ME102B Group 13 Final Report: Wack O Mole

Ryan Chiang[a], Santiago Hopkins[a], Anders Nilsen[a], Jonathan Rendon[a]

[a]*Department of Engineering, University of California at Berkeley*

December, 2024

Opportunity

61% of Americans are involved in some sort of game for at least 1 hour a week. It goes without saying that a fun pastime is indispensable for a well-rounded life. A Wack O Mole presents itself as a perfect demonstration of entertainment and challenge to the player. Added with a twist of a rotating plate, the game presents a challenging aspect to further test the player's reflexes and skill.

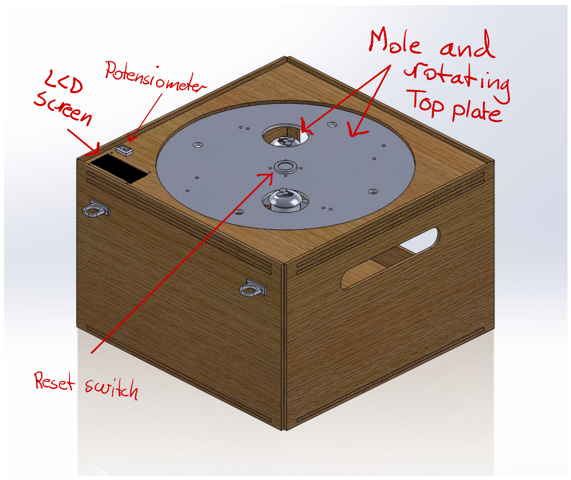
1. High Level Strategy

The Wack O Mole is equipped with two moles on a rotating plate. The plate reverses directions at random intervals to increase the complexity of the game-play. One of the moles is randomly selected at random intervals of time, triggering the mole to rise. The mole needs to be hit on a time limit that is set by the potentiometer; otherwise, one life is taken away from the starting three lives. Once the lives are depleted, the game stops, the LCD screen displays the final score, and the restart button must be pressed to start the game over.

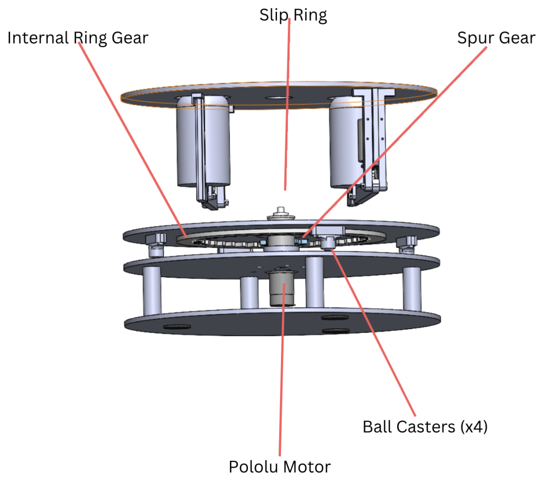
The Wack O Mole features moles that linearly actuate by independent solenoid linkage mechanisms. Due to budget constraints, two moles are installed: one mole would make it easy for the player to track. Increasing the number of moles will increase the cost of developing the design without making significant changes in game-play. The solenoid is active at random intervals of 2 to 7 seconds from a timer interrupt, triggering the mole to rise. The solenoid is deactivated when the limit switch installed inside of the mole heads is activated.

A potentiometer dial is installed on the top interface of the enclosure and functions to adjust the length of time that the solenoid is active. An LCD screen is also installed on the top of the enclosure and serves to display the current score and number of lives during game-play. It will also show the final score once the game is over. Lastly, a brushed DC motor is attached to a spur gear that drives a circular plate from an internal spur gear. The gear ratio is 4:1. An encoder is attached to the brushed DC motor to allow for closed-loop PID adjustments for the speed.

