# Jerry the New York Rat

Thor Oase, Ethan Pavlet, Greg Salazar, Conor Zachar

## Opportunity

Commercially available cat toys fail to stimulate cats without the presence of a human; they are often non-moving or rely on human interference. Cats are natural hunters and prefer to chase live rats, which is an impossibility for cats confined to the indoors. Our project seeks to create a robotic cat toy resembling a rat, which can move randomly throughout a room, change direction once it hits a wall, and even seek a home resting location.

## High-Level Strategy

Cats love to hunt, so our objective was to give them a toy that can simulate the thrill of the chase while staying safe inside. We decided to make an automated ‘rat’ for our cat to chase since it will provide exercise while not requiring attention from the owner. We succeeded in giving our rat bump sensors to allow it to run into and subsequently navigate away from walls. We overestimated the amount of speed that we’d be able to generate with these small motors, but ultimately this proved to be a good thing as our initial goal of two meters/second was much faster than is safe and realistic. Our final product achieved a speed 0.7 meters/second. Additionally, we added the ability for the rat to return to a beacon with infrared sensors, which was one of our stretch goals for this project.

## Integrated Physical Device

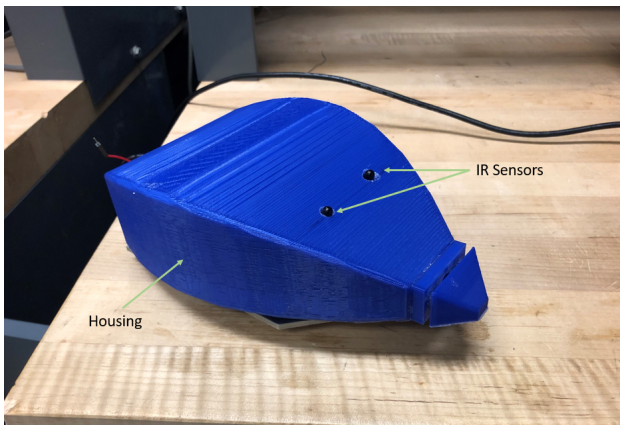



Figure 1: External view

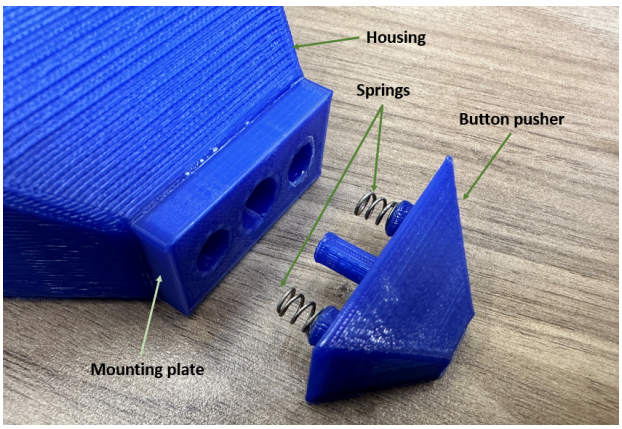



Figure 2: Button pusher subassembly

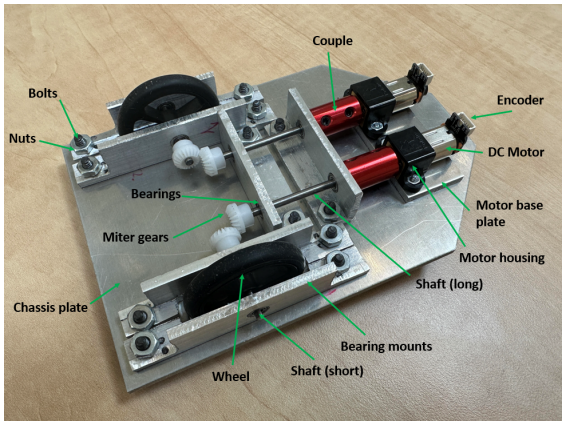



Figure 3: Transmission subsystem

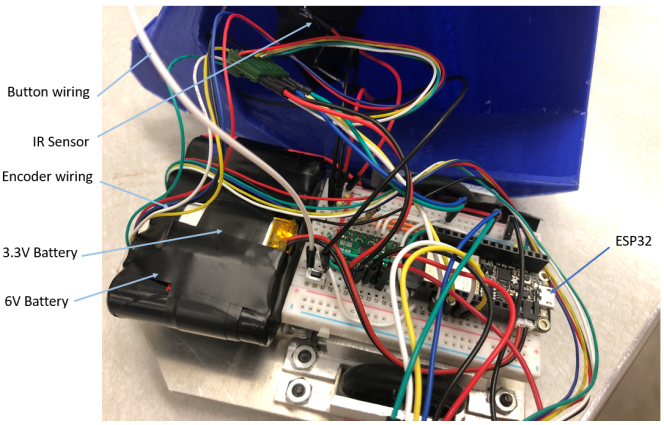



Figure 4: Electrical subsystem

# Function Critical Decisions & Calculations

Calculations for required motor torque:

$$\tau = \frac{P}{2\pi n}$$

$$P_{max} = IV = 6V * 0.34A = 2.04W$$

$$n_{max} = 340\ RPM = 340/60\ RPs = 5.6\ RPs$$

$$\tau_{max} = \frac{2.04}{2\pi * 5.6} Nm = 0.058\ N\,m = 0.59\ Kg\ cm$$

$$Stall\ torque,\ via\ spec\ sheet = 1.3\ Kg\ cm$$

$$Factor\ of\ Safety = \frac{1.3\ Kg\ cm}{0.59\ Kg\ cm} = 2.2$$

Therefore, the maximum torque required by our motor is within the motor specification.

Calculations for the springs to activate digital sensor (press button):

$$k = \frac{Gd^4}{8D^3N} = \frac{7*10^9\ Pa * (0.5*10^{-3}m)^4}{8*(2*10^{-3}m)^3*5} = 15234\ N/m$$

$$k_{eff} = 2 * k = 30468\ N/m$$

A light collision is estimated at 50N

$$\Delta x = \frac{F}{k_{eff}} = \frac{50N}{30468N/m} = 1.64mm$$

The button pusher will be compressed by 1.64mm, which is within our mechanical design guidelines to activate the sensor.

Calculations for the forces on our bearings:

$$W_{total} = 0.544kg * 9.81m/s^2 = 5.33N$$

$$W_{per\ wheel} = \frac{5.33N}{3} = 1.78N$$

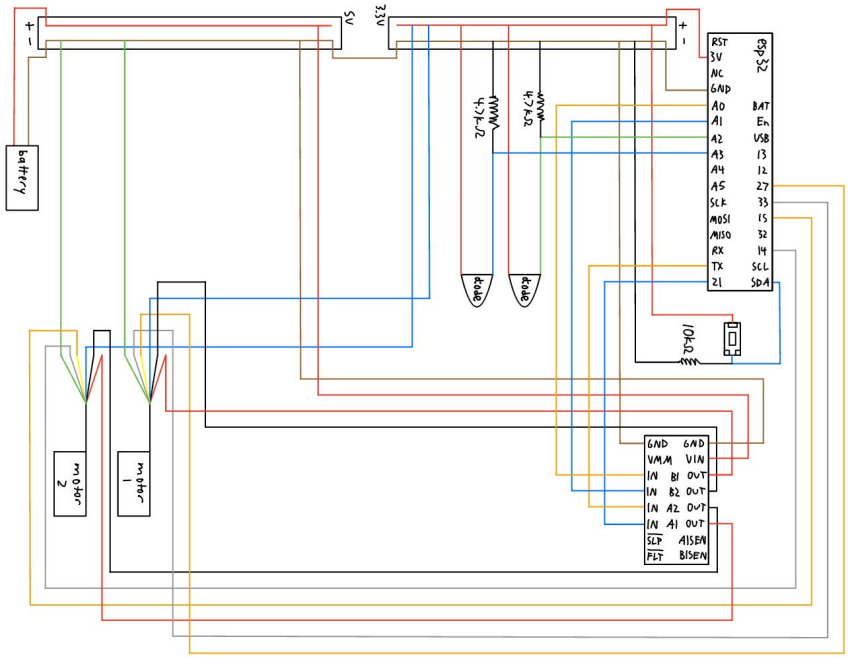
$$F_{radial} = \frac{1.78N}{2\ bearings/wheel} = 0.89N$$

