MEC ENG 102B Final Project Report

Mechatronics Design Fall 2022

Team 21: AutonoTech

Xiangyu Hong
Ziyi Wang

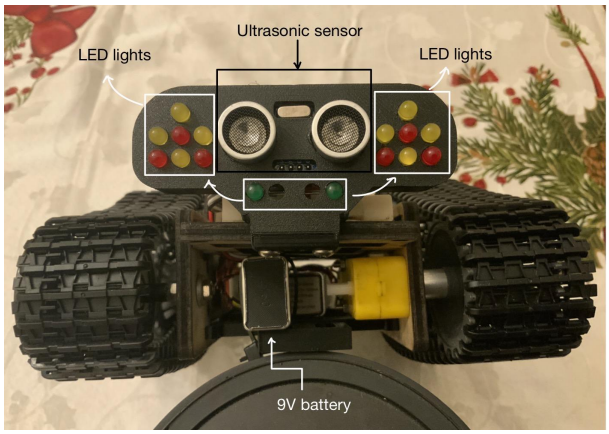
Prepared for Hannah Stuart
December 9, 2022

-- Opportunity --

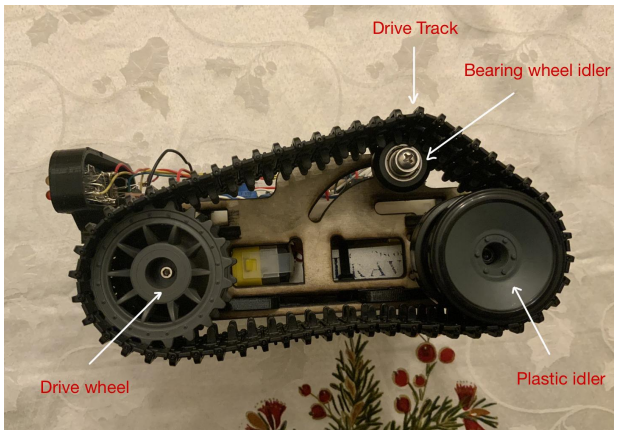
We want AutonoTech to drive autonomously, which we use an ultrasonic sensor to detect potential obstacles in the front, and AutonoTech decides whether to make a turn or keep driving forward. No matter it is in driving mode or stop mode, AutonoTech always uses an IMU to detect the surrounding temperature in order to decide whether there is a potential fire in the house. When the detected surrounding temperature is above 50℃, the buzzer will make an alarm sound, AutonoTech will stop, and red LEDs and yellow LEDs will flash one by one. The buzzer will stop the alarm when the user presses the button or the surrounding temperature drops to normal room temperature. (For demonstration purposes, we set a temperature threshold of 30℃) While AutonoTech is moving forward, it lights up the green LEDs which indicates the daytime running light. While AutonoTech is turning left, left-side yellow LEDs will flash and the buzzer will make a left-turn monophonic sound. While AutonoTech is turning right, right-side yellow LEDs will flash and the buzzer will make a right-turn monophonic sound. While AutonoTech is making a U-turn, all yellow LEDs and red LEDs will flash at the same time and the buzzer will make a two-tone sound. When AutonoTech makes three U-turns in 12 seconds, AutonoTech will treat itself as "Got Trapped", the buzzer will make Nokia music, and it will light up all red LEDs. AutonoTech also has an entertainment system, in which users can turn the potentiometer to switch songs. While AutonoTech stops, users can switch to Merry Christmas or by default no sound. While AutonoTech is driving, users can switch music between Super Mario Bros or Imperial March Star Wars or by default no sound. In addition for our initially desired functionality, we wanted to control the turning angle to be 90 degrees/turn and achieve 88 degrees/turn. For the U-turn, we wanted to control the turning angle to be 360 degrees/U-turn and achieve 350 degrees/U-turn.

--Integrated physical device --

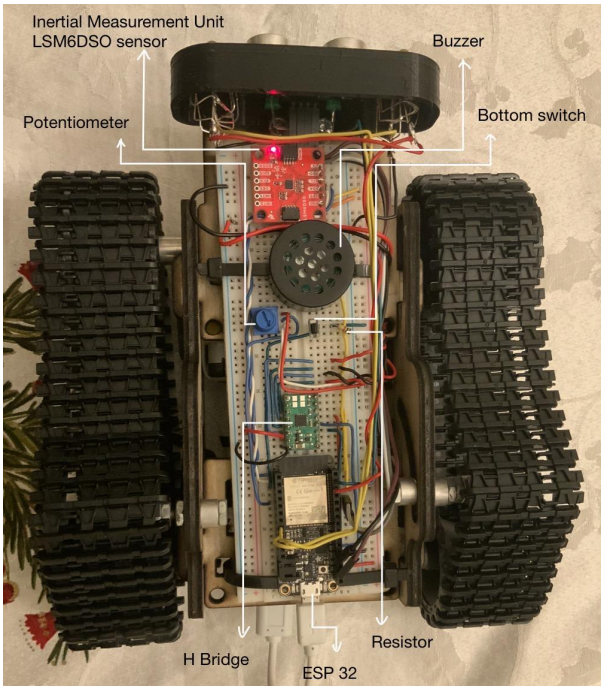
For this project, we use ESP 32 as the control board. The sensors we are using are Ultrasonic Sensor for navigation while detecting obstacles and IMU LSM6DSO sensor for checking the surrounding temperature. (Label on the Top view) For actuators, we have a switch press buttom, two DC motors one on each side of the robot which transmits power to drive the robot, 16 LED lights, and a buzzer that indicates the behavior and status of the robot. In addition to actuators, we have a potentiometer for the entertainment system which can switch songs. (Label on Front view and Top view) In addition, looking at the side and bottom view, the driving system is integrated with DC motor, gearbox, drive wheel, plastic idler, a slot for adjustable tensioner idler bearing, and drive track.



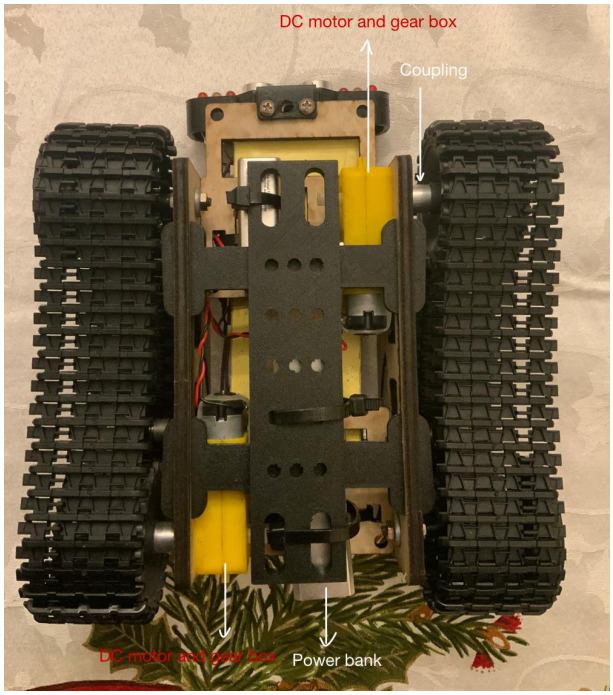
Front view



Side view



Top view



Bottom view

-- Function-critical decisions --

One of the function-critical components decisions we made was not adding bearings to support the shaft between the motor gearbox output and the drive wheels. In theory, for mechanism advantage, if the torque on the drive shaft is larger than the torque output on the gearbox, we will need to add bearings on both sides of the shaft to reduce the load and bending force to protect the drive shaft. However, in our cause:

$$Robot\ Max\ Weight\text{: } M = Mg = 2N$$

$$Drive\ Shaft/coupling\text{: } d = 2.8\ cm$$

$$Weight\ on\ each\ wheels: W = \frac{Mg}{4} = 0.5N$$

$$Torque\ on\ the\ drive\ shaft\text{: } T_d = W \cdot d = 0.5N \times 0.028m = 0.014\ Nm$$

