ROBOTICS AND MECHATRONICS 2

ECEA128-2P/E03

PROJECT DOCUMENTATION

Submitted to:

Engr. Glenn V. Magwili

Submitted by:

Gutierrez, Ma. Elenita T.

Honra, James Bernward

Louis, Jervin

Project Description

Autonomy in its most basic form is a go-to-goal conduct. To get to the destination place quickly, the robot must perform a lot of calculations and decisions. Specifically, a go-to-goal robot is a class of autonomous robot that has been programmed to find its way to a certain place or objective. These robots are frequently employed in fields including logistics, warehouse automation, and self-driving cars.

Go-to-goal robots employ sensors like cameras, lidar, or sonar to sense their environment and pinpoint where they are in relation to it. Then, they navigate toward their target using algorithms like path planning and obstacle avoidance. Some goal-directed robots can also learn from their surroundings and modify their behavior over time to perform better.

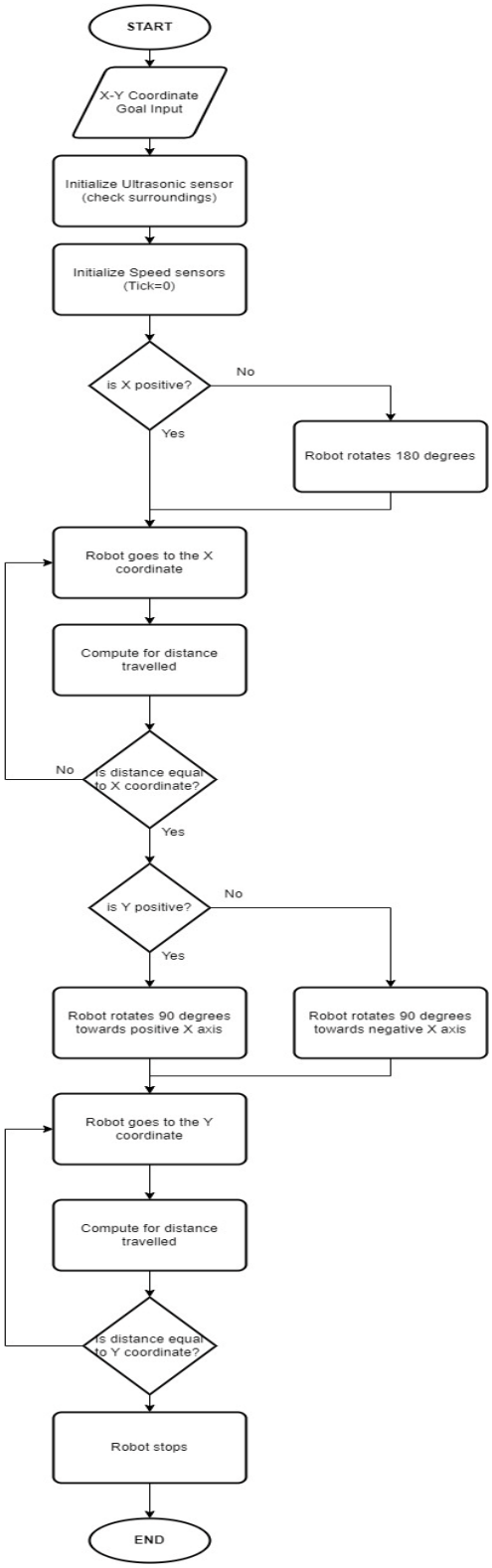
In this project, the sensors used are the ultrasonic sensors and the speed sensors. Particularly, the ultrasonic sensors sense the environment where the robot is placed, and the speed sensor counts the number of ticks to determine how far the robot has travelled. The robot is programmed though the Arduino Microcontroller.

