrams

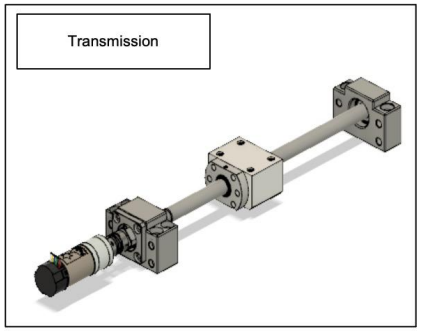
Final Prototype CAD model

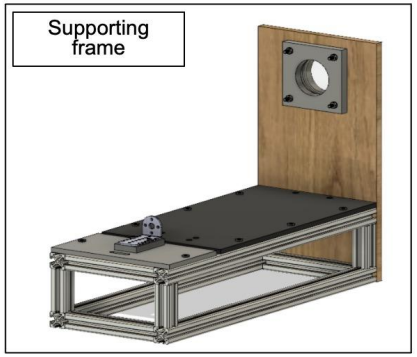
1. Assembly Overview

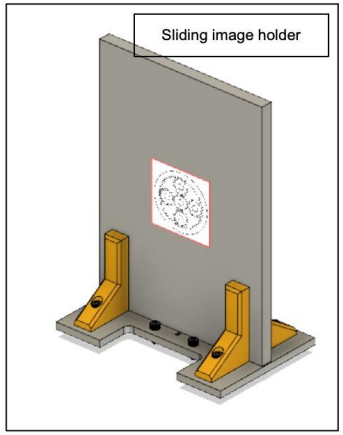




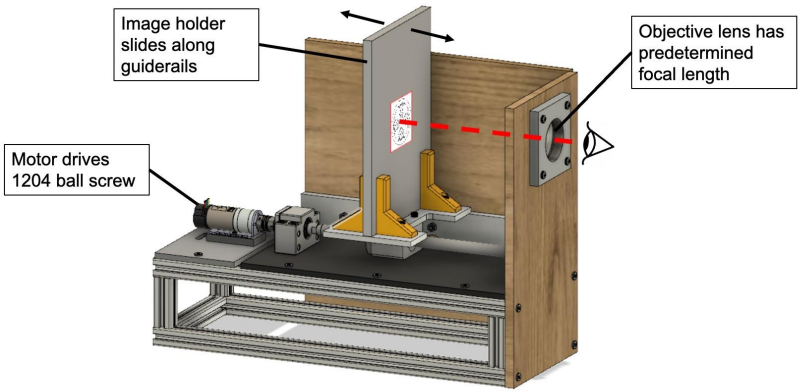
2. Parts Breakdown



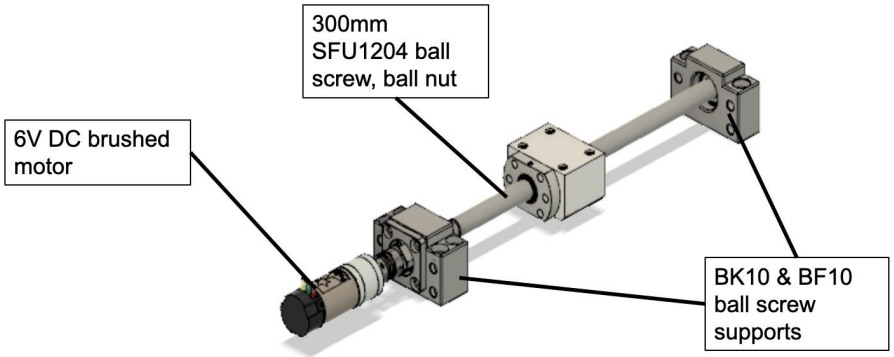




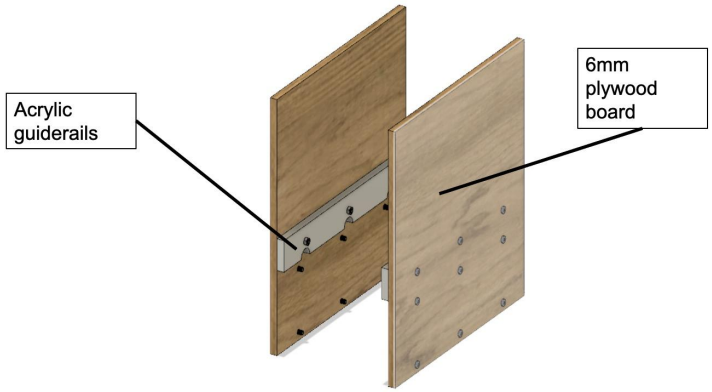
3. Functional breakdown

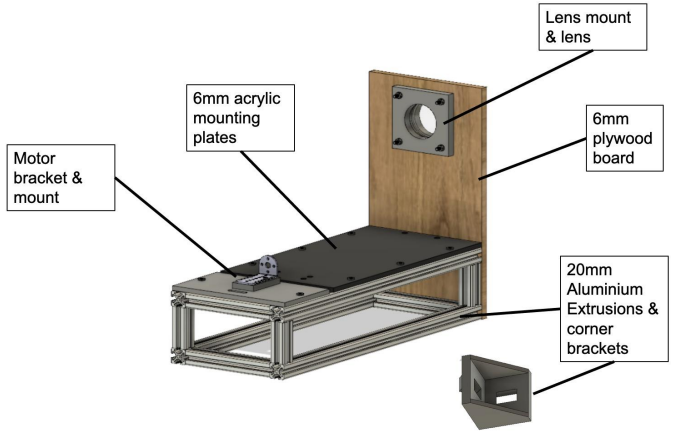


4. Transmission components

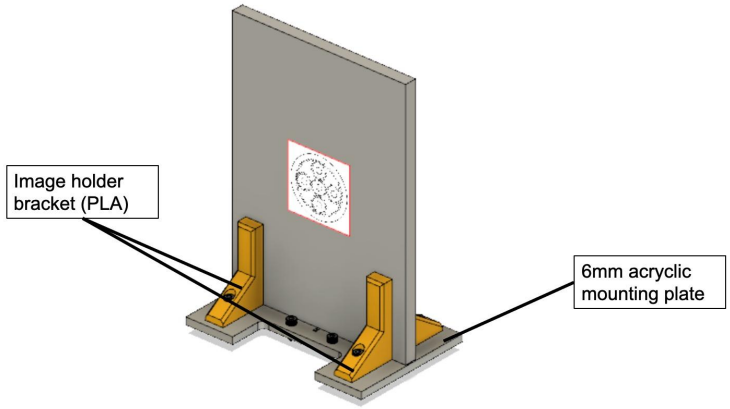


5. Frame Components





6. Image Holder Components



Appendix C: Code

```
#include <Arduino.h>
#include <ESP32Encoder.h>
#include <SPI.h>

//INPUTS -----------------------------------

#define SWT_FAR 13  // declare the limit switch far pin number
#define SWT_NEAR 16  // declare the limit switch near pin number
#define ROT_SW 34 // switch aka BTN1 (rotary encoder)
#define ROT_CLK 36 // CLK (rotary encoder)
#define ROT_DT 39 // DT (rotary encoder)
#define BTN2 21 // declare the user button2 (yellow button) pin number
#define ENCA 33 // encoder A (motor)
#define ENCB 27 // encoder B (motor)

//OUTPUTS ----------------------------------
#define PWM1 32 // PWM1 (motor driver)
#define PWM2 15 // PWM2 (motor driver)
#define MAX7219_CS 19 // CS/LOAD (digital read out)
#define MAX7219_CLK 18 // CLK (digital read out)
#define MAX7219_DIN 5 // DIN (digital read out)

// DISPLAY SETUP CODE ------------------------------------
enum {  MAX7219_REG_DECODE    = 0x09,
        MAX7219_REG_INTENSITY = 0x0A,
        MAX7219_REG_SCANLIMIT = 0x0B,
        MAX7219_REG_SHUTDOWN  = 0x0C,
        MAX7219_REG_DISPTEST  = 0x0F };

enum  { OFF = 0, ON  = 1 };

const byte DP = 0b10000000;
const byte neg = 0b00001010;
const byte H = 0b00001100;
const byte E = 0b00001011;
const byte L = 0b00001101;
```

```
const byte O = 0b00000000;
const byte blank = 0b00001111;


// read datasheet, https://cdn-shop.adafruit.com/datasheets/MAX7219.pdf change D3-D0

int length_of_float = 8;
int decimal_places = 2;

char DRO_string[9];

void set_register(byte reg, byte value)
{
    digitalWrite(MAX7219_CS, LOW);
    shiftOut(MAX7219_DIN, MAX7219_CLK, MSBFIRST, reg);
    shiftOut(MAX7219_DIN, MAX7219_CLK, MSBFIRST, value);
    digitalWrite(MAX7219_CS, HIGH);
}

void resetDisplay()
{
    set_register(MAX7219_REG_SHUTDOWN, OFF);   // turn off display
    set_register(MAX7219_REG_DISPTEST, OFF);   // turn off test mode
    set_register(MAX7219_REG_INTENSITY, 0x0D); // display intensity
}

void displayHello(){
    set_register(MAX7219_REG_SHUTDOWN, OFF);  // turn off display
    set_register(MAX7219_REG_SCANLIMIT, 7);   // scan limit 8 digits
    set_register(MAX7219_REG_DECODE, 0b11111111); // decode all digits

    set_register(8, H);
    set_register(7, E);
    set_register(6, L);
    set_register(5, L);
    set_register(4, O);
    set_register(3, blank);
```

```
    set_register(2, blank);
    set_register(1, blank);
    set_register(MAX7219_REG_SHUTDOWN, ON);   // Turn on display

}

void displayValue_int(String timeString){
    set_register(MAX7219_REG_SHUTDOWN, OFF);  // turn off display
