# Personal Storage Bot (Bot-ler)

Abdullah Alhamood, Fahd Althukair, Zachary Douglas

## Opportunity:

An occupied hand is something that most people experience as they're buying groceries, carrying objects around the house, or holding their coffee mug on their way to work. Our team brainstormed on how we can fix this common issue to free up our hands during any mundane tasks. To realize this opportunity, we have chosen to build a personal butler robot that follows you around inside your house with a small storage area inside of it.

## High-Level Strategy:

A thermal camera and Infrared sensors will be integrated and used to follow the person around. The thermal camera's data will be processed in our microcontroller to determine whether the user is in the left, center, or right to guide the path of the robot. The IR sensors will measure the distance in front of the robot so that it can stop at a certain distance if an object/person is in front of the robot. The robot uses tri-wheel design, where two wheels are driven by DC motors and a third idling wheel is used for support. The robot also has a separate transmission system for the lid that can be opened/closed with a button. Our desired functionality for the robot was to integrate the camera and IR sensors with the wheels to produce smooth movements. However, due to limitations within the microcontroller and the sensors' noise, it was difficult to achieve that goal. Nonetheless, our robot was able to change its direction depending on where it detected the user. Moreover, its stopping feature was consistent in our final version as it was able to stop before hitting an object/human. We also wanted the robot to be able to carry larger objects, but due to constraints with 3D printing and wheel transmission system, we built a smaller device that can only fit smaller things such as a coffee mug, phones, and wallets. Our robot was able to achieve the speed that we aimed for (1.5 to 3 m/s) to successfully follow a person who is walking/jogging.

## System Diagrams:

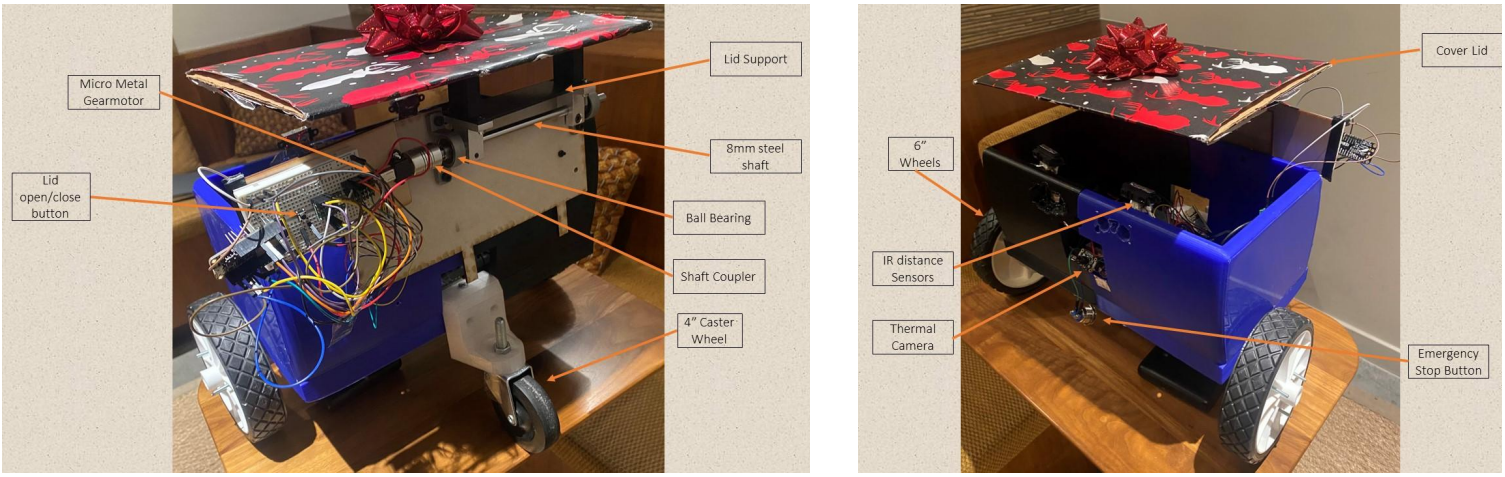



Figure 1: Lid Transmission System (Left) and Body Sensor Layout (Right)

