Automated Projector
Group 22 - Austin Nguyen & James Nguyen

1. Opportunity

In this project we aimed to highlight a single product and to improve its core functionality. We have accomplished this with a projector by automating its ability to maintain orientation with respect to an image surface and to remain in focus.

2. Strategy

The project works by attempting to face the projector head on with the wall in order to maintain a rectangular image, and adjusting focus level for image clarity. It utilizes two ultrasonic sensors that are placed next to each other and uses the two analog inputs to ensure that the projector is facing the wall head on by comparing the two sensor readings and rotating accordingly. Our initial desired functionality was to be able to actuate (rotate) the assembly at a low enough speed relative to its range of motion so that it does not actuate too suddenly. We wanted no more than 30 rpm for the rotational aspect and were able to maintain levels below based on adjusting the PWM into the motor. We also wanted the focus toggle actuation to not actuate too much and apply so much torque as to break something in our housing or on the actual projector. At first we wanted to specify an rpm that we would need to be under (around less than 1 rpm), but we realized that we could instead specify a range of encoder values the motor can actuate to in order to effectively reach certain points in our very small range of motion for the focus toggling.

3. Critical Decisions

With a goal of 30 RPM for the rotation transmission, and a gear train with a 2:1 ratio, we need a motor that can rotate at around 60 RPM under the load of the mechanism. The motors we use each have a max 410 RPM with no load. The 2:1 gear ratio results in an output RPM of less than 200 after considering the inertia of the entire mechanism. Because our goal is 30 RPM for the rotation transmission, there is more than enough power being supplied by the motors. We can scale down the PWM signals given to the motors in order to achieve the speeds we want, which is necessary because the RPM at max output is far too high. The gear train also halves the load torque on the motor shafts, effectively doubling the stall torque of our system, which gives our motors even more clearance in overcoming the torque of the device’s inertia.

4. Reflection

One thing that worked for us was that we both knew our strengths/weaknesses and distributed the work in accordance to that pretty well. One thing we wish we had done differently was fabricating physical prototypes earlier on in order to have more time to iterate and expose engineering problems and obstacles that we had to solve. Both party members also took this course along with several other lab/project based classes, resulting in a high workload semester. We recommend finding a better balance of courses to take alongside ME102B, because this course requires significant amounts of time to produce quality assignments, time that we struggled to find.