Features

The created go-to-goal have different features that is significant and can be applied to real life applications. These features are the following:

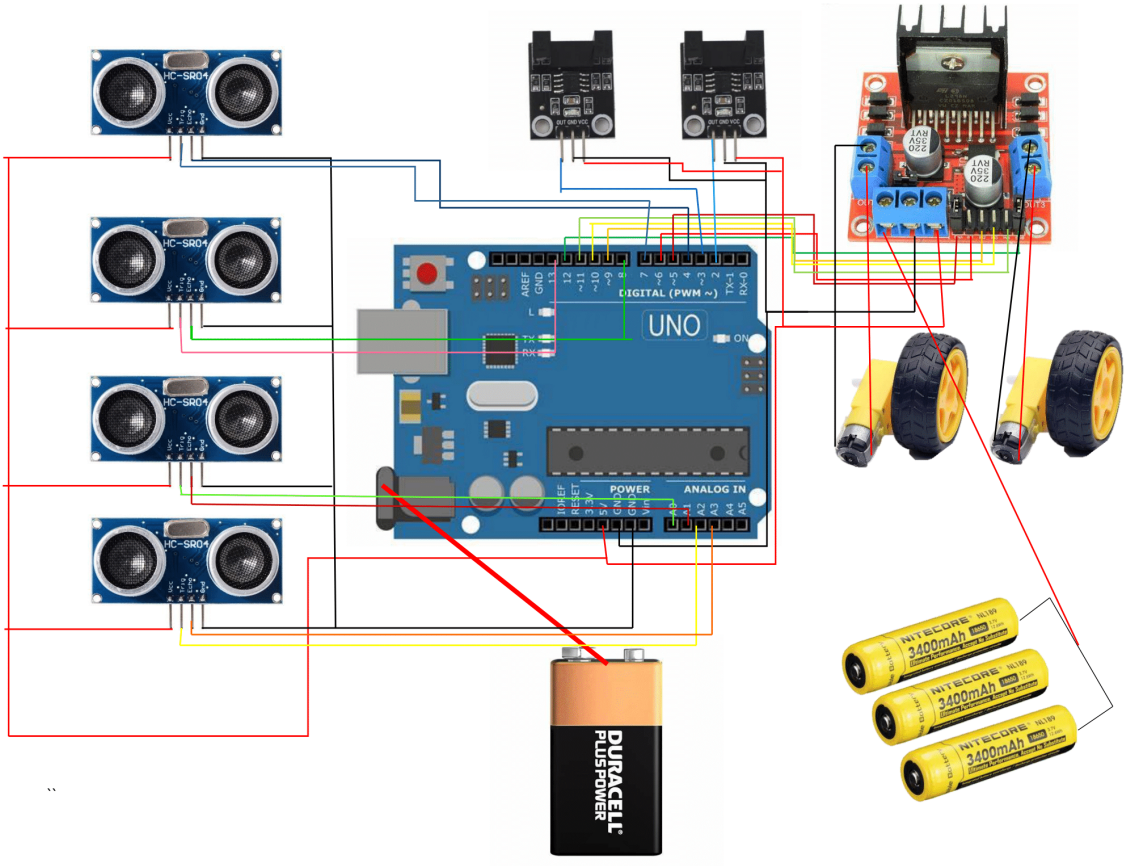
1. Sensing and Perception: A go-to-goal robot must have sensors that can perceive its environment and determine its current location. In this project, the ultrasonic sensor is used.
2. Navigation and Path Planning: A go-to-goal robot must be able to plan a path from its current location to its desired goal. This involves algorithms that is based on the ticks specified in the program.
3. Localization and Mapping: A go-to-goal robot must have the ability to localize itself within its environment. In the robot created, this is achieved by the measurement of the distance travelled.
4. Autonomous Control: A go-to-goal robot must be capable of making decisions and taking actions on its own, which means that it can automatically go to the desired x and y coordinates based on its programmed algorithm.

Block Diagram



The figure above shows the flow of the program from the moment its is uploaded to the board and the robot is turned on. Input of the goal coordinates will be needed for the robot to function along with the initialization of the sensors. The first part will be determining whether the x coordinate is positive or negative in which the robot will response accordingly. The robot will go to the said x coordinate while computing for the distance travelled which then will be compared to the inputted x-coordinate. The robot will stop when the distance travelled is equal to the inputted coordinate. Same process will occur when travelling y axis in which will involve determination whether it is positive or negative with corresponding actions along with the measuring and comparing of distance.