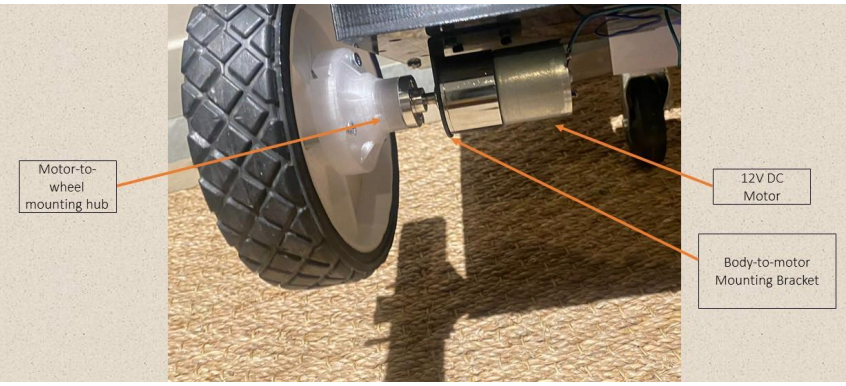



Figure 2: Wheel Transmission System

# Function-Critical Decisions:

**Wheels:-**

Since the motors are directly attached to the hub of the wheels, it is important to ensure that its shaft does not experience high stress that can break and ruin the motor. For safety, we will assume that the normal force that each wheel will experience is equal to the full weight of the body.

$$Stress = \frac{Weight}{Surface\ Area} = \frac{100}{\pi(0.003)^2 0.014} \approx 250\ MPa \quad \text{(Equation 1)}$$

To ensure that our motor shaft is not going to break in extreme cases, the Greartisan DC 12V motor will be used for the wheels as it possesses a hardened steel shaft which will endure the maximum stress the motor might experience. Additionally, the torque provided by the motor should be able to overcome friction experienced by the wheels in order to freely rotate the wheels and move the robot. With our tri-wheels configuration, the front wheels will approximately experience one fourth of the weight. Equation 2 is used to determine the friction torque from the ground. With our chosen motor able to provide 4.5 kg.cm, it will be able to produce enough torque.

$$Friction\ Torque = W \mu_s R = 25 \times 0.15 \times 0.076 = 0.285\ Nm = 2.9\ Kg.cm \quad \text{(Equation 2)}$$

**Lid:-**

Our Lid mechanism motor will need to support the weight of the lid and be able to rotate the rotary shaft to close and open the storage lid. The center of mass for the lid will approximately be half the distance of the lid width which is around 8 cm away from the back. This orientation will create the highest torque which our motor needs to overcome to be able to open the lid. The total weight of the lid system is around 0.5 N. The torque needed is computed using Equation 3 which can be created by our DC gearmotor found in our kit.

$$Lid\ Torque = WL = 0.5 \times 0.08 = 0.04\ Nm = 0.41\ Kg.cm \quad \text{(Equation 3)}$$