5. Photos

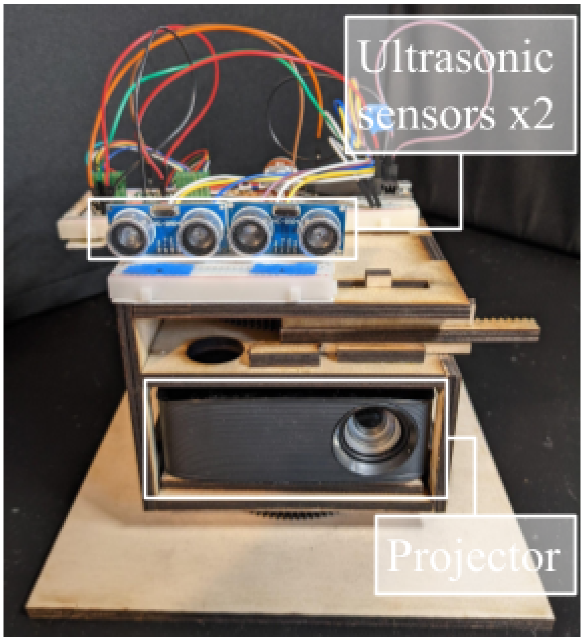


Figure 1. Front view

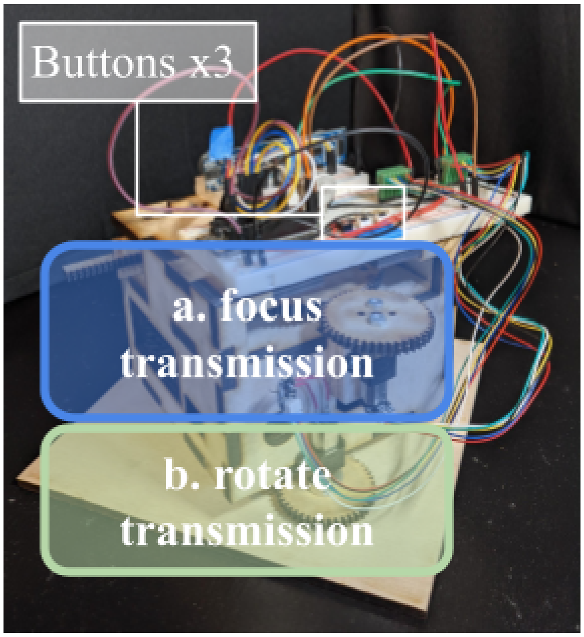


Figure 2. Back view

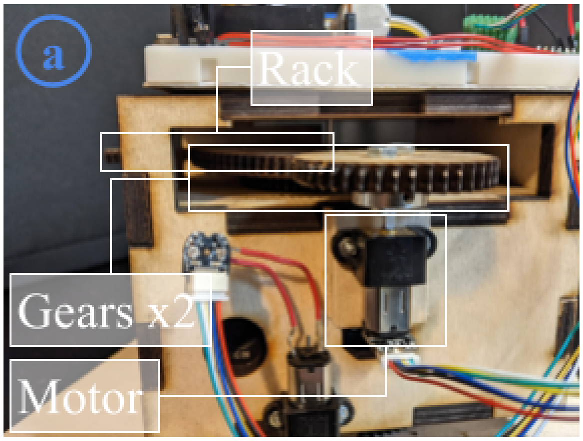


Figure 3. High back view

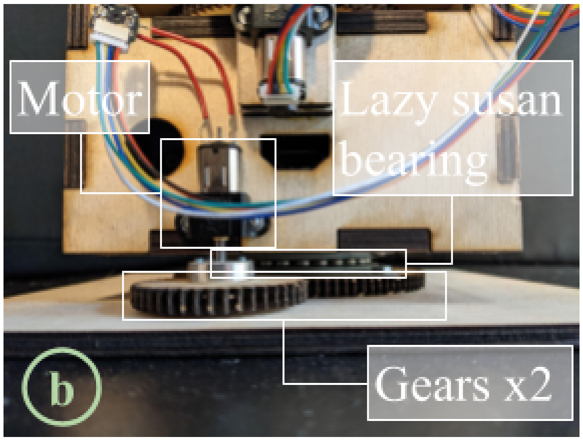


Figure 4. Low back view

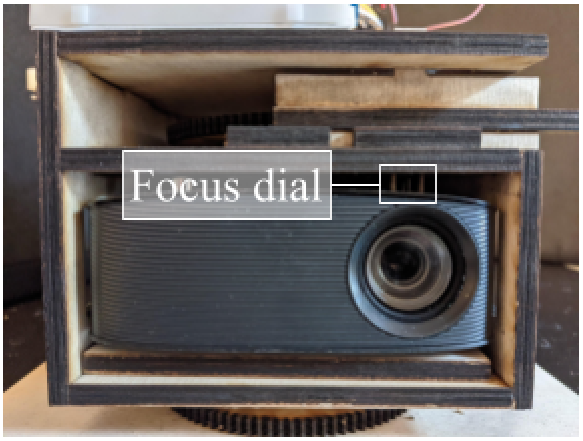


Figure 5. Zoomed front view

6. Circuit & State Machine

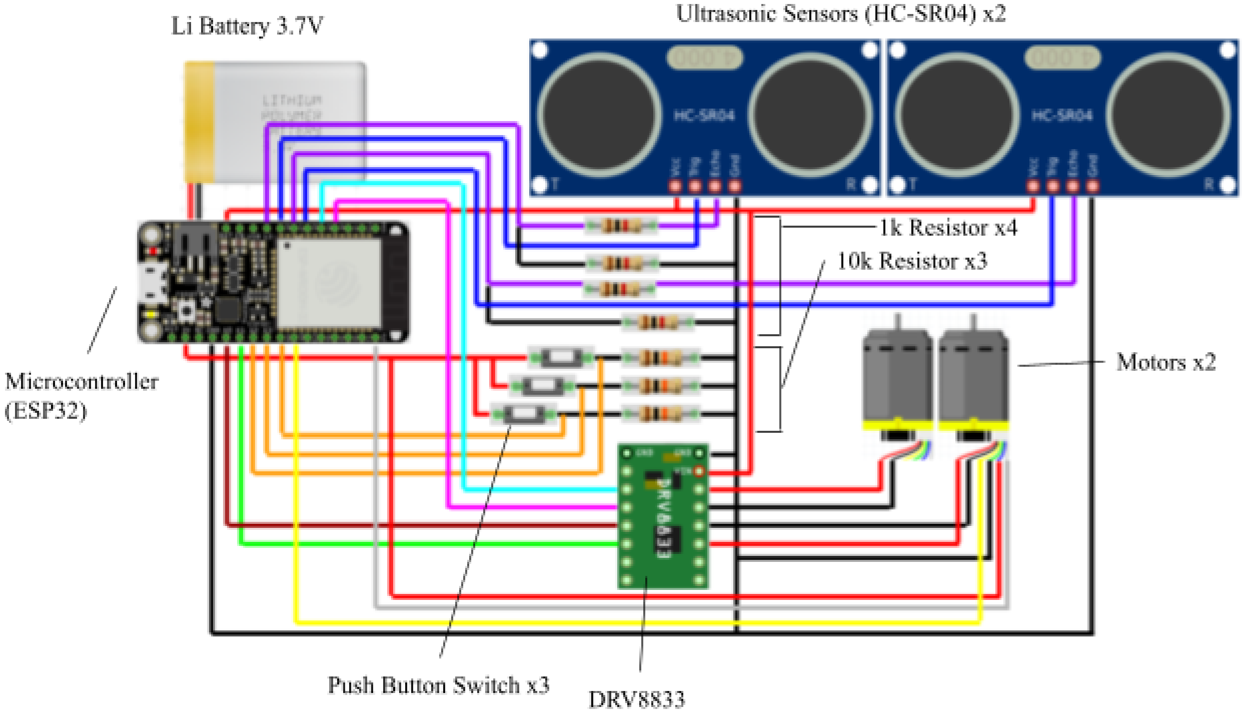


Figure 6. Circuit Diagram

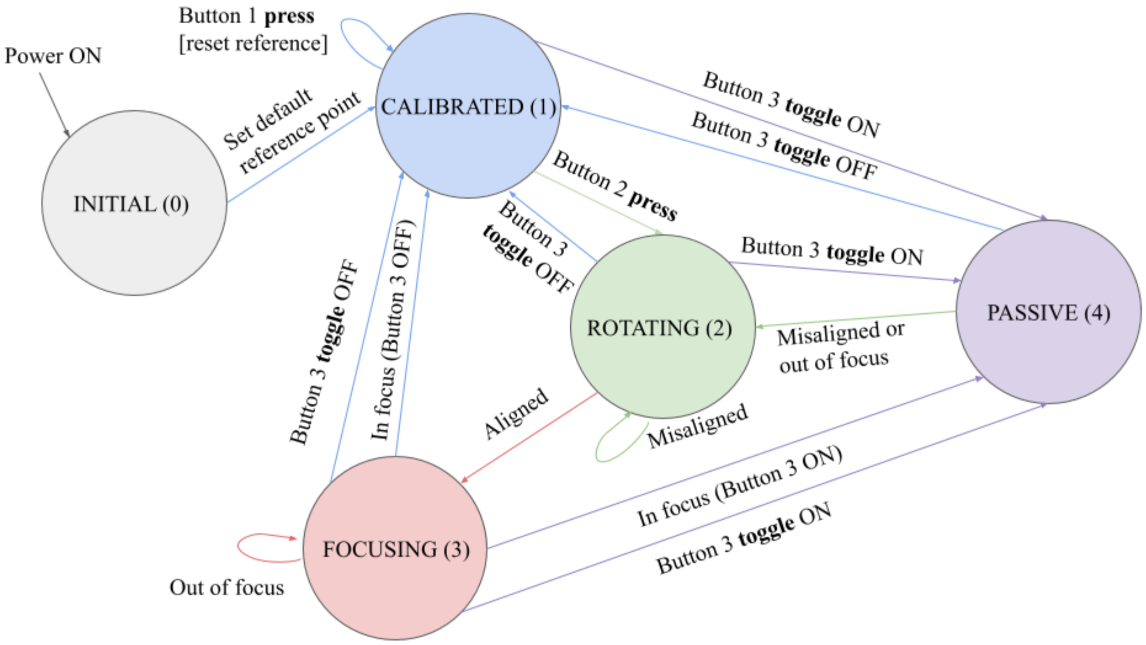


Figure 7. Finite State Machine

Appendix A
Bill of Materials

Part Number	Part Name	Quantity	Cost	Link/Source
1	ESP32 Microcontroller	1	$15.00	Adafruit / Lab Kit
2	DRV8833	1	$4.95	Adafruit / Lab Kit
3	6V 75:1 Pololu Motor	2	$33.90	Pololu / Lab Kit
4	Pololu Micro Metal Gearmotor Bracket pair	2	$2.95	Pololu / Lab Kit
5	Push button switch	3	$0.82	Mouser / Lab Kit
6	Ultrasonic sensor HC-SR04	2	$7.99	Amazon / Lab Kit
7	Lazy Susan Bearing 4”	1	$5.99	Amazon
8	3 mm Ball bearing	1	$8.99	Amazon
9	3 mm shaft	1	$6.49	Amazon
10	#4-40 x ½ screws & nuts	12	$2.56	Home Depot
11	12”x24”x0.25” plywood	2	$3.34	Jacobs Store