2. Assembly



((a)) Isometric View



((b)) Side View

Figure 1: Assembly Views

3. Function Critical Decisions

3.1. Identifying Maximum Mole Weight from Solenoid

The mass of the mole needed to be considered in the design of the solenoid link. The selected solenoid produces more than $60N$ of force and will require the mole to be under a calculated mass for smooth actuation. $60N$ is loaded $0.025m$ away from the pivot point, and an unknown force $0.2m$ away from the pivot point must be calculated using static mechanics to identify the maximum mole weight. The mass can be identified by utilizing the sum of moments equation:

$$\sum M = T_s + T_m$$

where T_s represents the torque of the solenoid and T_m represents the torque of the mole.

$$0 = F_s r_s + m_m g r_m$$

$$m_m = \frac{F_s r_s}{r_m} \frac{1}{g}$$

Plugging in $F_s = 60N$, $r_s = 0.025m$, $g = 9.8m/s^2$, and $r_m = 0.2m$ yields

$$m_m = 0.765kg = 765g$$

Therefore, an estimated maximum of $765g$ is theoretically allowed on the mole, not including friction.

3.2. DC Motor Selection

$$T_{\text{motor}} = \frac{T_{\text{load}}}{\text{gear ratio}} = \frac{9.62}{3.2} = 3.00 \text{ N}\cdot\text{m} \tag{1}$$

We're going with the *Pololu 100:1 Metal Gearmotor 37Dx57L mm 12V (Helical Pinion)*. This gives us just over 3 Nm of torque.

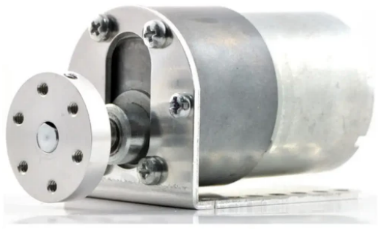


Figure 2: Pololu 100:1 Metal Gearmotor 37Dx57L mm 12V (Helical Pinion)