## Circuit Diagram:

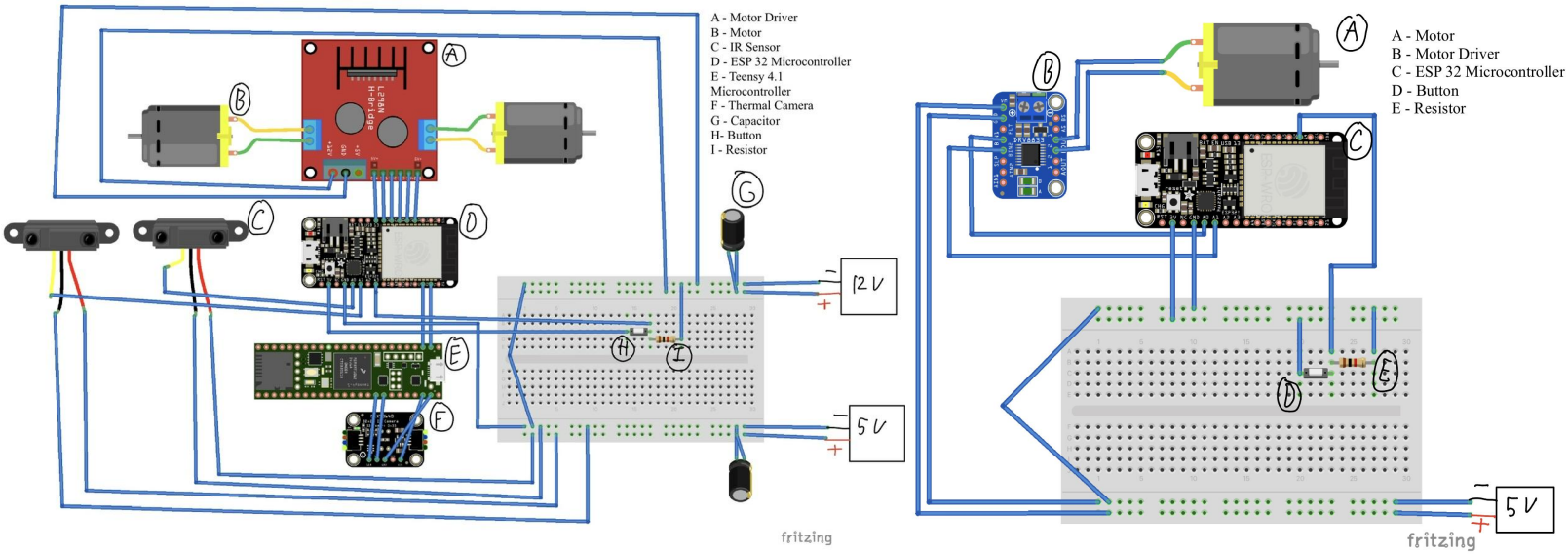



Figure 3: Robot Circuit (Left) and Lid Opening\Closing Circuit (Right)

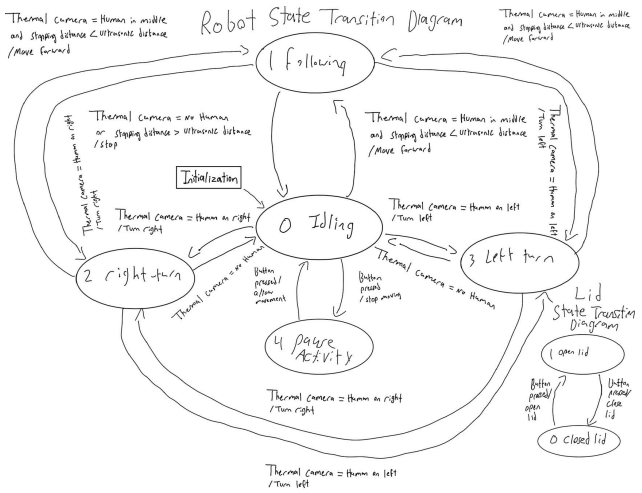



Figure 4: State Diagram of the robot

## Reflection:

Having a shared group chat and using google drive and documents to work together helped in easing teamwork. However, in the beginning we were doing duplicate research into the electronics of the system and were able to finish the electronics before the mechanical systems. We think that finishing the mechanical system first is important because code debugging takes more time, and you want a platform to test the electronics after they finish to tune in your code and make it work in time.

# Appendix

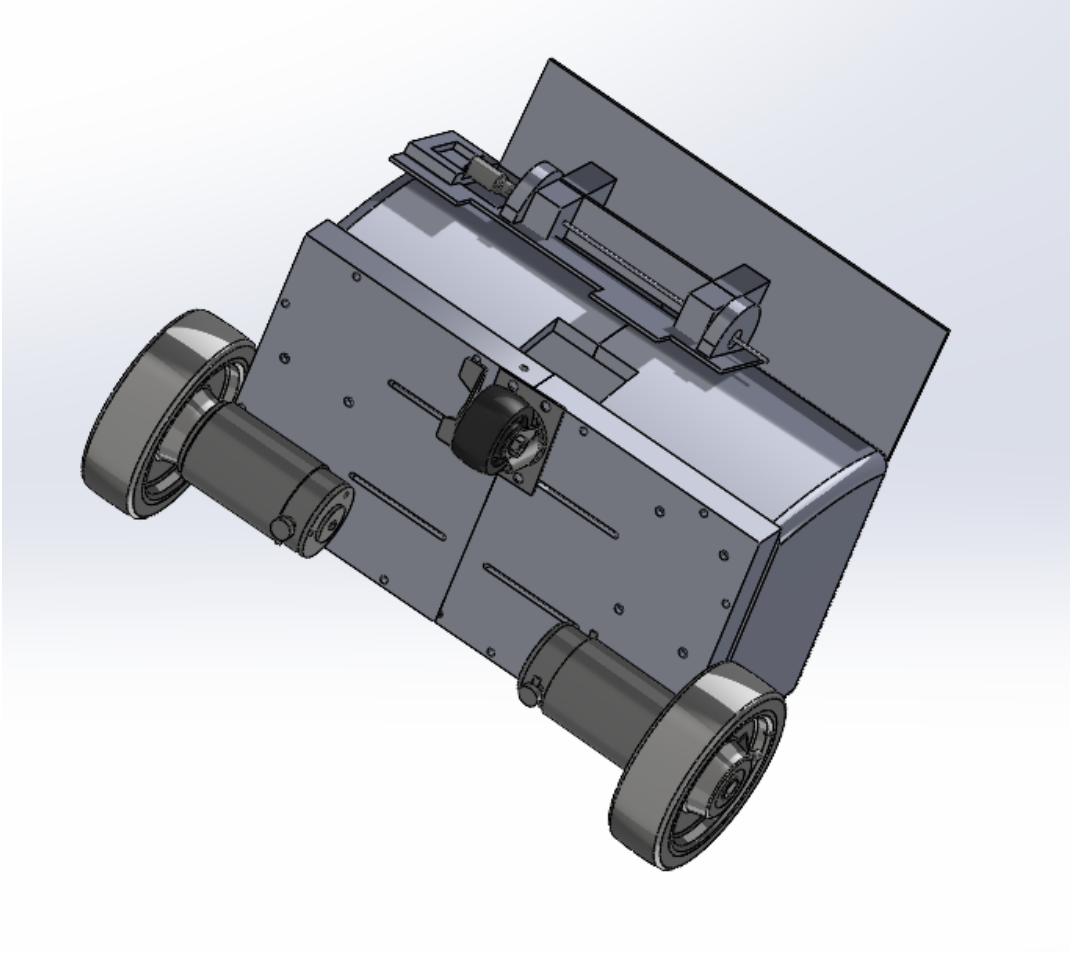
## Bill of Materials:

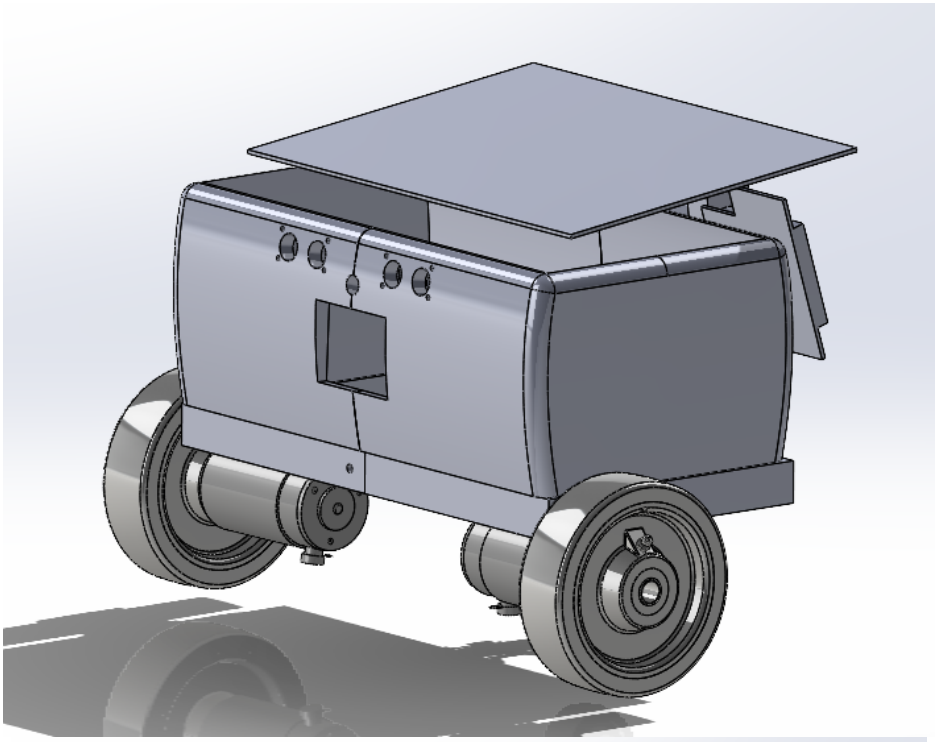
| Item | Notes | Quantity | Cost Each | Total Cost |
|---|---|---|---|---|
| ESP32 Microcontroller | Provided by ME102B MicroKit | 2 | 0 | 0 |
| L298N Motor Driver | https://a.co/d/9hiTTyh | 1 | 2.88 | 2.88 |
| Teensy 4.1 Microcontroller | https://a.co/d/1atm4gm | 1 | 45.99 | 45.99 |
| Thermal Camera MLX90640 | https://www.adafruit.com/product/4407 | 1 | 74.95 | 74.95 |
| Infrared Proximity Distance Sensor | https://a.co/d/4xxQ3Vc | 2 | 8.33 | 16.66 |
| USB to Terminal Block Connector | https://a.co/d/12wT8pC | 2 | 3.5 | 7 |
| 12V/5V Power Bank | https://a.co/d/99d17l0 | 1 | 28.99 | 28.99 |
| Capacitor | Provided by ME102B MicroKit (We used 100uf/10V for the 5V terminal and 10 uf/25V for the 12V terminal) | 2 | 0 | 0 |
| Breadboard | Provided by ME102B MicroKit | 2 | 0 | 0 |
| Resistor | Provided by ME102B MicroKit (1000 Ohms) | 2 | 0 | 0 |
| DC Motor | Provided by ME102B lab kit | 1 | 0 | 0 |
| Greartisan DC 12V 100RPM Gear Motor | https://a.co/d/4ZWN3wE | 2 | 14.99 | 29.98 |

