Bear-Pact me102b final report
Laser Light Show

Abstract

We developed a sound reactive laser light show device. A laser is traced to form multiple pre-defined patterns using single laser shown upon 2 consecutive mirrors: one deflecting the laser on the x-axis, one deflecting upon the y-axis. These mirrors are attached to precise DC motors which move rapidly to trace the output laser along the edge of a shape faster than the human eye can process, resulting in 2D shapes being drawn on the wall. The shapes being traced are cycled via a sound-reactive state machine. We also added a selector which can apply different diffraction gratings at the exit of the laser path. This allows our machine to cycle through different diffraction gratings using a stepper motor and belt to turn which grating is present at the exit.

Opportunity

Several of our group members are music festival enthusiasts. Through first-hand experience, we realized that one of the biggest attractions to festival environments is the professional laser light shows. We identified an opportunity: there is an experience that many people enjoy but is not accessible. We were driven to create a portable, cheap, safe party laser for festival level laser features in home like environments.

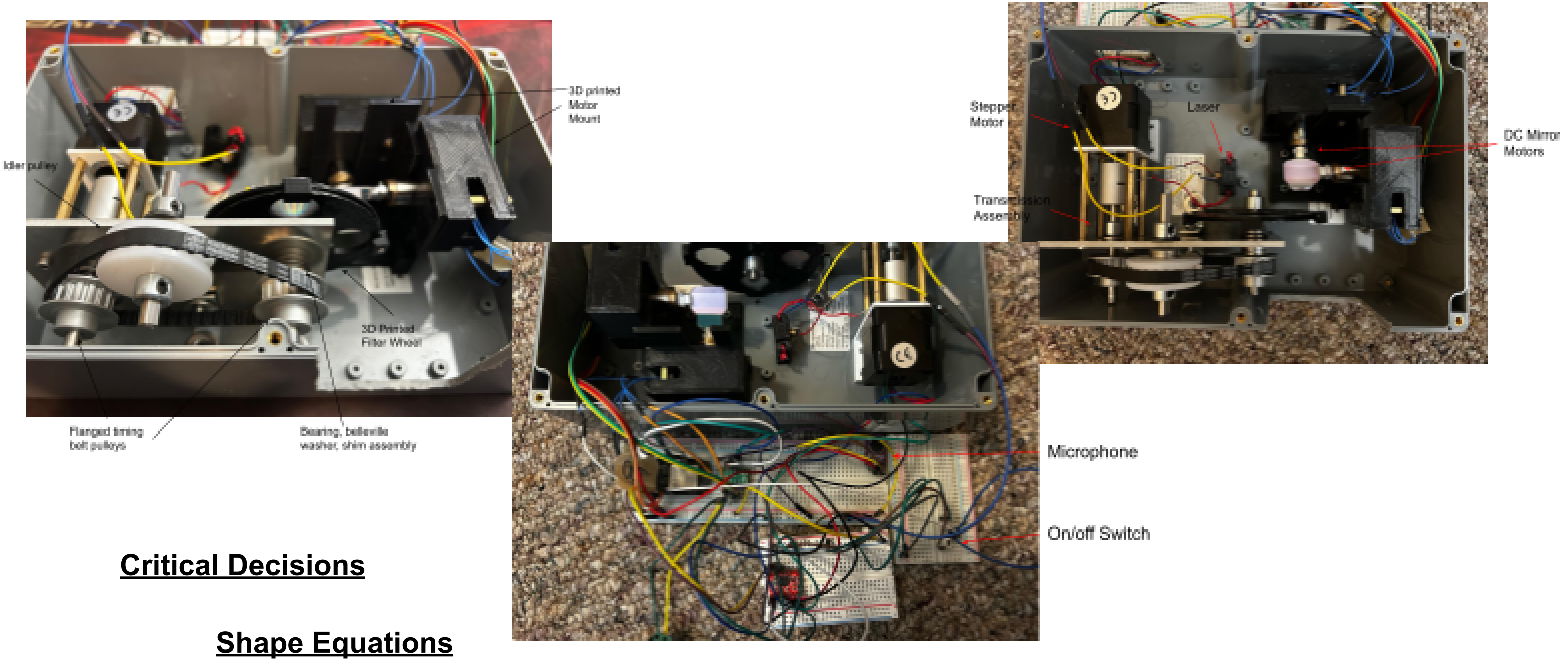
Beyond an enjoyment for laser lights and entertainment equipment, our group was motivated to create a project that was able to produce a stunning and mesmerizing display. We were determined to have a final product that looked well designed: contained electronics, easy user interface, simple start up, and overall great user experience.

Strategy

The primary functionality we hoped to achieve was laser projection of arbitrary shapes on an x-y axis. We achieved this by shining a laser onto 2 single axis mirrors attached to motors oscillating at high speeds. We defined several shapes as parametric equations wrapped inside functions. When these functions are called, the mirrors rotate at high speeds to deflect the laser to trace the defined path, resulting in each function projecting a different shape on the wall. We added more flavor to our patterns with a rotating selector which changed which diffraction grating the laser exited through; this allowed us to create the illusion of multiple lasers being projected. We initially hoped to trace great complex, arbitrary, and changing patterns; though we found predefined parametric equations much easier to work with and sufficient for our original goal.

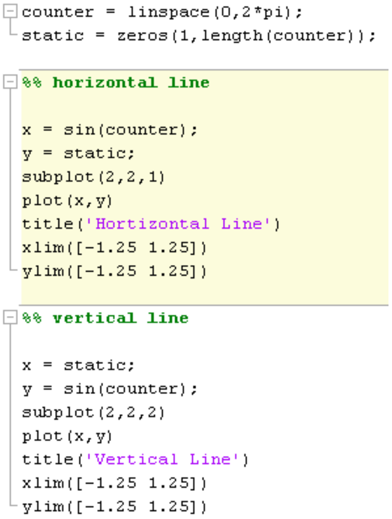
With several patterns and laser effects defined (states), we created a state machine that cycled through these patterns in an unpredictable and entertaining way. We decided to stay true to our original idea of “party laser” by having patterns cycle when a particular noise threshold was hit, causing patterns to change with the beat of the music. We attached a microphone to our esp32 and defined an interrupt (sound event) which triggered a new pattern (state change) whenever the sound got loud enough. To keep watchers engaged, we found it important to keep some level of unpredictability in the states we cycled through. Everytime a sound event occurs, a random number generator selects a random state to jump to.