4. Diagrams

4.1. Circuit Diagram and State Diagram

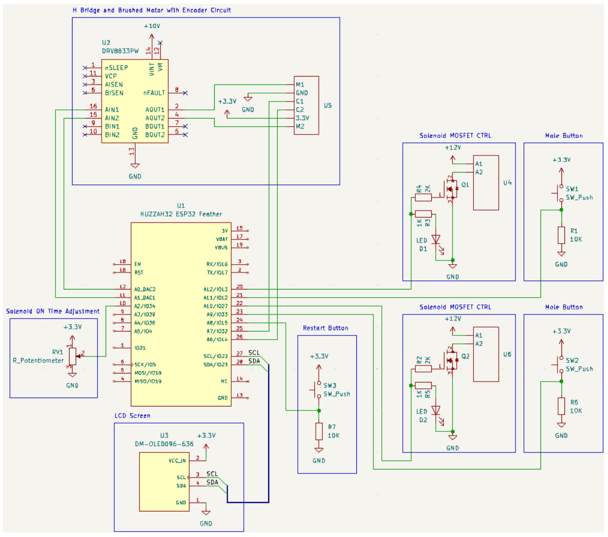


Figure 3: Wack O Mole Electrical Schematic

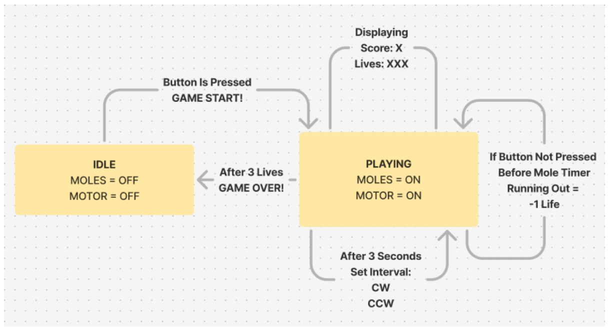
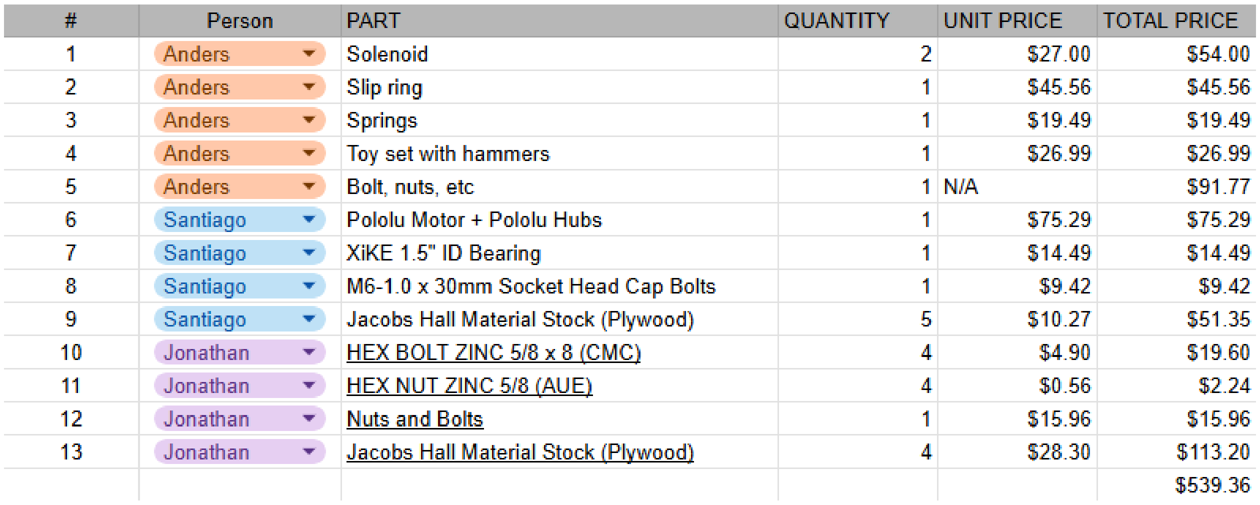


Figure 4: Wack O Mole State Diagram

5. Reflection

The Wack O Mole presented itself with many design routes that could be taken. Prototyping in the ideation phase was essential in balancing both the function and budget of the design. For example, the rotating plate presented itself with a few challenges in budget and fabrication. What worked best for our group was to separate and deliver several potential designs, reconvening and weighing each design based on function and cost, then iterating on a selected design. Once we had selected a design, a challenge had presented itself: the gears that we had designed would be too big to utilize forms of additive manufacturing such as FDM or SLA. It was ultimately decided to laser cut acrylic plates to create the gears. Our solenoid mechanism realized that the original mole was too heavy to be actuated. We had to reduce the weight of the mole to allow actuation to be feasible. A future note would be to first identify the actuation force of the solenoid and design around quantitative values before designing a mechanism. The plywood enclosure held the entire mechanism well and was able to withstand any movement and impact from the players.

Appendix A. Bill of Materials



#	Person	PART	QUANTITY	UNIT PRICE	TOTAL PRICE
1	Anders	Solenoid	2	$27.00	$54.00
2	Anders	Slip ring	1	$45.56	$45.56
3	Anders	Springs	1	$19.49	$19.49
4	Anders	Toy set with hammers	1	$26.99	$26.99
5	Anders	Bolt, nuts, etc	1	N/A	$91.77
6	Santiago	Pololu Motor + Pololu Hubs	1	$75.29	$75.29
7	Santiago	XiKE 1.5" ID Bearing	1	$14.49	$14.49
8	Santiago	M6-1.0 x 30mm Socket Head Cap Bolts	1	$9.42	$9.42
9	Santiago	Jacobs Hall Material Stock (Plywood)	5	$10.27	$51.35
10	Jonathan	HEX BOLT ZINC 5/8 x 8 (CMC)	4	$4.90	$19.60
11	Jonathan	HEX NUT ZINC 5/8 (AUE)	4	$0.56	$2.24
12	Jonathan	Nuts and Bolts	1	$15.96	$15.96
13	Jonathan	Jacobs Hall Material Stock (Plywood)	4	$28.30	$113.20
					$539.36

Figure A.5: Bill Of Materials

Appendix B. CAD Screenshots

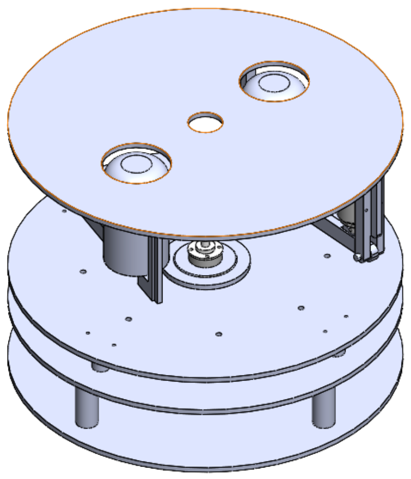


Figure B.6: Iso View of Assembly w/o Case

Appendix C. Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <ESP32Encoder.h>
#include <driver/gpio.h>