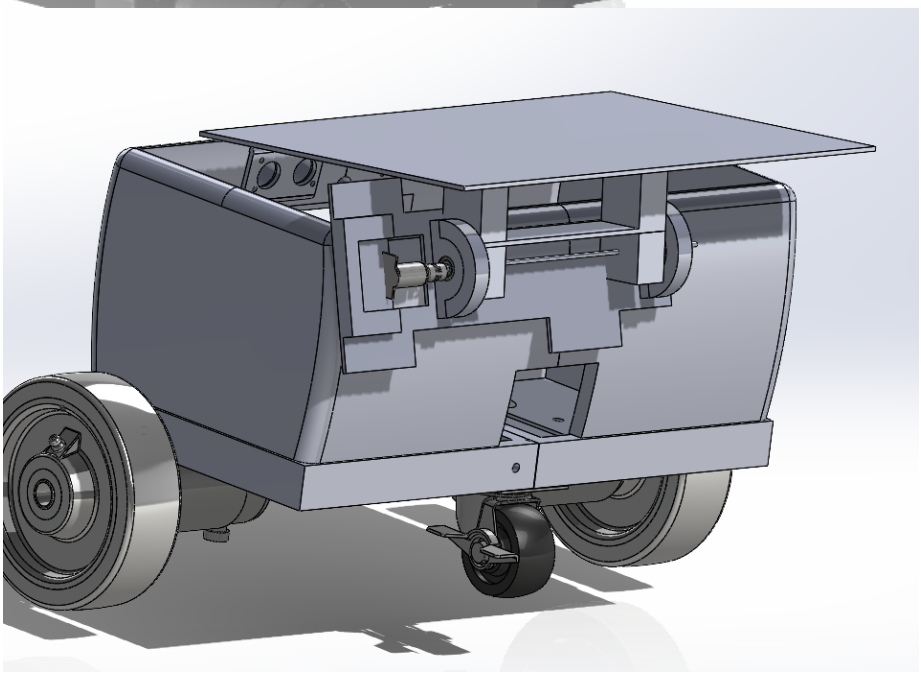
| | | | | |
|---|---|---|---|---|
| Gear Motors Mounting Bracket | https://a.co/d/bD43ouY | 2 | 8.99 | 17.98 |
| 3D Printed Body | | 1 | 14.84 | 14.84 |
| 3D Printed wheel Hub | | 2 | 2.67 | 5.34 |
| 3D Printed caster wheel attachment | | 1 | 3.08 | 3.08 |
| Caster Wheel | McMaster. #2834T25 | 1 | 9.00 | 9.00 |
| Lid System Bracket (laser cut) | | 1 | 13.40 | 13.40 |
| Jumper Cables | Provided by ME102B MicroKit | | 0 | 0 |
| M5x25mm Bolts and Nuts | Home Depot Model# 801378 | 12 | 1.15 | 13.8 |
| #8 3 in. Phillips Bugle-Head Self-Drilling Screws | Home Depot | 3 | 9.88 | 29.64 |
| #8 3 in. Phillips Bugle-Head Self-Drilling Screws | Home Depot | 1 | 1.25 | 1.25 |
| Steel Metal Push Button | https://a.co/d/bdw6k66 | 1 | 11.49 | 11.49 |
| 6” Wheel | https://www.acehardware.com/departments/lawn-and-garden/lawn-mowers/lawn- | 2 | 13.99 | 27.89 |

| | | | | |
|---|---|---|---|---|
| | mower-tires-and-wheels/71044 | | | |
| Lid Transmission System (Coupler, Shaft, Bearings, Linear Rail Support) | https://a.co/d/dgrGbQN | 1 | 26.99 | 26.99 |
| DRV8833 Dual Motor Driver | Provided by ME102B MicroKit | 1 | 0 | 0 |
| Push Button | Provided by ME102B MicroKit | 1 | 0 | 0 |
| Cost Sum | | | | $381.15 |

CAD Images:







## Software:

You can find the full code (Thermal camera algorithm and test codes for each electronic piece) at https://github.com/FahdAlthukair/ME102B-FinalProject. The first code screenshots are for the robot state transition, the second is for the lid:

FahdAlthukair First commit History

1 contributor

414 lines (371 sloc) | 10.8 KB

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

//Thermal camera attributes
HardwareSerial MySerial(1);
uint8_t arrayCenter = 10;