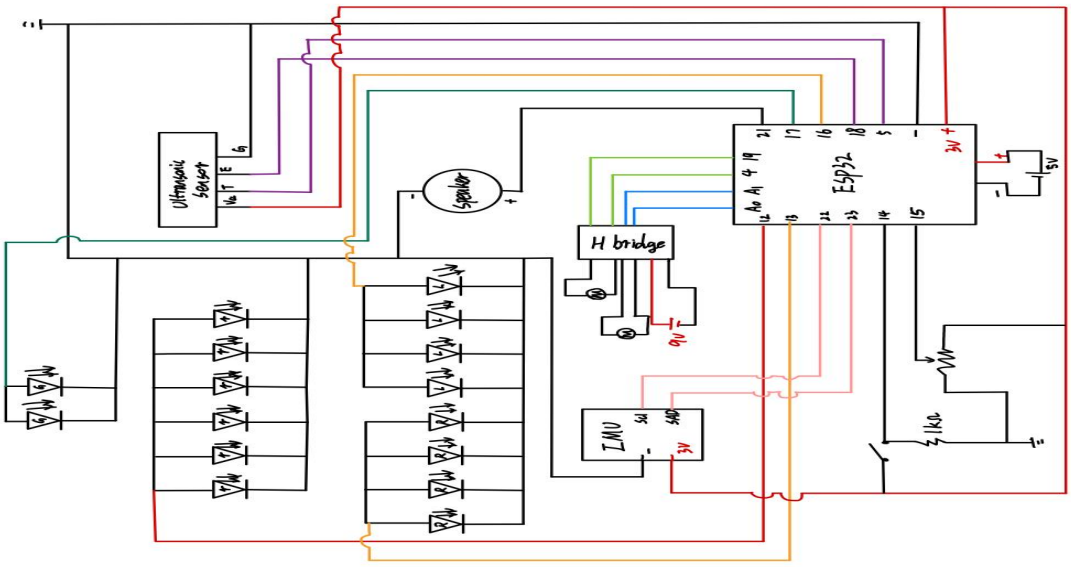
Torque on each motor gearbox shaft output T_g = 1.2 KGF.CM

$$T_g = 1.2KGF.CM \times \frac{0.098NM}{1KGF.CM} = 0.1176\ Nm$$

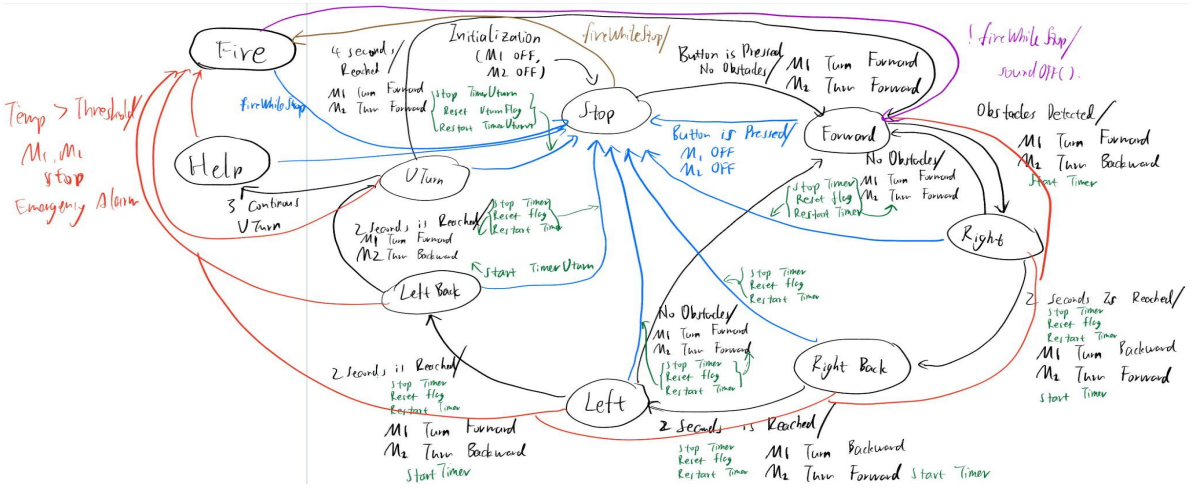
$$T_d < T_g$$

Based on the torque calculation above we conclude that the torque on the drive shaft is about $\frac{1}{10}$ of the motor gearbox shaft. Also, the weight of the robot is light. As a result, we won't necessarily need a bearing to support the shaft between the motor output and drive wheels. This can save our bill of materials.

-- Circuit diagram --



-- State Transition Diagram --

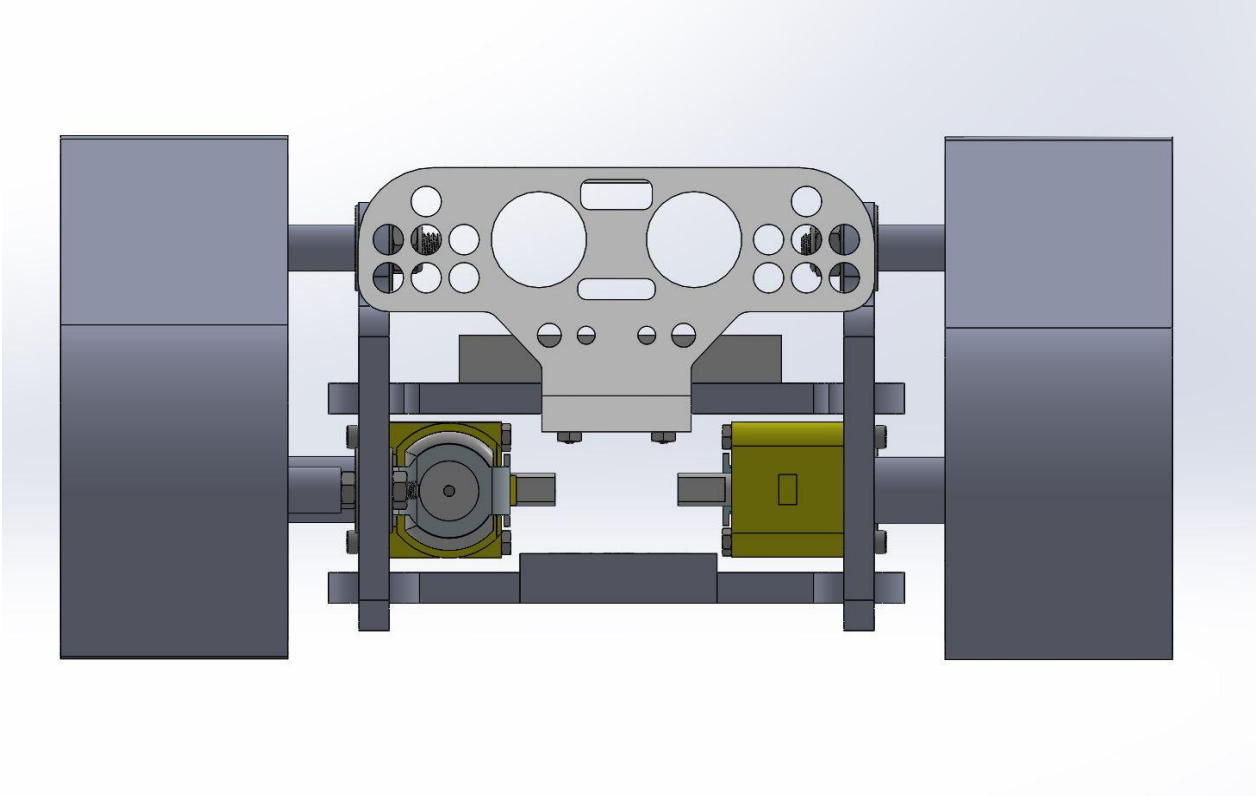


-- Reflection --

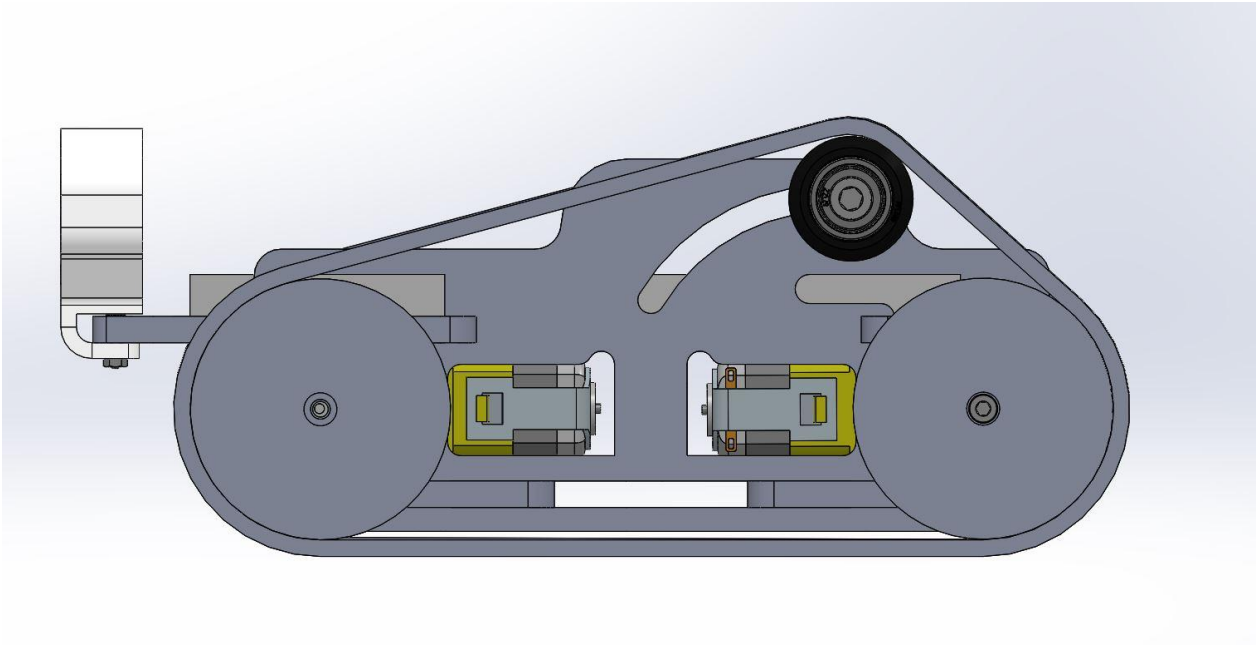
For a group of 2 people team, there are three strategies that worked well for our group such as building trust and respect for each other, clearly defining roles and responsibilities for each team member, and balancing work. On the other hand, what we wish we had done differently is real-time machine testing and tuning, because when it comes to real-time testing there will be a lot of unexpected problems or small issues that we could not visualize in computer design. Therefore, if we can spend more time on testing, tuning, and debugging I believe we will be more successful. In addition, the ultrasonic sensor has 2 weaknesses it doesn't detect corners very well because the 90° corner will reflect the ultrasonic sound waves to random direction which the ultrasonic sensor can't catch, and the ultrasonic sound waves won't be reflected if the sensor is so close to the obstacle. So if you are working with ultrasonic sensor, please consider install it on the middle of the vehicle and this should solve the second weakness. And if you have time, please consider install the ultrasonic sensor on a servo motor which the servo motor can swing the ultrasonic sensor in a small angle; then, you can calculate the average distance from your vehicle to the obstacles, and this should solve the first weakness because the vehicle won't be affected negatively by one 90° corner.