#define SOLENOID_2 13
#define BUTTON_2 12
#define SOLENOID_1 27
#define BUTTON_1 39

#define RESTART_BUTTON 15

#define BIN_1 26
#define BIN_2 25
#define ENCODER_PIN_A 14
#define ENCODER_PIN_B 32

#define POT 34

LiquidCrystal_I2C lcd(0x27, 20, 4);

// Motor
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;

// PID
float Kp = 15.0;
float Ki = 0.05;
float Kd = 0.5;

float integral = 0.0;
float previous_error = 0.0;
float derivative = 0.0;
float output = 0.0;

volatile float currentSpeed = 0.0;
volatile float desiredSpeed = 4000.0;

// Motor direction
volatile int motorDirection = random(0,2);
hw_timer_t* motorTimer = NULL;
portMUX_TYPE timerMux4 = portMUX_INITIALIZER_UNLOCKED;

// Solenoid
volatile bool button1Pressed = false;
volatile bool button2Pressed = false;
volatile unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 200;
volatile bool solenoidState = LOW;
volatile bool solenoid2State = LOW;
int score = 0;

volatile bool restartButtonPressed = false;

// Lives
volatile int lives = 3;
volatile bool livesChanged = false;
volatile int gameOver = false;
volatile int event = 0;

ESP32Encoder encoder;
volatile int64_t encoderCount = 0;
volatile bool deltaT = false;

hw_timer_t* encoderTimer = NULL;
hw_timer_t* solenoidTimer = NULL;
hw_timer_t* solenoidOffTimer = NULL;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux3 = portMUX_INITIALIZER_UNLOCKED;
```

```
int lastPotValue = -1;
unsigned long solenoidOffTime = 2000000;
volatile int chosenSolenoid = 1;

// 64-bit mapping: allows for larger digits from map() function
int64_t map64(int64_t x, int64_t in_min, int64_t in_max, int64_t out_min, int64_t out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

// ISR Declarations
void IRAM_ATTR ISR_button1_pressed();
void IRAM_ATTR ISR_button2_pressed();
void IRAM_ATTR ISR_restartbutton_pressed();
void IRAM_ATTR onMotorTimer();
void IRAM_ATTR onEncoderTimer();
void IRAM_ATTR onSolenoidTimer();
void IRAM_ATTR onSolenoidOffTimer();

void IRAM_ATTR onEncoderTimer() {
  portENTER_CRITICAL_ISR(&timerMux1);
  encoderCount = encoder.getCount();
  encoder.clearCount();
  deltaT = true;
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR onSolenoidTimer() {
  if (gameOver) {
    return;
  }

  int randomSolenoid = random(0, 2);
  portENTER_CRITICAL_ISR(&timerMux2);
  if (randomSolenoid == 0) {
    digitalWrite(SOLENOID_1, HIGH);
    solenoidState = HIGH;
    solenoid2State = LOW;
    chosenSolenoid = 1;
  } else {
    digitalWrite(SOLENOID_2, HIGH);
    solenoid2State = HIGH;
    solenoidState = LOW;
    chosenSolenoid = 2;
  }

  timerRestart(solenoidOffTimer);
  timerAlarmEnable(solenoidOffTimer);

  unsigned long newSolenoidInterval = random(1000000, 7000001);
  timerAlarmWrite(solenoidTimer, newSolenoidInterval, false);
  timerRestart(solenoidTimer);
  portEXIT_CRITICAL_ISR(&timerMux2);
}