//Initialize IR
#define IR_PIN_LEFT 26
#define IR_PIN_RIGHT 25


//Initialize Motors
#define ENA 13
#define IN1 12
#define IN2 27
#define IN3 33
#define IN4 15
#define ENB 32

// setting PWM properties
const int MOTOR_PWM_VOLTAGE = 170;
const int TURN_PWM_VOLTAGE = 200;

int stop_distance = 1000;
const int STOP_DISTANCE_1 = 1000;
const int STOP_DISTANCE_2 = 900;


// current IR distance reading
float distance_right = 0;
float distance_left = 0;

// prev IR distance reading
float prev_distance_right = 0;
float prev_distance_left = 0;

// Motor timing
unsigned long motorCm;
unsigned long motorPm;
```

```
const unsigned long MOTOR_PERIOD = 100;
//
//// Thermal timing
//unsigned long thermalTimer;
//const unsigned long THERMAL_PERIOD = 20;

//Motor btn setup
#define BTN 39  // declare the button pin number
bool motorsOn = true;

volatile bool buttonIsPressed = false;
volatile bool timePassed = true;
hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
  portENTER_CRITICAL_ISR(&timerMux);
  buttonIsPressed = true;
  timePassed = false;
  portEXIT_CRITICAL_ISR(&timerMux);
  timerStart(timer);

}

void IRAM_ATTR timer_isr() {  // the function to be called when interrupt is triggered
  portENTER_CRITICAL(&timerMux);
  timePassed = true;
  portEXIT_CRITICAL(&timerMux);
  timerStop(timer);
  timerRestart(timer);
}

//Set currentState variable
int prevState = 0;
int currentState = 0;

void setup() {
  Serial.begin(115200);
  MySerial.begin(9600, SERIAL_8N1, 16, 17);

  pinMode(IR_PIN_LEFT, INPUT);
  pinMode(IR_PIN_RIGHT, INPUT);

  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENB, OUTPUT);

  pinMode(BTN, INPUT);
  attachInterrupt(BTN, isr, RISING);
  timer = timerBegin(0, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics /
  timerAttachInterrupt(timer, &timer_isr, true);    // sets which function do you want to call when the interrupt
  timerAlarmWrite(timer, 200000, true);         // sets how many tics will you count to trigger the interrupt
  timerAlarmEnable(timer); // Enables timer
```

```
//  //Thermal Timer
//  thermalTimer = millis() + 50;

}

void loop() {
//  Serial.print("current state: ");
//  Serial.println(currentState);
  switch (currentState) {

    //Idling state. Waiting to see human or to get a button press
    case 0:
      //Check to see if the lid button is pressed to open
      if (checkMotorButton()) {
        //Move to motor closing state
        buttonIsPressed = false;
        prevState = 0;
        currentState = 4;
      }
      //Check to see if human is in the center with enough distance to follow
      else if (checkHumanCenter()) {
        //Move to following state
        prevState = 0;
        currentState = 1;
        startMotors();
      }
      //Check to see if human is on the right side
      else if (checkHumanRight()) {
        //Move to the turning right state
        prevState = 0;
        currentState = 2;
        startMotors();
      }
      //Check to see if human is on the left side
      else if (checkHumanLeft()) {
        //Move to turning left state
        prevState = 0;
        currentState = 3;
        startMotors();
      }
      prevState = 0;
      break;
    //Following state. Move forward to follow human
    case 1:
      //Check to see if the lid button is pressed to open
      if (checkMotorButton()) {
        //Move to lid opening state
        buttonIsPressed = false;
        prevState = 1;
        currentState = 4;
      }
      //Check to see if human is on the right side
      else if (checkHumanRight()) {
        //Move to the turning right state
        prevState = 1;
        currentState = 2;
      }
      //Check to see if human is on the left side
      else if (checkHumanLeft()) {
        //Move to turning left state
```