-- Appendices --

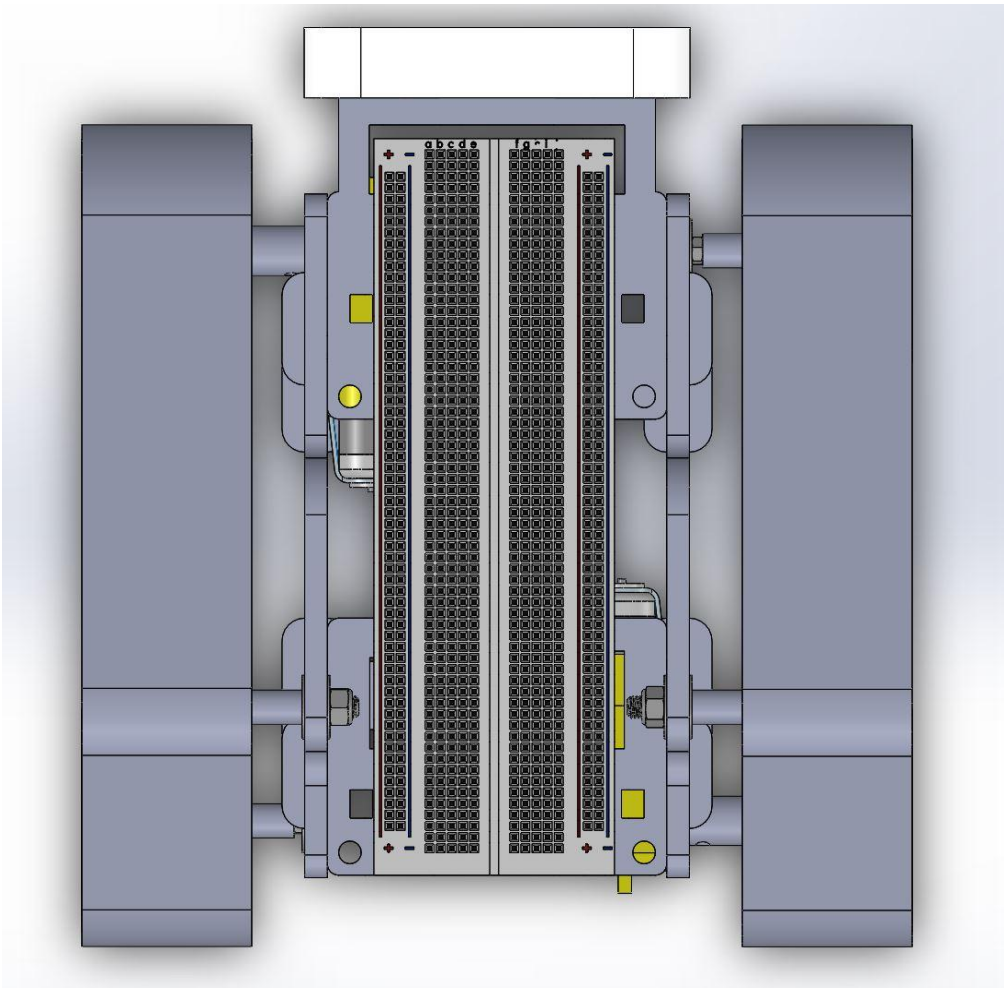
-- Images of the CAD --



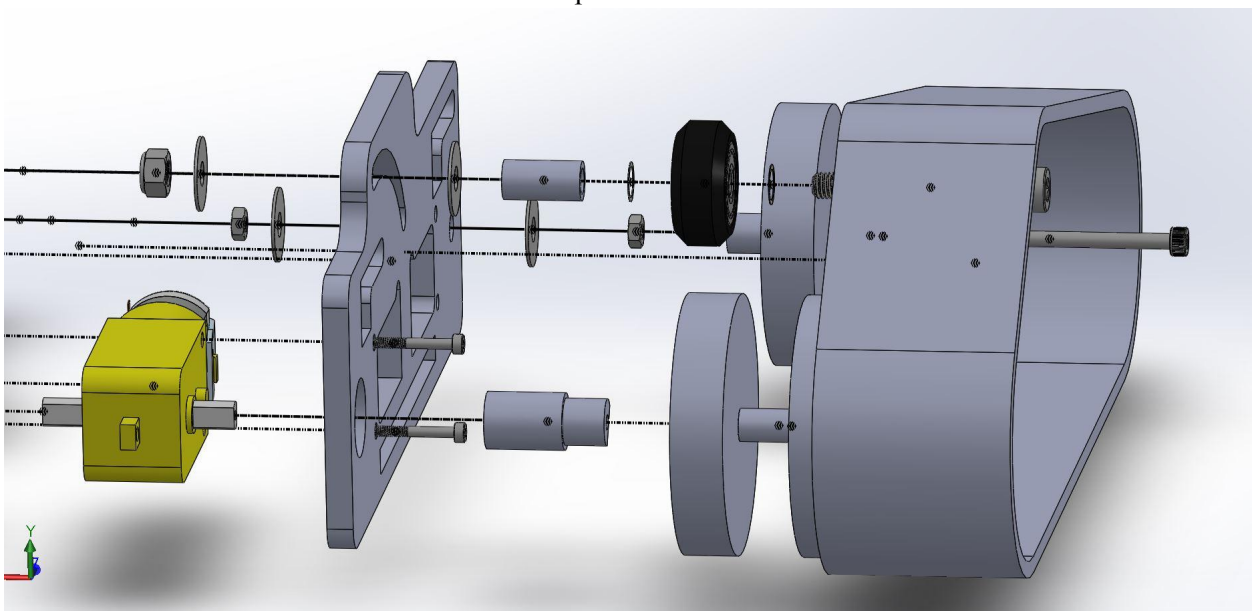
Tank Front view



Tank Side view



Tank Top view



Exploded view for mechanical transmission elements

-- Code --`

finalProject_withTone

```
#include <ESP32Encoder.h>
#include <Arduino.h>
#include "SparkFunLSM6DSO.h"
#include "Wire.h"
#define BIN_1 25
#define BIN_2 26
#define SPK 21  // speaker 21
#define LEFT_TRACK_1 4
#define LEFT_TRACK_2 19
#define L_LED 13
#define R_LED 12
#define RED_LED 16
#define GREEN_LED 17
#define POT 15
#define BTN 14  // declare the button ED pin number
//define sound speed in cm/uS
#define SOUND_SPEED 0.034
#define CM_TO_INCH 0.393701
//Music Tones
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
```

```
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
```

```
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
#define REST      0

// change this to make the song slower or faster
int nokiaTempo = 250;

// notes of the moledy followed by the duration.
```

```
// a 4 means a quarter note, 8 an eighteenth , 16 sixteenth, so on
// !!negative numbers are used to represent dotted notes,
// so -4 means a dotted quarter note, that is, a quarter plus an eighteenth!!
int nokiaMelody[] = {
  NOTE_E5, 8, NOTE_D5, 8, NOTE_FS4, 4, NOTE_GS4, 4,
  NOTE_CS5, 8, NOTE_B4, 8, NOTE_D4, 4, NOTE_E4, 4,
  NOTE_B4, 8, NOTE_A4, 8, NOTE_CS4, 4, NOTE_E4, 4,
  NOTE_A4, 2,
};

// sizeof gives the number of bytes, each int value is composed of two bytes (16 bits)
// there are two values per note (pitch and duration), so for each note there are four bytes
int nokiaNotes = sizeof(nokiaMelody) / sizeof(nokiaMelody[0]) / 2;

// this calculates the duration of a whole note in ms
int nokiaWholenote = (60000 * 4) / nokiaTempo;

int nokiaDivider = 0, nokiaNoteDuration = 0;

int nokiaPlay = 0;

int imMelody[] = {

  NOTE_A4,4, NOTE_A4,4, NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16,

  NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16, NOTE_A4,2,//4
  NOTE_E5,4, NOTE_E5,4, NOTE_E5,4, NOTE_F5,-8, NOTE_C5,16,
  NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16, NOTE_A4,2,

  NOTE_A5,4, NOTE_A4,-8, NOTE_A4,16, NOTE_A5,4, NOTE_GS5,-8, NOTE_G5,16, //7
  NOTE_DS5,16, NOTE_D5,16, NOTE_DS5,8, REST,8, NOTE_A4,8, NOTE_DS5,4, NOTE_D5,-8, NOTE_CS5,16,

  NOTE_C5,16, NOTE_B4,16, NOTE_C5,16, REST,8, NOTE_F4,8, NOTE_GS4,4, NOTE_F4,-8, NOTE_A4,-16,//9
  NOTE_C5,4, NOTE_A4,-8, NOTE_C5,16, NOTE_E5,2,

  NOTE_A5,4, NOTE_A4,-8, NOTE_A4,16, NOTE_A5,4, NOTE_GS5,-8, NOTE_G5,16, //7
  NOTE_DS5,16, NOTE_D5,16, NOTE_DS5,8, REST,8, NOTE_A4,8, NOTE_DS5,4, NOTE_D5,-8, NOTE_CS5,16,
```

```
  NOTE_C5,16, NOTE_B4,16, NOTE_C5,16, REST,8, NOTE_F4,8, NOTE_GS4,4, NOTE_F4,-8, NOTE_A4,-16,//9
  NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16, NOTE_A4,2,
};

int imTempo = 170;
int imNotes = sizeof(imMelody) / sizeof(imMelody[0]) / 2;
int imWholenote = (60000 * 4) / imTempo;
int imDivider = 0, imNoteDuration = 0;
int imPlay = 0;


int smMelody[] = {
  NOTE_E5,8, NOTE_E5,8, REST,8, NOTE_E5,8, REST,8, NOTE_C5,8, NOTE_E5,8, //1
  NOTE_G5,4, REST,4, NOTE_G4,8, REST,4,
  NOTE_C5,-4, NOTE_G4,8, REST,4, NOTE_E4,-4, // 3
  NOTE_A4,4, NOTE_B4,4, NOTE_AS4,8, NOTE_A4,4,
  NOTE_G4,-8, NOTE_E5,-8, NOTE_G5,-8, NOTE_A5,4, NOTE_F5,8, NOTE_G5,8,
  REST,8, NOTE_E5,4,NOTE_C5,8, NOTE_D5,8, NOTE_B4,-4,
  NOTE_C5,-4, NOTE_G4,8, REST,4, NOTE_E4,-4, // repeats from 3
  NOTE_A4,4, NOTE_B4,4, NOTE_AS4,8, NOTE_A4,4,
  NOTE_G4,-8, NOTE_E5,-8, NOTE_G5,-8, NOTE_A5,4, NOTE_F5,8, NOTE_G5,8,
  REST,8, NOTE_E5,4,NOTE_C5,8, NOTE_D5,8, NOTE_B4,-4,


  REST,4, NOTE_G5,8, NOTE_FS5,8, NOTE_F5,8, NOTE_DS5,4, NOTE_E5,8,//7
  REST,8, NOTE_GS4,8, NOTE_A4,8, NOTE_C4,8, REST,8, NOTE_A4,8, NOTE_C5,8, NOTE_D5,8,
  REST,4, NOTE_DS5,4, REST,8, NOTE_D5,-4,
  NOTE_C5,2, REST,2,

  REST,4, NOTE_G5,8, NOTE_FS5,8, NOTE_F5,8, NOTE_DS5,4, NOTE_E5,8,//repeats from 7
  REST,8, NOTE_GS4,8, NOTE_A4,8, NOTE_C4,8, REST,8, NOTE_A4,8, NOTE_C5,8, NOTE_D5,8,
  REST,4, NOTE_DS5,4, REST,8, NOTE_D5,-4,
  NOTE_C5,2, REST,2,

  NOTE_C5,8, NOTE_C5,4, NOTE_C5,8, REST,8, NOTE_C5,8, NOTE_D5,4,//11
  NOTE_E5,8, NOTE_C5,4, NOTE_A4,8, NOTE_G4,2,

  NOTE_C5,8, NOTE_C5,4, NOTE_C5,8, REST,8, NOTE_C5,8, NOTE_D5,8, NOTE_E5,8,//13
```

```
NOTE_E5,8, NOTE_C5,4, NOTE_A4,8, NOTE_G4,2,
NOTE_E5,8, NOTE_E5,8, REST,8, NOTE_E5,8, REST,8, NOTE_C5,8, NOTE_E5,4,
NOTE_G5,4, REST,4, NOTE_G4,4, REST,4,
NOTE_C5,-4, NOTE_G4,8, REST,4, NOTE_E4,-4, // 19

NOTE_A4,4, NOTE_B4,4, NOTE_AS4,8, NOTE_A4,4,
NOTE_G4,-8, NOTE_E5,-8, NOTE_G5,-8, NOTE_A5,4, NOTE_F5,8, NOTE_G5,8,
REST,8, NOTE_E5,4, NOTE_C5,8, NOTE_D5,8, NOTE_B4,-4,

NOTE_C5,-4, NOTE_G4,8, REST,4, NOTE_E4,-4, // repeats from 19
NOTE_A4,4, NOTE_B4,4, NOTE_AS4,8, NOTE_A4,4,
NOTE_G4,-8, NOTE_E5,-8, NOTE_G5,-8, NOTE_A5,4, NOTE_F5,8, NOTE_G5,8,
REST,8, NOTE_E5,4, NOTE_C5,8, NOTE_D5,8, NOTE_B4,-4,

NOTE_E5,8, NOTE_C5,4, NOTE_G4,8, REST,4, NOTE_GS4,4,//23
NOTE_A4,8, NOTE_F5,4, NOTE_F5,8, NOTE_A4,2,
NOTE_D5,-8, NOTE_A5,-8, NOTE_A5,-8, NOTE_A5,-8, NOTE_G5,-8, NOTE_F5,-8,

NOTE_E5,8, NOTE_C5,4, NOTE_A4,8, NOTE_G4,2, //26
NOTE_E5,8, NOTE_C5,4, NOTE_G4,8, REST,4, NOTE_GS4,4,
NOTE_A4,8, NOTE_F5,4, NOTE_F5,8, NOTE_A4,2,
NOTE_B4,8, NOTE_F5,4, NOTE_F5,8, NOTE_F5,-8, NOTE_E5,-8, NOTE_D5,-8,
NOTE_C5,8, NOTE_E4,4, NOTE_E4,8, NOTE_C4,2,

NOTE_E5,8, NOTE_C5,4, NOTE_G4,8, REST,4, NOTE_GS4,4,//repeats from 23
NOTE_A4,8, NOTE_F5,4, NOTE_F5,8, NOTE_A4,2,
NOTE_D5,-8, NOTE_A5,-8, NOTE_A5,-8, NOTE_A5,-8, NOTE_G5,-8, NOTE_F5,-8,

NOTE_E5,8, NOTE_C5,4, NOTE_A4,8, NOTE_G4,2, //26
NOTE_E5,8, NOTE_C5,4, NOTE_G4,8, REST,4, NOTE_GS4,4,
NOTE_A4,8, NOTE_F5,4, NOTE_F5,8, NOTE_A4,2,
NOTE_B4,8, NOTE_F5,4, NOTE_F5,8, NOTE_F5,-8, NOTE_E5,-8, NOTE_D5,-8,
