ME102B Final Report: Rubble Robot

Alex Chemali, Matthew Nelson, Alejandro Orozco

December 12, 2021

Device Opportunity

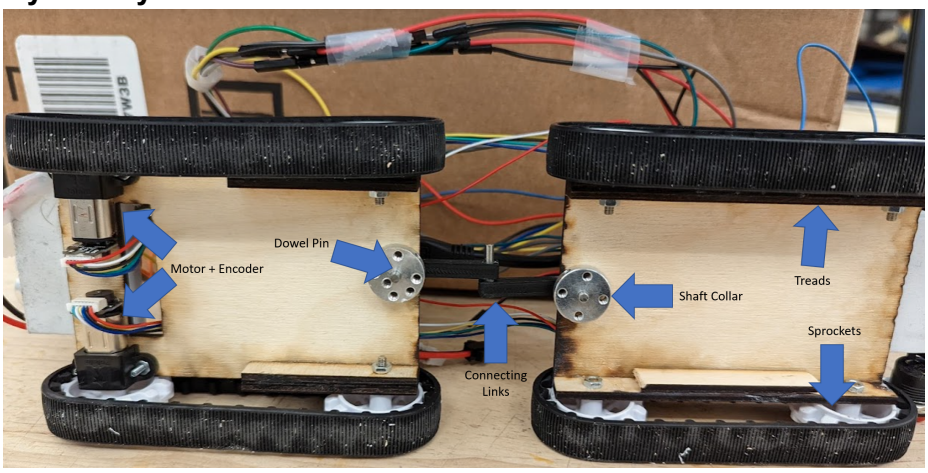
The early stages of emergency response are the most important. Especially for earthquakes, it is vital that emergency responders begin their search for those trapped under rubble or debris. Any time wasted clearing debris in an area where there might not be a survivor could cost critical time for others trapped under the debris. However, if a small and easily maneuverable robot (or multiple robots) could be sent out into the debris they would reduce the time wasted searching in areas where survivors may not be as well as protecting the lives of responders since there would be no risk to human harm if the robots were first exploring debris. The Rubble Robot provides an efficient way for first responders to explore areas remotely and receive information on whether or not there is someone trapped in debris before sending people to clear the debris.

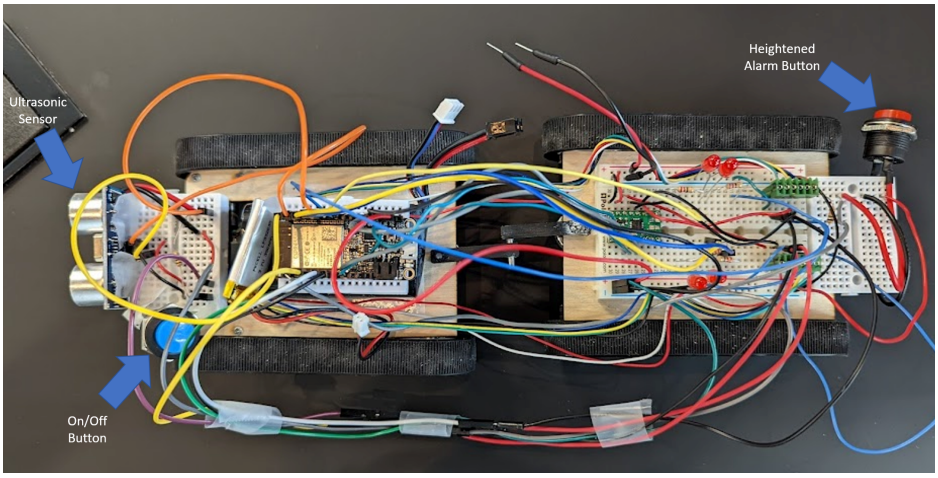
High Level Strategy

The implemented high level strategy is as follows: Rubble Robot will be placed down and turned on by the user and will drive forward and either timeout after five seconds and alarm or detect an obstacle and turn right until the alarm button is pressed which will put it into a heightened alarm state.

Originally, we wanted to have the Rubble Robot to be able to recognize whether the obstacle was a person or not using a thermal camera or some computer vision and alarm that way, but we settled on using another button to put the Rubble Robot in an alarm state. Also, we had hoped to implement some sort of remote communication but as that was not a requirement of the project we decided to not implement that in order to improve the other aspects of the device. However, we were able to achieve the driving speed and characteristics we wanted with the expectation of the turning state having some trouble due to the mechanical design of the linkage between the two cars.