```
        prevState = 1;
        currentState = 3;
      }
      //Check to see if human is no longer at the center or not at an accepted distance
      else if (checkHumanCenter() == false) {
        //Stop motors and move to idling state
        prevState = 1;
        currentState = 0;
        stopMotors();
      }
      else {
        //If none of the previous event checkers were triggered move forward
        moveForward();
      }
      prevState = 1;
      break;
    //Turning right state. Turn right until human is in the middle
    case 2:
      //Check to see if the lid button is pressed to open
      if (checkMotorButton()) {
        //Move to lid opening state
        buttonIsPressed = false;
        prevState = 2;
        currentState = 4;
      }
      //Check to see if human is in the center with enough distance to follow
      else if (checkHumanCenter()) {
        //Move to following state
        prevState = 2;
        currentState = 1;
      }
      //Check to see if human is on the left side
      else if (checkHumanLeft()) {
        //Move to turning left state
        prevState = 2;
        currentState = 3;
      }
      //Check to see if human is no longer at the right after checking if human is in the center
      else if (checkHumanRight() == false) {
        //Stop motors and move to the idling state
        prevState = 2;
        currentState = 0;
        stopMotors();
      } else {
        //If none of the previous event checkers were triggered turn right
        turnRight();
      }
      prevState = 2;
      break;
    //Turning left state. Turn left until human is in the middle
    case 3:
      //Check to see if the lid button is pressed to open
      if (checkMotorButton()) {
        //Move to lid opening state
        buttonIsPressed = false;
        prevState = 3;
        currentState = 4;
      }
      //Check to see if human is in the center with enough distance to follow
      else if (checkHumanCenter()) {
```

```
        //Move to following state
        prevState = 3;
        currentState = 1;
      }
      //Check to see if human is on the right side
      else if (checkHumanRight()) {
        //Move to the turning right state
        prevState = 3;
        currentState = 2;
      }
      //Check to see if human is no longer at the left after checking if human is in the center
      else if (checkHumanLeft() == false) {
        //Stop motors and move to idling state.
        prevState = 3;
        currentState = 0;
        stopMotors();
      } else {
        //If none of the previous event checkers were triggered turn left
        turnLeft();
      }
      prevState = 3;
      break;
    //Stop Motor State. Stop the motors until the button is pressed again.
    case 4:
      //Check to see if motors are on
      if (motorsOn) {
        //Stop motors
        stopMotors();
        motorsOn = false;
      }
      //Check to see if the motor button is pressed to start motors
      if (checkMotorButton()) {
        //Start motors and go back to idling state
        prevState = 4;
        currentState = 0;
        buttonIsPressed = false;
        startMotors();
        motorsOn = true;
      }
      prevState = 4;
      break;
    default:
      Serial.println("Robot Error");
      break;
  }


  delay(100);


}

bool checkMotorButton() {
  if (buttonIsPressed && timePassed) {
    Serial.println("Button is pressed");
    return true;
  }
  return false;
}

```

```
bool checkHumanCenter() {
    //Read from thermal camera
    thermalRead();
    readIR();
    if (arrayCenter > 2 && arrayCenter < 7 || (arrayCenter == 10) && (distance_left > 400 || distance_right > 400
      if (distance_left < stop_distance && distance_right < stop_distance) {
        stop_distance = STOP_DISTANCE_1;
        return true;
      }
      else {
        //Serial.println("Returned false due to stopping distance");
        stop_distance = STOP_DISTANCE_2;
        return false;
      }
    }
    return false;
}

bool checkHumanRight() {
    thermalRead();
    if (arrayCenter < 3) {
       readIR();
      if (distance_left < stop_distance && distance_right < stop_distance) {
        stop_distance = STOP_DISTANCE_1;
        return true;
      }
      else {
        //Serial.println("Returned false due to stopping distance");
        stop_distance = STOP_DISTANCE_2;
        return false;
      }
    }
    return false;
}

bool checkHumanLeft() {
    thermalRead();
    if (arrayCenter != 10 && arrayCenter > 6) {
      readIR();
      if (distance_left < stop_distance && distance_right < stop_distance) {
        stop_distance = STOP_DISTANCE_1;
        return true;
      }
      else {
        //Serial.println("Returned false due to stopping distance");
        stop_distance = STOP_DISTANCE_2;
        return false;
      }
    }
    return false;
}

void stopMotors() {
  motorCm = millis();
  if (currentState != prevState && motorCm > motorPm + MOTOR_PERIOD) {
    Serial.println("Stopping Motors");
   digitalWrite(IN1,LOW);
   digitalWrite(IN2,LOW);
   digitalWrite(IN3,LOW);
   digitalWrite(IN4,LOW);
```

