Remote Controlled Maze

Daniel Richards, Laura Wong, Kyle Woody

ME 102B Fall 2024

Opportunity

Our goal was to create an interactive activity that is enjoyable for all ages and promotes improved basic cognitive and fine motor control skills. This took the form of a rotating marble maze—a device that holds a course perpendicular to the ground and rotates it based on user input so that a marble within it can fall through. The primary utility of the device is as a form of entertainment, although it may also be useful as a physical therapy tool, as an exercise in hand-eye coordination and fine motor skills.

Strategy and Functionality

Our original high-level design was a maze with 2 degrees of rotational freedom that was parallel to the ground in its neutral position. Using a set of two motors and concentric gimbals, the maze would pitch and roll to move a marble through it. The desired user input was an IMU, with the maze movement corresponding with the rotation of the IMU when a button was held down. A potentiometer would be included to adjust the sensitivity of the controller, along with a button to reset the maze to its neutral position. These user input elements would be mounted separate from the microcontroller and motor electronics on a handheld board. To limit the actuation of the gimbals to within a tilt angle of ±15 degrees and enable feedback control of motor speed, magnetic encoders would be placed on the motors.

After considering several mechanisms and layouts for the maze, we eventually realized that it would be simpler and more cost-effective to eliminate one degree of freedom and mount the maze in a vertical position, rotating it about a single axis. Marbles would be dropped in from a hole at the top and would be guided to the opposite side of the maze. This approach would have much simpler transmission and control, allowing us to focus more on the mechanical design and user experience.

The final design uses the same user input as envisioned for the original maze, albeit with only one accelerometer axis being observed. An encoder is still used on the maze for position and velocity control, but the actuation did not need to be limited since the maze could rotate a full 360 degrees as opposed to the original design. Whereas the previous design had no motor velocity limit, the encoder was used to enforce a maximum angular velocity of 72 deg/s on the final design. The potentiometer was removed from the design because it was not necessary and would have resulted in a cluttered controller. To keep the center of gravity low and reduce the size and complexity of the enclosure, the motor is kept close to the bottom of the assembly and uses a simple geared transmission system to actuate the maze above. Lastly, position control was added to the final design, using the magnetic encoder and proportional-integral feedback to return the maze back to its upright position. This was necessary to conveniently drop in new marbles.

Function Critical Decisions & Calculations

<u>REQUIRED MOTOR TORQUE:</u> The most critical consideration was sourcing a motor that could safely operate at a top speed of 72 degrees per second — one full revolution of the maze in five seconds — and would be able to reach this speed from a standstill within 0.01 seconds. Since the user will likely be starting and stopping the maze regularly during operation, the motor will approach maximum torque frequently.

$\omega = 72\, deg/s = 1.26\, rad/s = \alpha t = 0.01\alpha\, \alpha = 126\, rad/s^2$

$\tau = I\alpha = \frac{1}{2}mr^2 * a = 0.5 * 0.0785kg * (0.0762m)^2 * 126\, rad/s^2 = 2.87\, N{\cdot}cm$

$\tau_{motor} = \frac{\tau}{0.6} = 4.78\, N{\cdot}cm = 0.487\, kg{\cdot}cm$

Assuming that the operational torque limit is 60% of the stall torque, the motor must have a stall torque rating of at least 0.487 kg·cm. The final selected motor has a stall torque rating of 4.7 kg·cm.

FORCES ON BEARINGS: Bearings were not considered a point of failure since they are made of steel. However, the bearing loads are necessary to determine so that the housing can be optimally designed. The bearing loads are calculated below and used as an input for the generative design of the housing in Autodesk Fusion.