Integrated Physical Systems





Function Critical Calculations

The most critical calculation for our device was whether the actuators had enough torque to drive the car. The two 6V DC motors we were using have a nominal torque of 3.25 kg-mm and nominal current of 0.4 A. We want to see if two of these motors can overcome the force of friction that is resultant from the weight of the electronics and chassis of the Rubble Robot.

Firstly, we calculate the mass of all the components:
Two Motors = 20 g, PETG parts = 50 cm^3 * 1.23 g/cm^3 = 61.5 g, Ultrasound sensor = 11 g, Button = 30 g, ESP32 = 9 g, 3.7V 350mAh Battery = 7 g, 7.4V 2000mAh Battery = 100 g, Breadboards and Wires= 100 g, Treads and sprockets (ABS) = 4 * (18.12 cm^3 * 1 g/cm^3) = 72.5 g, Screws, nuts, washers (for sprocket) and dowel pins = 20 g, Total = 431 g, Incorporating a 1.2 Factor of Safety we designed the motors to pull 500 g.

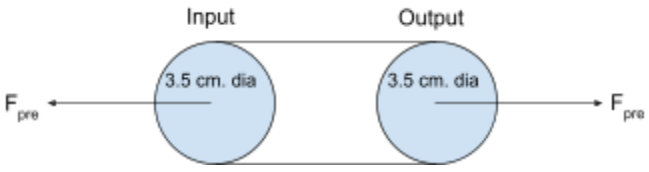
The sprocket and timing belt will create a friction force that the motors will have to overcome (approximating coefficient of friction to be 0.5). The motor operates at 80% of its max PWM since it will not be continuous for a long period of time and the radius of the sprocket is 35 mm.

$$F_f = \mu * m * g = 0.5 * 0.5 * 9.81 = 2.45\ N$$

$$\tau_{motors} = 2 * (0.8 * 1.3) = 2.08\ kg\ cm \text{ and}$$

$$F_{motors} = \tau_{motors}/r = 2.08\ kg\ cm/3.5\ cm = 0.6\ kgf = 5.9\ N$$

Since the force the motors produce is greater than the expected force of friction, our choice of actuators is appropriate. Additionally, we want to know how much pre-tension our sprockets need since we are using a timing belt transmission. We do not want the transmission to slack under larger torque conditions. Since we are using timing belts, we can assume no slipping on the sprockets. Ignoring inertial effects and gravity, we want to calculate the pretension force between the sprockets to prevent slackness. Analyzing one side of the Rubble Robot, the motor will input 1.04 kg cm torque and the sprockets have a diameter of 3.5 cm.