    set_register(MAX7219_REG_SCANLIMIT, 7);   // scan limit 8 digits
    set_register(MAX7219_REG_DECODE, 0b11111111); // decode all digits

    for (int i = 1; i < length_of_float+1; i++)
    {
      if (i < decimal_places + 1){
        set_register(i, timeString.charAt(length_of_float + 1 - i));
      }
      else if (i == decimal_places + 1){
        set_register(i, timeString.charAt(length_of_float - i) | DP);
      }
      else if (i > decimal_places + 1){
        if (timeString.charAt(length_of_float - i) == '-'){
          set_register(i, neg);
        }
        else{
          set_register(i, timeString.charAt(length_of_float - i));
        }
      }
    }
    set_register(MAX7219_REG_SHUTDOWN, ON);   // Turn on display
}

void displayValue(float value){
  char string[length_of_float + 1];
  dtostrf(value, length_of_float + 1, decimal_places, string);
  displayValue_int(string);
}
```

```
// FUNCTIONS ----------------------------------
ESP32Encoder encoder;

//Setup variables ----------------------------------
int state = 1;

const int SwitchDebounceT = 100; //ms
const int ButtonDebounceT = 50; //ms
const int MainLoopDeltaT = 10; //ms


volatile bool SwitchIsPressed_NEAR = false;
volatile bool SwitchIsPressed_FAR = false;

hw_timer_t * timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;

volatile bool ButtonIsPressed = false;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

volatile bool MainLoopT = false;
hw_timer_t * timer2 = NULL;
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;

// Prototypes ----------------------------------------
void rotary_encoder();
void motor_CW();
void motor_CCW();
void motor_stop();
void move_motor();
void move_motor_target();
int pwm_motor_speed(int abs_error);


//Setup variables ----------------------------------
volatile int enc_count = 0; // encoder count
```

```
// variables for rotary encoder
volatile boolean TurnDetected;
volatile boolean up;
static long virtualPosition=0;    // without STATIC it does not count correctly!!!

const int freq = 5000;
const int ledChannel1 = 0;
const int ledChannel2 = 1;
const int resolution = 8;

// variables for motor
bool move = false;
int target_position = 0;
int target_position_old = 0;
int current_position = 0;

int encoder_step = 50;
int goal_tolerance = 10;
int abs_goal_error = 0;
int max_error = 1000;
const int MIN_PWM_VOLTAGE = 80;
const int MAX_PWM_VOLTAGE = 255;

// float Kp = (MAX_PWM_VOLTAGE-50)/max_error;

//variables for diopter
float focal_length = 0.15; //m
float lens_to_eye = 0.02; //m
float total_length = 0.20; //measure the far_switch to lens
float object_to_lens = total_length;
float max_encoder_count = 12000;
float screw_ratio = total_length / max_encoder_count;  //We need to measure it, object_to_lens = total_length - CurrentPosition
* screw_ratio
float diopter = 0;

//Initialization -----------------------------------
```

```
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  portEXIT_CRITICAL_ISR(&timerMux0);
}