Bearings A & B

$W_{15T} = ((0.212oz)(20\%) + 3.2g + 4g)(9.81) = 0.082N$

$\Sigma F_y = 0 = A_y + B_y - F_{15T/21T,y} - W_{15T}$

$\Sigma M_A = 0 = M_{F_{15T/21T,y}} + M_b + M_{W15T}$

$A_y = F_{15T/21T,y} - B_y = 0.24N$

Bearings C & D

$W_{21T} = ((8.159g)(20\%) + 3.2g + 4g)(9.81) = 0.087N$

$\Sigma F_y = 0 = C_y + D_y + F_{15T/21T,y} - F_{21T/15T,y} - W_{21T}$

$\Sigma M_C = 0 = M_D + M_{F_{21T/15T,y}} + M_{F_{15T/21T,y}} + M_{W21T}$

$C_y = F_{21T/15T,y} + W_{21T} - D_y - F_{15T/21T,y} = 0.029N$

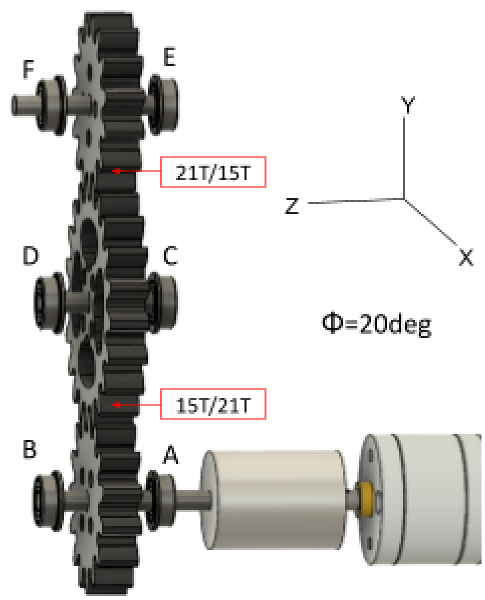
Bearings E & F

$W_{maze} = 0.77N$

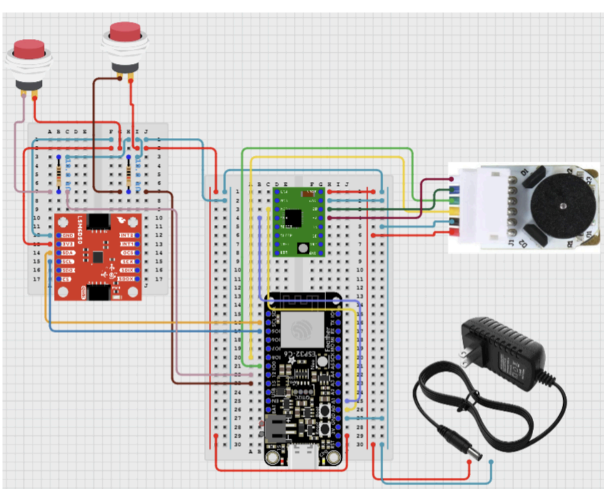
$\Sigma F_y = 0 = E_y + G_y + F_{21T/15T,y} - W_{15T} - W_{maze}$