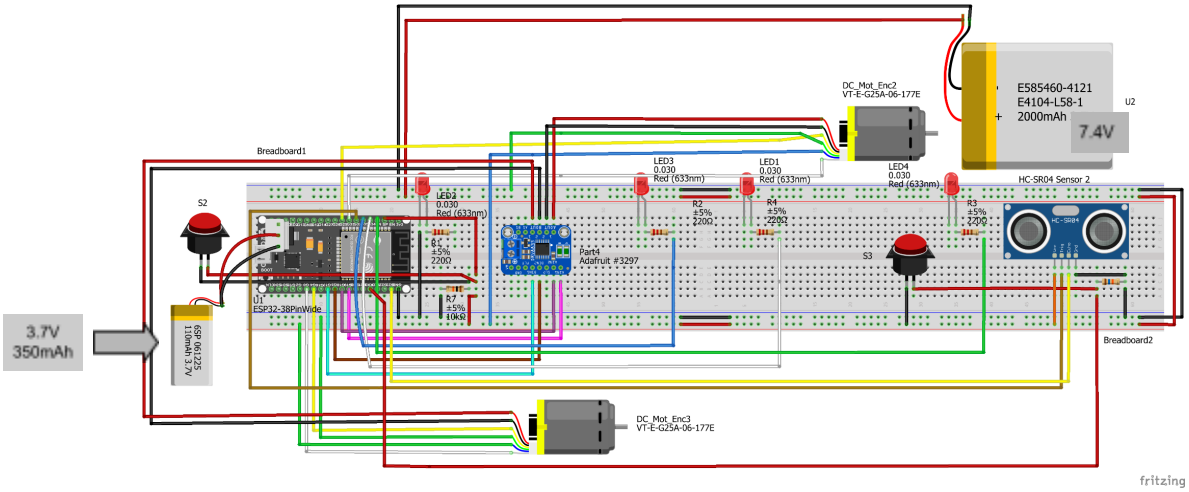
$$T_2 = F_{pre}/2 - (\tau/d)$$

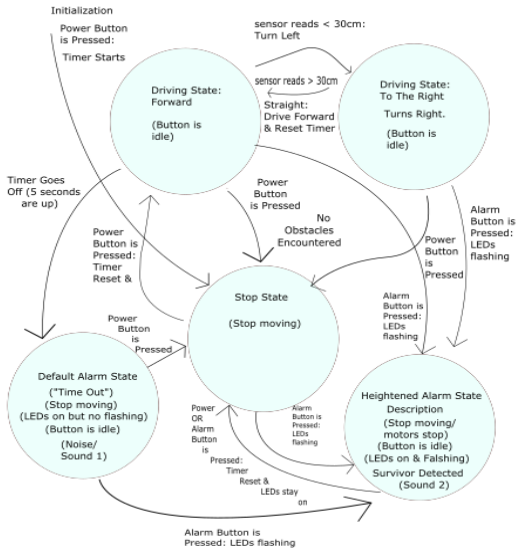
$T_2 \leq 0$ is the condition for slackness, so minimum pretension force:

$$0 = (F_{pre}/2) - (\tau/d) \text{ so } F_{pre} = 0.6\,kgf$$

We can stretch out the two sprockets instead of idlers since the minimum pretension force is not very high. This allows us to keep our design compact by not including idlers as well as keeping it lightweight.

Circuit and State Diagram





Reflection

Narrowing down the ideas or directions for this project was difficult. However, once we made a decision, it was imperative that we began planning ahead on sourcing parts and materials. As the lead time for those was the bottleneck for our project. Thus we designed the systems in the order of longest lead time first. While our design was complete it was rushed because of this, so if we had spent more time choosing a project with these concerns in mind it would have gone smoother.