void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  ButtonIsPressed = true;                                  // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}
void IRAM_ATTR onTime2() {
  portENTER_CRITICAL_ISR(&timerMux2);
  MainLoopT = true;                                 // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux2);
}
void TimerInterruptInit0() {                        //With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer0 = timerBegin(0, 80000, true);                 // divides the frequency by the prescaler: 80,000,000 / 80000 = 1,000 tics
/ sec, 1ms
  timerAttachInterrupt(timer0, &onTime0, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(timer0, SwitchDebounceT, true);          // sets how many tics will you count to trigger the interrupt, 200 * 1
us = 200 ms
  timerAlarmEnable(timer0);                         // Enables timer
}
void TimerInterruptInit1() {                        //With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer1 = timerBegin(1, 80000, true);                 // divides the frequency by the prescaler: 80,000,000 / 80000 = 1,000 tics
/ sec, 1ms
  timerAttachInterrupt(timer1, &onTime1, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(timer1, ButtonDebounceT, true);          // sets how many tics will you count to trigger the interrupt, 200 *
1ms = 200 ms
  timerAlarmEnable(timer1);                         // Enables timer
}
void TimerInterruptInit2() {                        //With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer2 = timerBegin(2, 80000, true);                 // divides the frequency by the prescaler: 80,000,000 / 80000 = 1,000 tics
/ sec, 1ms
  timerAttachInterrupt(timer2, &onTime2, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(timer2, MainLoopDeltaT, true);          // sets how many tics will you count to trigger the interrupt, 200 *
1ms = 200 ms
  timerAlarmEnable(timer2);                         // Enables timer
```

```
}

void IRAM_ATTR SWT_NEAR_isr() {  // the function to be called when interrupt is triggered
  timerStart(timer0);
  SwitchIsPressed_NEAR = true;
}

void IRAM_ATTR SWT_FAR_isr() {  // the function to be called when interrupt is triggered
  timerStart(timer0);
  SwitchIsPressed_FAR = true;
}

void IRAM_ATTR ROT_SW_isr() {  // the function to be called when interrupt is triggered
  timerStart(timer1);
}

void IRAM_ATTR ROT_ENC_isr ()  {                    // Interrupt service routine is executed when a HIGH to LOW transition is
detected on CLK
  if (digitalRead(ROT_CLK))
    up = digitalRead(ROT_DT);
  else
    up = !digitalRead(ROT_DT);
  TurnDetected = true;
}


bool CheckForSwitchPress() {
   if (SwitchIsPressed_NEAR == true || SwitchIsPressed_FAR == true) {
        return true;}
    else {
        return false;}
}
bool CheckForButtonPress() {
  if (ButtonIsPressed) {
        return true;}
    else {
        return false;}
```

```
}

void SwitchTZeroing() {
        portENTER_CRITICAL_ISR(&timerMux0);
        SwitchIsPressed_NEAR = false;
        SwitchIsPressed_FAR = false;