NOTE_C5,8, NOTE_E4,4, NOTE_E4,8, NOTE_C4,2,
NOTE_C5,8, NOTE_C5,4, NOTE_C5,8, REST,8, NOTE_C5,8, NOTE_D5,8, NOTE_E5,8,
REST,1,

NOTE_C5,8, NOTE_C5,4, NOTE_C5,8, REST,8, NOTE_C5,8, NOTE_D5,4, //33
NOTE_E5,8, NOTE_C5,4, NOTE_A4,8, NOTE_G4,2,
```

```
  NOTE_E5,8, NOTE_E5,8, REST,8, NOTE_E5,8, REST,8, NOTE_C5,8, NOTE_E5,4,
  NOTE_G5,4, REST,4, NOTE_G4,4, REST,4,
  NOTE_E5,8, NOTE_C5,4, NOTE_G4,8, REST,4, NOTE_GS4,4,
  NOTE_A4,8, NOTE_F5,4, NOTE_F5,8, NOTE_A4,2,
  NOTE_D5,-8, NOTE_A5,-8, NOTE_A5,-8, NOTE_A5,-8, NOTE_G5,-8, NOTE_F5,-8,

  NOTE_E5,8, NOTE_C5,4, NOTE_A4,8, NOTE_G4,2, //40
  NOTE_E5,8, NOTE_C5,4, NOTE_G4,8, REST,4, NOTE_GS4,4,
  NOTE_A4,8, NOTE_F5,4, NOTE_F5,8, NOTE_A4,2,
  NOTE_B4,8, NOTE_F5,4, NOTE_F5,8, NOTE_F5,-8, NOTE_E5,-8, NOTE_D5,-8,
  NOTE_C5,8, NOTE_E4,4, NOTE_E4,8, NOTE_C4,2,

  //game over sound
  NOTE_C5,-4, NOTE_G4,-4, NOTE_E4,4, //45
  NOTE_A4,-8, NOTE_B4,-8, NOTE_A4,-8, NOTE_GS4,-8, NOTE_AS4,-8, NOTE_GS4,-8,
  NOTE_G4,8, NOTE_D4,8, NOTE_E4,-2,

};

int smTempo = 280;
int smNotes = sizeof(smMelody) / sizeof(smMelody[0]) / 2;
int smWholenote = (60000 * 4) / smTempo;
int smDivider = 0, smNoteDuration = 0;
int smPlay = 0;

int mcMelody[] = {

  NOTE_C5,4, //1
  NOTE_F5,4, NOTE_F5,8, NOTE_G5,8, NOTE_F5,8, NOTE_E5,8,
  NOTE_D5,4, NOTE_D5,4, NOTE_D5,4,
  NOTE_G5,4, NOTE_G5,8, NOTE_A5,8, NOTE_G5,8, NOTE_F5,8,
  NOTE_E5,4, NOTE_C5,4, NOTE_C5,4,
  NOTE_A5,4, NOTE_A5,8, NOTE_AS5,8, NOTE_A5,8, NOTE_G5,8,
  NOTE_F5,4, NOTE_D5,4, NOTE_C5,8, NOTE_C5,8,
  NOTE_D5,4, NOTE_G5,4, NOTE_E5,4,

  NOTE_F5,2, NOTE_C5,4, //8
  NOTE_F5,4, NOTE_F5,8, NOTE_G5,8, NOTE_F5,8, NOTE_E5,8,
```

```
NOTE_D5,4, NOTE_D5,4, NOTE_D5,4,
NOTE_G5,4, NOTE_G5,8, NOTE_A5,8, NOTE_G5,8, NOTE_F5,8,
NOTE_E5,4, NOTE_C5,4, NOTE_C5,4,
NOTE_A5,4, NOTE_A5,8, NOTE_AS5,8, NOTE_A5,8, NOTE_G5,8,
NOTE_F5,4, NOTE_D5,4, NOTE_C5,8, NOTE_C5,8,
NOTE_D5,4, NOTE_G5,4, NOTE_E5,4,
NOTE_F5,2, NOTE_C5,4,

NOTE_F5,4, NOTE_F5,4, NOTE_F5,4,//17
NOTE_E5,2, NOTE_E5,4,
NOTE_F5,4, NOTE_E5,4, NOTE_D5,4,
NOTE_C5,2, NOTE_A5,4,
NOTE_AS5,4, NOTE_A5,4, NOTE_G5,4,
NOTE_C6,4, NOTE_C5,4, NOTE_C5,8, NOTE_C5,8,
NOTE_D5,4, NOTE_G5,4, NOTE_E5,4,
NOTE_F5,2, NOTE_C5,4,
NOTE_F5,4, NOTE_F5,8, NOTE_G5,8, NOTE_F5,8, NOTE_E5,8,
NOTE_D5,4, NOTE_D5,4, NOTE_D5,4,

NOTE_G5,4, NOTE_G5,8, NOTE_A5,8, NOTE_G5,8, NOTE_F5,8, //27
NOTE_E5,4, NOTE_C5,4, NOTE_C5,4,
NOTE_A5,4, NOTE_A5,8, NOTE_AS5,8, NOTE_A5,8, NOTE_G5,8,
NOTE_F5,4, NOTE_D5,4, NOTE_C5,8, NOTE_C5,8,
NOTE_D5,4, NOTE_G5,4, NOTE_E5,4,
NOTE_F5,2, NOTE_C5,4,
NOTE_F5,4, NOTE_F5,4, NOTE_F5,4,
NOTE_E5,2, NOTE_E5,4,
NOTE_F5,4, NOTE_E5,4, NOTE_D5,4,

NOTE_C5,2, NOTE_A5,4,//36
NOTE_AS5,4, NOTE_A5,4, NOTE_G5,4,
NOTE_C6,4, NOTE_C5,4, NOTE_C5,8, NOTE_C5,8,
NOTE_D5,4, NOTE_G5,4, NOTE_E5,4,
NOTE_F5,2, NOTE_C5,4,
NOTE_F5,4, NOTE_F5,8, NOTE_G5,8, NOTE_F5,8, NOTE_E5,8,
NOTE_D5,4, NOTE_D5,4, NOTE_D5,4,
NOTE_G5,4, NOTE_G5,8, NOTE_A5,8, NOTE_G5,8, NOTE_F5,8,
NOTE_E5,4, NOTE_C5,4, NOTE_C5,4,
```

```
  NOTE_A5,4, NOTE_A5,8, NOTE_AS5,8, NOTE_A5,8, NOTE_G5,8,//45
  NOTE_F5,4, NOTE_D5,4, NOTE_C5,8, NOTE_C5,8,
  NOTE_D5,4, NOTE_G5,4, NOTE_E5,4,
  NOTE_F5,2, NOTE_C5,4,
  NOTE_F5,4, NOTE_F5,8, NOTE_G5,8, NOTE_F5,8, NOTE_E5,8,
  NOTE_D5,4, NOTE_D5,4, NOTE_D5,4,
  NOTE_G5,4, NOTE_G5,8, NOTE_A5,8, NOTE_G5,8, NOTE_F5,8,
  NOTE_E5,4, NOTE_C5,4, NOTE_C5,4,

  NOTE_A5,4, NOTE_A5,8, NOTE_AS5,8, NOTE_A5,8, NOTE_G5,8, //53
  NOTE_F5,4, NOTE_D5,4, NOTE_C5,8, NOTE_C5,8,
  NOTE_D5,4, NOTE_G5,4, NOTE_E5,4,
  NOTE_F5,2, REST,4
};

int mcTempo = 190;
int mcNotes = sizeof(mcMelody) / sizeof(mcMelody[0]) / 2;
int mcWholenote = (60000 * 4) / mcTempo;
int mcDivider = 0, mcNoteDuration = 0;
int mcPlay = 0;

//#include "Tone_sounds_library.h"

ESP32Encoder encoder;

const int trigPin = 5;
const int echoPin = 18;
const int disThreshold = 20;
const int buzzer = 21; //buzzer to arduino pin 9
const int fireThreshold = 25;

long duration;
float distanceCm;
//Control Vehicle
int state = 0; //Stop State
int _speed = 0;
int val = 0;
bool turnRight = false;
```

```
bool turnLeft = false;
bool uTurn = false;
bool shutUp = false;
bool fireWhileStop = false;

//Setup interrupt variables ----------------------------
volatile int count = 0; // encoder count
volatile bool deltaT0 = false;     // check timer interrupt 0
volatile bool deltaT1 = false;     // check timer interrupt 1
volatile bool deltaT2 = false;     // check timer interrupt 1
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;
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;
volatile bool buttonIsPressed = false;

// setting PWM properties ---------------------------
const int freq = 5000;
const int ledChannel_1 = 5;
const int ledChannel_2 = 2;
const int ledChannel_3 = 3;
const int ledChannel_4 = 4;
const int pwmChannel = 0;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 200; //180 ok with return
const int NOM_PWM_VOLTAGE = 150;

LSM6DSO myIMU;

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

//Initialization -----------------------------------
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
```

```
  deltaT0 = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  deltaT1 = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR onTime2() {
  portENTER_CRITICAL_ISR(&timerMux2);
  deltaT2 = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux2);
}

// Timer for turning
void TimerInterruptInit0() {
  timer0 = timerBegin(0, 80, true);  // timer 1, 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, 1200000, true); // 1200000 * 1 us = 1200 ms = 1.2 s, autoreload true
  timerAlarmEnable(timer0); // enable
}

// Timer for U turn
void TimerInterruptInit1() {
  timer1 = timerBegin(1, 80, true);  // timer 1, 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, 2400000, true); // 2400000 * 1 us = 2400 ms = 2.4 s, autoreload true