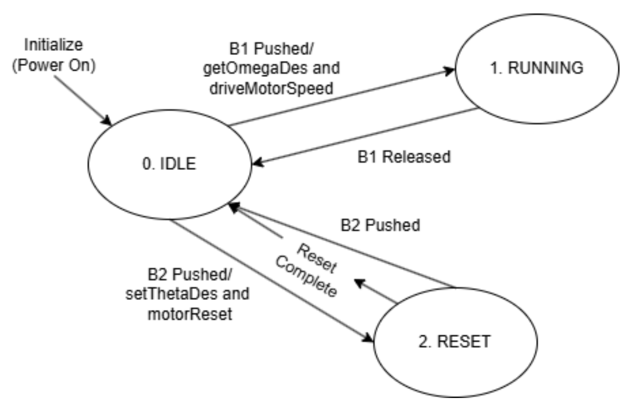
$\rightarrow G_y = 0.021$

$E_y = W_{15T} - G_y - F_{21T/15T,y} = -0.51N$



Diagrams

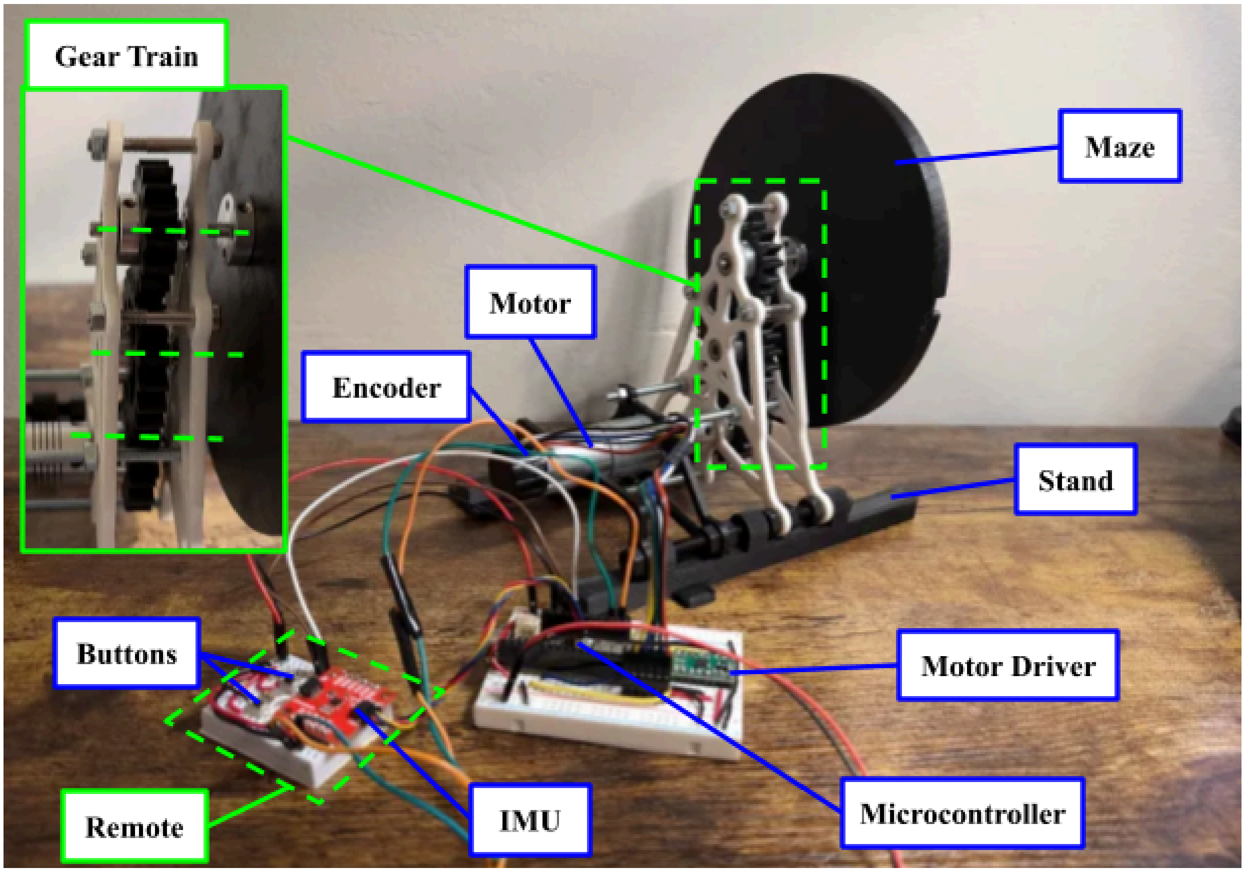




(Left) Circuit Diagram.

(Right) State Transition Diagram.

Integrated Assembly



Reflection

The project went smoothly and we encountered no major obstacles. The main area where we strove for a simple design was in the transmission, where we reduced the degrees of freedom of the maze's motion with the aim of focusing our resources onto a single well-designed, robust mechanism. However, some parts of the maze's structural housing were probably overdesigned. These parts were made using a Markforged printer with carbon reinforcements, and resulted in unnecessary cost and manufacturing lead time.

Desired improvements to this design include the addition of a limit switch to allow the maze to zero its position by itself, addition of spacers on the threaded screws connecting the two sides of the transmission housing to prevent over-torquing, and further tweaking of the control algorithm to improve the response at low user inputs. In addition, assembly of the housing was difficult in some areas, and the use of nuts on a threaded rod as the sole locating feature for some components was especially troublesome. It would be better for the maze's base to have tabs that act as locating features for the motor and transmission housings. Lastly, fasteners significantly loosened after extended operation, causing the transmission system to slip. In future designs, the use of secondary fastener retention is paramount, as well as machining flats onto the shafts to reduce the amount of preload needed to clamp the gears and minimize wear.