Schematic Diagram/Wiring Diagram

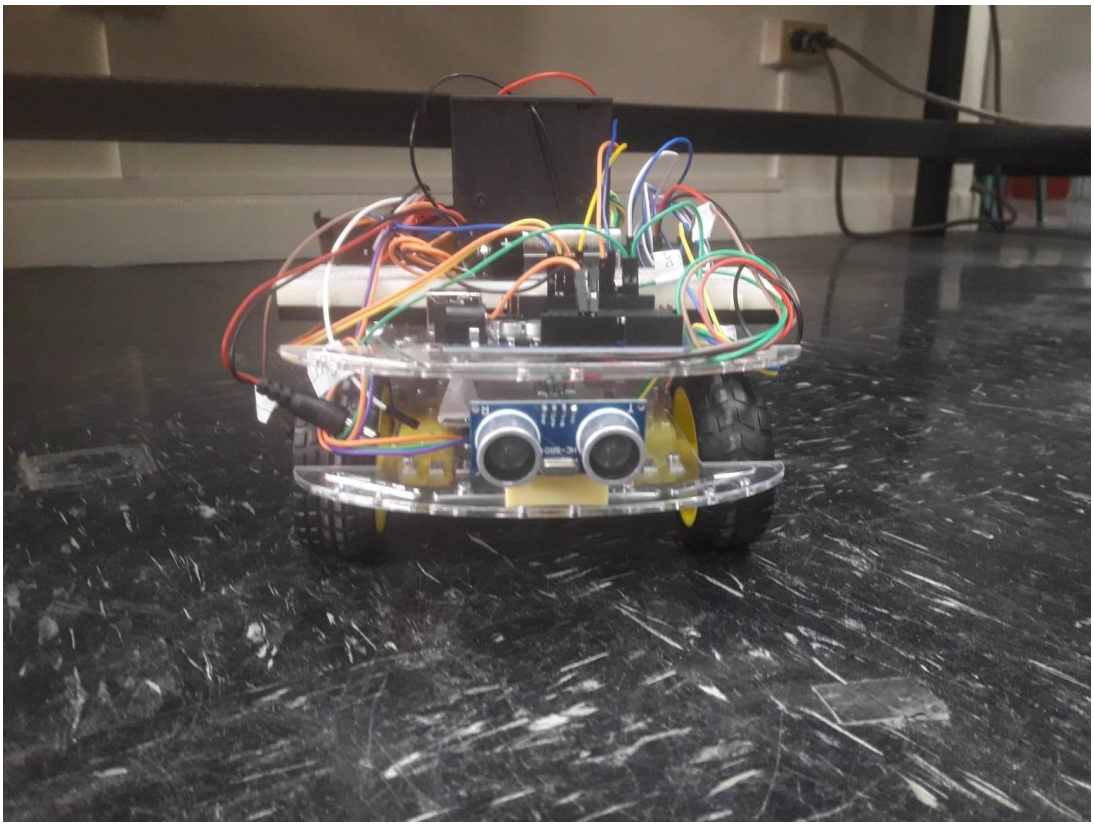


The figure above shows the schematic or wiring diagram of the constructed Go-to-Goal Robot. As seen in the picture, the DC motors are connected to the motor driver and the pins of the motor driver are connected to the Arduino Uno in order to control the speed and direction of the DC motors. The speed sensors are also connected to pin 2 and 3 of the Arduino in order to count the number of ticks to yield the distance travelled by the robot. The ultrasonic sensors are also connected and act as the sensing device to measure the environment the robot has been placed.