Appendix B

CAD

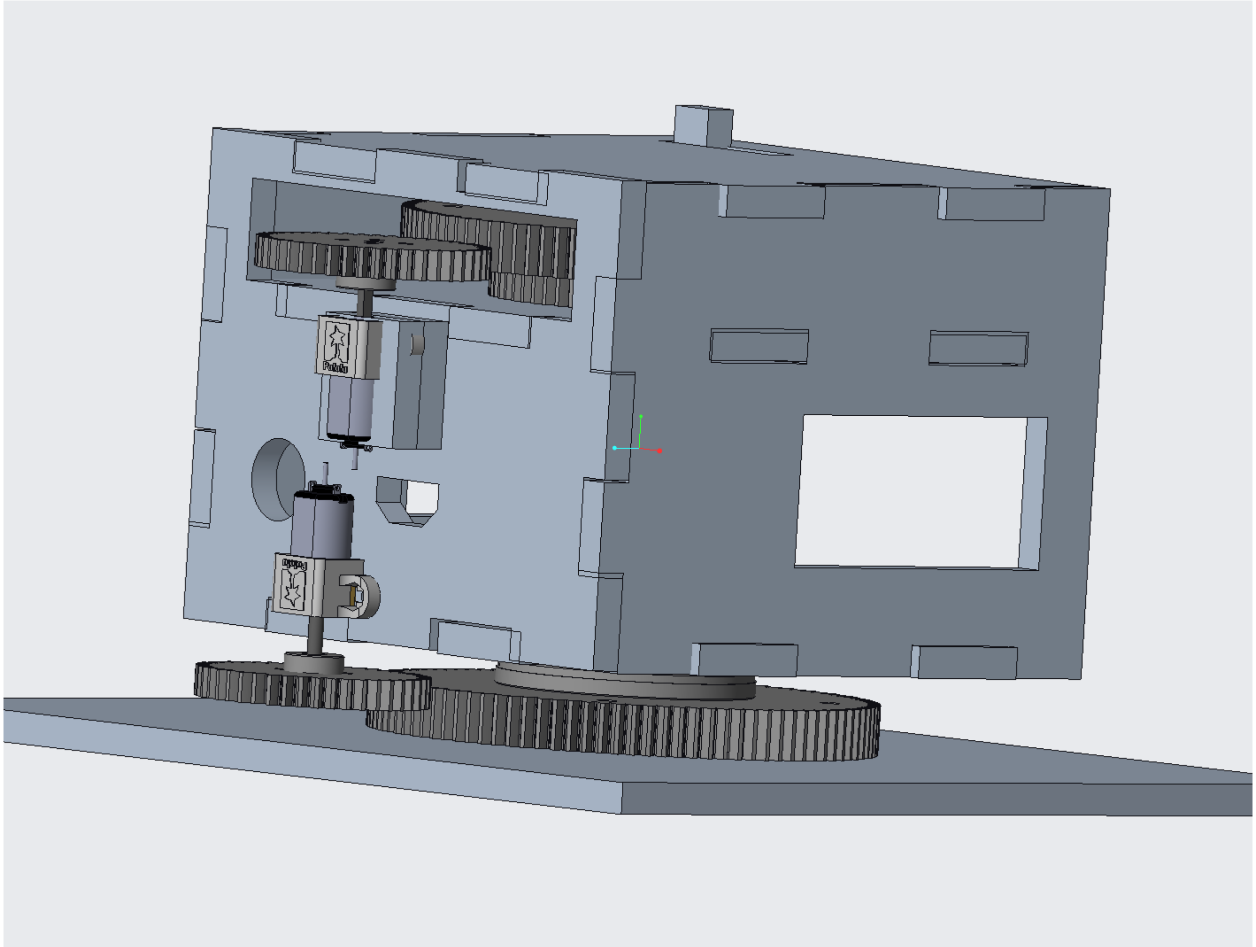


Figure 8. Motor and gear mechanisms for projector/housing rotation (bottom). Motor and gear mechanism for focus toggling (top)

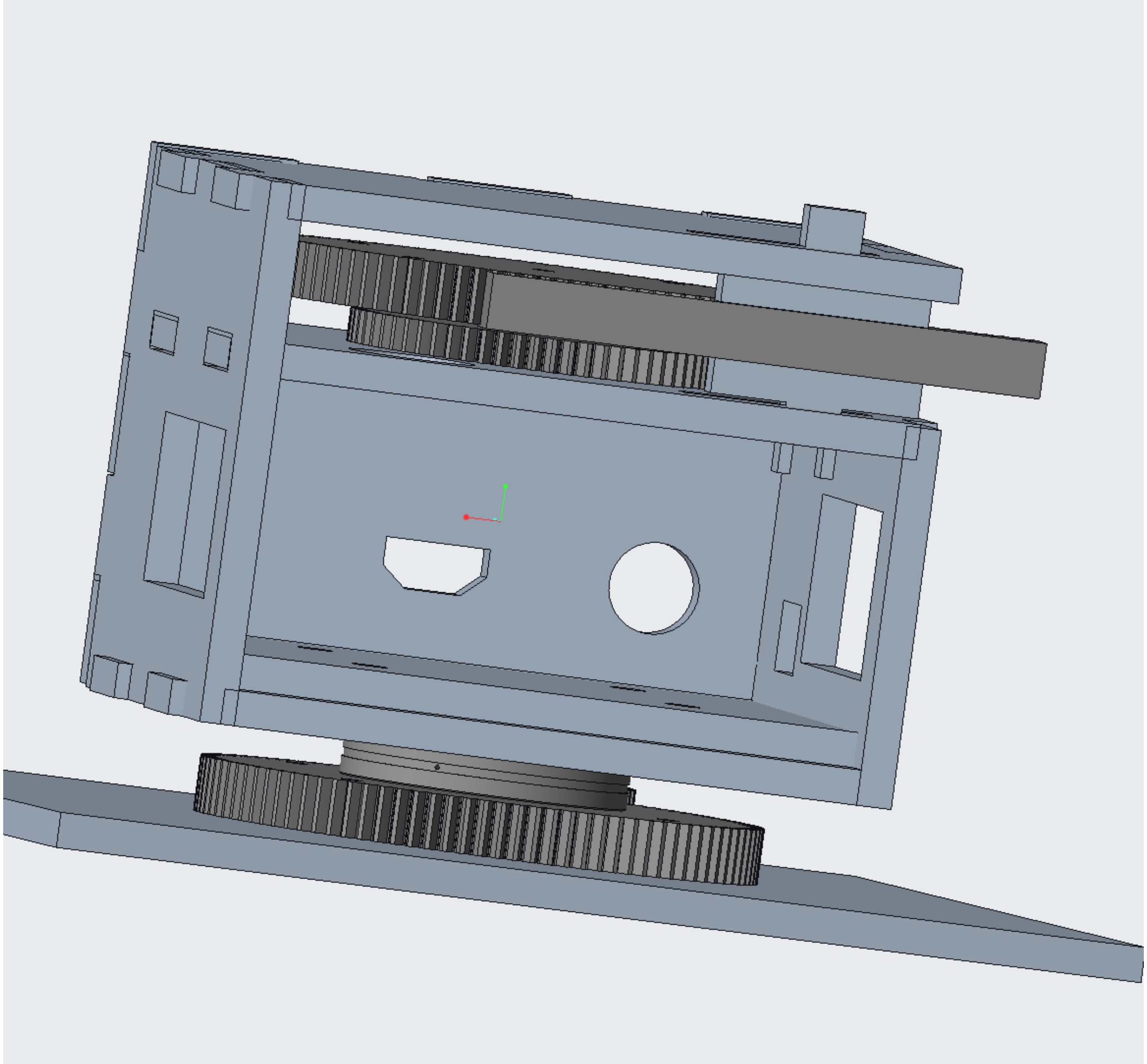
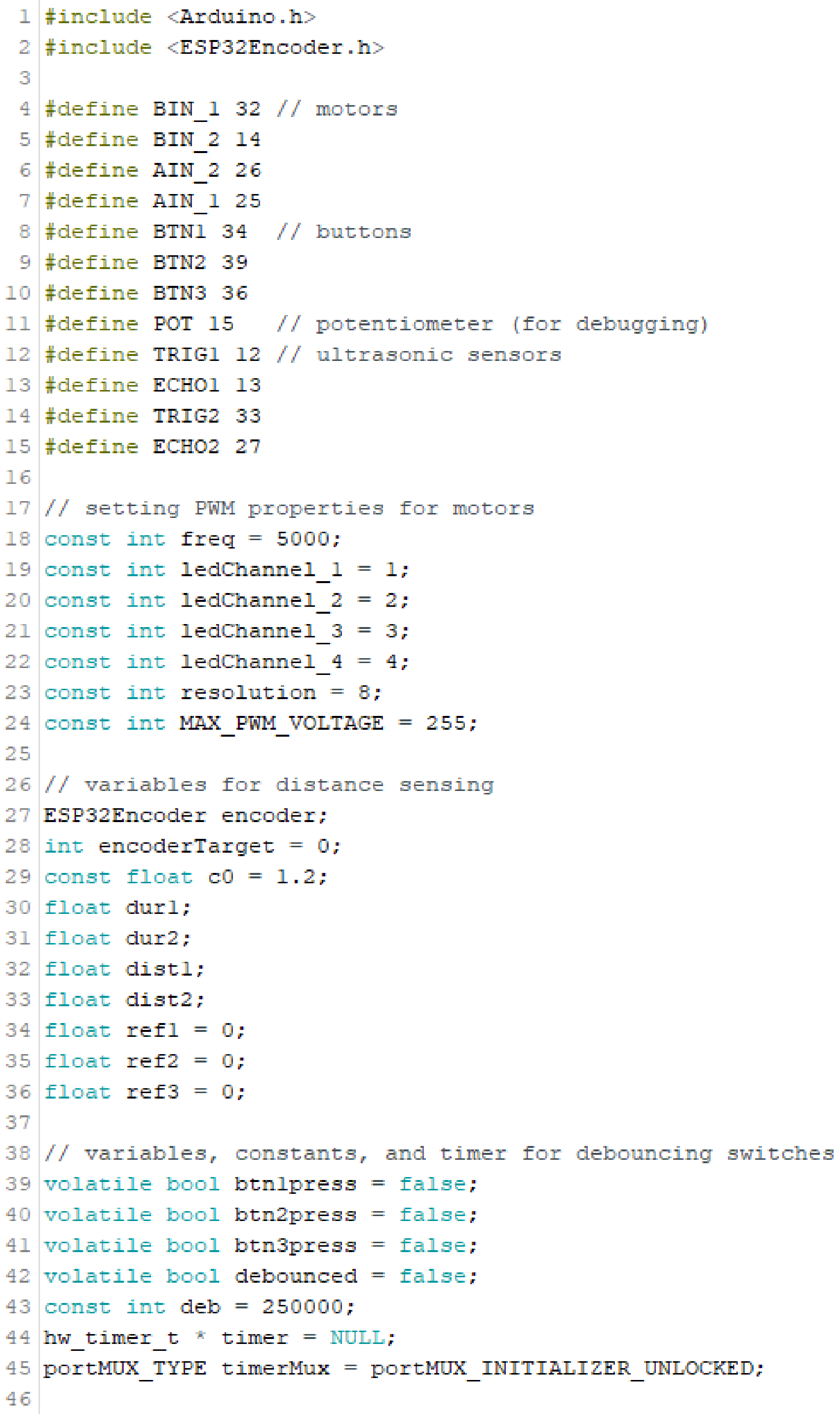