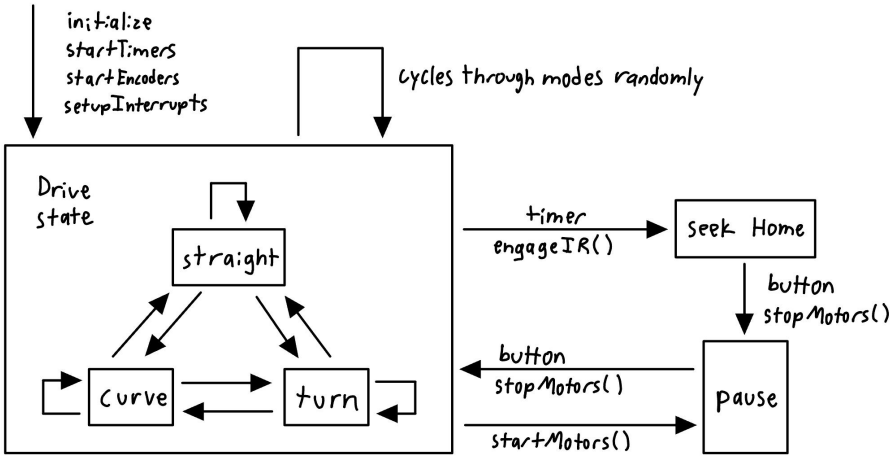
$$Radial\ load\ rating,\ via\ spec\ sheet = 35N$$

$$Factor\ of\ Safety = \frac{35N}{0.89N} = 39$$

Therefore, our radial load is well within the specification of the bearings.

# Circuit Diagram




# Reflection

What worked well:

Our group had effective communication, which was enabled by establishing weekly team meetings. We assigned a mechanical subteam as well as a coding lead, which enabled us to effectively divide work and have multiple workflows simultaneously.

What could be improved:

We conducted our design step-by-step, and ordered parts on an as-needed basis, whenever they needed to be assembled. However, this led to us needing to re-order our wheels in a different size and design a new shape for our bearing mounts to fit them. We recommend making a comprehensive electrical/mechanical design and purchasing everything early on, to ensure parts fit as intended without delays. Furthermore, feel like we could have delved deeper into the mechanical complexity of our rat and tried to add more features to keep it from getting caught on walls, especially if it goes backwards, and attaining a more sleek and robust design.

# Appendix

## BOM

| Mechanical | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Item** | **Component name** | **Key specs** | **Manufacturer** | **Manufacturer Part #** | **Raw materials required** | **Machinery required** | **Link** | **Weight per, g** | **Quantity** | **Cost per** | notes | received |
| 1 | Chassis | Rectangular, 6” long, 4” wide | XinDaQi | | Aluminum 6061 | Waterjet | Link | | 1 | $ 15.99 | | y |
| 2 | Back wheels | 40mm D, 7mm W, 3mm shaft | Pololu | 8946T61 | N/A | N/A | Link | 3.1 | 2 | $ 3.95 | pololu | y |
| 3 | Motor | 1kg*cm Torque, 6V | Digikey | 4641 | N/A | N/A | Link | 9.5 | 2 | $ 12.50 | supplied | y |
| 4 | Shaft | 3mm x 400mm | McMaster | 1327K504 | N/A | Mill | Link | | 1 | $ 12.19 | mcmaster | y |
| 5 | Couple | 3mm to 3mm | Uxcell | | N/A | Hex wrench | Link | 45 | 2 | $ 5.99 | amazon | y |
| 6 | Miter gears | 5/16” inner D | McMaster | 2810N1 | N/A | N/A | Link | 20 | 4 | $ 3.20 | mcmaster | y |
| 7 | Front Wheel | | Uxcell | | N/A | N/A | Link | 10 | 1 | $ 5.49 | amazon | y |
| 8 | Ball bearings | 3mm inner D, 7mm outer | Uxcell | 683ZZ | N/A | N/A | Link | 10 | 8 | $ 1.50 | amazon | y |
| 9 | Bearing Mounts | 1" x 1" L Brackets, 1/8" thick, 13" long | DOUYUDAU | | Aluminum 6061 | Bandsaw, Mill | Link | | 2 | $ 3.67 | amazon | y |
| **Electrical** | | | | | | | | | | | | |
| **Item** | **Component name** | **Key specs** | **Manufacturer** | **Manufacturer Part #** | **Raw materials required** | **Machinery required** | **Link** | **Weight per, g** | **Quantity** | **Cost per** | | |
| 10 | ESP32 | - | espressif | | - | | **Link** | 10 | 1 | $ - | supplied | y |
| 11 | Buttons | - | Adafruit | 1528-4431-ND | - | Solder | **Link** | 5 | 4 | $ 4.99 | amazon | y |
| 12 | Potentiometer | - | Lab Kit | | - | | **Link** | 1 | 1 | $ - | supplied | y |
| 13 | 6V Battery | 6V | Tenergy | B001BCOWLY | - | | **Link** | 140 | 1 | $ 12.00 | amazon | y |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | IR sensor | - | Gikfun | LYSB01HGIQ8NG-ELECTRNCS | | | Link | 9 | 2 | $ 3.07 | amazon | y |
| 15 | Breadboard | | | | | | Link | | | | | |