```
   motorPm = motorCm;
  }

}

void moveForward() {
  motorCm = millis();
  if (currentState != prevState && motorCm > motorPm + MOTOR_PERIOD) {
    Serial.println("Moving Forward");
     analogWrite(ENA, MOTOR_PWM_VOLTAGE);
     analogWrite(ENB, MOTOR_PWM_VOLTAGE);
     motorPm = motorCm;
  }
}

void turnRight() {
  motorCm = millis();
  if (currentState != prevState && motorCm > motorPm + MOTOR_PERIOD) {
    Serial.println("Turning Right");
//    analogWrite(ENA, MOTOR_PWM_VOLTAGE);
//    analogWrite(ENB, TURN_PWM_VOLTAGE);
    analogWrite(ENA, 0);
    analogWrite(ENB, TURN_PWM_VOLTAGE);
    motorPm = motorCm;
  }
}

void turnLeft() {
  motorCm = millis();
  if (currentState != prevState && motorCm > motorPm + MOTOR_PERIOD) {
    Serial.println("Turning left");
//    analogWrite(ENA, TURN_PWM_VOLTAGE);
//    analogWrite(ENB, MOTOR_PWM_VOLTAGE);
    analogWrite(ENA, TURN_PWM_VOLTAGE);
    analogWrite(ENB, 0);
    motorPm = motorCm;
  }
}

void startMotors() {
  motorCm = millis();
  if (currentState != prevState && motorCm > motorPm + MOTOR_PERIOD) {
    Serial.println("Starting Motors");
     digitalWrite(IN1,HIGH);
     digitalWrite(IN2,LOW);
     digitalWrite(IN3,HIGH);
     digitalWrite(IN4,LOW);
     motorPm = motorCm;
  }
}

void thermalRead() {
  if (MySerial.available() > 0) {
    arrayCenter = MySerial.read();
//     thermalTimer += THERMAL_PERIOD;
     Serial.print("Array center is: ");
     Serial.println(arrayCenter);
//     Serial.println();
//     Serial.print("Array center is: ");
//     Serial.println(arrayCenter);
```

```
//     Serial.println();
  }
}

void readIR() {
  distance_right = 0;
  distance_left = 0;
  for (int i = 0; i < 25; i++) {
    distance_right += analogRead(IR_PIN_RIGHT);
    distance_left += analogRead(IR_PIN_LEFT);
  }
  distance_right /= 25;
  distance_left /= 25;
}
```

```
#define BIN_1 26
#define BIN_2 25
#define LED_PIN 13
const int buttonPin = 14;
 int  red = 0;

  int blue = 0;


int buttonState = 0;

// declare timer variables ----------------------------
unsigned long currentTime = 0;
unsigned long lastCycleTime = 0;
unsigned long lastStepTime = 0;
const int cycleDuration = 4000; // duration of full cycle in ms

// setting PWM properties ----------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
int MAX_PWM_VOLTAGE = 255;

void setup() {
    pinMode(LED_PIN, OUTPUT);t
    digitalWrite(LED_PIN, LOW);
  pinMode(buttonPin, INPUT);
    Serial.begin(115200);


    ledcSetup(ledChannel_1, freq, resolution);
    ledcSetup(ledChannel_2, freq, resolution);


    ledcAttachPin(BIN_1, ledChannel_1);
    ledcAttachPin(BIN_2, ledChannel_2);

    lastCycleTime = millis();
    lastCycleTime = millis();
    lastStepTime = millis();

}

void loop() {
    buttonState = digitalRead(buttonPin);
    currentTime = millis();

    // The state of the lid is called red. This event checker first checks to see the state of red, if it is below 1 then the lid is closed. if it is above 1 then the lid is open.
    // then the event checker checks to see if the button has been pressed, if it has been pressed it then it performs the action of turning the motor!
if (red < 1){
  if( buttonState == HIGH){

    // turn LED on:
      ledcWrite(ledChannel_1, MAX_PWM_VOLTAGE);
      ledcWrite(ledChannel_2, LOW);
    Serial.println("LID OPEN");
    delay(1100);
      // turn LED off:
      red = red+2;
     ;

      ledcWrite(ledChannel_1, LOW);
      ledcWrite(ledChannel_2, LOW);

    delay(1000);
      buttonState = digitalRead(buttonPin);


  }}
// event checker to see if the lid is open, and if it is then to see if the button has been pressed!

  if (red > 1);{
  if( buttonState == HIGH){

    // turn LED on:
      ledcWrite(ledChannel_1, LOW);
      ledcWrite(ledChannel_2, MAX_PWM_VOLTAGE);
```

```
      buttonState = digitalRead(buttonPin);


  }}
// event checker to see if the lid is open, and if it is then to see if the button has been pressed!

  if (red > 1);{
  if( buttonState == HIGH){

    // turn LED on:
      ledcWrite(ledChannel_1, LOW);
      ledcWrite(ledChannel_2, MAX_PWM_VOLTAGE);

    delay(1100);
      // turn LED off:
      red = red-2;
     ;

      ledcWrite(ledChannel_1, LOW);
      ledcWrite(ledChannel_2, LOW);
Serial.println("LID CLOSED");
    delay(1000);
      buttonState = digitalRead(buttonPin);


  }}









  }

```