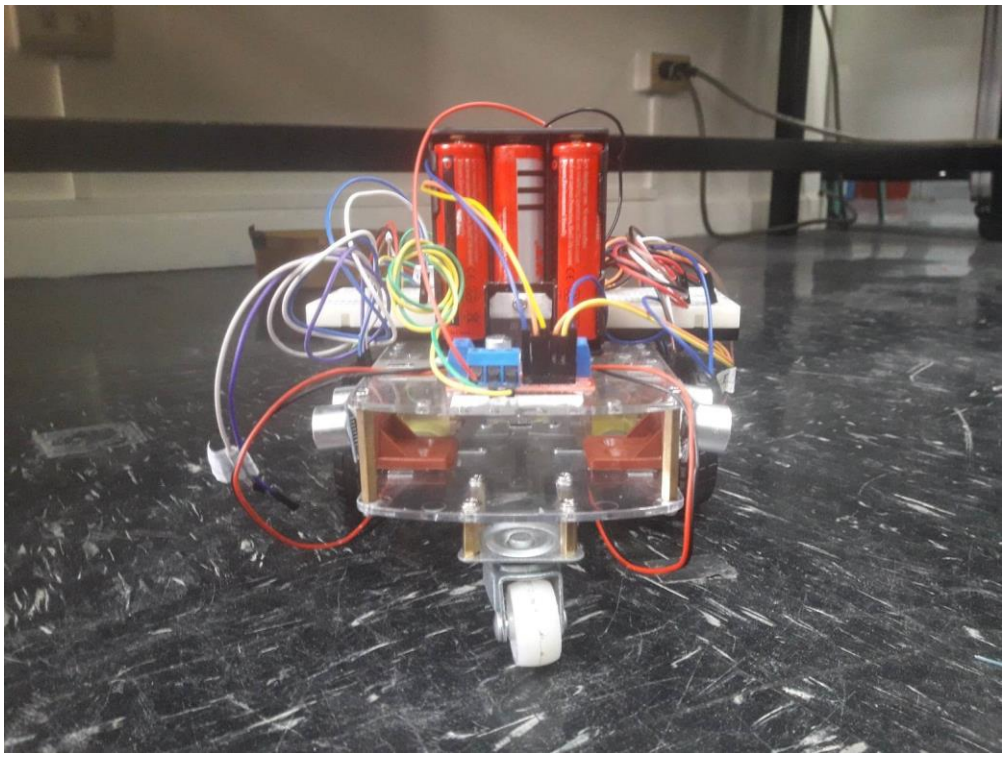
Tabulated Bill of Materials

Material	Quantity	Price (Php)
Arduino Uno	1	350
L298N Motor Driver	1	75
LM393 Speed Sensor	2	198
DC Motor with wheel	2	198
Free Wheel	1	10
Ultrasonic Sensor	4	240
Bracket	4	20
9V Battery	1	200
Lithium Ion Battery	3	210
Breadboard	1	65
Jumper Wires	1 pack	90
Robot Chassis	1	250
Total		1906