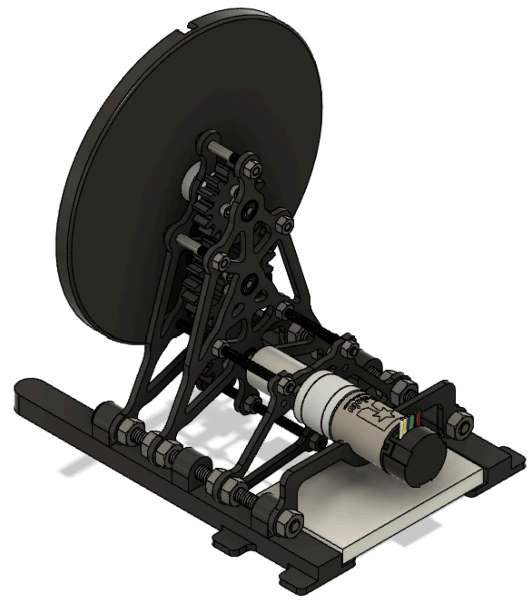
Appendix 1: BOM

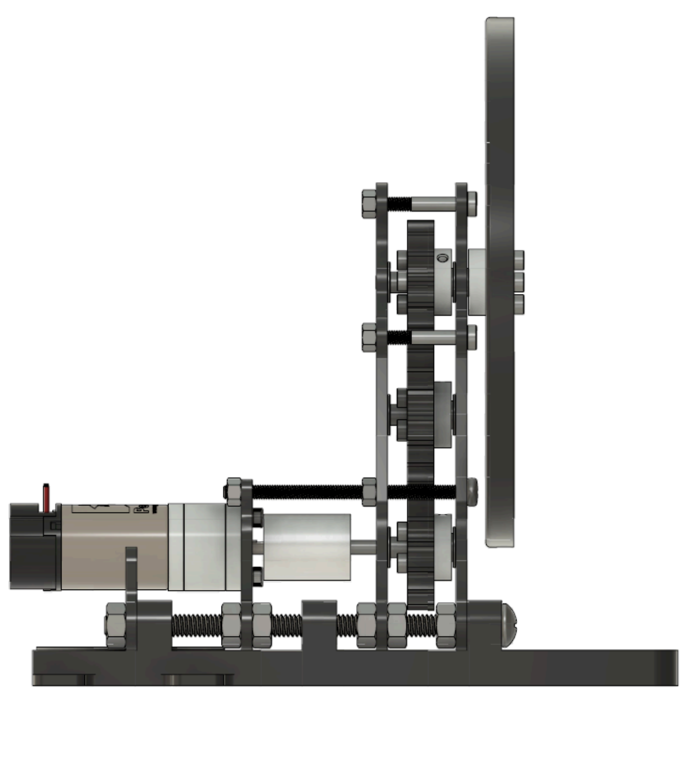
Item Name	Description	Price (ea.)	Qty.	Vendor	Link to Item	Notes	Subtotal
Onyx filament	Markforged filament (priced per cm^3)	$ 0.27	122	Jacobs Hall	https://store.jacobshall.org/products/markforged-onyx-filament-per-cubic-inch?_pos=1&_sid=f22dbf23a&_ss=r	Planned usage based on 20% infill	$ 32.94
1/16" acrylic	12" x 12" acrylic sheet	$ 5.14	1	McMaster	https://www.mcmaster.com/8589K11/		$ 5.14
4mm shaft	4mm dia x 200 mm length steel shaft	$ 5.25	1	McMaster	https://www.mcmaster.com/1327K507/	Will be cut into the lengths needed for our design	$ 5.25
7/64" Ball Bearing Ball	Steel wear-resistant ball, 7/64" dia (pkg of 100)	$ 7.99	1	McMaster	https://www.mcmaster.com/9529K35/	One needed in total	$ 7.99
25d metal gearmotor	12V motor with encoder and 4mm shaft	$ 45.95	1	Pololu	https://www.pololu.com/product/4868		$ 45.95
Pololu Universal 4mm shaft mount	Mounting hub for 4mm shafts, compatible with M3 screws	$ 8.49	2	Pololu	https://www.pololu.com/product/1997	2 Pack	$ 16.98
Stainless Steel ring shim	4mm ID, 8mm OD (pkg of 10)	$ 11.82	1	McMaster	https://www.mcmaster.com/90214A151/	Four needed in total	$ 11.82

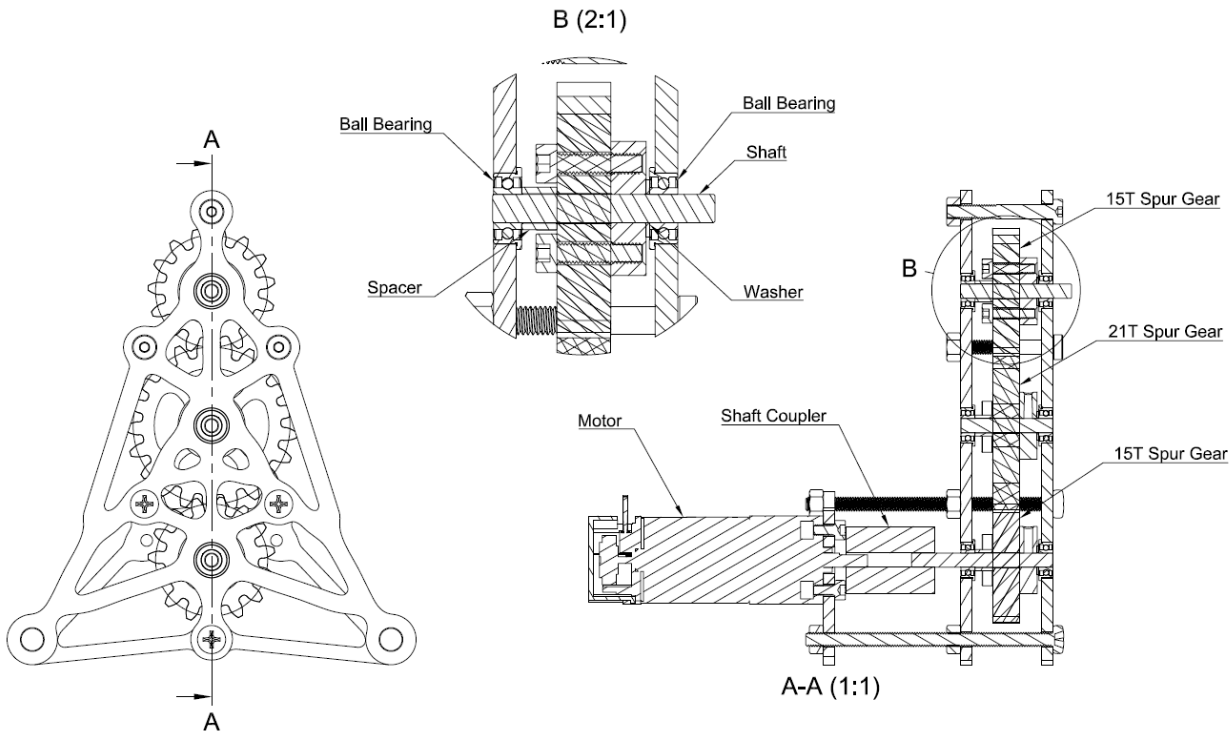