Images

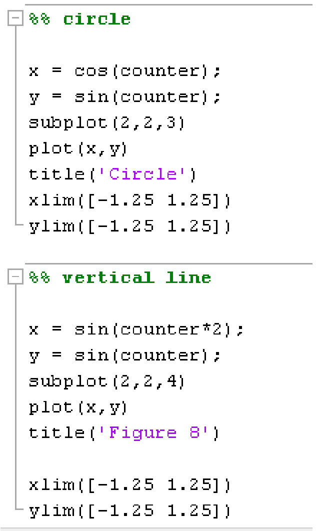


Critical Decisions

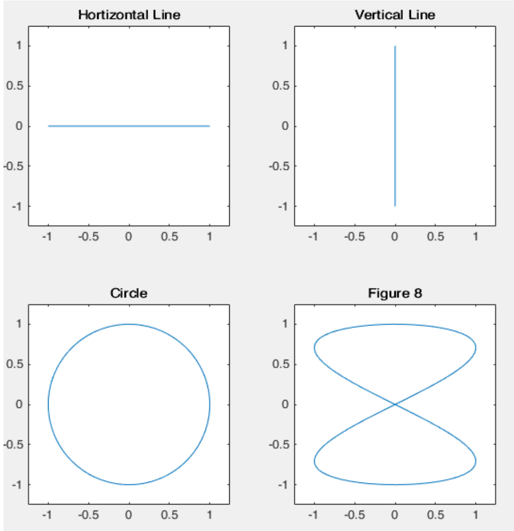
Shape Equations

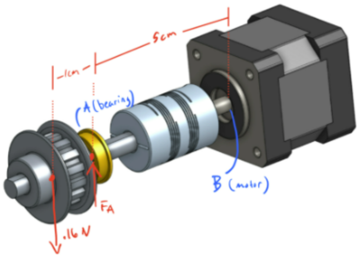


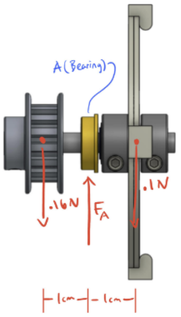
```
counter = linspace(0,2*pi);
static = zeros(1,length(counter));

%% horizontal line

x = sin(counter);
y = static;
subplot(2,2,1)
plot(x,y)
title('Hortizontal Line')
xlim([-1.25 1.25])
ylim([-1.25 1.25])

%% vertical line

x = static;
y = sin(counter);
subplot(2,2,2)
plot(x,y)
title('Vertical Line')
xlim([-1.25 1.25])
ylim([-1.25 1.25])
```

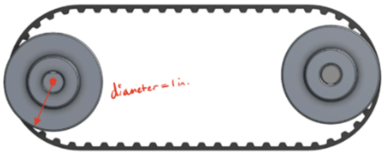


```
%% circle

x = cos(counter);
y = sin(counter);
subplot(2,2,3)
plot(x,y)
title('Circle')
xlim([-1.25 1.25])
ylim([-1.25 1.25])

%% vertical line

x = sin(counter*2);
y = sin(counter);
subplot(2,2,4)
plot(x,y)
title('Figure 8')

xlim([-1.25 1.25])
ylim([-1.25 1.25])
```









τ_{in} = max input torque = 10 N/cm
= 25.4 N/in

$F_i = -\Delta\tau/2 \quad \Delta\tau = \tau_{in} / r$

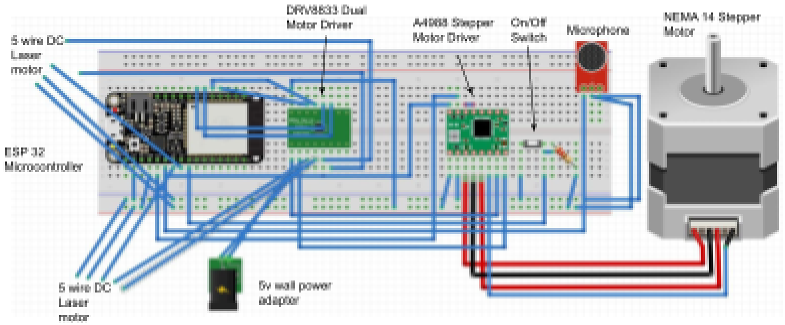
$\tau_{out} = (F_i - \Delta\tau/2) * r$
$= -\Delta\tau * r$

$F_A = (.16*6) / 5 = .19$ N in +z direction
$F_B = (.16*1) / 5 = .032$ N in -z direction

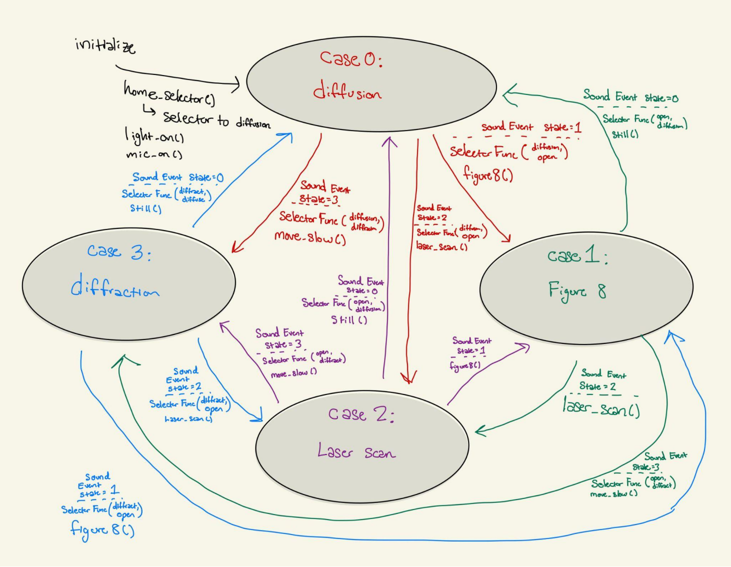
$F_A = (.1*2) / 1 + (.16*2) / 1 =$
.52 N in -z direction