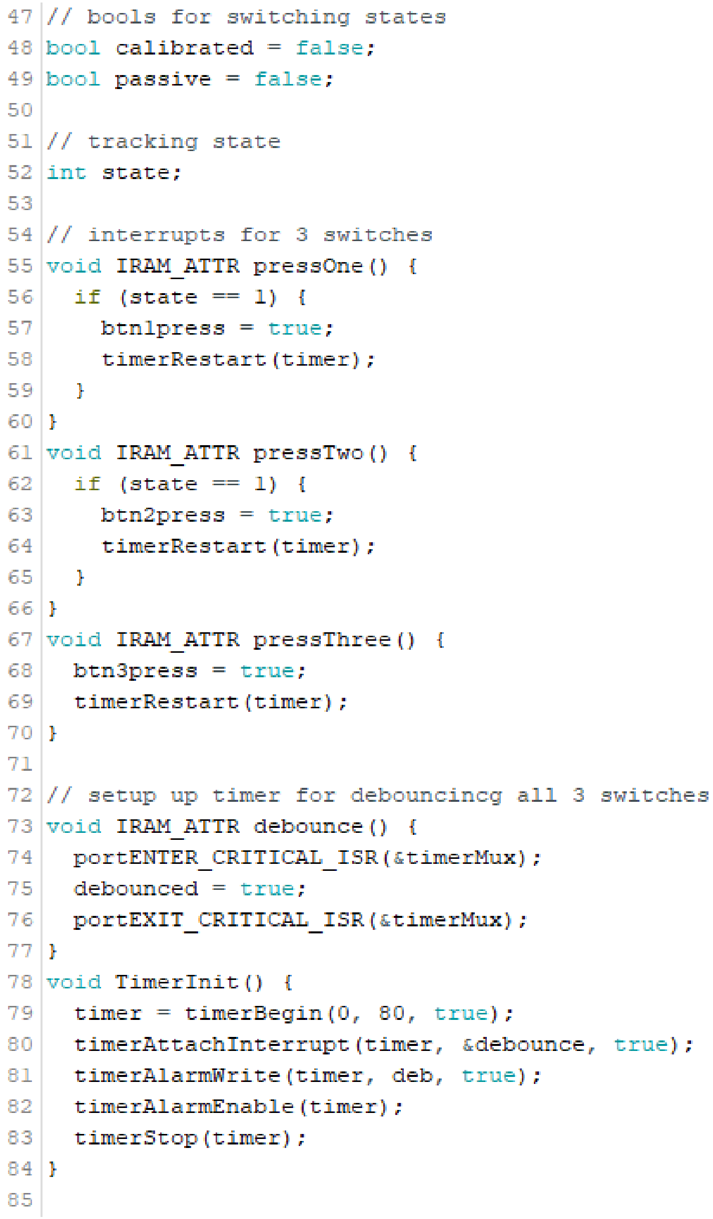
Figure 9. Rack and pinion gear mechanism along with slider attachments for focus toggling.

