زانکۆی سەلاحەدین- هەولێر

Salahaddin University- Erbil

Go to Goal Robot

Research project

Submitted to the department of (Electrical Engineering) in partial fulfilment of the requirements for the degree of B.A in (Electrical Engineering)

By:

Abdulghafar Othman

Saad Mohammed

Nuraddin Kanhan

Khalil Mohammed

Supervised by:

Mr. Ahmed Khalid

Acknowledgment:

Special thanks for our supervisor Mr. Ahmed Khalid. And we also thank our families and colleagues who helped us to create this project.

SUPERVISOR'S CERTIFICATION

I certify that this Project, titled **'Go to Goal Robot'** and presented by 'Abdulghafar Othman, Saad Mohammed, Nuraddin Kanhan and Khaleel Mohammed' was prepared under my supervision at the University of Salahaddin – Erbil as a partial requirement for the degree of Bachelors of Science in Electrical Engineering.

Name: Mr. Ahmed Kh. Ahmed

Signature: 

Date: 19 / 05 / 2019

Abstract:

Our robot is a mobile robot that can work in closed area, the robot has an Arduino Uno that control the sensor and actuators, and has four ultrasonic sensors that uses to find the location of the robot inside the frame, two encoder sensors that work with two ticks and two motors together to find the distance that the robot has taken, the robot designed to find it is location and should go to that point goal that we marked.

List of contents: Page

Chapter 1:

Introduction to Mobile Robot

1.1 Introduction:

Go to goal robot is basically a mobile robot that can traverse and turn, Find its wanted position. A mobile robot needs locomotion mechanisms that enables it move unbounded throughout its environment. There are a large variety of possible ways to move, and so the selection of robot's approach to locomotion is an important aspect of mobile robot design. In laboratory, there are research robots that can walk, jump, run, slide, skate, swim, fly, and, of course, roll. Most of these locomotion mechanisms have been inspired by their bio-logical counterparts.

There is, however, one exception: the actively powered wheel is a human invention that achieves extremely high efficiency on flat ground. This mechanism is not completely foreign to biological systems. Our bipedal walking system can be approximated by a rolling polygon, with sides equal in length d to the span of the step (figure 1.1). As the step size decreases, the polygon approaches a circle or wheel. But nature did not develop a fully rotating, actively powered joint, which is the technology necessary for wheeled locomotion.

Biological systems succeed in moving through a wide variety of harsh environments. Therefore, it can be desirable to copy their selection of locomotion mechanisms. However, replicating nature in this regard is extremely difficult for several reasons. To begin with, mechanical complexity is easily achieved in biological systems through structural replication. Cell division, in combination with specialization, can readily produce a millipede with several hundred legs and several tens of thousands of individually sensed cilia. In man-made structures, each part must be fabricated individually, and so no such economies of scale exist. Additionally, the cell is a microscopic building block that enables extreme miniaturization. With very small size and weight, insects achieve a level of robustness that we have not been able to match with human fabrication techniques. Finally, the biological energy storage system and the muscular and hydraulic activation systems used by large animals and insects achieve torque, response time, and conversion efficiencies that far exceed similarly scaled man-made systems.

Owing to these limitations, mobile robots generally locomotion either using wheeled mechanisms, a well-known human technology for vehicles, or using a small number of articulated legs, the simplest of the biological approaches to locomotion. In general, legged locomotion requires higher degrees of freedom and therefore greater mechanical complexity than wheeled locomotion. Wheels, in addition to being simple, are extremely well suited to flat ground. On flat surfaces wheeled loco-motion is one to two orders of magnitude more efficient than legged locomotion. The rail-way is ideally engineered for wheeled locomotion

because rolling friction is minimized on a hard and flat steel surface. But as the surface becomes soft, wheeled locomotion accumulates inefficiencies due to rolling friction whereas legged locomotion suffers much less because it consists only of point contacts with the ground. This will lead to the dramatic loss of efficiency in the case of a tire on soft ground. [1]

In effect, the efficiency of wheeled locomotion depends greatly on environmental qualities, particularly the flatness and hardness of the ground, while the efficiency of legged locomotion depends on the leg mass and body mass, both of which the robot must support at various points in a legged gait.