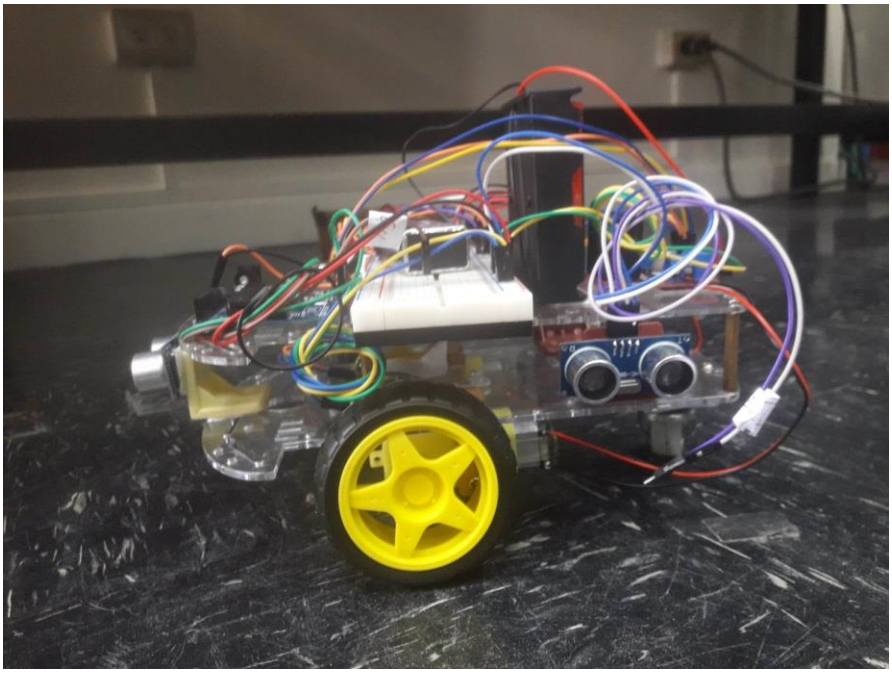
Pictures of Prototype



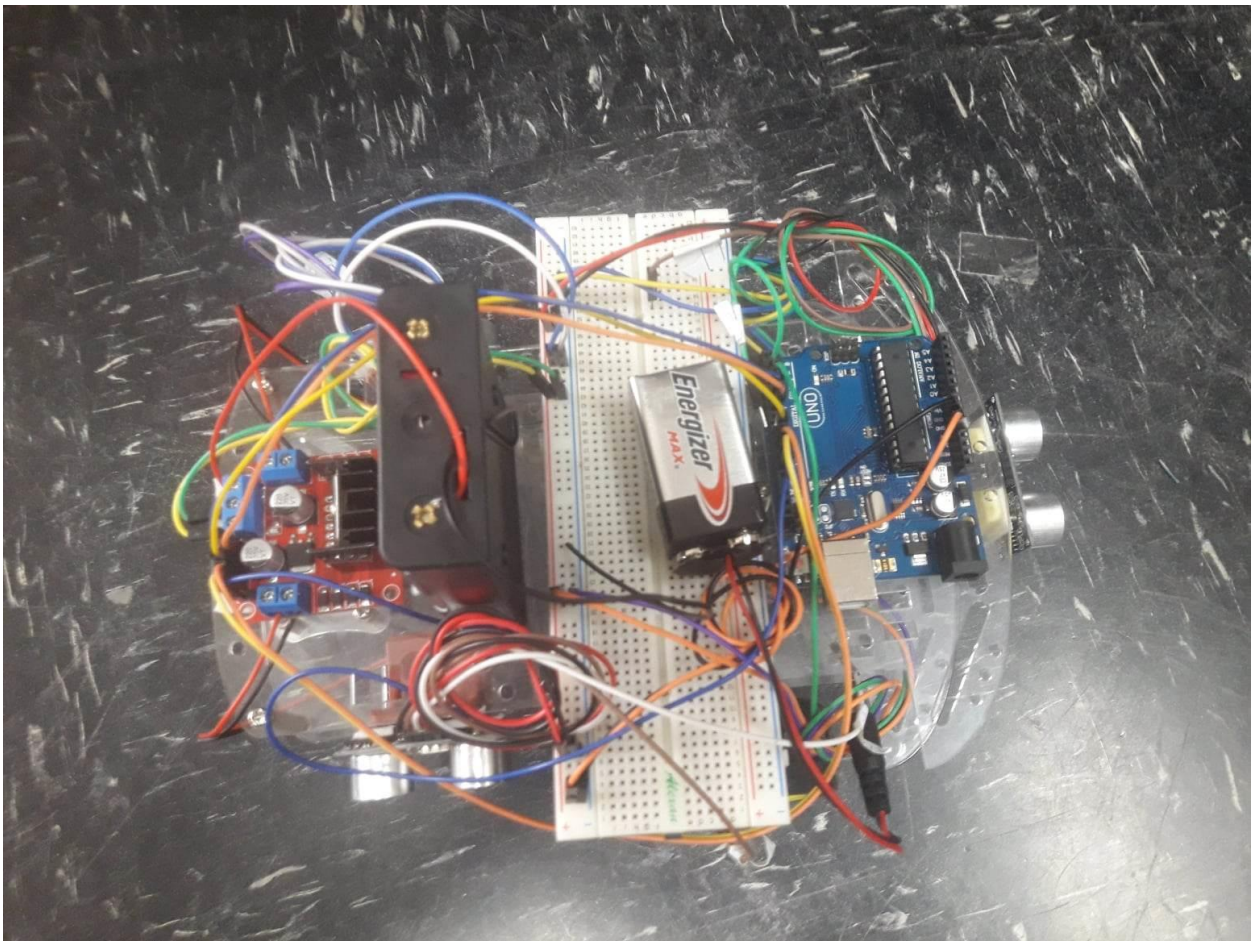
Front View



Back View



Side View



Top View

Calibration Sensors

- Motor Driver

```
// Motor A connections
int enA = 5;
int in1 = 6;
int in2 = 9;
// Motor B connections
int in3 = 10;
int in4 = 11;
int enB = 12;


void setup() {
  // Set all the motor control pins to outputs
  pinMode(enA, OUTPUT);
```

```
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);


  // Turn off motors - Initial state
  analogWrite(enA, 0);
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  analogWrite(enB, 0);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);

}

void loop() {
  directionControl();
  delay(1000);

}

// This function lets you control spinning direction of motors
void directionControl() {
  // Set motors to maximum speed
  // For PWM maximum possible values are 0 to 255
  analogWrite(enA, 255);
  analogWrite(enB, 255);

  // Turn on motor A & B
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);

  delay(2000);

  // Reverse
  analogWrite(enA, 255);
  analogWrite(enB, 255);

  // Turn on motor A & B
```

```
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);

  delay(2000);

  // Turn off motors
  analogWrite(enA, 0);
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  analogWrite(enB, 0);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);

}
```

- Ultrasonic Sensor

```
// Define pins
#define trigPin 9
#define echoPin 10

// Define variables
long duration;
int distance;