$F_{pre} = 2 F_i = 2*(\Delta\tau/2) = \Delta\tau = 25.4 / .5 = 50.8$ N

Circuit Diagram



State Machine

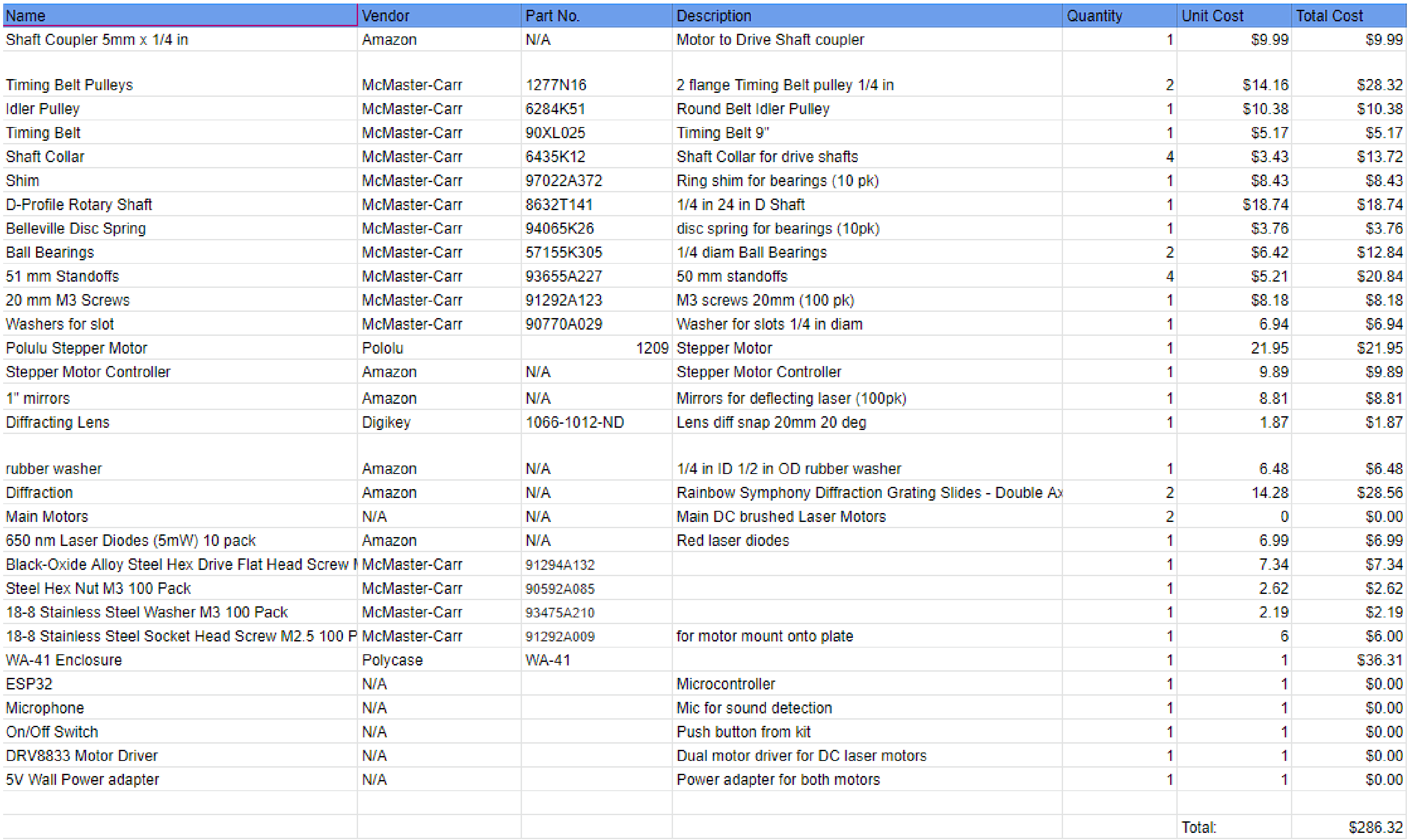


Reflection

One of the main goals we had throughout the semester was building a project we know would work on demo day without fail. This meant we needed to write software that was simple, and use components that were reliable. Simplicity is key. This worked well for much of our design decisions: sound reaction based on a simple threshold and microphone; a single event to monitor and trigger state transition. We understood the sensitivity of these instruments and designed our code accordingly. A few attempts to work without simple components where simpler options existed failed. We tried to use a magnetometer for homing of the selector instead of a limit switch- but the magnetometer device was not accurate enough for our purposes at such close range. Additionally we did not verify compatibility with our diffraction grating and laser, resulting in the projection being too dim for the human eye. Finally, we cut corners and used an available stepper motor to move the selector instead of a brushed motor with an encoder which would have removed the need for external homing techniques. We ended up with an unreliable, and loud selector which wouldn't move within our tolerance and sometimes set off additional sound events. Overall, make sure the components you chose are compatible and simple enough to work with.