Appendix C

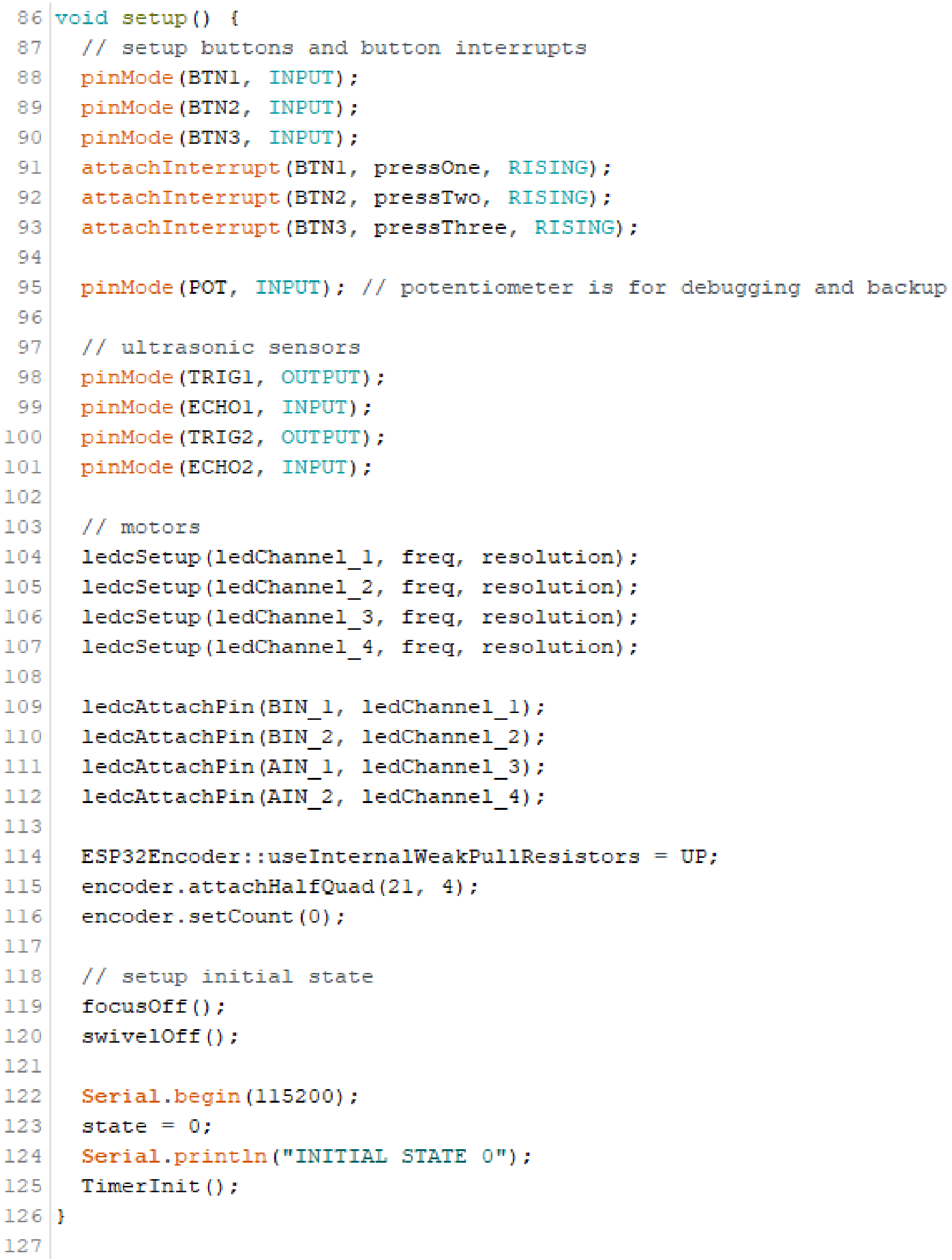
Arduino code



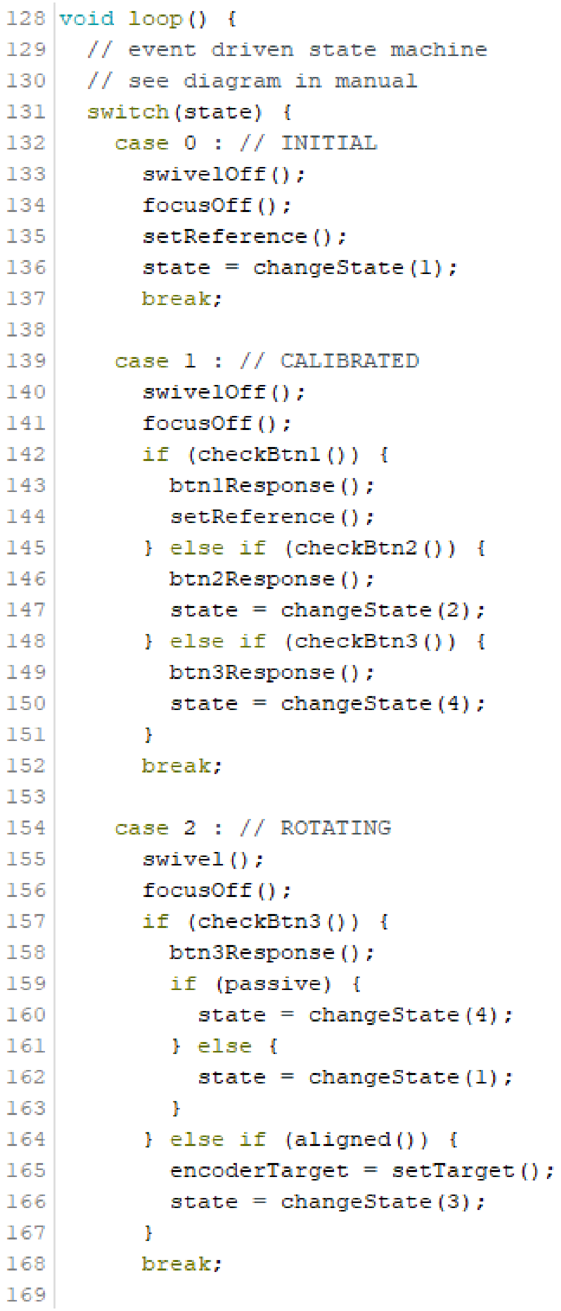
```
#include <Arduino.h>
#include <ESP32Encoder.h>

#define BIN_1 32 // motors
#define BIN_2 14
#define AIN_2 26
#define AIN_1 25
#define BTN1 34  // buttons
#define BTN2 39
#define BTN3 36
#define POT 15   // potentiometer (for debugging)
#define TRIG1 12 // ultrasonic sensors
#define ECHO1 13
#define TRIG2 33
#define ECHO2 27

// setting PWM properties for motors
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int ledChannel_3 = 3;
const int ledChannel_4 = 4;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;

// variables for distance sensing
ESP32Encoder encoder;
int encoderTarget = 0;
const float c0 = 1.2;
float dur1;
float dur2;
float dist1;
float dist2;
float ref1 = 0;
float ref2 = 0;
float ref3 = 0;

// variables, constants, and timer for debouncing switches
volatile bool btn1press = false;
volatile bool btn2press = false;
volatile bool btn3press = false;
volatile bool debounced = false;
const int deb = 250000;
hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

```



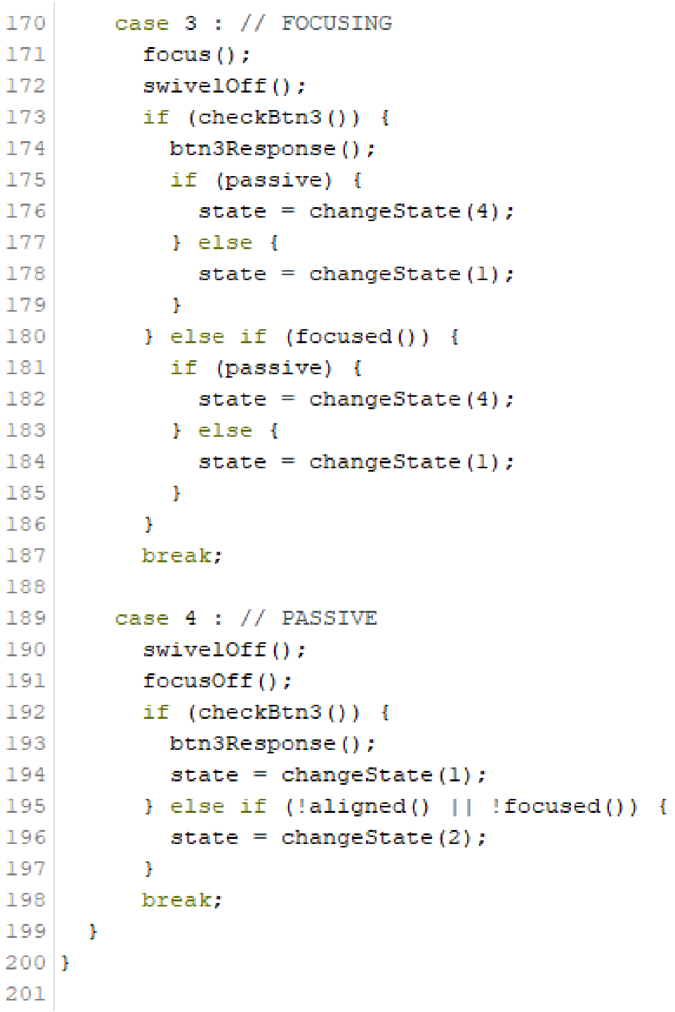
```
// bools for switching states
bool calibrated = false;
bool passive = false;

// tracking state
int state;

// interrupts for 3 switches
void IRAM_ATTR pressOne() {
  if (state == 1) {
    btn1press = true;
    timerRestart(timer);
  }
}
void IRAM_ATTR pressTwo() {
  if (state == 1) {
    btn2press = true;
    timerRestart(timer);
  }
}
void IRAM_ATTR pressThree() {
  btn3press = true;
  timerRestart(timer);
}

// setup up timer for debouncincg all 3 switches
void IRAM_ATTR debounce() {
  portENTER_CRITICAL_ISR(&timerMux);
  debounced = true;
  portEXIT_CRITICAL_ISR(&timerMux);
}
void TimerInit() {
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &debounce, true);
  timerAlarmWrite(timer, deb, true);
  timerAlarmEnable(timer);
  timerStop(timer);
}