  timerAlarmEnable(timer1); // enable
}

// Timer for Asking Help
void TimerInterruptInit2() {
  timer2 = timerBegin(2, 80, true);  // timer 1, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer2, &onTime2, true); // edge (not level) triggered
  timerAlarmWrite(timer2, 12000000, true); // 12000000 * 1 us = 12000 ms = 12 s, autoreload true
  timerAlarmEnable(timer2); // enable
```

```
}

void setup() {
  Serial.begin(115200); // Starts the serial communication
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  pinMode(BTN, INPUT);  // configures the specified pin to behave either as an input or an output
  pinMode(POT, INPUT);
  pinMode(L_LED, OUTPUT);
  pinMode(R_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  attachInterrupt(BTN, isr, RISING);
  Wire.begin();
  myIMU.begin();
  myIMU.initialize(BASIC_SETTINGS);

  ESP32Encoder::useInternalWeakPullResistors = 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

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

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);
  ledcAttachPin(LEFT_TRACK_1, ledChannel_3);
  ledcAttachPin(LEFT_TRACK_2, ledChannel_4);
  // speaker
  ledcAttachPin(SPK, pwmChannel);

  TimerInterruptInit0();
```

```
  timerStop(timer0);
  TimerInterruptInit1();
  timerStop(timer1);
  TimerInterruptInit2();
  timerStop(timer2);
}

void loop() {
  delay(50);
  Serial.println(myIMU.readTempC());
  val = analogRead(POT);
  switch (state) {
    case 0:                          //Stop
      ledcWrite(ledChannel_1, LOW);
      ledcWrite(ledChannel_2, LOW);
      ledcWrite(ledChannel_3, LOW);
      ledcWrite(ledChannel_4, LOW);
      Serial.println("State 0: idling with button pressed,");
      if ( val >= 3000 ) {
        if ( mcPlay == mcNotes * 2 ) {
          mcPlay = 0;
        }

        mcDivider = mcMelody[mcPlay + 1];
        if (mcDivider > 0) {
          // regular note, just proceed
          mcNoteDuration = (mcWholenote) / mcDivider;
        } else if (mcDivider < 0) {
          // dotted notes are represented with negative durations!!
          mcNoteDuration = (mcWholenote) / abs(mcDivider);
          mcNoteDuration *= 1.5; // increases the duration in half for dotted notes
        }
        // we only play the note for 90% of the duration, leaving 10% as a pause
        tone(buzzer, mcMelody[mcPlay], mcNoteDuration * 0.9);

        // Wait for the specief duration before playing the next note.
        delay(mcNoteDuration);
```

```
    // stop the waveform generation before the next note.
    noTone(buzzer);

    mcPlay = mcPlay + 2;
  }
  //      Event checker for button press
  if ( myIMU.readTempC() >= fireThreshold && !shutUp ) {
    fireWhileStop = true;
    state = 8;
  }
  if ( myIMU.readTempC() < fireThreshold && shutUp ) {
    shutUp = false;
  }
  if ( CheckForButtonPress() ) {
    //        Service to state 1
    state = 1;
  }
  break;

case 1:                         //Drive Forward
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, MAX_PWM_VOLTAGE);
  ledcWrite(ledChannel_3, LOW);
  ledcWrite(ledChannel_4, MAX_PWM_VOLTAGE);
  Serial.println("State 1: Vehicle is Moving Forward,");
  digitalWrite(GREEN_LED, HIGH);
  if ( val <= 1500 ) {
    if ( imPlay == imNotes * 2 ) {
      imPlay = 0;
    }

    imDivider = imMelody[imPlay + 1];
    if (imDivider > 0) {
      // regular note, just proceed
      imNoteDuration = (imWholenote) / imDivider;
    } else if (imDivider < 0) {
      // dotted notes are represented with negative durations!!
      imNoteDuration = (imWholenote) / abs(imDivider);
```

```
      imNoteDuration *= 1.5; // increases the duration in half for dotted notes
    }
    // we only play the note for 90% of the duration, leaving 10% as a pause
    tone(buzzer, imMelody[imPlay], imNoteDuration * 0.9);

    // Wait for the specief duration before playing the next note.
    delay(imNoteDuration);

    // stop the waveform generation before the next note.
    noTone(buzzer);

    imPlay = imPlay + 2;
  }
  else if ( val < 3000 && val > 1500 ) {
    if ( smPlay == smNotes * 2 ) {
      smPlay = 0;
    }

    smDivider = smMelody[smPlay + 1];
    if (smDivider > 0) {
      // regular note, just proceed
      smNoteDuration = (smWholenote) / smDivider;
    } else if (smDivider < 0) {
      // dotted notes are represented with negative durations!!
      smNoteDuration = (smWholenote) / abs(smDivider);
      smNoteDuration *= 1.5; // increases the duration in half for dotted notes
    }
    // we only play the note for 90% of the duration, leaving 10% as a pause
    tone(buzzer, smMelody[smPlay], smNoteDuration * 0.9);

    // Wait for the specief duration before playing the next note.
    delay(smNoteDuration);

    // stop the waveform generation before the next note.
    noTone(buzzer);

    smPlay = smPlay + 2;
  }
```

```
    //      Event checker for distance
    if ( myIMU.readTempC() >= fireThreshold ) {
      TimeOut0();
      sound_off();
      digitalWrite(GREEN_LED, LOW);
      state = 8;
    }
    if ( CheckForDistance() ) {
      //        Service to state 2 and turning Timer start
      timerStart(timer0);
      digitalWrite(GREEN_LED, LOW);
      state = 2;
    }
    if ( deltaT2 ) {
      TimeOut2();
    }
    //        Event checker for button press
    if ( CheckForButtonPress() ) {
      //        Service to state 0 and turning Timer stop
      TimeOut0();
      sound_off();
      digitalWrite(GREEN_LED, LOW);
      state = 0;
    }
    break;

  case 2:                         //LEFT Turn
    ledcWrite(ledChannel_1, MAX_PWM_VOLTAGE);
    ledcWrite(ledChannel_2, LOW);