# CAD

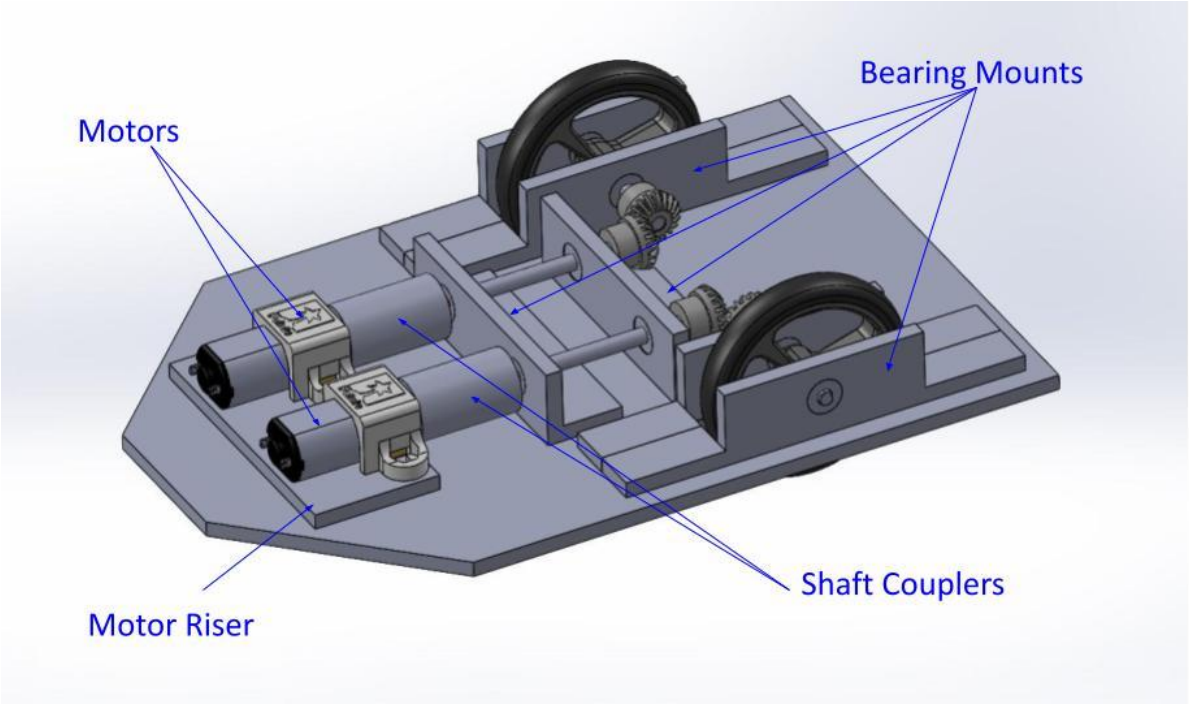



Figure 1: Isometric View

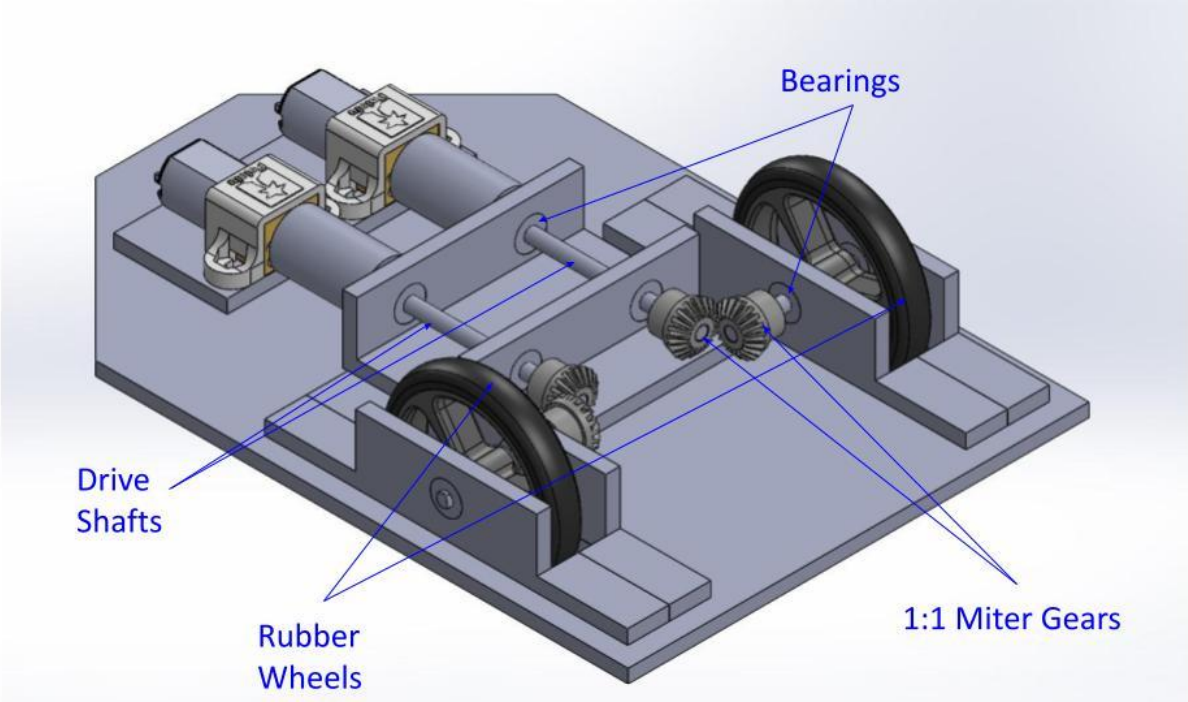



Figure 2: Shifted Isometric View

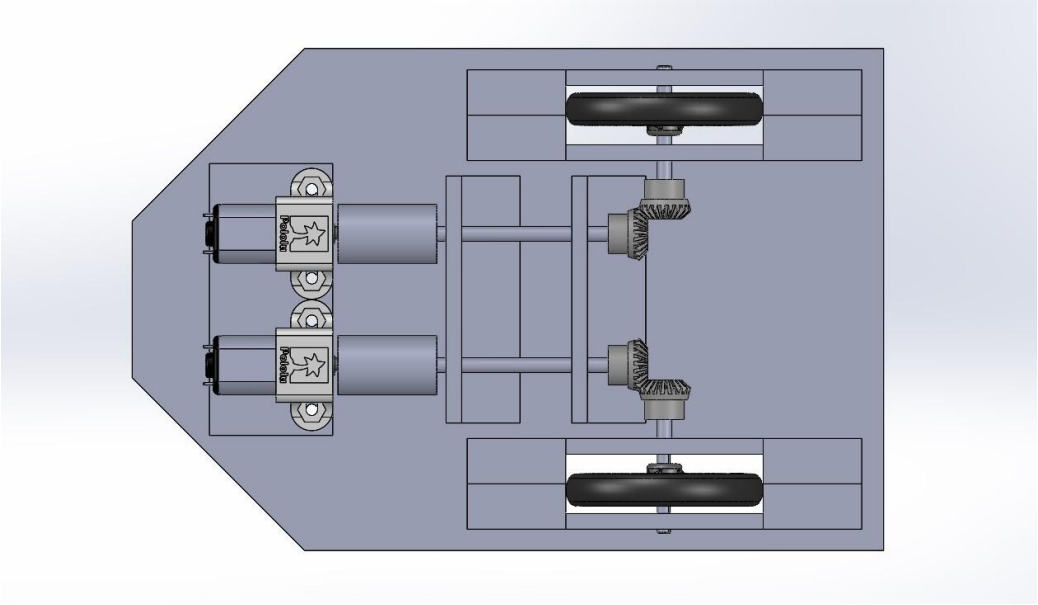


Figure 3: Top View

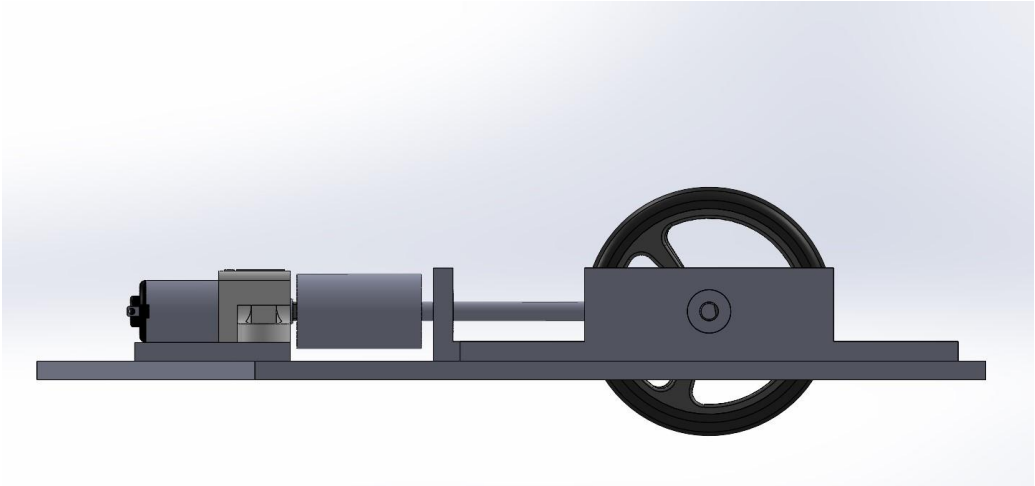


Figure 4: Side View

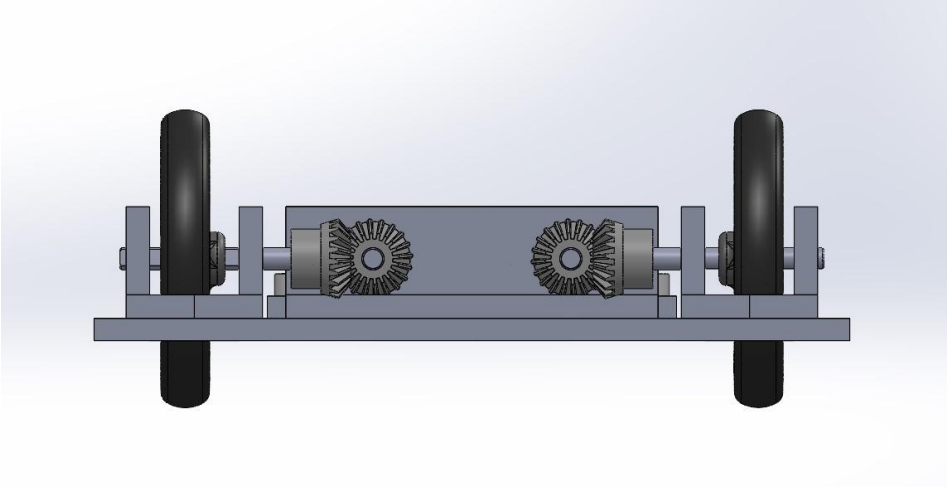


Figure 5: Back View

## Code Screenshots

jerry_code.ino

```
#include <ESP32Encoder.h>

#define BTN 23
#define LF_PIN 26
#define LR_PIN 25
#define RF_PIN 17
#define RR_PIN 21
#define ENCODERL1 27
#define ENCODERL2 33
#define ENCODERR1 14
#define ENCODERR2 15
#define MAX_PWM_VOLTAGE 200
#define END_STATE 10
#define AMBIENT 100

int state = 0;
int nextState;
int speed = 10;

//IR Sensor Setup
#define LEFT_IR_PIN 39
#define RIGHT_IR_PIN 34
int readingLeft;
int readingRight;
int difference;

//encoder and control variable setup
ESP32Encoder encoderL;
ESP32Encoder encoderR;
volatile int countR = 0;
volatile int countL = 0;
volatile int errorL = 0;
volatile int errorR = 0;
```

```
34  volatile int DR = 0;
35  volatile int DL = 0;
36  int Kp = 65;
37
38  //debounce and button setup
39  hw_timer_t * timerDebounce = NULL;
40  portMUX_TYPE timerMuxDebounce = portMUX_INITIALIZER_UNLOCKED;
41  volatile bool debounceT = true;
42  volatile bool buttonIsPressed = false;
43
44  //state swap timer setup
45  hw_timer_t * timer0 = NULL;
46  portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
47  volatile bool swap = false;
48
49  //motor control timer and variable setup
50  hw_timer_t * timerMotorControl = NULL;
51  portMUX_TYPE timerMuxMotorControl = portMUX_INITIALIZER_UNLOCKED;
52  volatile bool deltaT = false;
53
54  //playtime timer and variable setup
55  hw_timer_t * timerPlaytime = NULL;
56  portMUX_TYPE timerMuxPlaytime = portMUX_INITIALIZER_UNLOCKED;