```



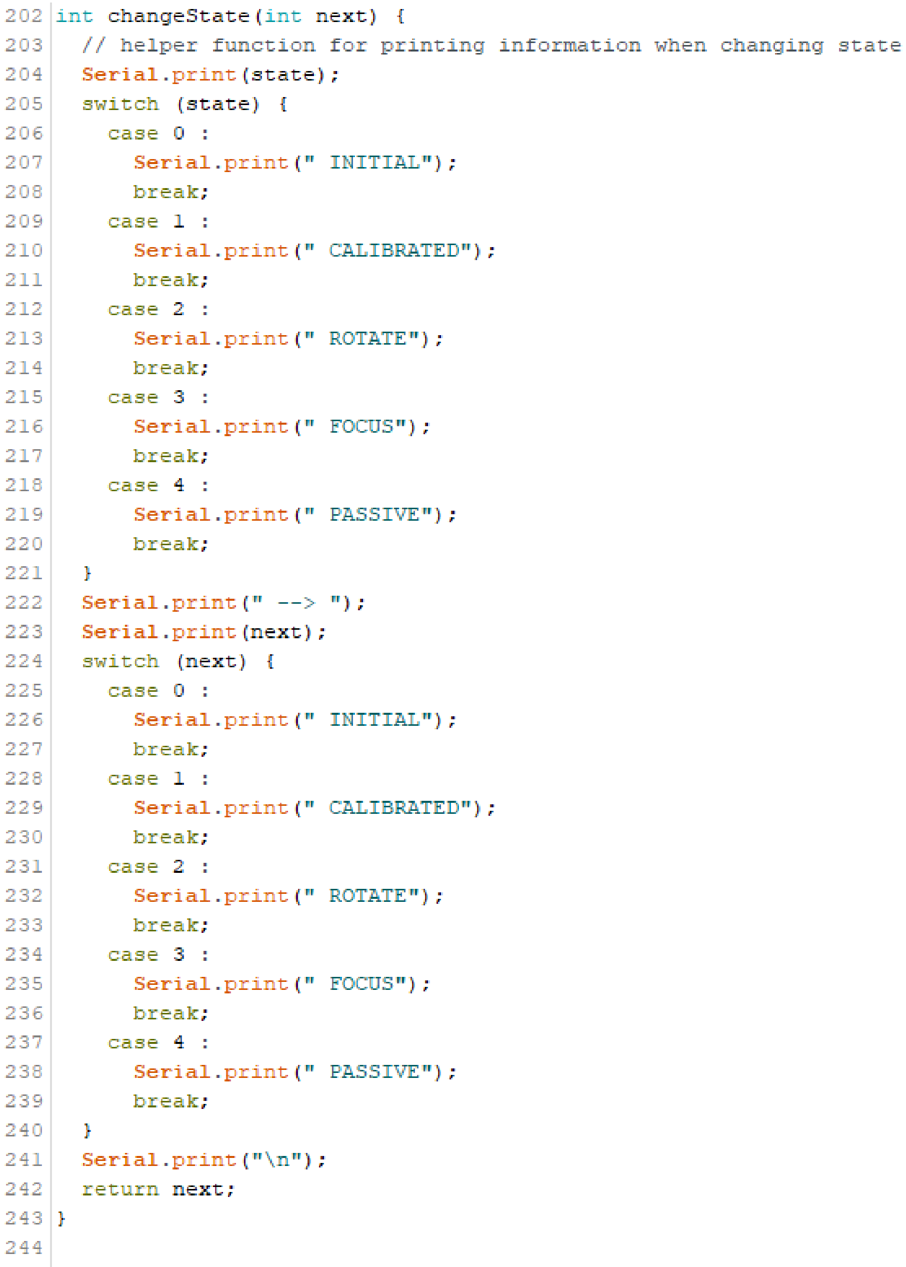
```
void setup() {
  // setup buttons and button interrupts
  pinMode(BTN1, INPUT);
  pinMode(BTN2, INPUT);
  pinMode(BTN3, INPUT);
  attachInterrupt(BTN1, pressOne, RISING);
  attachInterrupt(BTN2, pressTwo, RISING);
  attachInterrupt(BTN3, pressThree, RISING);

  pinMode(POT, INPUT); // potentiometer is for debugging and backup

  // ultrasonic sensors
  pinMode(TRIG1, OUTPUT);
  pinMode(ECHO1, INPUT);
  pinMode(TRIG2, OUTPUT);
  pinMode(ECHO2, INPUT);

  // motors
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);
  ledcSetup(ledChannel_3, freq, resolution);
  ledcSetup(ledChannel_4, freq, resolution);

  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);
  ledcAttachPin(AIN_1, ledChannel_3);
  ledcAttachPin(AIN_2, ledChannel_4);

  ESP32Encoder::useInternalWeakPullResistors = UP;
  encoder.attachHalfQuad(21, 4);
  encoder.setCount(0);

  // setup initial state
  focusOff();
  swivelOff();

  Serial.begin(115200);
  state = 0;
  Serial.println("INITIAL STATE 0");
  TimerInit();
}