        portEXIT_CRITICAL_ISR(&timerMux0);
        timerStop(timer0);
}
void ButtonTZeroing() {
        portENTER_CRITICAL_ISR(&timerMux1);
        ButtonIsPressed = false;
        portEXIT_CRITICAL_ISR(&timerMux1);
        timerStop(timer1);
}
void MainLoopTZeroing() {
        portENTER_CRITICAL_ISR(&timerMux2);
        MainLoopT = false;
        portEXIT_CRITICAL_ISR(&timerMux2);
}

void ControlPosition(){
    rotary_encoder();
    move_motor_target();
    displayValue(float(encoder.getCount()));
}

void rotary_encoder(){
  // if (!(digitalRead(ROT_SW))) {      // check if pushbutton is pressed
  //   virtualPosition=0;              // if YES, then reset counter to ZERO
  // //  Serial.print ("Reset  = ");      // Using the word RESET instead of COUNT here to find out a buggy encoder
  // }
  if (TurnDetected){        // do this only if rotation was detected
    if (up)
      virtualPosition--;
    else
      virtualPosition++;
```

```
    TurnDetected = false;          // do NOT repeat IF loop until new rotation detected
    // Serial.print ("Count = ");
    // Serial.println (virtualPosition);
  }
}

void Reset(){
  virtualPosition = 0;
  diopter = 0;
}

void Zeroing(){
  ledcWrite(ledChannel1, MAX_PWM_VOLTAGE);
  ledcWrite(ledChannel2, 0);
}

void motor_CW(){
  // Serial.println("Direction: Clockwise  ");
  ledcWrite(ledChannel1, pwm_motor_speed(abs_goal_error));
  ledcWrite(ledChannel2, 0);
}
void motor_stop(){
  // Serial.println("Direction: STOP  ");
  ledcWrite(ledChannel1, 0);
  ledcWrite(ledChannel2, 0);
}
void motor_CCW(){
  // Serial.println("Direction: Anti-Clockwise  ");
  ledcWrite(ledChannel1, 0);
  ledcWrite(ledChannel2, pwm_motor_speed(abs_goal_error));
  }

void move_motor_target(){
  current_position = encoder.getCount();
  abs_goal_error = abs(target_position - current_position);
  if (virtualPosition != 0 && move == false) {
    target_position = current_position + encoder_step*virtualPosition;
```

```
    target_position_old= target_position;
    move = true;
  }
  else if (virtualPosition != 0 && move == true) {
    target_position = target_position_old + encoder_step*virtualPosition;
  }

  if (((target_position - current_position) > goal_tolerance) && move == true && digitalRead(SWT_FAR) == 1) {
      motor_CCW();
    }
  else if (((current_position - target_position) > goal_tolerance) && move == true && digitalRead(SWT_NEAR) == 1) {
      motor_CW();
    }
  else if (((current_position - target_position) < goal_tolerance) && move == true) {
      motor_stop();
      move = false;
      virtualPosition = 0;
  }
 Serial.print("PWM Motor Speed: ");
 Serial.print(pwm_motor_speed(abs_goal_error));
 Serial.print(" Virtual Position: " );
 Serial.print(virtualPosition);
 Serial.print(" Target position: ");
 Serial.print (target_position);
 Serial.print(" Current position: ");
 Serial.println(current_position);
}

int pwm_motor_speed(int abs_error){
  int pwm = 0;
  if (abs_error > 0 && abs_error < max_error) {