57  // volatile bool playtimesOver = false;
58  // ^^ perhaps necessary if state swap doesn't work
59
60  //PWM setup variables
61  const int LF_channel = 1;
62  const int LR_channel = 2;
63  const int RF_channel = 3;
64  const int RR_channel = 4;
65  const int freq = 5000;
```

```
66  const int resolution = 8;
67
68  //interrupt function setup
69  void IRAM_ATTR debouncer(){
70    portENTER_CRITICAL_ISR(&timerMuxDebounce);
71    debounceT = true;
72    portEXIT_CRITICAL_ISR(&timerMuxDebounce);
73  }
74
75  void IRAM_ATTR switchState(){
76    portENTER_CRITICAL_ISR(&timerMux0);
77    swap = true;
78    portEXIT_CRITICAL_ISR(&timerMux0);
79  }
80
81  void IRAM_ATTR updateError(){
82    portENTER_CRITICAL_ISR(&timerMuxMotorControl);
83    countR = encoderR.getCount();
84    countL = encoderL.getCount();
85    encoderR.clearCount();
86    encoderL.clearCount();
87    deltaT = true;
88    portEXIT_CRITICAL_ISR(&timerMuxMotorControl);
89  }
90
91  void IRAM_ATTR endGame(){
92      portENTER_CRITICAL_ISR(&timerMuxPlaytime);
93      state = END_STATE;
94      portEXIT_CRITICAL_ISR(&timerMuxPlaytime);
95  }
96
97  void IRAM_ATTR buttonResponse() {
```

```
  if (debounceT) {
    debounceT = false;
    buttonIsPressed = true;
    timerStop(timerDebounce);
    timerWrite(timerDebounce, 0);
    timerStart(timerDebounce);
  }
}

bool checkForButtonPress(){
  if (buttonIsPressed){
    buttonIsPressed = false;
    return true;
  } else {
    return false;
  }
}

//initialization functions for cleanliness

void initializeEncoders() {

  encoderL.attachHalfQuad(ENCODERL1, ENCODERL2);
  encoderL.setCount(0);
  encoderR.attachHalfQuad(ENCODERR1, ENCODERR2);
  encoderR.setCount(0);
}

void initializeTimers(){
  //button debouncer