Appendix

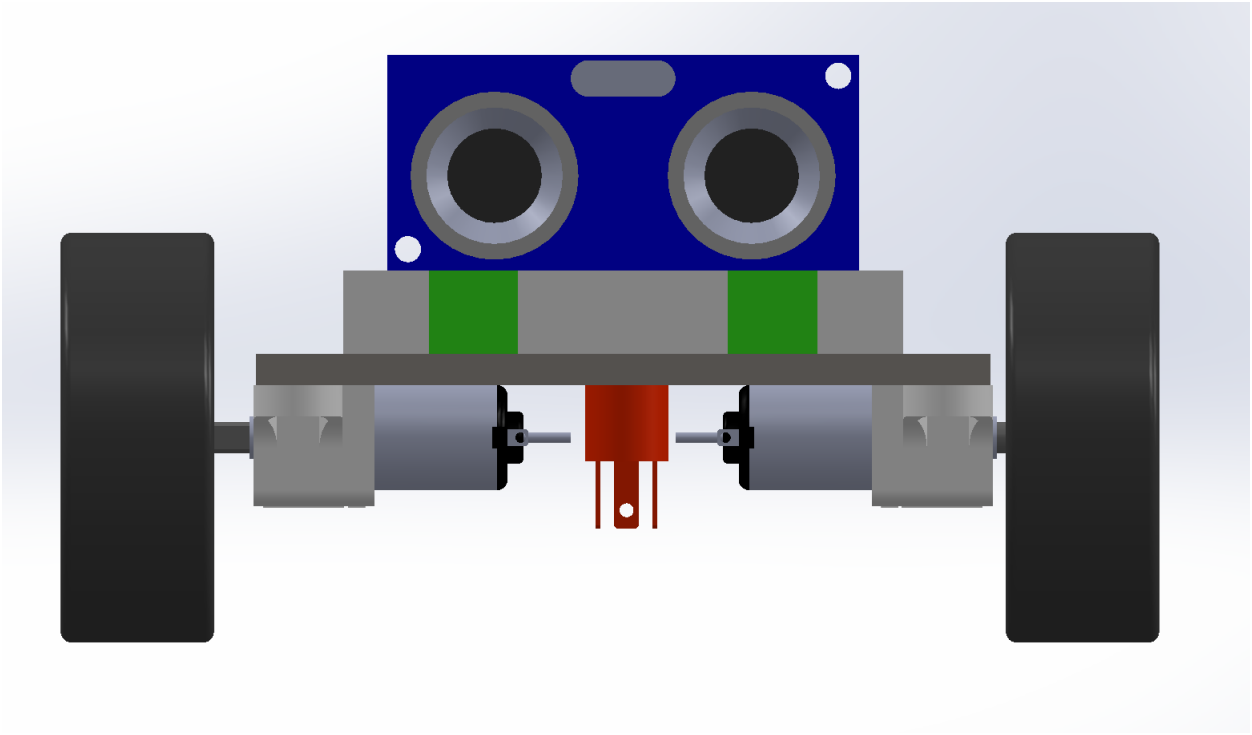
BOM

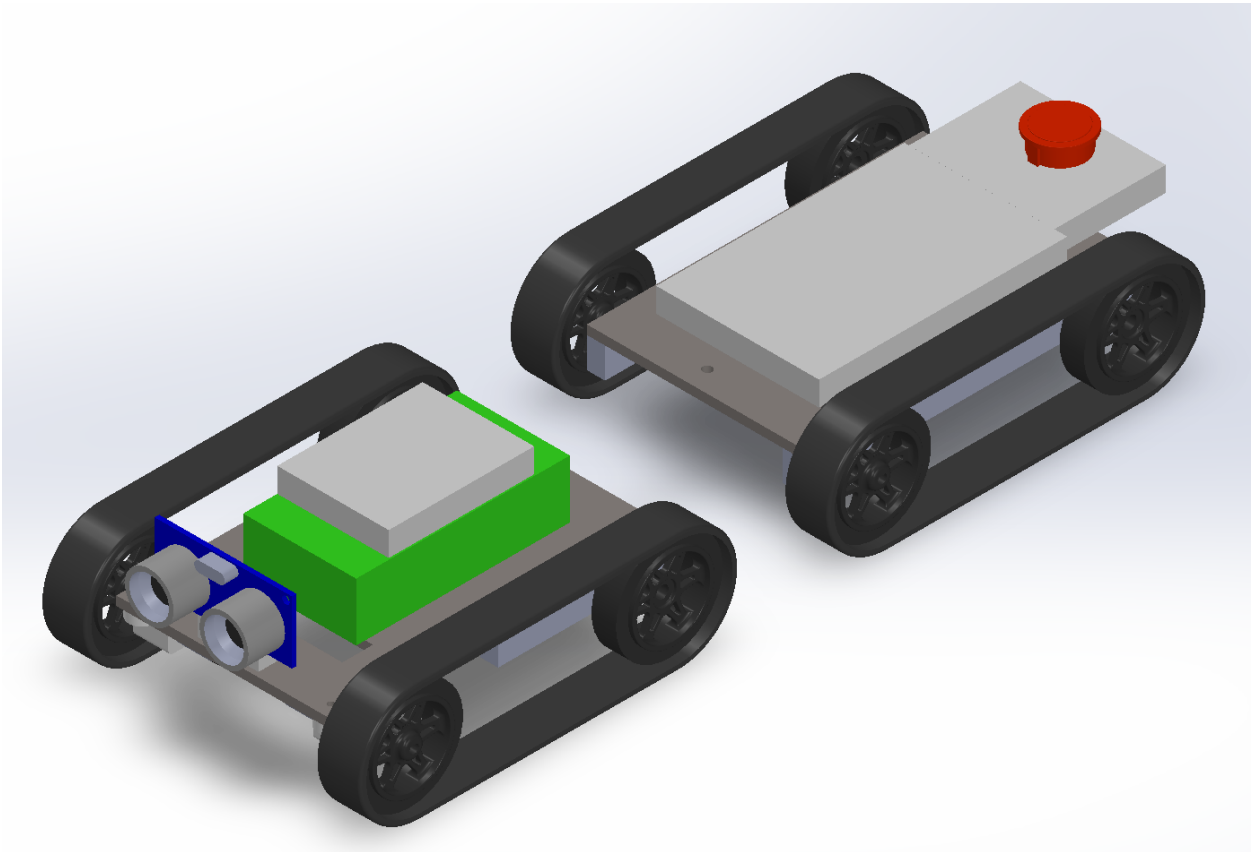
Name	Quantity	Unit Cost	Comments
HUZZAH32 Feather Microcontroller (ESP32)	1	$10. 99	In Kit
Pololu 30T Track Set - White	2	$14.95	Comes with fasteners, sprockets, and belt
Replacement Sprocket Set for Zumo Chassis	1	$5.95	For extra idler sprockets
Lithium Ion Polymer Battery – 3.7V 290mAh	2	$5.95	In Kit
KABUDA Multicolored Dupont Wire 20 Pin Male to Female, 20 Pin Male to Male, 20 Pin Female to Female, Breadboard Jumper Wires Ribbon Cables Kit	1 set	$9.99	In Kit
3 Foot micro-USB Data Sync and Power Charge Cable	1	$2.95	In Kit
Red, Yellow, Green LED	4	$0.15	In Kit
Micro Metal Gearmotor HP 6V with Extended Motor Shaft	2	$16.95	In Kit