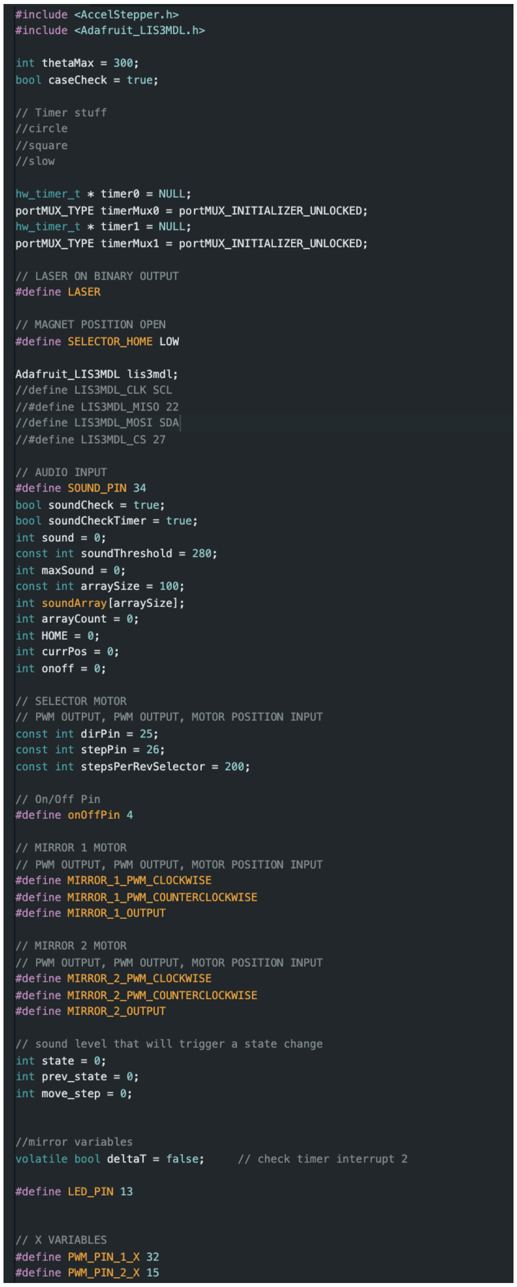
Appendix

BOM

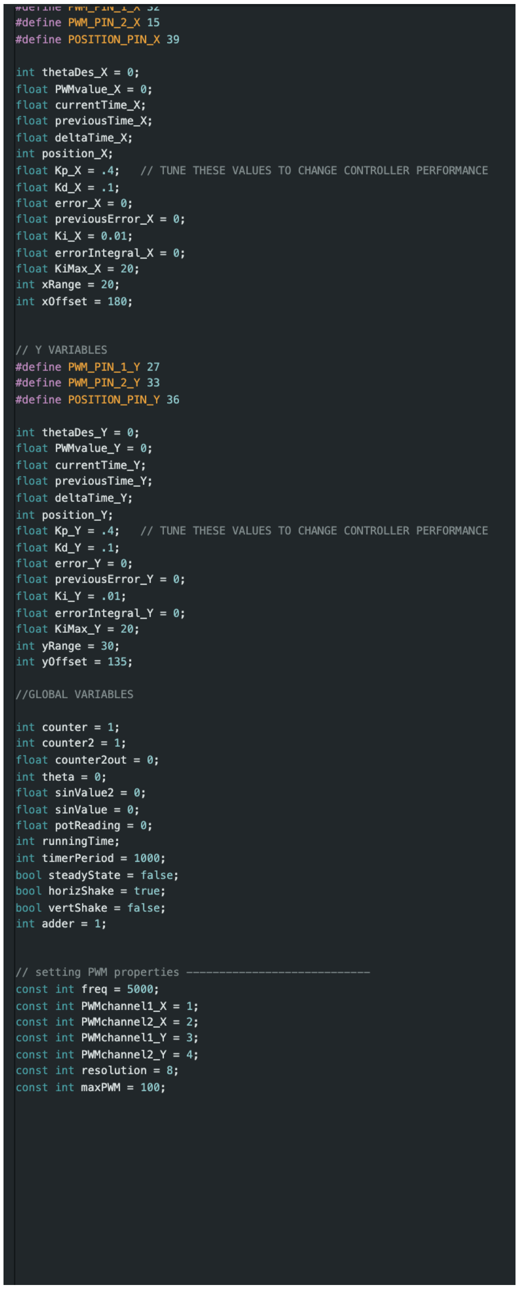


Name	Vendor	Part No.	Description	Quantity	Unit Cost	Total Cost
Shaft Coupler 5mm x 1/4 in	Amazon	N/A	Motor to Drive Shaft coupler	1	$9.99	$9.99
Timing Belt Pulleys	McMaster-Carr	1277N16	2 flange Timing Belt pulley 1/4 in	2	$14.16	$28.32
Idler Pulley	McMaster-Carr	6284K51	Round Belt Idler Pulley	1	$10.38	$10.38
Timing Belt	McMaster-Carr	90XL025	Timing Belt 9"	1	$5.17	$5.17
Shaft Collar	McMaster-Carr	6435K12	Shaft Collar for drive shafts	4	$3.43	$13.72
Shim	McMaster-Carr	97022A372	Ring shim for bearings (10 pk)	1	$8.43	$8.43
D-Profile Rotary Shaft	McMaster-Carr	8632T141	1/4 in 24 in D Shaft	1	$18.74	$18.74
Belleville Disc Spring	McMaster-Carr	94065K26	disc spring for bearings (10pk)	1	$3.76	$3.76
Ball Bearings	McMaster-Carr	57155K305	1/4 diam Ball Bearings	2	$6.42	$12.84
51 mm Standoffs	McMaster-Carr	93655A227	50 mm standoffs	4	$5.21	$20.84
20 mm M3 Screws	McMaster-Carr	91292A123	M3 screws 20mm (100 pk)	1	$8.18	$8.18
Washers for slot	McMaster-Carr	90770A029	Washer for slots 1/4 in diam	1	6.94	$6.94
Polulu Stepper Motor	Pololu	1209	Stepper Motor	1	21.95	$21.95
Stepper Motor Controller	Amazon	N/A	Stepper Motor Controller	1	9.89	$9.89
1" mirrors	Amazon	N/A	Mirrors for deflecting laser (100pk)	1	8.81	$8.81
Diffracting Lens	Digikey	1066-1012-ND	Lens diff snap 20mm 20 deg	1	1.87	$1.87
rubber washer	Amazon	N/A	1/4 in ID 1/2 in OD rubber washer	1	6.48	$6.48
Diffraction	Amazon	N/A	Rainbow Symphony Diffraction Grating Slides - Double Ax	2	14.28	$28.56
Main Motors	N/A	N/A	Main DC brushed Laser Motors	2	0	$0.00
650 nm Laser Diodes (5mW) 10 pack	Amazon	N/A	Red laser diodes	1	6.99	$6.99
Black-Oxide Alloy Steel Hex Drive Flat Head Screw I	McMaster-Carr	91294A132		1	7.34	$7.34
Steel Hex Nut M3 100 Pack	McMaster-Carr	90592A085		1	2.62	$2.62
18-8 Stainless Steel Washer M3 100 Pack	McMaster-Carr	93475A210		1	2.19	$2.19
18-8 Stainless Steel Socket Head Screw M2.5 100 P	McMaster-Carr	91292A009	for motor mount onto plate	1	6	$6.00
WA-41 Enclosure	Polycase	WA-41		1	1	$36.31
ESP32	N/A		Microcontroller	1	1	$0.00
Microphone	N/A		Mic for sound detection	1	1	$0.00
On/Off Switch	N/A		Push button from kit	1	1	$0.00
DRV8833 Motor Driver	N/A		Dual motor driver for DC laser motors	1	1	$0.00
5V Wall Power adapter	N/A		Power adapter for both motors	1	1	$0.00
					Total:	$286.32