    pwm = map(abs_error, 0, max_error, MIN_PWM_VOLTAGE, MAX_PWM_VOLTAGE);
  }
  else if (abs_error >= max_error) {
    pwm = MAX_PWM_VOLTAGE;
  }
  return pwm;
```

```
}

void CalculateDiopter(){
  object_to_lens = total_length - current_position * screw_ratio;
  diopter = 1 / (1 / (1/focal_length - 1/object_to_lens) - lens_to_eye);
  Serial.print ("Diopter = ");
  Serial.println (diopter);
  resetDisplay();                 // reset the MAX2719 display
  displayValue(float(diopter));   // display the diopter value
}


void setup() {
  pinMode(SWT_FAR, INPUT);
  pinMode(SWT_NEAR, INPUT);
  pinMode(ROT_SW, INPUT);
  pinMode(ROT_CLK, INPUT);
  pinMode(ROT_DT, INPUT);
  pinMode(BTN2, INPUT);
  pinMode(ENCA, INPUT);
  pinMode(ENCB, INPUT);

  pinMode(PWM1, OUTPUT);
  pinMode(PWM2, OUTPUT);
  pinMode(MAX7219_DIN, OUTPUT);   // serial data-in
  pinMode(MAX7219_CS, OUTPUT);    // chip-select, active low
  pinMode(MAX7219_CLK, OUTPUT);   // serial clock
  digitalWrite(MAX7219_CS, HIGH);
  resetDisplay();                 // reset the MAX2719 display
  attachInterrupt(SWT_NEAR, SWT_NEAR_isr, RISING);
  attachInterrupt(SWT_FAR, SWT_FAR_isr, RISING);
  attachInterrupt(ROT_SW, ROT_SW_isr, RISING);
  attachInterrupt (ROT_CLK,ROT_ENC_isr,FALLING);
  ledcSetup(ledChannel1, freq, resolution);
  ledcSetup(ledChannel2, freq, resolution);
  ledcAttachPin(PWM1, ledChannel1);
  ledcAttachPin(PWM2, ledChannel2);
```

```
  TimerInterruptInit0();
  timerStop(timer0);
  TimerInterruptInit1();
  timerStop(timer1);
  TimerInterruptInit2();

  ESP32Encoder::useInternalWeakPullResistors=UP;
  encoder.attachHalfQuad(ENCA, ENCB);
  encoder.setCount(0);  // set starting count value after attaching
  Serial.begin(115200);
  Serial.println("Encoder Start = " + String((int32_t)encoder.getCount()));
  displayHello();
}

void loop() {
  if (MainLoopT) {
    MainLoopTZeroing();
    switch (state) {
      case 1: //Start
        if (SwitchIsPressed_NEAR || digitalRead(SWT_NEAR) == LOW) {
          state = 3;
          Serial.println("1->3 ControlPosition()");
        }
        else {
          state = 2;
          Serial.println("1->2 Zeroing();");

        }
      break;

      case 2: //Zeroing
        Zeroing();
        if (SwitchIsPressed_NEAR) {
          Serial.println("2->3 StopMotor(); ControlPosition();");
          motor_stop();
          encoder.setCount(0);  // set starting count value after attaching
```

```
          state = 3;
        }
      break;

      case 3: //Position Control
        ControlPosition();
        if (CheckForButtonPress()) {
          state = 4;
          Serial.println("3->4 CalculateDiopter(); DROshowDiopter(); StopMotor();");
          CalculateDiopter();
          motor_stop();
        }
      break;

      case 4: //Reset
        if (CheckForButtonPress()) {
          state = 1;
          Serial.println("4->1 Reset();");
          Reset();
          resetDisplay();
        }
      break;
     }
    if (CheckForSwitchPress()){SwitchTZeroing();}
    if (CheckForButtonPress()){ButtonTZeroing();}
   }
}
```