DRV8833 Dual Motor Driver Carrier	1	$6.95	In Kit
Pololu Micro Metal Gearmotor Bracket Pair	2	$2.95	In Kit
Momentary Push Buttons	1 set (10 pcs)	$9.99	Only need 2, Amazon
Dowel Pins, M2 M3 M4 Stainless Steel	1 (75 Pack)	$11.99	Amazon
400-Point Breadboard	1	$2.49	Pololu
170-Point Breadboard	2	$2.95	Pololu
Ultrasonic Sensor	1	$3.95	In Kit
7.4V Li-ion Battery	1 (2 pack)	$24.99	Amazon
Shaft Collar	1 (15 pcs)	$7.99	Amazon

Total (including S&H but not including items in microkit) = $120.28

CAD





Code

```
#include <ESP32Encoder.h>
#include <NewPing.h>

#define BIN_1 26
#define BIN_2 25
#define AIN_1 13
#define AIN_2 12
#define LED_PIN 13
#define BTN_Motor 18
#define BTN_Alarm 23
#define echoPin 21 // attach pin 15 Arduino to pin Echo of HC-SR04
#define trigPin 17 //attach pin 32 Arduino to pin Trig of HC-SR04
const byte led1 = 19;
const byte led2 = 16;
const byte led3 = 5;
const byte led4 = 4;
const byte speakerPin = 22;

byte state = 0;

// setting up ultrasound sesnor object//////////////////////////////////////////////////////
#define maximum_distance 200
boolean goesForward = false;
int distance = 100;
NewPing sonar(trigPin, echoPin, maximum_distance);
//////////////////////////////////////////////////////////////////////////////////////////////


volatile bool timeOut = false;    // check timer interrupt 1
hw_timer_t * timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  timeOut = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}

volatile bool doneWaiting = false;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  doneWaiting = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

// setting PWM properties ----------------------------
const int freq = 5000;
const int ledChannel_1 = 1; //Right Motor
const int ledChannel_2 = 2; //Right Motor
const int ledChannel_3 = 3; //Left Motor
const int ledChannel_4 = 4; //Left Motor
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;
const int DES_PWM_VOLTAGE = 150;
// ---------------------------------------------------

// setting up Button interrupts ----------------------------------------------------------------
volatile bool Alarm_buttonIsPressed = false;
volatile bool Power_buttonIsPressed = false;

void IRAM_ATTR isr_power() {  // the function to be called when interrupt is triggered
    Power_buttonIsPressed = true;
}

void IRAM_ATTR isr_alarm() {  // the function to be called when interrupt is triggered
    Alarm_buttonIsPressed = true;
}
// ---------------------------------------------------------------------------------------------


```

```
// SETUP SETUP SETUP SETUP SETUP SETUP
void setup() {
  Serial.begin(115200);

  pinMode(BTN_Motor, INPUT); // configures the specified pin to behave either as an input or an output
  pinMode(BTN_Alarm, INPUT); // configures the specified pin to behave either as an input or an output
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW); // sets the initial state of LED as turned-off
  attachInterrupt(BTN_Motor, isr_power, RISING);
  attachInterrupt(BTN_Alarm, isr_alarm, RISING);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT

  // configure LED PWM functionalitites
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);
  ledcSetup(ledChannel_3, freq, resolution);
  ledcSetup(ledChannel_4, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(BIN_1, ledChannel_1); //Right
  ledcAttachPin(BIN_2, ledChannel_2); //Right
  ledcAttachPin(AIN_1, ledChannel_3); //Left
  ledcAttachPin(AIN_2, ledChannel_4); //Left


  // initilize timer
  timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
  timerAlarmWrite(timer0, 5000000, true); // 5000000 * 1 us = 2 s, autoreload true
  timerAlarmEnable(timer0); // enable

  timer1 = timerBegin(1, 80, true);  // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
  timerAlarmWrite(timer1, 3000000, true); // 3000000 * 1 us = 2 s, autoreload true
  timerAlarmEnable(timer1); // enable

}


// LOOP LOOP LOOP LOOP LOOP LOOP LOOP
void loop() {
  Serial.println(readPing());
  switch (state) {

    case 0 : // motor is stopped
      if (Power_buttonPressEvent()) {
        state = 1;
        startMotorResponse();
        resetTimer();
        digitalWrite(led1,LOW);
        digitalWrite(led2,LOW);
        digitalWrite(led3,LOW);
        digitalWrite(led4,LOW);
      } else if (Alarm_buttonPressEvent()) {
        state = 2;
        stopMotorResponse();
        digitalWrite(led1,LOW);
        digitalWrite(led2,LOW);
        digitalWrite(led3,LOW);
        digitalWrite(led4,LOW);
      }
      break;