void IRAM_ATTR onSolenoidOffTimer() {
  portENTER_CRITICAL_ISR(&timerMux3);
  if (chosenSolenoid == 1 && solenoidState == HIGH) {
    digitalWrite(SOLENOID_1, LOW);
    solenoidState = LOW;
    lives--;
    livesChanged = true;
    if (lives <= 0) {
      gameOver = true;
      event = 1;
    }
  } else if (chosenSolenoid == 2 && solenoid2State == HIGH) {
    digitalWrite(SOLENOID_2, LOW);
    solenoid2State = LOW;
    lives--;
    livesChanged = true;
    if (lives <= 0) {
      gameOver = true;
      event = 1;
    }
  }
  timerAlarmDisable(solenoidOffTimer);
  portEXIT_CRITICAL_ISR(&timerMux3);
}

void IRAM_ATTR onMotorTimer() {
  if (gameOver) {
    return;
  }

  portENTER_CRITICAL_ISR(&timerMux4);
  motorDirection = random(0, 2);
  desiredSpeed = 4000;
  portEXIT_CRITICAL_ISR(&timerMux4);
}

void setup() {
  Serial.begin(9600);

  uint32_t seed = esp_random();
  randomSeed(seed);

  pinMode(SOLENOID_1, OUTPUT);
  pinMode(SOLENOID_2, OUTPUT);
  pinMode(BUTTON_1, INPUT_PULLUP);
  pinMode(BUTTON_2, INPUT_PULLUP);
  pinMode(RESTART_BUTTON, INPUT_PULLUP);
  pinMode(POT, INPUT);

  attachInterrupt(digitalPinToInterrupt(BUTTON_1), ISR_button1_pressed, RISING);
  attachInterrupt(digitalPinToInterrupt(BUTTON_2), ISR_button2_pressed, RISING);
  attachInterrupt(digitalPinToInterrupt(RESTART_BUTTON), ISR_restartbutton_pressed, RISING);

  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0, 1);
  lcd.print("Score: 0");
  lcd.setCursor(0, 2);
  lcd.print("Lives: ");
  lcd.print(lives);

  ledcSetup(ledChannel_1, freq, resolution);
  ledcAttachPin(BIN_1, ledChannel_1);

  ledcSetup(ledChannel_2, freq, resolution);
  ledcAttachPin(BIN_2, ledChannel_2);

  encoder.attachHalfQuad(ENCODER_PIN_A, ENCODER_PIN_B);
  encoder.setCount(0);

  encoderTimer = timerBegin(0, 80, true);
  timerAttachInterrupt(encoderTimer, &onEncoderTimer, true);
  timerAlarmWrite(encoderTimer, 10000, true);
  timerAlarmEnable(encoderTimer);

  solenoidTimer = timerBegin(1, 80, true);
  timerAttachInterrupt(solenoidTimer, &onSolenoidTimer, true);
  unsigned long initialSolenoidInterval = random(1000000, 7000001);
  timerAlarmWrite(solenoidTimer, initialSolenoidInterval, false);
  timerAlarmEnable(solenoidTimer);

  solenoidOffTimer = timerBegin(2, 80, true);
  timerAttachInterrupt(solenoidOffTimer, &onSolenoidOffTimer, true);
  timerAlarmWrite(solenoidOffTimer, solenoidOffTime, false);

  motorTimer = timerBegin(3, 80, true);
  timerAttachInterrupt(motorTimer, &onMotorTimer, true);
  timerAlarmWrite(motorTimer, 1000000, true);
  timerAlarmEnable(motorTimer);
  ledcWrite(ledChannel_1, 0);
  ledcWrite(ledChannel_2, 0);
}

void loop() {
  int potValue = analogRead(POT);

  switch (event) {
    case 0:
      if (button1Pressed) {
        button1Pressed = false;
        if (chosenSolenoid == 1 && solenoidState == HIGH) {
          score++;
          Serial.print("Score: ");
          Serial.println(score);

          lcd.setCursor(7, 1);
          lcd.print(score);
          lcd.print("   ");

          digitalWrite(SOLENOID_1, LOW);
          solenoidState = LOW;
          timerAlarmDisable(solenoidOffTimer);
        }
      }

      if (button2Pressed) {
        button2Pressed = false;
        if (chosenSolenoid == 2 && solenoid2State == HIGH) {
          score++;
          Serial.print("Score: ");
          Serial.println(score);

          lcd.setCursor(7, 1);
          lcd.print(score);
          lcd.print("   ");

          digitalWrite(SOLENOID_2, LOW);
          solenoid2State = LOW;
          timerAlarmDisable(solenoidOffTimer);
        }
      }

      if (livesChanged) {
        livesChanged = false;
        lcd.setCursor(7, 2);
        lcd.print(lives);
        lcd.print("   ");
      }