```



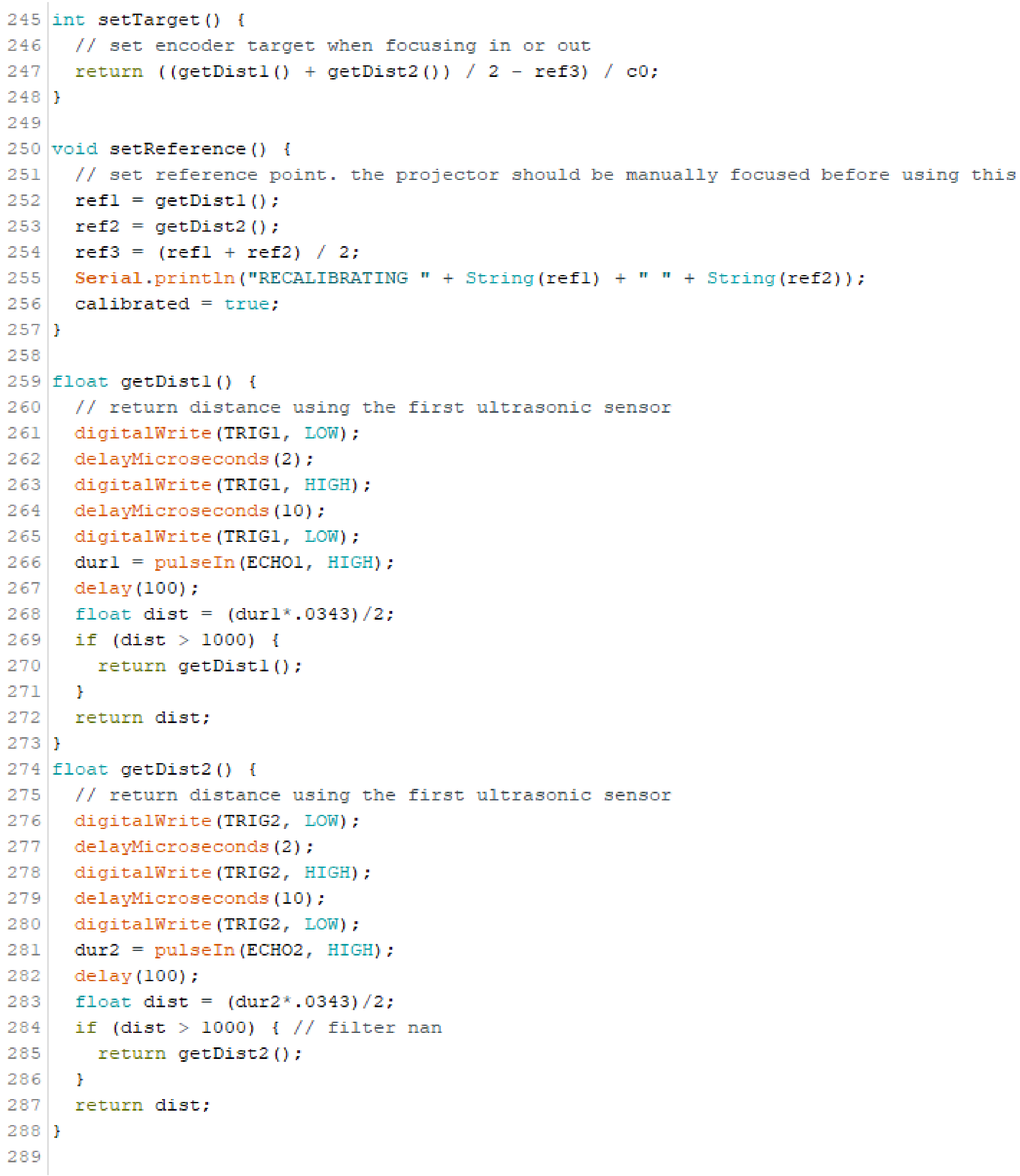
```
void loop() {
  // event driven state machine
  // see diagram in manual
  switch(state) {
    case 0 : // INITIAL
      swivelOff();
      focusOff();
      setReference();
      state = changeState(1);
      break;

    case 1 : // CALIBRATED
      swivelOff();
      focusOff();
      if (checkBtn1()) {
        btn1Response();
        setReference();
      } else if (checkBtn2()) {
        btn2Response();
        state = changeState(2);
      } else if (checkBtn3()) {
        btn3Response();
        state = changeState(4);
      }
      break;

    case 2 : // ROTATING
      swivel();
      focusOff();
      if (checkBtn3()) {
        btn3Response();
        if (passive) {
          state = changeState(4);
        } else {
          state = changeState(1);
        }
      } else if (aligned()) {
        encoderTarget = setTarget();
        state = changeState(3);
      }
      break;

```



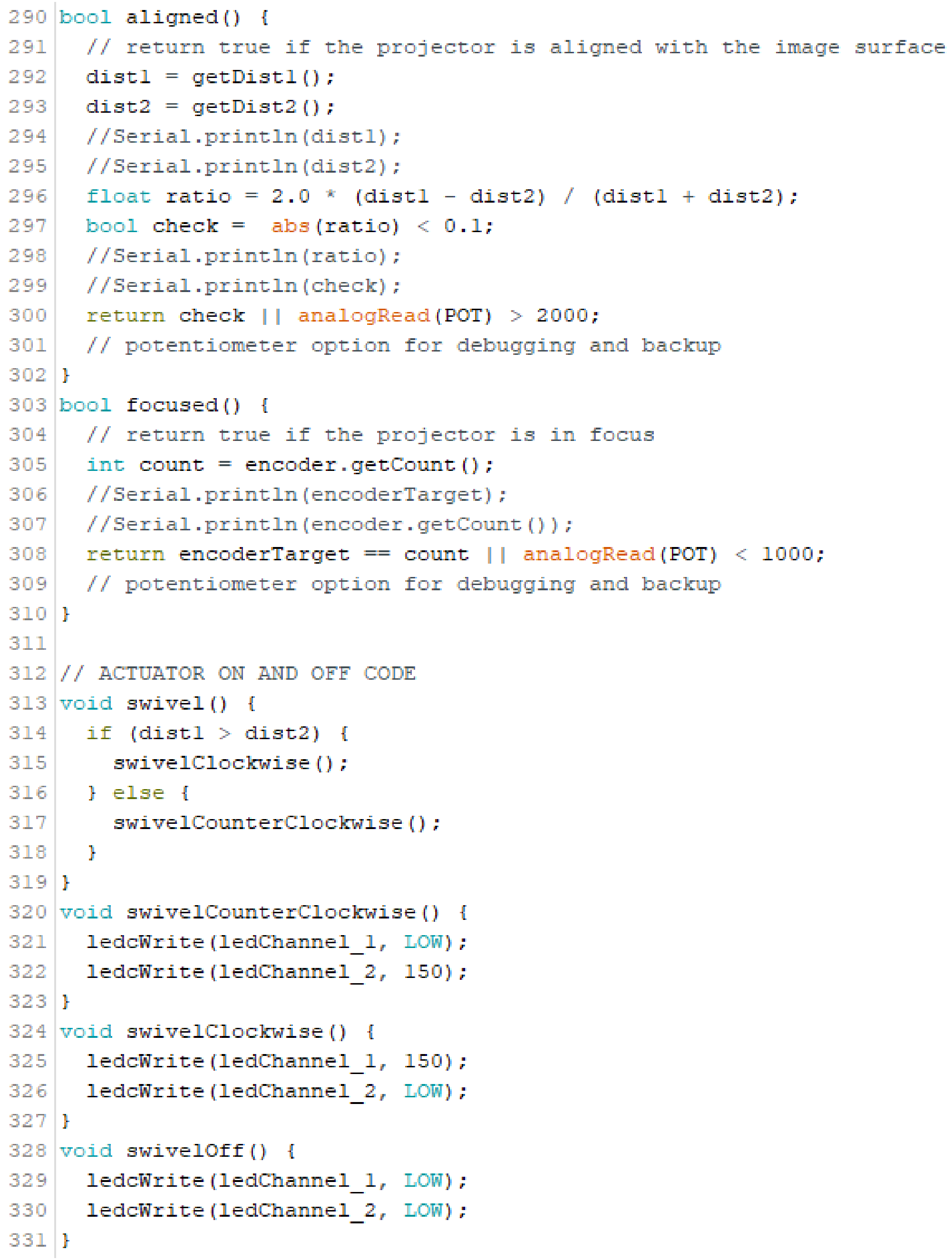
```
    case 3 : // FOCUSING
      focus();
      swivelOff();
      if (checkBtn3()) {
        btn3Response();
        if (passive) {
          state = changeState(4);
        } else {
          state = changeState(1);
        }
      } else if (focused()) {
        if (passive) {
          state = changeState(4);
        } else {
          state = changeState(1);
        }
      }
      break;

    case 4 : // PASSIVE
      swivelOff();
      focusOff();
      if (checkBtn3()) {
        btn3Response();
        state = changeState(1);
      } else if (!aligned() || !focused()) {
        state = changeState(2);
      }
      break;
  }
}