void setup() {
  // Initialize serial communication
  Serial.begin(9600);

  // Define pin modes
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Wait for sensor to settle
  delay(1000);
}

void loop() {
  // Send ultrasonic pulse
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
```

```
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Measure duration of echo pulse
  duration = pulseIn(echoPin, HIGH);

  // Calculate distance
  distance = duration * 0.034 / 2;

  // Print distance to serial monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  // Wait for a moment before measuring again
  delay(500);
}
```

- Speed Sensor

```
// Define speed sensor pins
int sensor1 = 2;
int sensor2 = 3;

// Define variables for distance and speed calculations
unsigned int ticks1;
unsigned int ticks2;
double distanceR;
double distanceL;

// Define constants and wheel dimensions
const double pi = 3.14159265358979238463;
int wheelrad = 3.3;
int wheeldist = 14;

void setup()
{
  // Set speed sensor pins as inputs
  pinMode(sensor1, INPUT);
  pinMode(sensor2, INPUT);

  pinMode(sensor1, INPUT_PULLUP);
```

```
  attachInterrupt(digitalPinToInterrupt(sensor1), LeftTick, RISING);
  pinMode(sensor2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(sensor2), RightTick, RISING);

  // Start serial communication
  Serial.begin(9600);

}

void loop()
{

  distanceR = (2*pi*wheelrad) * (ticks2/20);
  distanceL = (2*pi*wheelrad) * (ticks1/20);

}

void LeftTick()
{
  ticks1++;
}

void RightTick()
{
  ticks2++;
}
```