      // Drive Motor: PID Control
      if (deltaT) {
        portENTER_CRITICAL(&timerMux1);
        deltaT = false;
        int64_t count = encoderCount;
        portEXIT_CRITICAL(&timerMux1);

        currentSpeed = (float)count / 0.010;
        float error = abs(desiredSpeed) - abs(currentSpeed);
        integral += error * 0.010;
        derivative = (error - previous_error) / 0.010;
        previous_error = error;

        output = Kp * error + Ki * integral + Kd * derivative;
        output = constrain(output, 0, 255);

        if (motorDirection == 0) {
          ledcWrite(ledChannel_1, (int)output);
          ledcWrite(ledChannel_2, 0);
        } else {
          ledcWrite(ledChannel_1, 0);
          ledcWrite(ledChannel_2, (int)output);
        }
      }

      // POT to Solenoid ON Time Adjustment
      if (abs(potValue - lastPotValue) > 100) {
        lastPotValue = potValue;
        unsigned long newSolenoidOffTime = (unsigned long)map64((int64_t)potValue, 0, 4095, 150000, 2000000);
        solenoidOffTime = newSolenoidOffTime;

        portENTER_CRITICAL(&timerMux3);
        timerAlarmWrite(solenoidOffTimer, solenoidOffTime, false);
        portEXIT_CRITICAL(&timerMux3);

        Serial.print("Potentiometer Value: ");
        Serial.print(potValue);
        Serial.print(", New Solenoid Off Time: ");
        Serial.print(solenoidOffTime / 1000000.0);
        Serial.println("s");
      }
      break;

    case 1:
      timerAlarmDisable(solenoidTimer);
      timerAlarmDisable(solenoidOffTimer);
      timerAlarmDisable(encoderTimer);
      timerAlarmDisable(motorTimer);

      ledcWrite(ledChannel_1, 0);
      ledcWrite(ledChannel_2, 0);

      digitalWrite(SOLENOID_1, LOW);
      digitalWrite(SOLENOID_2, LOW);

      Serial.print("Game Over! Final Score: ");
      Serial.println(score);

      lcd.clear();
      lcd.setCursor(0, 1);
      lcd.print("Game Over!");
      lcd.setCursor(0, 2);
      lcd.print("Final Score: ");
      lcd.print(score);
      event = 2;
      break;

    case 2:
      if (!gameOver) {
        event = 0;
        lives = 3;
        score = 0;

        lcd.clear();
        lcd.setCursor(0, 1);
        lcd.print("Score: 0");
        lcd.setCursor(0, 2);
        lcd.print("Lives: ");
        lcd.print(lives);

        delay(3000);

        timerAlarmEnable(solenoidTimer);
        timerAlarmEnable(solenoidOffTimer);
        timerAlarmEnable(encoderTimer);
        timerAlarmEnable(motorTimer);

      }
      break;

    default:
      break;
  }

  delay(10);
}
```

```
void IRAM_ATTR ISR_button1_pressed() {
  if (gameOver) {
    return;
  }
  unsigned long currentTime = millis();
  if ((currentTime - lastDebounceTime) > debounceDelay) {
    lastDebounceTime = currentTime;
    button1Pressed = true;
  }
}

void IRAM_ATTR ISR_button2_pressed() {
  if (gameOver) {
    return;
  }
  unsigned long currentTime = millis();
  if ((currentTime - lastDebounceTime) > debounceDelay) {
    lastDebounceTime = currentTime;
    button2Pressed = true;
  }
}

void IRAM_ATTR ISR_restartbutton_pressed() {
  if (!gameOver) {
    return;
  }
  unsigned long currentTime = millis();
  if ((currentTime - lastDebounceTime) > debounceDelay) {
    lastDebounceTime = currentTime;
    gameOver = false;
  }
}
```