  timerDebounce = timerBegin(0,80,true);
  timerAttachInterrupt(timerDebounce, &debouncer, true);
```

```
  timerAlarmWrite(timerDebounce, 500000, true);

  //switch state every 0.5 seconds (assuming no button press)
  timer0 = timerBegin(1,80,true);
  timerAttachInterrupt(timer0, &switchState, true);
  timerAlarmWrite(timer0, 1000000, true);

  //encoder timer for motor control
  timerMotorControl = timerBegin(2,80,true);
  timerAttachInterrupt(timerMotorControl, &updateError, true);
  timerAlarmWrite(timerMotorControl, 10000, true);

  //timer to send jerry home
  timerPlaytime = timerBegin(3,80, true);
  timerAttachInterrupt(timerPlaytime, &endGame, true);
  timerAlarmWrite(timerPlaytime, 100000000, true); //10 000 000 => ten seconds

  timerAlarmEnable(timer0);
  timerAlarmEnable(timerDebounce);
  timerAlarmEnable(timerMotorControl);
  timerAlarmEnable(timerPlaytime);
}

void setup(){
  ESP32Encoder::useInternalWeakPullResistors = UP;

  pinMode(BTN, INPUT);
  pinMode(LEFT_IR_PIN, INPUT);
  pinMode(RIGHT_IR_PIN, INPUT);

  //pwm channels for motor control
  ledcSetup(LF_channel, freq, resolution);
```

```
  ledcSetup(LR_channel, freq, resolution);
  ledcSetup(RF_channel, freq, resolution);
  ledcSetup(RR_channel, freq, resolution);

  ledcAttachPin(LF_PIN, LF_channel);
  ledcAttachPin(LR_PIN, LR_channel);
  ledcAttachPin(RF_PIN, RF_channel);
  ledcAttachPin(RR_PIN, RR_channel);

  initializeTimers();

  initializeEncoders();
  randomSeed(analogRead(0));

  attachInterrupt(BTN, buttonResponse, RISING);
  Serial.begin(115200);
  //make sure no setup loop exists, great for debugging and otherwise inconsequential
  Serial.println(42);
}


void loop() {
  //swap state on timer, unless in "off" state (state 100)
  nextState = random(0,9);
  Serial.println(state);