```



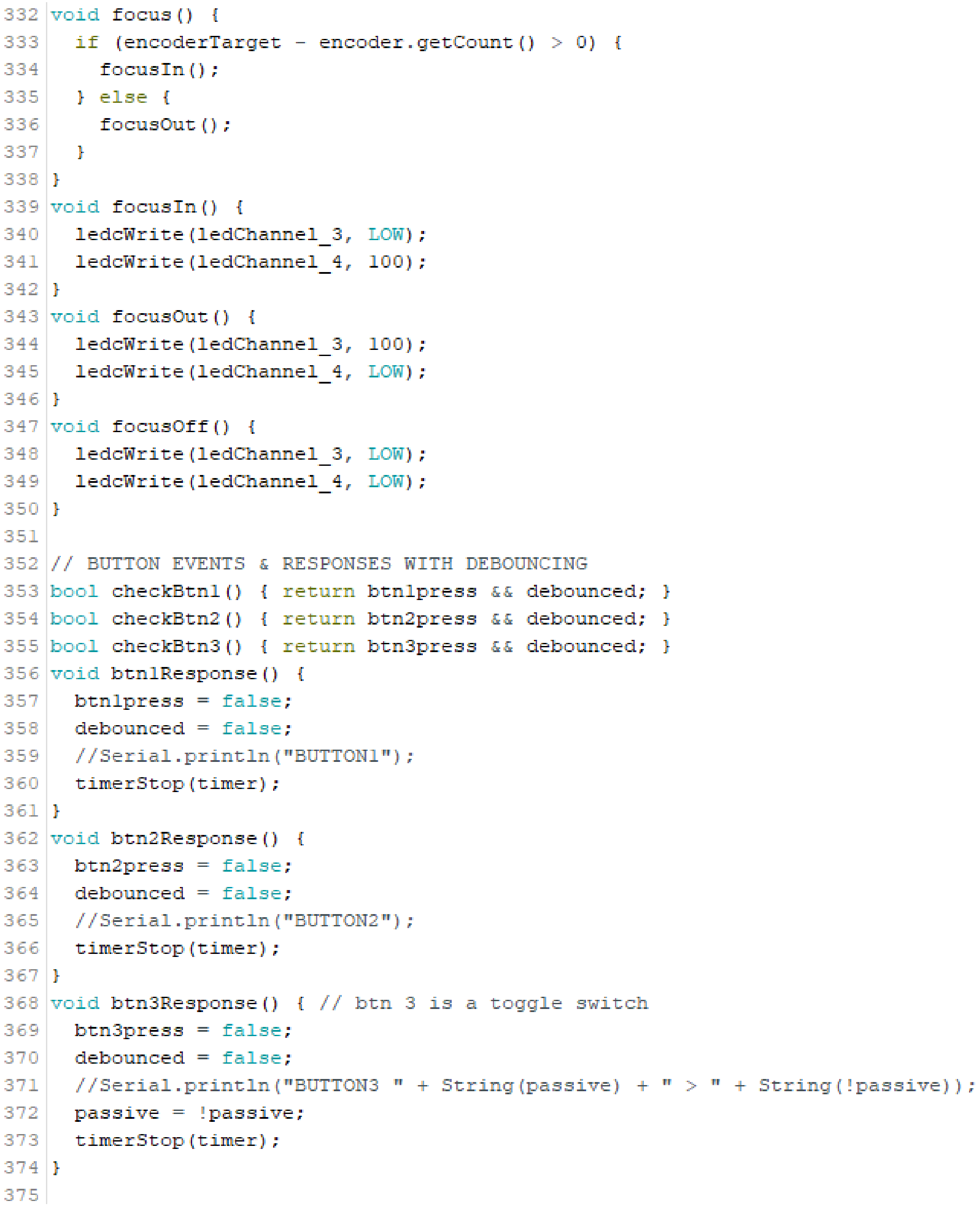
```
int changeState(int next) {
  // helper function for printing information when changing state
  Serial.print(state);
  switch (state) {
    case 0 :
      Serial.print(" INITIAL");
      break;
    case 1 :
      Serial.print(" CALIBRATED");
      break;
    case 2 :
      Serial.print(" ROTATE");
      break;
    case 3 :
      Serial.print(" FOCUS");
      break;
    case 4 :
      Serial.print(" PASSIVE");
      break;
  }
  Serial.print(" --> ");
  Serial.print(next);
  switch (next) {
    case 0 :
      Serial.print(" INITIAL");
      break;
    case 1 :
      Serial.print(" CALIBRATED");
      break;
    case 2 :
      Serial.print(" ROTATE");
      break;
    case 3 :
      Serial.print(" FOCUS");
      break;
    case 4 :
      Serial.print(" PASSIVE");
      break;
  }
  Serial.print("\n");
  return next;
}

```



```
int setTarget() {
  // set encoder target when focusing in or out
  return ((getDist1() + getDist2()) / 2 - ref3) / c0;
}

void setReference() {
  // set reference point. the projector should be manually focused before using this
  ref1 = getDist1();
  ref2 = getDist2();
  ref3 = (ref1 + ref2) / 2;
  Serial.println("RECALIBRATING " + String(ref1) + " " + String(ref2));
  calibrated = true;
}

float getDist1() {
  // return distance using the first ultrasonic sensor
  digitalWrite(TRIG1, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG1, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG1, LOW);
  dur1 = pulseIn(ECHO1, HIGH);
  delay(100);
  float dist = (dur1*.0343)/2;
  if (dist > 1000) {
    return getDist1();
  }
  return dist;
}
float getDist2() {
  // return distance using the first ultrasonic sensor
  digitalWrite(TRIG2, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG2, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG2, LOW);
  dur2 = pulseIn(ECHO2, HIGH);
  delay(100);
  float dist = (dur2*.0343)/2;
  if (dist > 1000) { // filter nan
    return getDist2();
  }
  return dist;
}

```



```
bool aligned() {
  // return true if the projector is aligned with the image surface
  dist1 = getDist1();
  dist2 = getDist2();
  //Serial.println(dist1);
  //Serial.println(dist2);
  float ratio = 2.0 * (dist1 - dist2) / (dist1 + dist2);
  bool check =  abs(ratio) < 0.1;
  //Serial.println(ratio);
  //Serial.println(check);
  return check || analogRead(POT) > 2000;
  // potentiometer option for debugging and backup
}
bool focused() {
  // return true if the projector is in focus
  int count = encoder.getCount();
  //Serial.println(encoderTarget);
  //Serial.println(encoder.getCount());
  return encoderTarget == count || analogRead(POT) < 1000;
  // potentiometer option for debugging and backup
}

// ACTUATOR ON AND OFF CODE
void swivel() {
  if (dist1 > dist2) {
    swivelClockwise();
  } else {
    swivelCounterClockwise();
  }
}
void swivelCounterClockwise() {
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, 150);
}
void swivelClockwise() {
  ledcWrite(ledChannel_1, 150);
  ledcWrite(ledChannel_2, LOW);
}
void swivelOff() {
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, LOW);
}
```



```
void focus() {
  if (encoderTarget - encoder.getCount() > 0) {
    focusIn();
  } else {
    focusOut();
  }
}
void focusIn() {
  ledcWrite(ledChannel_3, LOW);
  ledcWrite(ledChannel_4, 100);
}
void focusOut() {
  ledcWrite(ledChannel_3, 100);
  ledcWrite(ledChannel_4, LOW);
}
void focusOff() {
  ledcWrite(ledChannel_3, LOW);
  ledcWrite(ledChannel_4, LOW);
}

// BUTTON EVENTS & RESPONSES WITH DEBOUNCING
bool checkBtn1() { return btn1press && debounced; }
bool checkBtn2() { return btn2press && debounced; }
bool checkBtn3() { return btn3press && debounced; }
void btn1Response() {
  btn1press = false;
  debounced = false;
  //Serial.println("BUTTON1");
  timerStop(timer);
}
void btn2Response() {
  btn2press = false;
  debounced = false;
  //Serial.println("BUTTON2");
  timerStop(timer);
}
void btn3Response() { // btn 3 is a toggle switch
  btn3press = false;
  debounced = false;
  //Serial.println("BUTTON3 " + String(passive) + " > " + String(!passive));
  passive = !passive;
  timerStop(timer);
}

```