    ledcWrite(ledChannel_3, LOW);
    ledcWrite(ledChannel_4, MAX_PWM_VOLTAGE);
    digitalWrite(L_LED, HIGH);
    soundR_on();
    delay(100);
    digitalWrite(L_LED, LOW);
    sound_off();
    delay(100);
    Serial.println("State 2: Vehicle Detected Obstacles & Turning Right.");
```

```
    if ( myIMU.readTempC() >= fireThreshold ) {
      TimeOut0();
      sound_off();
      state = 8;
    }
    //        Event checker for turning time
    if ( deltaT0 ) {
      //        Service to state 3 and turning timer stop and restart
      TimeOut0();
      state = 3;
      timerStart(timer0);
    }
    //        Event checker for distance
    if ( CheckForDistance() == false ) {
      //        Service to state 1 and turning timer stop
      TimeOut0();
      sound_off();
      state = 1;
    }
    //        Event checker for buttom press
    if ( CheckForButtonPress() ) {
      //        Service to state 0 and turning timer stop
      TimeOut0();
      sound_off();
      state = 0;
    }
    break;

  case 3:                       //LeftBack Turn
    //Since we can't find a space for the vehicle to pass through when turing to the right,
    //We don't need to check distance when turning back to the origianl orientation.
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, MAX_PWM_VOLTAGE);
    ledcWrite(ledChannel_3, MAX_PWM_VOLTAGE);
    ledcWrite(ledChannel_4, LOW);
    digitalWrite(R_LED, HIGH);
    soundRB_on();
    delay(100);
```

```
    digitalWrite(R_LED, LOW);
    sound_off();
    delay(100);
    Serial.println("State 3: Vehicle Turning Back to Original Pos after Right Turn Reached 90 degrees,");
    if ( myIMU.readTempC() >= fireThreshold ) {
      TimeOut0();
      sound_off();
      state = 8;
    }
    //        Event checker for turning time
    if ( deltaT0 ) {
      //        Service to state 4, turing timer stop and restart
      TimeOut0();
      state = 4;
      timerStart(timer0);
    }
    //        Event checker for button press
    if ( CheckForButtonPress() ) {
      //        Service to state 0, turning timer stop
      TimeOut0();
      sound_off();
      state = 0;
    }
    break;

  case 4:                       //Right Turn
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, MAX_PWM_VOLTAGE);
    ledcWrite(ledChannel_3, MAX_PWM_VOLTAGE);
    ledcWrite(ledChannel_4, LOW);
    digitalWrite(R_LED, HIGH);
    soundL_on();
    delay(100);
    digitalWrite(R_LED, LOW);
    sound_off();
    delay(100);
    Serial.println("State 4: Vehicle Detected Obstacles & Turning Left,");
    if ( myIMU.readTempC() >= fireThreshold ) {
```

```
      TimeOut0();
      sound_off();
      state = 8;
    }
    //        Event checker for turning time
    if ( deltaT0 ) {
      //        Service to state 5, tunring timer stop and restart
      TimeOut0();
      state = 5;
      timerStart(timer0);
    }
    //        Event checker for distance
    if ( CheckForDistance() == false ) {
      //        Service to state 1 and turning timer stop
      TimeOut0();
      sound_off();
      state = 1;
    }
    //        Event checker for button press
    if ( CheckForButtonPress() ) {
      //       Service to state 0 and turning timer stop
      TimeOut0();
      sound_off();
      state = 0;
    }
    break;

  case 5:                       //RightBack Turn
    ledcWrite(ledChannel_1, MAX_PWM_VOLTAGE);
    ledcWrite(ledChannel_2, LOW);
    ledcWrite(ledChannel_3, LOW);
    ledcWrite(ledChannel_4, MAX_PWM_VOLTAGE);
    digitalWrite(L_LED, HIGH);
    soundLB_on();
    delay(100);
    digitalWrite(L_LED, LOW);
    sound_off();
    delay(100);
```

```
    Serial.println("State 5: Vehicle Turning Back to Original Pos after Left Turn Reached 90 degrees,");
    if ( myIMU.readTempC() >= fireThreshold ) {
      TimeOut0();
      sound_off();
      state = 8;
    }
    //      Event checker for turning time
    if ( deltaT0 ) {
      //        Service to state 6, turing timer stop and restart
      TimeOut0();
      state = 6;
      timerStart(timer1);
      timerStart(timer2);
    }
    //      Event checker for button press
    if ( CheckForButtonPress() ) {
      //        Service to state 0 and turning timer stop
      TimeOut0();
      sound_off();
      state = 0;
    }
    break;

  case 6:                       //U Turn
    ledcWrite(ledChannel_1, MAX_PWM_VOLTAGE);
    ledcWrite(ledChannel_2, LOW);
    ledcWrite(ledChannel_3, LOW);
    ledcWrite(ledChannel_4, MAX_PWM_VOLTAGE);
    digitalWrite(L_LED, HIGH);
    digitalWrite(R_LED, HIGH);
    soundL_on();
    delay(100);
    digitalWrite(L_LED, LOW);
    digitalWrite(R_LED, LOW);
    soundR_on();
    delay(100);
    Serial.println("State 6: Vehicle is Making U Turn,");
    if ( myIMU.readTempC() >= fireThreshold ) {
```

```
      TimeOut1();
      TimeOut2();
      sound_off();
      state = 8;
    }
    //      Event checker for turning time
    if ( deltaT2 ) {
      TimeOut1();
      TimeOut2();
      sound_off();
      state = 7;
    }
    if ( deltaT1 ) {
      //        Service to state 1 and U turn timer stop
      TimeOut1();
      sound_off();
      state = 1;
    }
    //      Event Checker for buttom press
    if ( CheckForButtonPress() ) {
      //        Service to state 0 and U turn timer stop
      TimeOut1();
      sound_off();
      state = 0;
    }
    break;

  case 7:                       //Waiting For Help
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, LOW);
    ledcWrite(ledChannel_3, LOW);
    ledcWrite(ledChannel_4, LOW);
    Serial.println("State 7: Can't escape! Please Help!,");
    digitalWrite(RED_LED, HIGH);
    if ( nokiaPlay == nokiaNotes * 2 ) {
      nokiaPlay = 0;
    }
```