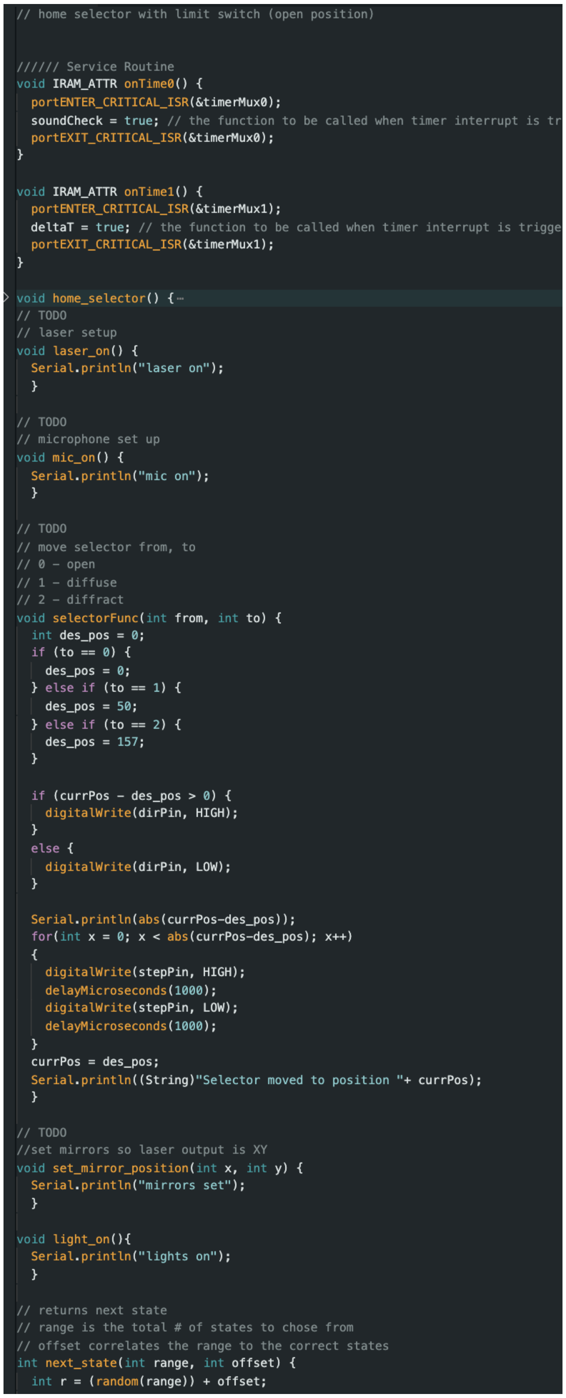
CODE



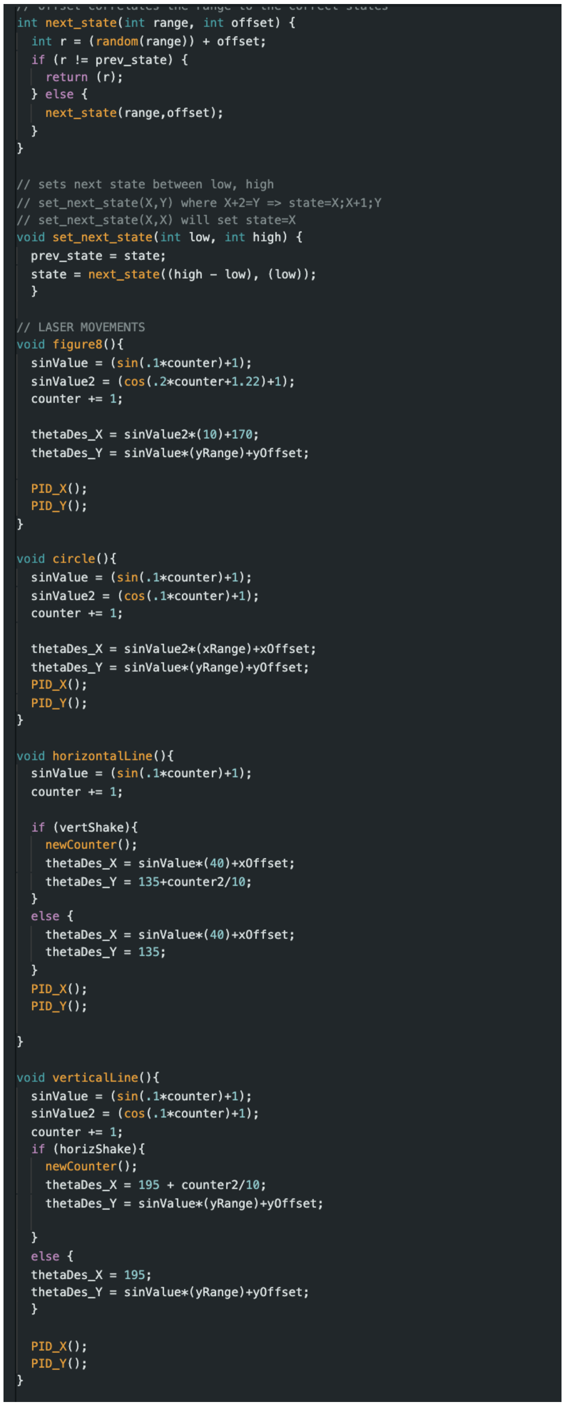
```
#include <AccelStepper.h>
#include <Adafruit_LIS3MDL.h>

int thetaMax = 300;
bool caseCheck = true;

// Timer stuff
//circle
//square
//slow

hw_timer_t * timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

// LASER ON BINARY OUTPUT
#define LASER

// MAGNET POSITION OPEN
#define SELECTOR_HOME LOW

Adafruit_LIS3MDL lis3mdl;
//define LIS3MDL_CLK SCL
//#define LIS3MDL_MISO 22
//define LIS3MDL_MOSI SDA
//#define LIS3MDL_CS 27

// AUDIO INPUT
#define SOUND_PIN 34
bool soundCheck = true;
bool soundCheckTimer = true;
int sound = 0;
const int soundThreshold = 280;
int maxSound = 0;
const int arraySize = 100;
int soundArray[arraySize];
int arrayCount = 0;
int HOME = 0;
int currPos = 0;
int onoff = 0;

// SELECTOR MOTOR
// PWM OUTPUT, PWM OUTPUT, MOTOR POSITION INPUT
const int dirPin = 25;
const int stepPin = 26;
const int stepsPerRevSelector = 200;

// On/Off Pin
#define onOffPin 4

// MIRROR 1 MOTOR
// PWM OUTPUT, PWM OUTPUT, MOTOR POSITION INPUT
#define MIRROR_1_PWM_CLOCKWISE
#define MIRROR_1_PWM_COUNTERCLOCKWISE
#define MIRROR_1_OUTPUT

// MIRROR 2 MOTOR
// PWM OUTPUT, PWM OUTPUT, MOTOR POSITION INPUT
#define MIRROR_2_PWM_CLOCKWISE
#define MIRROR_2_PWM_COUNTERCLOCKWISE
#define MIRROR_2_OUTPUT

// sound level that will trigger a state change
int state = 0;
int prev_state = 0;
int move_step = 0;


//mirror variables
volatile bool deltaT = false;     // check timer interrupt 2

#define LED_PIN 13


// X VARIABLES
#define PWM_PIN_1_X 32
#define PWM_PIN_2_X 15
```



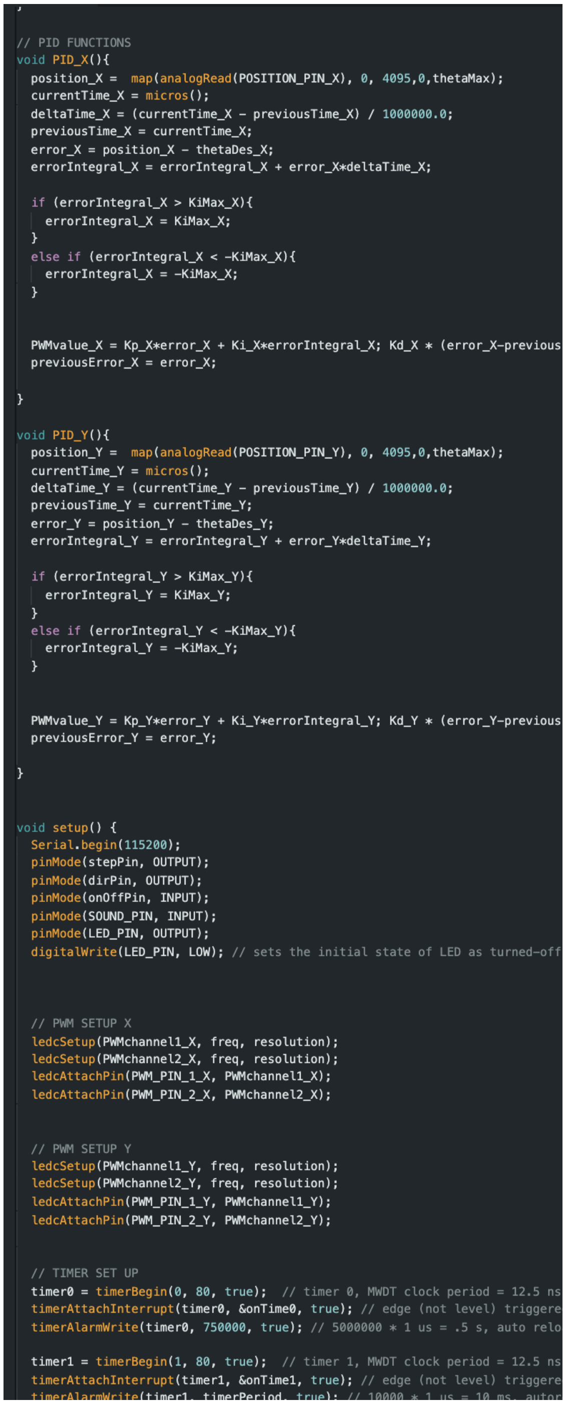
```
#define PWM_PIN_1_X 32
#define PWM_PIN_2_X 15
#define POSITION_PIN_X 39

int thetaDes_X = 0;
float PWMvalue_X = 0;
float currentTime_X;
float previousTime_X;
float deltaTime_X;
int position_X;
float Kp_X = .4;   // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
float Kd_X = .1;
float error_X = 0;
float previousError_X = 0;
float Ki_X = 0.01;
float errorIntegral_X = 0;
float KiMax_X = 20;
int xRange = 20;
int xOffset = 180;


// Y VARIABLES
#define PWM_PIN_1_Y 27
#define PWM_PIN_2_Y 33
#define POSITION_PIN_Y 36

int thetaDes_Y = 0;
float PWMvalue_Y = 0;
float currentTime_Y;
float previousTime_Y;
float deltaTime_Y;
int position_Y;
float Kp_Y = .4;   // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
float Kd_Y = .1;
float error_Y = 0;
float previousError_Y = 0;
float Ki_Y = .01;
float errorIntegral_Y = 0;
float KiMax_Y = 20;
int yRange = 30;
int yOffset = 135;

//GLOBAL VARIABLES

int counter = 1;
int counter2 = 1;
float counter2out = 0;
int theta = 0;
float sinValue2 = 0;
float sinValue = 0;
float potReading = 0;
int runningTime;
int timerPeriod = 1000;
bool steadyState = false;
bool horizShake = true;
bool vertShake = false;
int adder = 1;


// setting PWM properties ----------------------------
const int freq = 5000;
const int PWMchannel1_X = 1;
const int PWMchannel2_X = 2;
const int PWMchannel1_Y = 3;
const int PWMchannel2_Y = 4;
const int resolution = 8;
const int maxPWM = 100;
```