Flexible Shaft Connector	4mm-to-4mm aluminum shaft connector	$ 7.99	1	Amazon	https://www.amazon.com/uxcell-Aluminum-Coupling-Flexible-Connector/dp/B07G6QBR8Y?th=1		$ 7.99
M3 Screw	12 mm lg, alloy steel (pkg of 100)	$ 11.29	1	McMaster	https://www.mcmaster.com/91290A117/	12 needed in total	$ 11.29
M3 Screw (short)	6 mm lg, zinc-coated steel (pkg of 50)	$ 6.45	1	McMaster	https://www.mcmaster.com/91274A102/	Two needed in total	$ 6.45
M4 Nut	Zinc-plated hex nut (pkg of 50)	$ 7.14	1	McMaster	https://www.mcmaster.com/90725A025/	Nine needed in total	$ 7.14
M4 Hex Head Screw	30mm lg stainless steel (pkg of 25)	$ 7.86	1	McMaster	https://www.mcmaster.com/92855A425/	Three needed in total	$ 7.86
M4 Philips Head Screw	70 mm lg zinc-plated steel (pkg of 10)	$ 4.69	1	McMaster	https://www.mcmaster.com/92005A248/	Three needed in total	$ 4.69
Ball Bearing	Flanged, shielded, for 4mm shaft diameter	$ 7.49	1	Amazon	https://www.amazon.com/uxcell-Silver-Premium-Flanged-Bearing/dp/B00G9W20VU		$ 7.49
Unthreaded spacer	Aluminum, 5mm lg, for 4mm dia shaft	$ 2.02	3	McMaster	https://www.mcmaster.com/94669A008/		$ 6.06
1/4"-20 Hex Nut	Black-oxide steel (pkg of 50)	$ 8.90	1	McMaster	https://www.mcmaster.com/95479A111/	12 needed in total	$ 8.90
1/4"-20 Phillips Head Screw	Black-oxide steel, 4.5" lg	$ 2.22	2	McMaster	https://www.mcmaster.com/91249A646/		$ 4.44