```

```
    case 1 : //forward driving
      if (Alarm_buttonPressEvent()) {
        state = 2;
        stopMotorResponse();
      } else if (readPing() < 20) {
        state = 3;
        resetWaiting();
//        turnRight();
        stopMotorResponse();
      } else if (Power_buttonPressEvent()) {
        state = 0;
        stopMotorResponse();
      } else if (timeOutEvent()) {
        state = 0;
        digitalWrite(led1, HIGH);
        digitalWrite(led2, HIGH);
        digitalWrite(led3, HIGH);
        digitalWrite(led4, HIGH);
        stopMotorResponse();
      }
      break;

    case 2 : //alarm state
      digitalWrite(led1, HIGH);
      digitalWrite(led2, HIGH);
      digitalWrite(led3, HIGH);
      digitalWrite(led4, HIGH);
      delay(200);
      digitalWrite(led1,LOW);
      digitalWrite(led2,LOW);
      digitalWrite(led3,LOW);
      digitalWrite(led4,LOW);
      delay(200);

      if (Alarm_buttonPressEvent() || Power_buttonPressEvent()) {
        state = 0;
        stopMotorResponse();
      }
      break;

      case 3 : // turning right
        if (doneWaiting) {
          turnRight();
          doneWaiting = false;
        } else {
          if (readPing() > 20) {
            state = 1;
            resetTimer();
            startMotorResponse();
          } else if (Power_buttonPressEvent()) {
            state = 0;
            stopMotorResponse();
          } else if (Alarm_buttonPressEvent()) {
            state = 2;
            stopMotorResponse();
          }
        }
        break;

  }
}



//Event Checkers
bool Power_buttonPressEvent() {
  if (Power_buttonIsPressed == true){
    Power_buttonIsPressed = false;
    return true;
  }
  else {
    return false;
  }
}
```

```
222
223 bool Alarm_buttonPressEvent() {
224   if (Alarm_buttonIsPressed == true){
225     Alarm_buttonIsPressed = false;
226     return true;
227   }
228   else {
229     return false;
230   }
231 }
232
233 bool timeOutEvent() {
234   if (timeOut == true){
235     timeOut = false;
236     return true;
237   }
238   else {
239     return false;
240   }
241 }
242
243 //Event Service Responses
244 void stopMotorResponse() {
245   ledcWrite(ledChannel_2, LOW);
246   ledcWrite(ledChannel_1, LOW);
247   ledcWrite(ledChannel_4, LOW);
248   ledcWrite(ledChannel_3, LOW);
249   digitalWrite(LED_PIN, LOW);
250 }
251
252 void startMotorResponse() {
253   ledcWrite(ledChannel_1, LOW);
254   ledcWrite(ledChannel_2, DES_PWM_VOLTAGE);
255   ledcWrite(ledChannel_3, LOW);
256   ledcWrite(ledChannel_4, DES_PWM_VOLTAGE);
257   digitalWrite(LED_PIN, HIGH);
258 }
259
260 void turnRight() {
261   ledcWrite(ledChannel_1, LOW);
262   ledcWrite(ledChannel_2, 0); //RIGHT MOTOR
263   ledcWrite(ledChannel_3, LOW);
264   ledcWrite(ledChannel_4, 180);
265   digitalWrite(LED_PIN, HIGH);
266 }
267
268 void resetTimer() {
269   timeOut = false;
270   timerWrite(timer0, 0);
271 }
272
273 void resetWaiting() {
274   doneWaiting = false;
275   timerWrite(timer1, 0);
276 }
277
278 int readPing(){
279   delay(10);
280   int cm = sonar.ping_cm();
281   if (cm==0){
282     cm=250;
283   }
284   return cm;
285 }
```