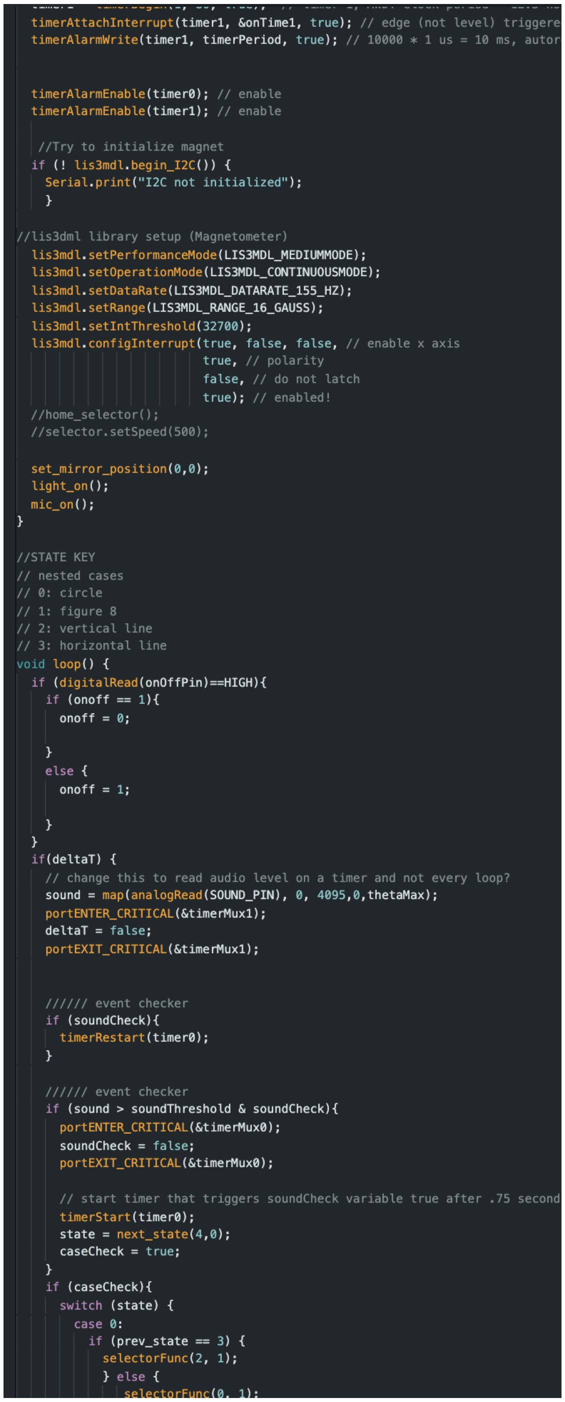
```
// home selector with limit switch (open position)


////// Service Routine
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  soundCheck = true; // the function to be called when timer interrupt is tr
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  deltaT = true; // the function to be called when timer interrupt is trigge
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void home_selector() {⋯
// TODO
// laser setup
void laser_on() {
  Serial.println("laser on");
  }

// TODO
// microphone set up
void mic_on() {
  Serial.println("mic on");
  }

// TODO
// move selector from, to
// 0 - open
// 1 - diffuse
// 2 - diffract
void selectorFunc(int from, int to) {
  int des_pos = 0;
  if (to == 0) {
    des_pos = 0;
  } else if (to == 1) {
    des_pos = 50;
  } else if (to == 2) {
    des_pos = 157;
  }

  if (currPos - des_pos > 0) {
    digitalWrite(dirPin, HIGH);
  }
  else {
    digitalWrite(dirPin, LOW);
  }

  Serial.println(abs(currPos-des_pos));
  for(int x = 0; x < abs(currPos-des_pos); x++)
  {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(1000);
  }
  currPos = des_pos;
  Serial.println((String)"Selector moved to position "+ currPos);
  }

// TODO
//set mirrors so laser output is XY
void set_mirror_position(int x, int y) {
  Serial.println("mirrors set");
  }

void light_on(){
  Serial.println("lights on");
  }

// returns next state
// range is the total # of states to chose from
// offset correlates the range to the correct states
int next_state(int range, int offset) {
  int r = (random(range)) + offset;
```