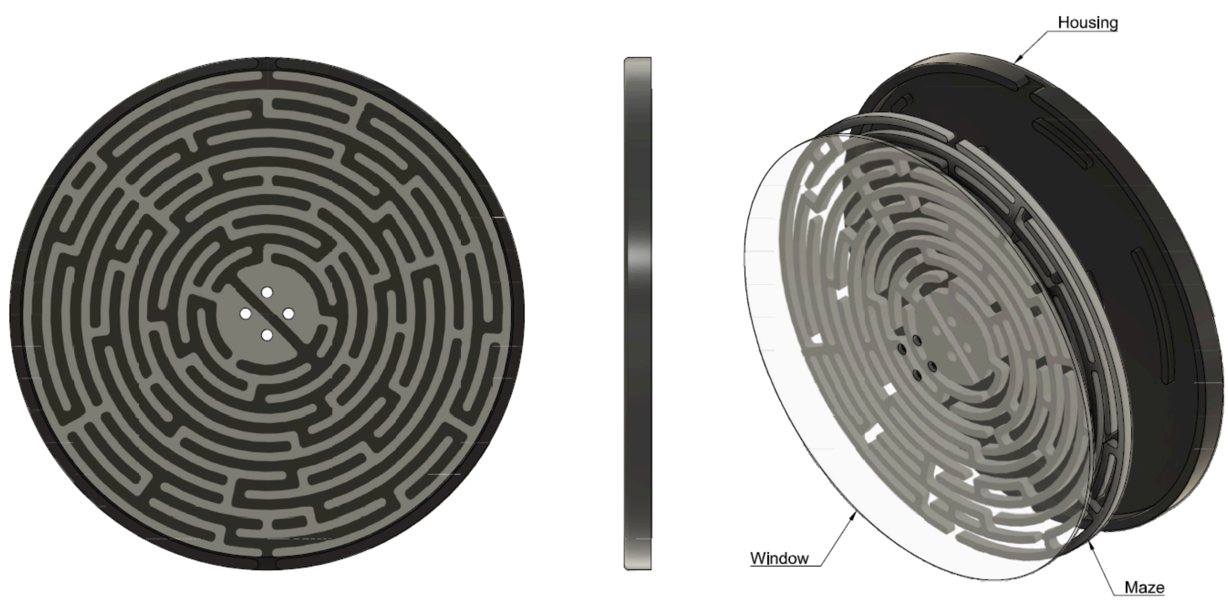
1/4"-20 Phillips Head Screw	Black-oxide steel, 4.5" lg	$ 2.22	2	McMaster	https://www.mcmaster.com/91249A646/		$ 4.44
12V Power Supply	Power supply for motor	$ -	1	Pre-owned	-	-	$ -
Small breadboard	Breadboard for remote	$ -	1	Pre-owned	-	-	$ -
Medium breadboard	Breadboard for motor electronics and esp32	$ -	1	Pre-owned	-	-	$ -
ESP32	Microcontroller	$ -	1	Pre-owned	-	-	$ -
LSM6DS0 IMU	6-axis DOF IMU module	$ -	1	Pre-owned	-	-	$ -
ESP32 cable	Adapter from USB-A on computer to USB-C on esp32	$ -	1	Pre-owned	-	-	$ -
ESP32 cable	Adapter from USB-A on computer to USB-C on esp32	$ -	1	Pre-owned	-	-	$ -
Wiring and resistors	Includes jumper cables and solid-core wire	$ -	1	Pre-owned	-	-	$ -
Push buttons	Momentary push buttons	$ -	1	Pre-owned	-	-	$ -