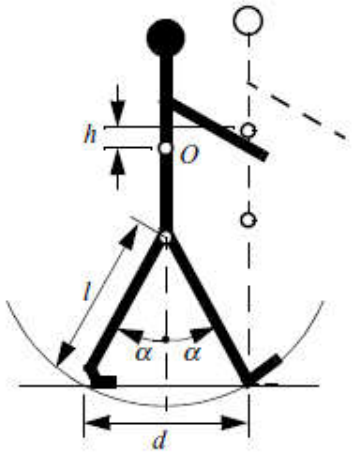

Figure 1.1
A biped walking system can be approximated by a rolling polygon, with sides equal in length *d* to the span of the step. As the step size decreases, the polygon approaches a circle or wheel with the radius *l*.

1.2 Key issues for locomotion:

Locomotion is the complement of manipulation. In manipulation, the robot arm is fixed but moves objects in the workspace by imparting force to them. In locomotion, the environment is fixed and the robot moves by imparting force to the environment. In both cases, the scientific basis is the study of actuators that generate interaction forces, and mechanisms that implement desired kinematic and dynamic properties. Locomotion and manipulation thus share the same core issues of stability, contact characteristics, and environmental type:

• Stability

- Number and geometry of contact points

- Center of gravity

- Static/dynamic stability

- Inclination of terrain

• Characteristics of contact

- contact point/path size and shape

- Angle of contact

- Friction

• Type of environment

- Structure

- Medium, (e.g. water, air, soft or hard ground)

1.3 Wheeled Mobile Robots:

The wheel has been by far the most popular locomotion mechanism in mobile robotics and in man-made vehicles in general. It can achieve very good efficiencies, and does so with a relatively simple mechanical implementation. In addition, balance is not usually a research problem in wheeled robot designs, because wheeled robots are almost always designed so that all wheels are in ground contact at all times. Thus, three wheels are sufficient to guarantee stable balance, although, as we shall see below, two-wheeled robots can also be stable. When more than three wheels are used, a suspension system is required to allow all wheels to maintain ground contact when the robot encounters uneven terrain.

Instead of worrying about balance, wheeled robot research tends to focus on the problems of traction and stability, maneuverability, and control: can the robot wheels provide sufficient traction and stability for the robot to cover all of the desired terrain, and does the robot's wheeled configuration enable sufficient control over the velocity of the robot?

There are four major wheel classes, as shown in figure 1.2. They differ widely in their kinematics, and therefore the choice of wheel type has a large effect on the overall kinematics of the mobile robot. The standard wheel and the castor wheel have a primary axis of rotation and are thus highly directional. To move in a different direction, the wheel must be steered first along a vertical axis. The key difference between these two wheels is that the standard wheel can accomplish this steering motion with no side effects, as the center of rotation passes through the contact patch with the ground, whereas the castor wheel rotates around an offset axis, causing a force to be imparted to the robot chassis during steering. [1]

1.4 Stability:

Surprisingly, the minimum number of wheels required for static stability is two. As shown above, a two-wheel differential-drive robot can achieve static stability if the center of mass is below the wheel axle. Cye is a commercial mobile robot that uses this wheel configuration. However, under ordinary circumstances such a solution requires wheel diameters that are impractically large. Dynamics can also cause a two-wheeled robot to strike the floor with a third point of contact, for instance, with sufficiently high motor torques from stand-still. Conventionally, static stability requires a minimum of three wheels, with the additional caveat that the center of gravity must be contained within the triangle formed by the ground contact points of the wheels. Stability can be further improved by adding more wheels, although once the number of contact points exceeds three, the hyper static nature of the geometry will require some form of flexible suspension on uneven terrain.