  // speed = random(6,12);

  if (swap && state!=100 && state!=10) {
    swap = false;
    timerWrite(timer0,0);
    timerStart(timer0);
    state = nextState;
```

```
194   }
195   //toggle between on and off, always resetting to state 0
196   if (checkForButtonPress()){
197     if (state!=100){
198       state=100;
199       timerStop(timer0);
200       timerStop(timerPlaytime);
201     } else {
202       state=0;
203       timerWrite(timer0, 0);
204       timerStart(timer0);
205       timerWrite(timerPlaytime, 0);
206       timerStart(timerPlaytime);
207     }
208   }
209
210   switch (state) {
211     case 0: //moving straight
212       drive(-speed,-speed);
213       break;
214     case 1: //turning left tank
215       drive(-speed,speed);
216       break;
217     case 2: //turning right tank
218       drive(speed,-speed);
219       break;
220     case 3: //turning left curve
221       drive(-speed,-speed/2);
222       break;
223     case 4: //turning right curve
224       drive(-speed/2,-speed);
225       break;
226     case 5: //moving straight
```

```
      drive(-speed,-speed);
      break;
    case 6: //moving straight (faster)
      drive(-speed*3/2,-speed*3/2);
      break;
    case 7: //moving backwards
      drive(speed,speed);
      break;
    case 8: //back left curve
      drive(speed/2,speed);
      break;
    case 9: //back right curve
      drive(speed,speed/2);
      break;
    case END_STATE:
      readingLeft = analogRead(LEFT_IR_PIN);
      readingRight = analogRead(RIGHT_IR_PIN);
      difference = readingLeft-readingRight;

      if (difference > 400) {
        //if left stronger, turn left
        drive(-10,10);
        //Serial.println(-1);
      } else if (difference < -700) {
        //if right much stronger, turn right