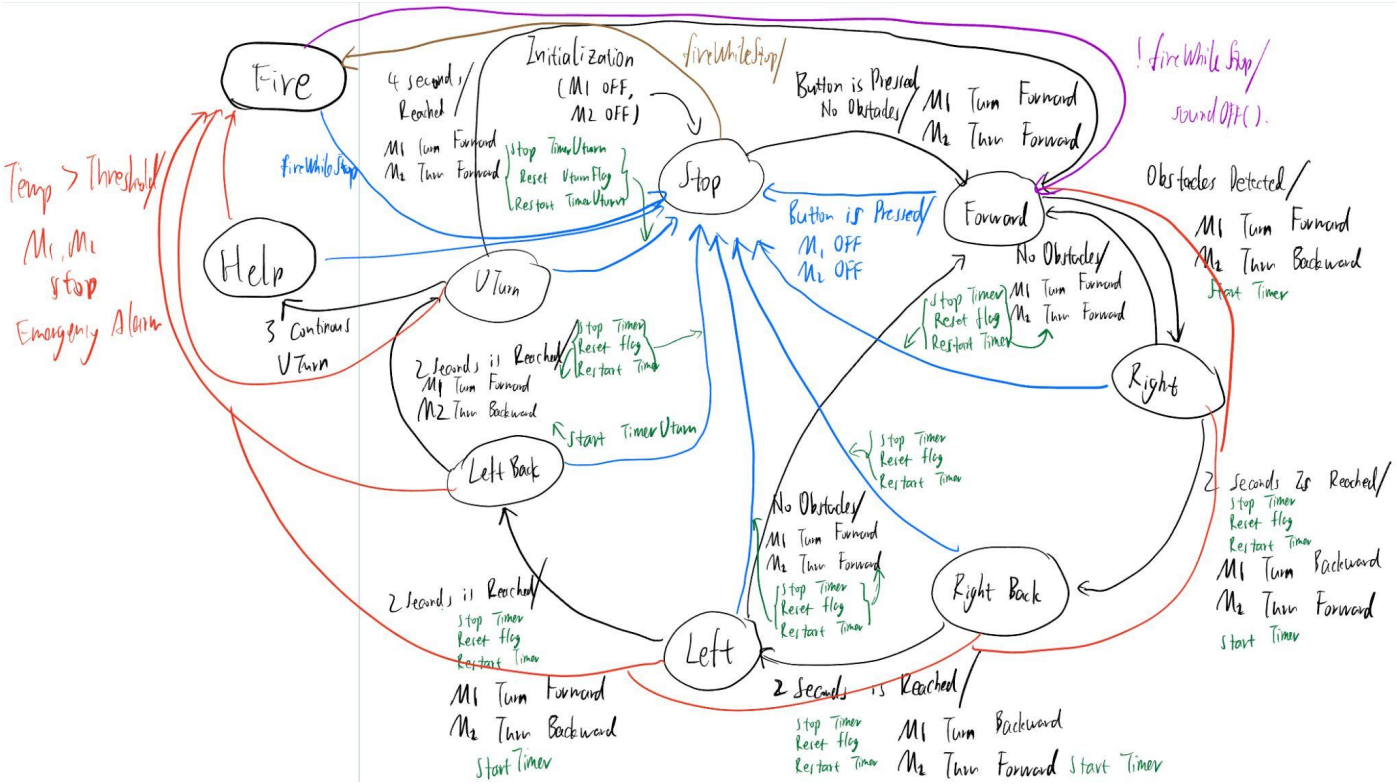
```
    nokiaDivider = nokiaMelody[nokiaPlay + 1];
    if (nokiaDivider > 0) {
      // regular note, just proceed
      nokiaNoteDuration = (nokiaWholenote) / nokiaDivider;
    } else if (nokiaDivider < 0) {
      // dotted notes are represented with negative durations!!
      nokiaNoteDuration = (nokiaWholenote) / abs(nokiaDivider);
      nokiaNoteDuration *= 1.5; // increases the duration in half for dotted notes
    }
    // we only play the note for 90% of the duration, leaving 10% as a pause
    tone(buzzer, nokiaMelody[nokiaPlay], nokiaNoteDuration * 0.9);

    // Wait for the specief duration before playing the next note.
    delay(nokiaNoteDuration);

    // stop the waveform generation before the next note.
    noTone(buzzer);

    nokiaPlay = nokiaPlay + 2;

    if ( myIMU.readTempC() >= fireThreshold ) {
      sound_off();
      digitalWrite(RED_LED, LOW);
      state = 8;
    }
    if ( CheckForButtonPress() ) {
      //        Service to state 0 and U turn timer stop
      sound_off();
      digitalWrite(RED_LED, LOW);
      state = 0;
    }
    break;

  case 8:                         //There is a Fire!!!!
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, LOW);
    ledcWrite(ledChannel_3, LOW);
    ledcWrite(ledChannel_4, LOW);
```

```
      Serial.println("State 8: There is a Fire!!!!");
      digitalWrite(R_LED, LOW);
      digitalWrite(L_LED, LOW);
      digitalWrite(RED_LED, HIGH);
      soundL_on();
      delay(500);
      digitalWrite(RED_LED, LOW);
      digitalWrite(R_LED, HIGH);
      digitalWrite(L_LED, HIGH);
      soundR_on();
      delay(500);
      if ( myIMU.readTempC() < fireThreshold && fireWhileStop ) {
        sound_off();
        digitalWrite(R_LED, LOW);
        digitalWrite(L_LED, LOW);
        fireWhileStop = false;
        state = 0;
      }
      else if ( myIMU.readTempC() < fireThreshold ) {
        sound_off();
        digitalWrite(R_LED, LOW);
        digitalWrite(L_LED, LOW);
        state = 1;
      }
      if ( CheckForButtonPress() ) {
        //        Service to state 0 and U turn timer stop
        sound_off();
        shutUp = true;
        digitalWrite(R_LED, LOW);
        digitalWrite(L_LED, LOW);
        state = 0;
      }
      break;


  }
```

```
}

// Event checker 1 "For button Press"
bool CheckForButtonPress() {
  if (buttonIsPressed == true) {
    buttonIsPressed = false;
    return true;
  }
  return false;
}

// Event checker 2 "Check for distance"
bool CheckForDistance() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance
  distanceCm = duration * SOUND_SPEED / 2;

  // Prints the distance in the Serial Monitor
  Serial.print("Distance (cm): ");
  Serial.println(distanceCm);

  if (distanceCm <= disThreshold ) {
    return true;
  }
  return false;
}
```

```
void soundR_on() {
  ledcAttachPin(SPK, pwmChannel);
  ledcWriteTone(pwmChannel, 600);
}

void soundL_on() {
  ledcAttachPin(SPK, pwmChannel);
  ledcWriteTone(pwmChannel, 800);
}

void soundRB_on() {
  ledcAttachPin(SPK, pwmChannel);
  ledcWriteTone(pwmChannel, 800);
}

void soundLB_on() {
  ledcAttachPin(SPK, pwmChannel);
  ledcWriteTone(pwmChannel, 600);
}

void sound_off() {
  ledcDetachPin(SPK);
  ledcWriteTone(pwmChannel, 0);
}

// Timer for turning/change directions
void TimeOut0() {
  portENTER_CRITICAL(&timerMux0);
  deltaT0 = false; // reset interruptCounter flag to false
  portEXIT_CRITICAL(&timerMux0);
  timerStop(timer0);
  timerRestart(timer0);
}

// Timer for U turn
void TimeOut1() {
  portENTER_CRITICAL(&timerMux1);
  deltaT1 = false; // reset interruptCounter flag to false
  portEXIT_CRITICAL(&timerMux1);
  timerStop(timer1);
  timerRestart(timer1);
}

void TimeOut2() {
  portENTER_CRITICAL(&timerMux2);
  deltaT2 = false; // reset interruptCounter flag to false
  portEXIT_CRITICAL(&timerMux2);
  timerStop(timer2);
  timerRestart(timer2);
}
```