1.5 Maneuverability:

Some robots are omnidirectional, meaning that they can move at any time in any direction along the ground plane regardless of the orientation of the robot around its vertical axis. This level of maneuverability requires wheels that can move in more than just one direction, and so omnidirectional robots usually employ Swedish or spherical wheels that are powered. A good example is Uranus. This robot uses four Swedish wheels to rotate and translate independently and without constraints.

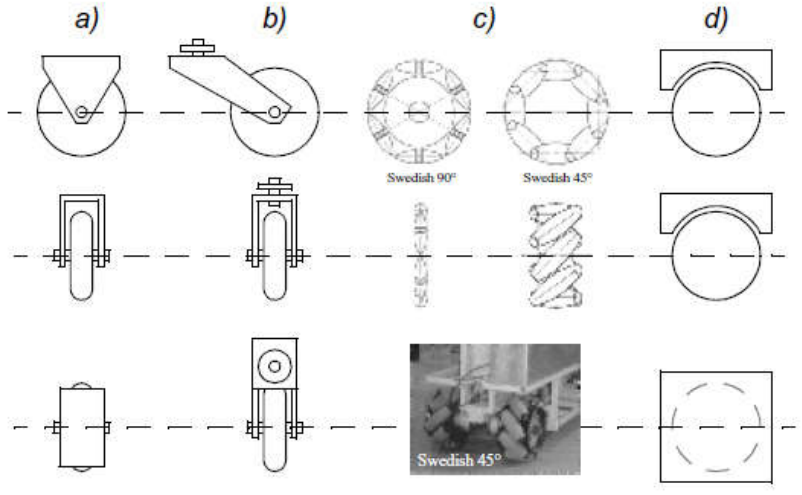

Figure 1.2
The four basic wheel types. (a) Standard wheel: two degrees of freedom; rotation around the (motorized) wheel axle and the contact point.(b) castor wheel: two degrees of freedom; rotation around an offset steering joint. (c) Swedish wheel: three degrees of freedom; rotation around the (motorized) wheel axle, around the rollers, and around the contact point. (d) Ball or spherical wheel: realization technically difficult.

Chapter 2:

Survey of Mobile Robot

2.1 Introduction:

Mobile robot including go to goal; can be defined as mechanical system capable of moving in its environment in an autonomous manner. For that purpose, it must be equipped with:

- *Sensors* that will help in gaining knowledge of its surroundings and determine its location, here we have used ultra-sonic other tools like GPS can be used in open area.
- *Actuators* which will allow it to move;
- an *intelligence* (or algorithm, regulator), which will allow it to compute, based on the data gathered by sensors, the commands to send actuators in order to perform a given task.

2.2 components:

- Arduino Uno.
- Ultrasonic sensor.
- Encoder sensor.
- Motor drive.
- DC motor
- Chassis.
- Ticks.

2.2.1 Arduino Uno: it's an open source microcontroller board based on the microchip Atmega328P microcontroller and developed by 10rduino.cc. The board is equipped with sets of digital and analog input/output pins that may be interfaced to various expansion boards and other circuits. It has a USB connection, a power jack, an ICSP header, and a reset button.