Appendix 2: CAD






B (2:1)
Ball Bearing
Ball Bearing
Shaft
Spacer
Washer
A
A
15T Spur Gear
B
21T Spur Gear
Motor
Shaft Coupler
15T Spur Gear
A-A (1:1)


Housing
Window
Maze

Appendix 3: Code

```
#include <ESP32Encoder.h>
#include <Adafruit_LSM6DSOX.h>
#include <Wire.h>
#define BIN_1 26
#define BIN_2 25
#define LED_PIN 13
#define BTN 12
#define BTN2 13

ESP32Encoder encoder;
Adafruit_LSM6DSOX lsm6dso;

float omegaSpeed = 0;
float omegaDes = 0;
float omegaMax = 45;  // CHANGE THIS VALUE TO YOUR MEASURED MAXIMUM SPEED
float accel_y = 0;
float D = 0;
float error = 0;
int theta = 0;
int thetaMax = 4123;
int thetaRange = 50;
int thetaDes = 0;
float errorSum = 0;

float Kp = 5;  // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
float Ki = 0.05; //note effective value is Ki/10
float resetKp = 1;
float resetKi = 0.05;

//Setup interrupt variables ----------------------------
volatile int count = 0;                    // encoder count
hw_timer_t* timer0 = NULL;
hw_timer_t* timer1 = NULL;
hw_timer_t * timer2 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

// setting PWM properties ----------------------------
const int freq = 20000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
```

```
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;

// button setup -----------------------------------
byte state = 0;
volatile bool DEBOUNCINGflag = false;
volatile bool BUTTONflag = false;
volatile bool DEBOUNCING2flag = false;
volatile bool BUTTON2flag = false;
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR isr() {  // Button 1 isr
  BUTTONflag = true;
}

void IRAM_ATTR isr2() {  // Button 2 isr
  BUTTON2flag = true;
}

void IRAM_ATTR onTime2() {  // Debouncing for button 1
  portENTER_CRITICAL_ISR(&timerMux2);
  DEBOUNCINGflag = false;
  portEXIT_CRITICAL_ISR(&timerMux2);
  timerStop(timer2);
  BUTTONflag = false;
}

void IRAM_ATTR onTime0() {    // Debouncing for button 2
  portENTER_CRITICAL_ISR(&timerMux0);
  DEBOUNCING2flag = false;
  portEXIT_CRITICAL_ISR(&timerMux0);
  timerStop(timer0);
  BUTTON2flag = false;
}

void IRAM_ATTR onTime1() {  // Updates encoder values and resets count to
zero
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount();
  theta = count + theta;
  if (theta < 0) {
    theta += thetaMax;
```

```
  }
  if (theta > thetaMax) {
    theta -= thetaMax;
  }
  //Serial.println(theta);
  encoder.clearCount();
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void setup() {
  // put your setup code here, to run once:
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);  // sets the initial state of LED as
turned-off

  pinMode(BTN, INPUT);                // configures the specified pin to
behave either as an input or an output
  attachInterrupt(BTN, isr, RISING);
  attachInterrupt(BTN2, isr2, RISING);

  timer2 = timerBegin(1000000);
  timerAttachInterrupt(timer2, &onTime2);
  timerAlarm(timer2, 150000, true, 0);
  timerStop(timer2);

  timer0 = timerBegin(1000000);
  timerAttachInterrupt(timer0, &onTime0);
  timerAlarm(timer0, 150000, true, 0);
  timerStop(timer0);

  Wire.begin(22, 14);
  Wire.setClock(400000);


  Serial.begin(115200);
  ESP32Encoder::useInternalWeakPullResistors = puType::up;  // Enable the
weak pull up resistors
  encoder.attachHalfQuad(27, 33);                           // Attache pins
for use as encoder pins
  encoder.setCount(0);                                      // set starting
count value after attaching

  if (!lsm6dso.begin_I2C(0x6B)) {  // Use the address 0x6B found by the
```

```
scanner
    Serial.println("Failed to find LSM6DSO chip");
    while (1) {
        Serial.println("Check wiring and reset board");
        delay(1000);
    }
  }
  Serial.println("LSM6DSO Found!");

  // Set IMU ranges and rates
  lsm6dso.setAccelRange(LSM6DS_ACCEL_RANGE_2_G);
  lsm6dso.setGyroRange(LSM6DS_GYRO_RANGE_250_DPS);
  lsm6dso.setAccelDataRate(LSM6DS_RATE_104_HZ);
  lsm6dso.setGyroDataRate(LSM6DS_RATE_104_HZ);


  // Method 2
  // configure PWM functionalitites with attaching the channel to the GPIO
to be controller
  ledcAttach(BIN_1, freq, resolution);
  ledcAttach(BIN_2, freq, resolution);

  timer1 = timerBegin(1000000);            // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer1, &onTime1);  // Attach onTimer1 function to
our timer.
  timerAlarm(timer1, 10000, true, 0);      // 10000 * 1 us = 10 ms,
autoreload true
}

void loop() {
  switch (state) {
    case 0: //motor is stopped
      if (CheckForButtonPress() == true) { // Event checker: button 1 is
pressed
          state = 1;  // Service function: Move to state 1
          Serial.println("Button 1 pressed, moving to state 1");
      }
      else if (CheckForButton2Press() == true) { // Event checker: button 2
is pressed
        state = 2; // Service function: move to state 2
        setThetaDes(); // Service function: Determines if motor travels CW
or CCW to zero position
        Serial.println("Button 2 pressed, moving to state 2");
```

```
      }
      else {
        driveMotorSpeed(); // Service function: Uses PWM control to match
motor speed with omegaDes
      }
      break;

    case 1: //motor is running
      if (digitalRead(BTN) == false) { // Event checker: button 1 is
released
        state = 0; // Service function: Move to state 0
        omegaDes = 0; // Service function: sets omegaDes to zero
        Serial.println("Button 1 released, moving to state 0");
      }
      else {
        getOmegaDes(); // Service function: Interperts data from the IMU to
set omegaDes
        driveMotorSpeed(); // Service function: Uses PWM control to match
motor speed with omegaDes
      }
    break;

    case 2: // the motor is being reset
      if (CheckForButton2Press() == true) { // Event checker: button 2 is
pressed
        state = 0; // Service function: Move to state zero, aborting the
motor reset
        omegaDes = 0; // Service function: sets omegaDes to zero
        Serial.println("Button 2 pressed, moving to state 0");
      }
      else if (abs(theta) < thetaRange) { // Event checker: The motor is
close to its reset position
        state = 0;  // Service function: Move to state zero, the reset is
complete
        omegaDes = 0; // Service function: sets omegaDes to zero
        Serial.println("Motor position is reset, moving to state 0");
      }
      else {
        motorReset(); // Service functio: moves the motor towards position
zero
      }
    break;
  }
```

```
}

//Other functions

bool CheckForButtonPress() {  // Event checker: Returns true if button 1 is
pressed and is not being debounced
  if (BUTTONflag==true && DEBOUNCINGflag==false) {
    portENTER_CRITICAL_ISR(&timerMux2);
    DEBOUNCINGflag = true;
    portEXIT_CRITICAL_ISR(&timerMux2);
    timerStart(timer2);
    return true;
  }
  else {
    return false;
  }
}

bool CheckForButton2Press() { // Event checker: Returns true if button 2 is
pressed and is not being debounced
  if (BUTTON2flag==true && DEBOUNCING2flag==false) {
    portENTER_CRITICAL_ISR(&timerMux0);
    DEBOUNCING2flag = true;
    portEXIT_CRITICAL_ISR(&timerMux0);
    timerStart(timer0);
    return true;
  }
  else {
    return false;
  }
}

void driveMotorSpeed() { // Service function: PWM control of the motor to
match the speed omegaDes
  omegaSpeed = count;
  error = omegaDes - omegaSpeed;
    errorSum = errorSum + error/10;
    D = Kp*error + Ki*errorSum;

    if (D > MAX_PWM_VOLTAGE) {
    D = MAX_PWM_VOLTAGE;
    errorSum = errorSum - error/10;
  } else if (D < -MAX_PWM_VOLTAGE) {
```