-- Event-driven program --



The Blue line indicates “Button is Pressed/ M1 Turn OFF, M2 Turn OFF, SoundOff()”.

The green part applies to the black part and blue part.

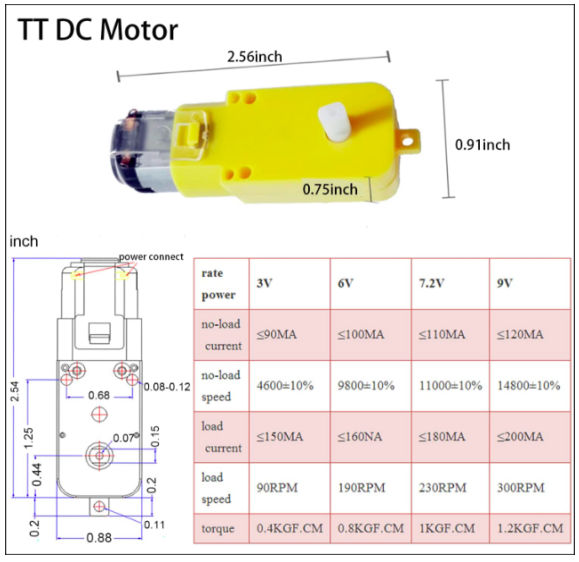
-- Bill Of Material --

Source Link	Part Name	Vendor	Quantity	Price
https://www.amazon.com/130MOTOR-DC3V-6C-Motor-Smart-Robot/dp/B0B8S6B1WR/ref=sr_1_13?crid=PC4PPY346FKK&keywords=TT+Motor&qid=1666314519&qu=eyJxc2MiOiI0Ljc2IiwicXNhIjoiNC42MiIsInFzcCI6IjQuMzkifQ%3D%3D&s=toys-and-games&sprefix=tt+motor+%2Ctoys-and-games%2C248&sr=1-13	TT Motor w. Coupling - LYH-DZZW8507-2pcs	Amazon	x2	$4.99

https://www.mcmaster.com/91290A130/	Alloy Steel Socket Head Screw M3 30mm - 91290A130	McMaster	x4	$13.16
https://www.mcmaster.com/90591A121/	Zinc-Plated Steel Hex Nut M3 - 90591A121	McMaster	x4	$1.90
https://www.mcmaster.com/94209A356	M3 8 screw	McMaster	x4	$8.00
https://www.mcmaster.com/92503A180/	Mil. Spec. shim - 92503A180	McMaster	x8	$12.40
https://www.amazon.com/Redrex-Printer-Plastic-Bearing-Tarantula/dp/B07Q5WN3GK/ref=asc_df_B07Q5WN3GK/?tag=hyprod-20&linkCode=df0&hvadid=509393708826&hvpos=&hvnetw=g&hvrand=4393676780055879849&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9032083&hvtargid=pla-732625003384&psc=1	Ball wheels - 3R0DXD7T7	Amazon	x2	$12.99
https://www.mcmaster.com/91290A196/	Alloy Steel Socket Head Screws - 91290A196	McMaster	x4	$7.78
https://www.mcmaster.com/90576A104/	Metric Medium-Strength Steel Nylon-Insert Locknuts—Class 8 - 90576A104	McMaster	x4	$6.14
https://www.mcmaster.com/91017A464/	Shims for Lengthening Screw Shoulders - 91017A464	McMaster	x4	$11.14
https://www.mcmaster.com/91290A137/	Alloy Steel Socket Head Screw M3 - 91290A137	McMaster	x2	$5.83
https://www.mcmaster.com/90592A085/	Steel Hex Nut M3 - 90592A085	McMaster	x4	$2.62
https://www.mcmaster.	Steel Hex Nut M2 - 90592A075	McMaster	x4	$4.00

com/90592A075/				
https://www.mcmaster.com/91292A032/	18-8 Stainless Steel Socket Head Screw M2 - 91292A032	McMaster	x4	$6.21
https://www.mcmaster.com/91292A833/	18-8 Stainless Steel Socket Head Screw M2 - 91292A833	McMaster	x2	$7.91
https://www.aliexpress.us/item/2255800903458655.html?spm=a2g0o.productlist.0.0.478651f6Orz4nl&algo_pvid=25735ead-efdf-4eef-90ee-d84ab85b755c&algo_exp_id=25735ead-efdf-4eef-90ee-d84ab85b755c-41&pdp_ext_f=%7B%22sku_id%22%3A%2210000014306734908%22%7D&pdp_npi=2%40dis%21USD%216.99%216.64%21%21%21%21%21%402101d91e16663894727113852e100b%2110000014306734908%21sea&curPageLogUid=BqEERltklyL5	Plastic Idler	AliExpress	x2	$6.50
https://www.amazon.com/Energizer-Alkaline-Volt-Battery-2-Count/dp/B004R16728/ref=sr_1_6?keywords=9v+battery&qid=1666392683&qu=eyJxc2MiOiI0LjUwIiwicXNhIjoiNC4yNCIsInFzcCI6IjQuMTIifQ%3D%3D&sr=8-6	9V Battery	Amazon	x1	$7.25
https://www.everythingpromo.com/power-bank-portable-battery-charger-2000-mah-aluminium?gclid=Cj0KCQjwh	2000 MAH ALUMINIUM Power Bank	Everything Promo	x1	$3.13

smaBhCvARIsAIbEbH47AmU3rI_wJsl29uPzTQ3qFY7I8F2v_cViLERiqJtb6o8OBL2sDbcaAm3WEALw_wcB				
	1/4" x 18" x 30" Plywood*	Jacob	x1	$6
	ESP32	Lab Kit	x1	
	Ultrasonic Sensor	Lab Kit	x1	
	Buzzer	Lab Kit	x1	
	LSM6DSO Sensor	Lab Kit	x1	
	Potentiometer	Lab Kit	x1	
	H Bridge	Lab Kit	x1	
	LED	Lab Kit	x16	
	Wires	Lab Kit	Several	
	Breadboard	Lab Kit	x1	
	Ultrasonic Sensor & LED Mount	3D Print	x1	



rate power	3V	6V	7.2V	9V
no-load current	≤90MA	≤100MA	≤110MA	≤120MA
no-load speed	4600±10%	9800±10%	11000±10%	14800±10%
load current	≤150MA	≤160NA	≤180MA	≤200MA
load speed	90RPM	190RPM	230RPM	300RPM
torque	0.4KGF.CM	0.8KGF.CM	1KGF.CM	1.2KGF.CM

Motor datasheet