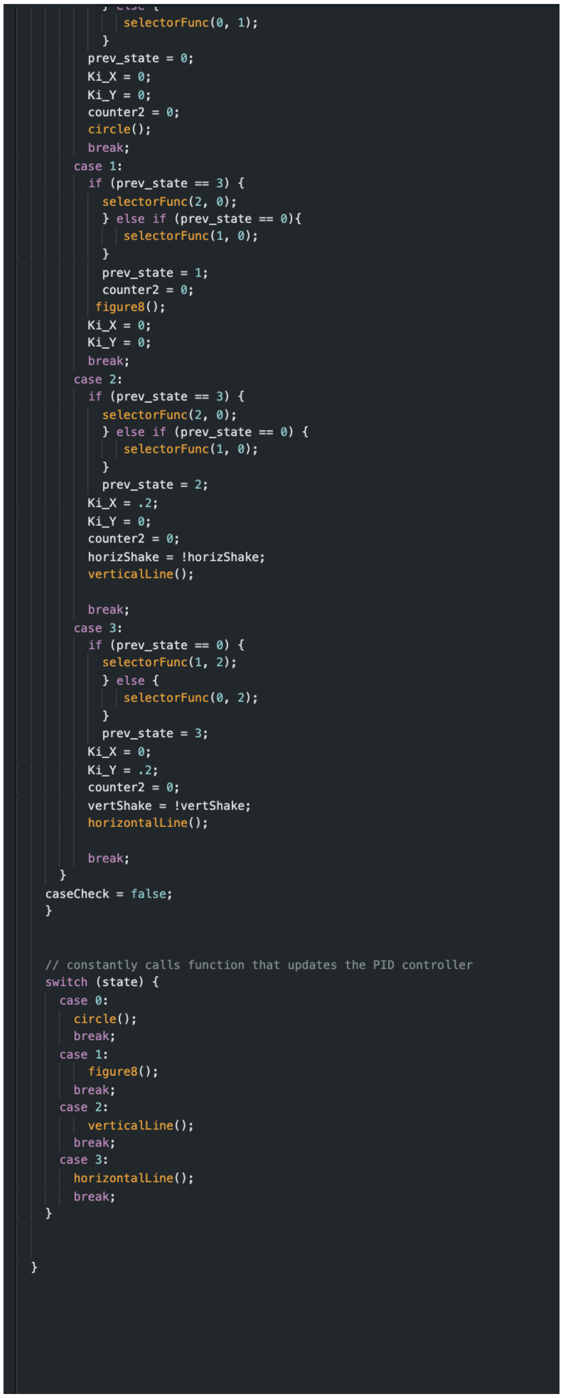
```
// offset correlates the range to the correct states
int next_state(int range, int offset) {
  int r = (random(range)) + offset;
  if (r != prev_state) {
    return (r);
  } else {
    next_state(range,offset);
  }
}

// sets next state between low, high
// set_next_state(X,Y) where X+2=Y => state=X;X+1;Y
// set_next_state(X,X) will set state=X
void set_next_state(int low, int high) {
  prev_state = state;
  state = next_state((high - low), (low));
  }

// LASER MOVEMENTS
void figure8(){
  sinValue = (sin(.1*counter)+1);
  sinValue2 = (cos(.2*counter+1.22)+1);
  counter += 1;

  thetaDes_X = sinValue2*(10)+170;
  thetaDes_Y = sinValue*(yRange)+yOffset;

  PID_X();
  PID_Y();
}

void circle(){
  sinValue = (sin(.1*counter)+1);
  sinValue2 = (cos(.1*counter)+1);
  counter += 1;

  thetaDes_X = sinValue2*(xRange)+xOffset;
  thetaDes_Y = sinValue*(yRange)+yOffset;
  PID_X();
  PID_Y();
}

void horizontalLine(){
  sinValue = (sin(.1*counter)+1);
  counter += 1;

  if (vertShake){
    newCounter();
    thetaDes_X = sinValue*(40)+xOffset;
    thetaDes_Y = 135+counter2/10;
  }
  else {
    thetaDes_X = sinValue*(40)+xOffset;
    thetaDes_Y = 135;
  }
  PID_X();
  PID_Y();

}

void verticalLine(){
  sinValue = (sin(.1*counter)+1);
  sinValue2 = (cos(.1*counter)+1);
  counter += 1;
  if (horizShake){
    newCounter();
    thetaDes_X = 195 + counter2/10;
    thetaDes_Y = sinValue*(yRange)+yOffset;

  }
  else {
  thetaDes_X = 195;
  thetaDes_Y = sinValue*(yRange)+yOffset;
  }

  PID_X();
  PID_Y();
}
```