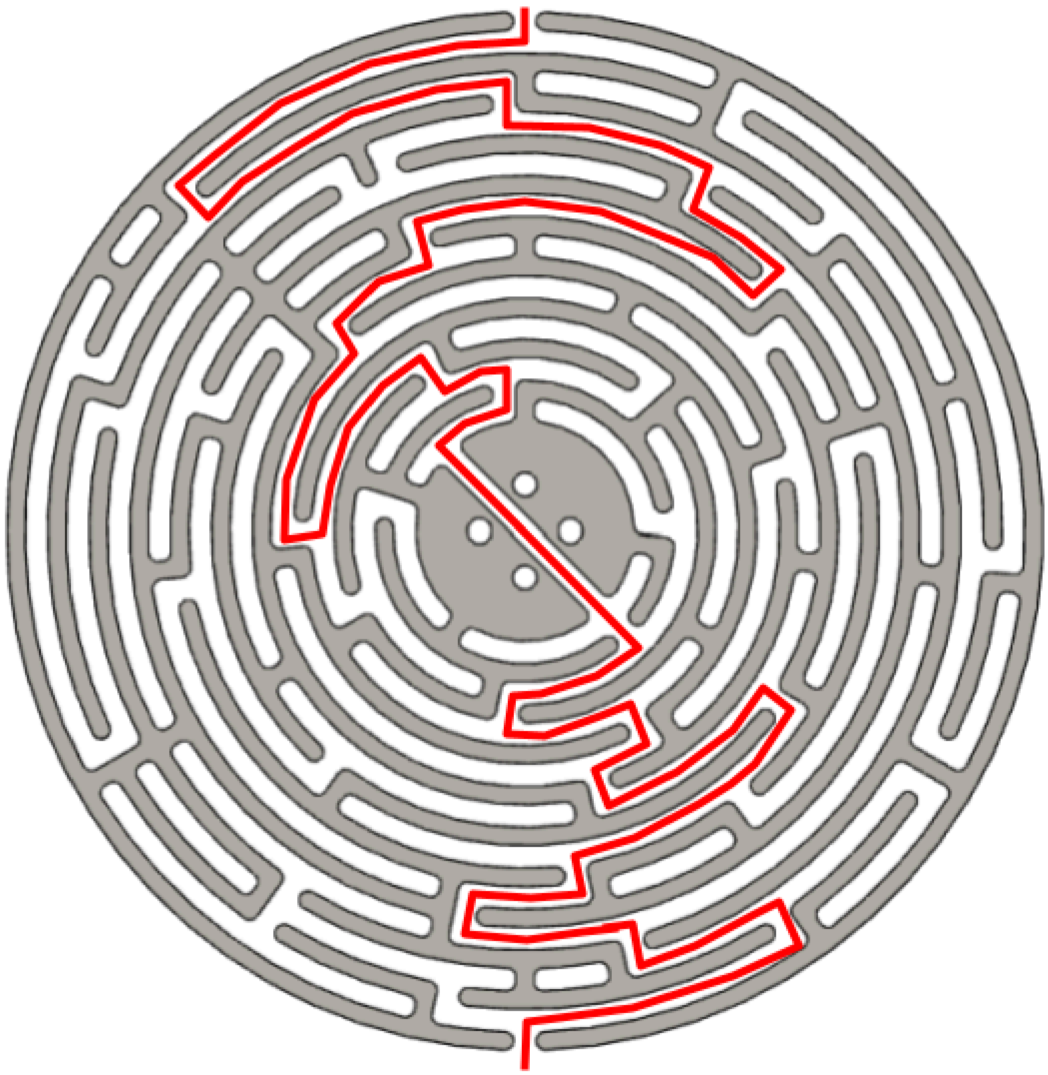
```
    D = -MAX_PWM_VOLTAGE;
    errorSum = errorSum - error/10;
  }

  if (D > 0) {
    ledcWrite(BIN_1, LOW);
    ledcWrite(BIN_2, D);
  } else if (D < 0) {
    ledcWrite(BIN_2, LOW);
    ledcWrite(BIN_1, -D);
  } else {
    ledcWrite(BIN_2, LOW);
    ledcWrite(BIN_1, LOW);
  }
}
void motorReset() { // Service function: Returns the motor to the position
it was in when the program was started
  error = theta - thetaDes;
    errorSum = errorSum + error/10;
    D = resetKp*error + resetKi*errorSum;
    if (D > MAX_PWM_VOLTAGE) {
    D = MAX_PWM_VOLTAGE;
    errorSum = errorSum - error/10;
  } else if (D < -MAX_PWM_VOLTAGE) {
    D = -MAX_PWM_VOLTAGE;
    errorSum = errorSum - error/10;
  }
  if (D > 0) {
    ledcWrite(BIN_1, LOW);
    ledcWrite(BIN_2, D);
  } else if (D < 0) {
    ledcWrite(BIN_2, LOW);
    ledcWrite(BIN_1, -D);
  } else {
    ledcWrite(BIN_2, LOW);
    ledcWrite(BIN_1, LOW);
  }
}

void setThetaDes() {  // Service function: determines if the motor should
move CW or CCW to return to the zero position
  if (theta > thetaMax/2) {
    thetaDes = 0;
```

```
  }
  else {
    thetaDes = thetaMax;
  }
}
void getOmegaDes() { // Service function: Uses IMU data to set omegaDes
    sensors_event_t accel;
  sensors_event_t gyro;
  sensors_event_t temp;
  lsm6dso.getEvent(&accel, &gyro, &temp);
  accel_y = accel.acceleration.y;
  omegaDes = -accel_y*omegaMax/9.81;
}
```

Appendix 4: Demo Maze Solution



Appendix 5: User Manual

To operate the maze, hold down the controller's right button. The maze will move while this button is depressed. To control the motion of the maze, rotate the controller from side to side about its long axis. The speed of the maze's movement is proportional to the tilt angle of the controller.

When pressed, the left button resets the position of the maze to where it was located at startup. It is recommended to manually reset the maze before removing power to the device, as this will allow the reset functionality to work nominally during future use.