Go-to-Goal Code

```
// Define motor control pins
int enA = 5;
int in1 = 6;
int in2 = 9;
int in3 = 10;
int in4 = 11;
int enB = 12;

// Define speed sensor pins
int sensor1 = 2;
int sensor2 = 3;

// Define variables for distance and speed calculations
unsigned int ticks1;
unsigned int ticks2;
double distanceRx;
double distanceLx;
double distanceRy;
double distanceLy;
double distanceRrotx;
double distanceLrotx;
double distanceRroty;
double distanceLroty;

// Define constants and wheel dimensions
const double pi = 3.141592653589793238463;
int wheelrad = 3.3;
```

```
int wheeldist = 14;
double wheelangleA = 110;
double wheelangleB = 110;
double wheelangleC = 110;
int xstage = 0;
int ystage = 0;
int prexstage = 0;
int preystage = 0;

// Define desired goal
double xf = 100;
int negx = 0;
double yf = 0;
int negy = 0;

void setup() {
  // Set motor control pins as outputs
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(enB, OUTPUT);

  // Set speed sensor pins as inputs
  pinMode(sensor1, INPUT);
  pinMode(sensor2, INPUT);
```

```
  pinMode(sensor1, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(sensor1), LeftTick, RISING);
  pinMode(sensor2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(sensor2), RightTick, RISING);

  // Start serial communication
  Serial.begin(9600);
}

void loop() {

  if(negx == 0 && xstage == 0 && ystage == 0 && preystage == 0)
  {
    distanceRx = (2*pi*wheelrad) * (ticks2/20);
    distanceLx = (2*pi*wheelrad) * (ticks1/20);

    if (distanceRx < xf && distanceLx < xf)
    {
      analogWrite(enA, 255);
      digitalWrite(in1, HIGH);
      digitalWrite(in2, LOW);
      analogWrite(enB, 128);
      digitalWrite(in3, HIGH);
      digitalWrite(in4, LOW);
    }
    if (distanceRx >= xf && distanceLx >= xf)
    {
      analogWrite(enA, 0);
```

```
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    analogWrite(enB, 0);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);

    ticks1=0;
    ticks2=0;
    xstage=1;

    delay(1000);
  }
}

if(negx == 1 && xstage == 0 && ystage == 0 && preystage == 0 && prexstage == 0)
{
  distanceLrotx = (2*pi*wheelrad) * (ticks1/20);


  if(distanceLrotx < wheelangleB)
  {
    analogWrite(enA, 150);
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    analogWrite(enB, 0);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  }
```

```
  if (distanceLrotx >= wheelangleB)
  {
    analogWrite(enA, 0);
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    analogWrite(enB, 0);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);

    ticks1=0;
    ticks2=0;
    prexstage=1;

    delay(1000);


  }
}

  if(negx == 1 && prexstage == 1 && ystage == 0 && preystage == 0)
  {

    distanceRx = (2*pi*wheelrad) * (ticks2/20);
    distanceLx = (2*pi*wheelrad) * (ticks1/20);

    if (distanceRx < xf && distanceLx < xf)
    {
```

```
    analogWrite(enA, 255);
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    analogWrite(enB, 128);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  }
  if (distanceRx >= xf && distanceLx >= xf)
  {
    analogWrite(enA, 0);
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    analogWrite(enB, 0);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);

    ticks1=0;
    ticks2=0;
    xstage=1;

    delay(1000);
  }
}

if (negy == 0 && preystage == 0 && xstage == 1)
{
  distanceRroty = (2*pi*wheelrad) * (ticks2/20);
  distanceLroty = (2*pi*wheelrad) * (ticks1/20);
```

```
  if(distanceLroty < wheelangleA)
  {
   analogWrite(enA, 150);
   digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
  }

  if (distanceLroty >= wheelangleA)
  {
    analogWrite(enA, 0);
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);


    ticks1=0;
    ticks2=0;
    preystage=1;

    delay(1000);
  }
}

 if (negy == 1 && preystage == 0 && xstage == 1)
 {
  distanceLroty = (2*pi*wheelrad) * (ticks1/20);

  if(distanceLroty < wheelangleC)
```

```
  {
   analogWrite(enA, 150);
   digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
  }

  if (distanceLroty >= wheelangleC)
  {
    analogWrite(enA, 0);
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    analogWrite(enB, 0);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);

    ticks1=0;
    ticks2=0;
    preystage=1;

    delay(1000);
  }
}

 if (ystage == 0 && preystage == 1)
 {
   distanceRy = (2*pi*wheelrad) * (ticks2/20);
   distanceLy = (2*pi*wheelrad) * (ticks1/20);
```

```
  if (distanceRy < yf && distanceLy < yf)
  {
    analogWrite(enA, 255);
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    analogWrite(enB, 128);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  }
  if (distanceRy >= yf && distanceLy >= yf)
  {
    analogWrite(enA, 0);
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    analogWrite(enB, 0);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);

    ticks1=0;
    ticks2=0;
    ystage=1;

    delay(1000);
  }
}
```

```
}

// Read sensor 1
  void LeftTick()
  {
    ticks1++;
  }

  // Read sensor 2
  void RightTick()
  {
    ticks2++;
  }
```