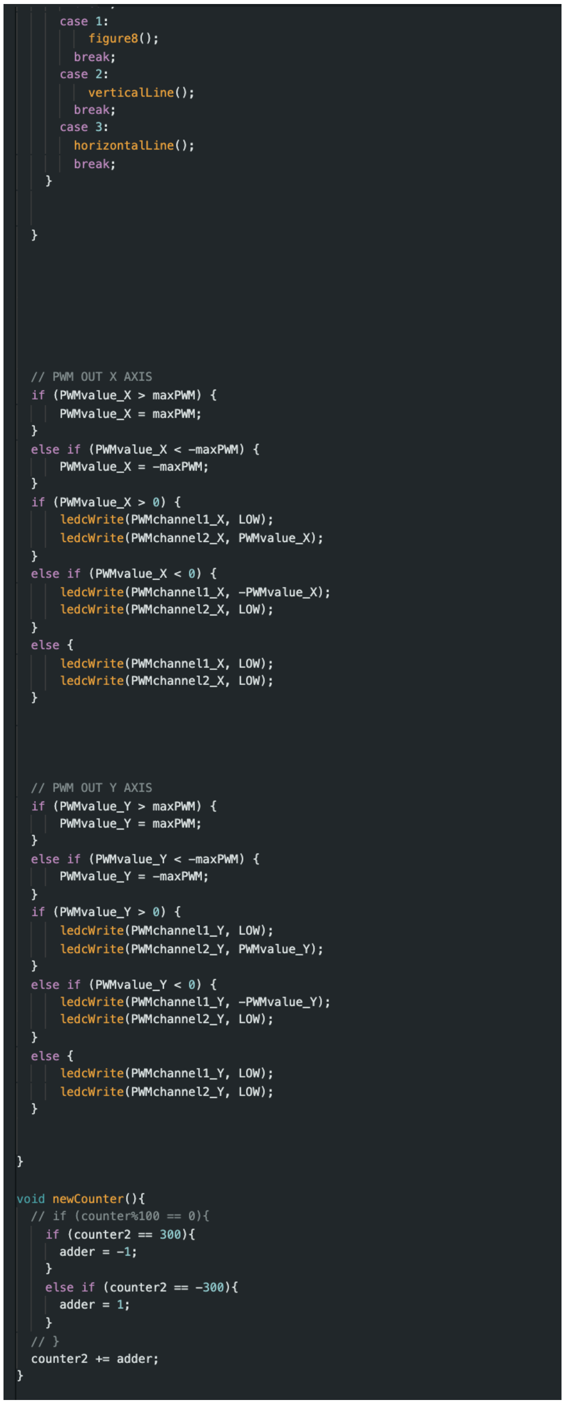
```
}

// PID FUNCTIONS
void PID_X(){
  position_X =  map(analogRead(POSITION_PIN_X), 0, 4095,0,thetaMax);
  currentTime_X = micros();
  deltaTime_X = (currentTime_X - previousTime_X) / 1000000.0;
  previousTime_X = currentTime_X;
  error_X = position_X - thetaDes_X;
  errorIntegral_X = errorIntegral_X + error_X*deltaTime_X;

  if (errorIntegral_X > KiMax_X){
    errorIntegral_X = KiMax_X;
  }
  else if (errorIntegral_X < -KiMax_X){
    errorIntegral_X = -KiMax_X;
  }


  PWMvalue_X = Kp_X*error_X + Ki_X*errorIntegral_X; Kd_X * (error_X-previous
  previousError_X = error_X;

}

void PID_Y(){
  position_Y =  map(analogRead(POSITION_PIN_Y), 0, 4095,0,thetaMax);
  currentTime_Y = micros();
  deltaTime_Y = (currentTime_Y - previousTime_Y) / 1000000.0;
  previousTime_Y = currentTime_Y;
  error_Y = position_Y - thetaDes_Y;
  errorIntegral_Y = errorIntegral_Y + error_Y*deltaTime_Y;

  if (errorIntegral_Y > KiMax_Y){
    errorIntegral_Y = KiMax_Y;
  }
  else if (errorIntegral_Y < -KiMax_Y){
    errorIntegral_Y = -KiMax_Y;
  }


  PWMvalue_Y = Kp_Y*error_Y + Ki_Y*errorIntegral_Y; Kd_Y * (error_Y-previous
  previousError_Y = error_Y;

}


void setup() {
  Serial.begin(115200);
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(onOffPin, INPUT);
  pinMode(SOUND_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW); // sets the initial state of LED as turned-off



  // PWM SETUP X
  ledcSetup(PWMchannel1_X, freq, resolution);
  ledcSetup(PWMchannel2_X, freq, resolution);
  ledcAttachPin(PWM_PIN_1_X, PWMchannel1_X);
  ledcAttachPin(PWM_PIN_2_X, PWMchannel2_X);


  // PWM SETUP Y
  ledcSetup(PWMchannel1_Y, freq, resolution);
  ledcSetup(PWMchannel2_Y, freq, resolution);
  ledcAttachPin(PWM_PIN_1_Y, PWMchannel1_Y);
  ledcAttachPin(PWM_PIN_2_Y, PWMchannel2_Y);


  // TIMER SET UP
  timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns
  timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggere
  timerAlarmWrite(timer0, 750000, true); // 5000000 * 1 us = .5 s, auto relo

  timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggere
  timerAlarmWrite(timer1, timerPeriod, true); // 10000 * 1 us = 10 ms, autor
```



```
  timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggere
  timerAlarmWrite(timer1, timerPeriod, true); // 10000 * 1 us = 10 ms, autor


  timerAlarmEnable(timer0); // enable
  timerAlarmEnable(timer1); // enable

   //Try to initialize magnet
  if (! lis3mdl.begin_I2C()) {
    Serial.print("I2C not initialized");
    }

//lis3dml library setup (Magnetometer)
  lis3mdl.setPerformanceMode(LIS3MDL_MEDIUMMODE);
  lis3mdl.setOperationMode(LIS3MDL_CONTINUOUSMODE);
  lis3mdl.setDataRate(LIS3MDL_DATARATE_155_HZ);
  lis3mdl.setRange(LIS3MDL_RANGE_16_GAUSS);
  lis3mdl.setIntThreshold(32700);
  lis3mdl.configInterrupt(true, false, false, // enable x axis
                          true, // polarity
                          false, // do not latch
                          true); // enabled!
  //home_selector();
  //selector.setSpeed(500);

  set_mirror_position(0,0);
  light_on();
  mic_on();
}

//STATE KEY
// nested cases
// 0: circle
// 1: figure 8
// 2: vertical line
// 3: horizontal line
void loop() {
  if (digitalRead(onOffPin)==HIGH){
    if (onoff == 1){
      onoff = 0;

    }
    else {
      onoff = 1;

    }
  }
  if(deltaT) {
    // change this to read audio level on a timer and not every loop?
    sound = map(analogRead(SOUND_PIN), 0, 4095,0,thetaMax);
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);



    ////// event checker
    if (soundCheck){
      timerRestart(timer0);
    }

    ////// event checker
    if (sound > soundThreshold & soundCheck){
      portENTER_CRITICAL(&timerMux0);
      soundCheck = false;
      portEXIT_CRITICAL(&timerMux0);

      // start timer that triggers soundCheck variable true after .75 second
      timerStart(timer0);
      state = next_state(4,0);
      caseCheck = true;
    }
    if (caseCheck){
      switch (state) {
        case 0:
          if (prev_state == 3) {
            selectorFunc(2, 1);
            } else {
              selectorFunc(0, 1);
```



```
          } else {
             selectorFunc(0, 1);
          }
        prev_state = 0;
        Ki_X = 0;
        Ki_Y = 0;
        counter2 = 0;
        circle();
        break;
      case 1:
        if (prev_state == 3) {
          selectorFunc(2, 0);
          } else if (prev_state == 0){
             selectorFunc(1, 0);
          }
          prev_state = 1;
          counter2 = 0;
         figure8();
        Ki_X = 0;
        Ki_Y = 0;
        break;
      case 2:
        if (prev_state == 3) {
          selectorFunc(2, 0);
          } else if (prev_state == 0) {
             selectorFunc(1, 0);
          }
          prev_state = 2;
        Ki_X = .2;
        Ki_Y = 0;
        counter2 = 0;
        horizShake = !horizShake;
        verticalLine();

        break;
      case 3:
        if (prev_state == 0) {
          selectorFunc(1, 2);
          } else {
             selectorFunc(0, 2);
          }
          prev_state = 3;
        Ki_X = 0;
        Ki_Y = .2;
        counter2 = 0;
        vertShake = !vertShake;
        horizontalLine();

        break;
    }
  caseCheck = false;
  }


  // constantly calls function that updates the PID controller
  switch (state) {
    case 0:
      circle();
      break;
    case 1:
        figure8();
      break;
    case 2:
        verticalLine();
      break;
    case 3:
      horizontalLine();
      break;
  }


}
```



```
      case 1:
          figure8();
        break;
      case 2:
          verticalLine();
        break;
      case 3:
        horizontalLine();
        break;
    }


  }







  // PWM OUT X AXIS
  if (PWMvalue_X > maxPWM) {
      PWMvalue_X = maxPWM;
  }
  else if (PWMvalue_X < -maxPWM) {
      PWMvalue_X = -maxPWM;
  }
  if (PWMvalue_X > 0) {
      ledcWrite(PWMchannel1_X, LOW);
      ledcWrite(PWMchannel2_X, PWMvalue_X);
  }
  else if (PWMvalue_X < 0) {
      ledcWrite(PWMchannel1_X, -PWMvalue_X);
      ledcWrite(PWMchannel2_X, LOW);
  }
  else {
      ledcWrite(PWMchannel1_X, LOW);
      ledcWrite(PWMchannel2_X, LOW);
  }



  // PWM OUT Y AXIS
  if (PWMvalue_Y > maxPWM) {
      PWMvalue_Y = maxPWM;
  }
  else if (PWMvalue_Y < -maxPWM) {
      PWMvalue_Y = -maxPWM;
  }
  if (PWMvalue_Y > 0) {
      ledcWrite(PWMchannel1_Y, LOW);
      ledcWrite(PWMchannel2_Y, PWMvalue_Y);
  }
  else if (PWMvalue_Y < 0) {
      ledcWrite(PWMchannel1_Y, -PWMvalue_Y);
      ledcWrite(PWMchannel2_Y, LOW);
  }
  else {
      ledcWrite(PWMchannel1_Y, LOW);
      ledcWrite(PWMchannel2_Y, LOW);
  }


}

void newCounter(){
  // if (counter%100 == 0){
    if (counter2 == 300){
      adder = -1;
    }
    else if (counter2 == -300){
      adder = 1;
    }
  // }
  counter2 += adder;
}
```

CAD