        drive(10,-10);
        //Serial.println(1);
      } else if (readingLeft < 700) {
        //if no significant readings, keep rotating
        drive(-10,10);
        //Serial.println(-2);
      } else {
        //else, go straight
```

```
260             drive(-10, -10);
261             //Serial.println(0);
262           }
263           break;
264         default:
265           drive(0,0);
266           break;
267       }
268   }
269
270   //motor control function
271   void drive(int L_command,int R_command){
272     if (deltaT) {
273       portENTER_CRITICAL(&timerMuxMotorControl);
274       deltaT = false;
275       portEXIT_CRITICAL(&timerMuxMotorControl);
276       errorL = L_command - countL;
277       errorR = R_command - countR;
278     }
279
280     DR = Kp*errorR;
281     DL = Kp*errorL;
282
283     if (DR > MAX_PWM_VOLTAGE) {
284       DR = MAX_PWM_VOLTAGE;
285     } else if (DR < -MAX_PWM_VOLTAGE) {
286       DR = -MAX_PWM_VOLTAGE;
287     }
288     if (DL > MAX_PWM_VOLTAGE) {
289       DL = MAX_PWM_VOLTAGE;
290     } else if (DL < -MAX_PWM_VOLTAGE) {
291       DL = -MAX_PWM_VOLTAGE;
292     }
```

```
293
294   if (DR > 0) {
295     ledcWrite(RR_channel, LOW);
296     ledcWrite(RF_channel, DR);
297     }
298   else if (DR < 0) {
299     ledcWrite(RR_channel, -DR);
300     ledcWrite(RF_channel, LOW);
301     }
302   else {
303     ledcWrite(RR_channel, LOW);
304     ledcWrite(RF_channel, LOW);
305   }
306
307   if (DL > 0) {
308     ledcWrite(LR_channel, LOW);
309     ledcWrite(LF_channel, DL);
310   }
311   else if (DL < 0) {
312     ledcWrite(LR_channel, -DL);
313     ledcWrite(LF_channel, LOW);
314   }
315   else {
316     ledcWrite(LR_channel, LOW);
317     ledcWrite(LF_channel, LOW);
318   }
319 }
```