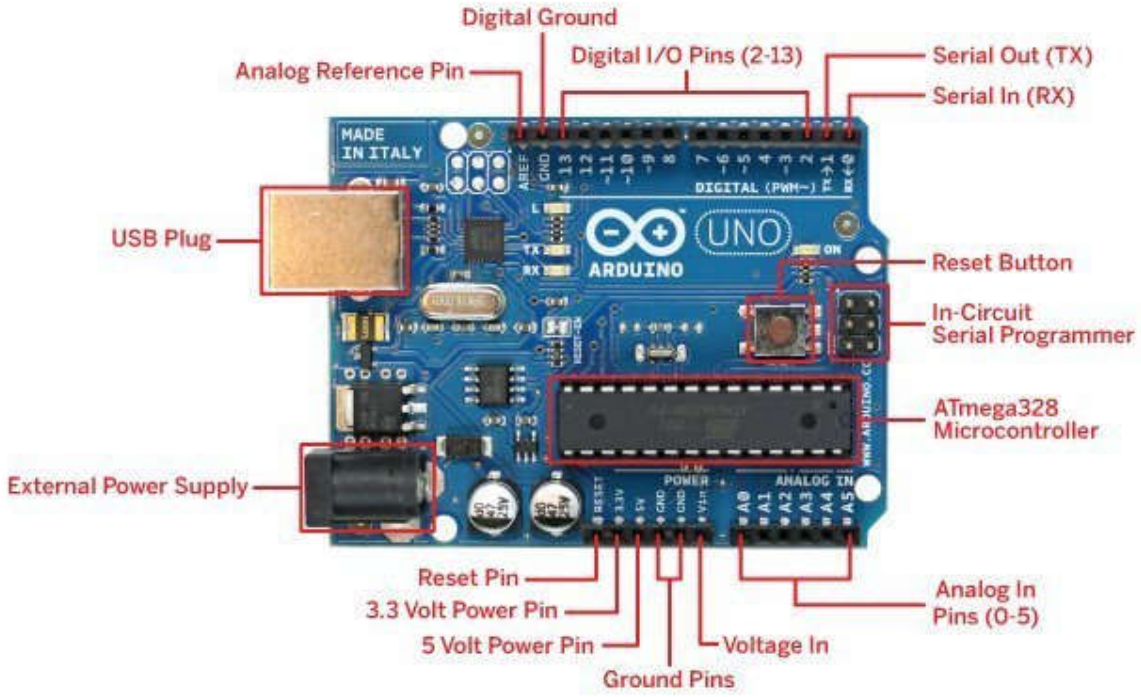

Figure 2.2.1: Arduino Uno board

2.2.2 Ultrasonic sensor: An Ultrasonic sensor is a device that can measure the distance to an object by using sound waves. It measures distance by sending out a sound wave at a specific frequency and listening for that sound wave to bounce back. It is important to understand that some objects might not be detected, a gadget that uses sonar to determine the distance of an object just like the method that bats use. It offers excellent non-contact range detection with high accuracy and stable readings in easy-to-use package from 2 cm to 400 cm or 1 to 13 feet.

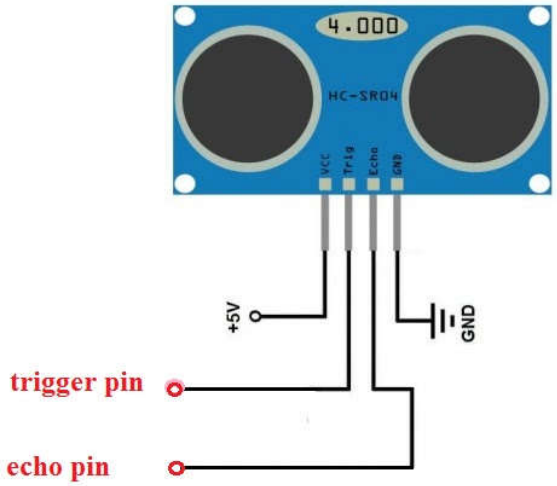

Figure 2.2.2: Ultrasonic sensor

Example 2.2.1: this example shows that how to measure the distance by ultrasonic sensor.

Code 2.2.1:

```
// defines pins numbers
12ons tint trigPin = 11;
12ons tint echoPin = 12;

// defines variables
long duration;
int distance;

void setup() {
pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
pinMode(echoPin, INPUT); // Sets the echoPin as an Input
Serial.begin(9600); // Starts the serial communication
}

void loop() {
// 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);

// Calculating the distance
```

```
29. distance= duration*0.034/2;
30.
31. // Prints the distance on the Serial Monitor
32. Serial.print("Distance: ");
33. Serial.println(distance);
34. }
```

Connection diagram 2.2.1:

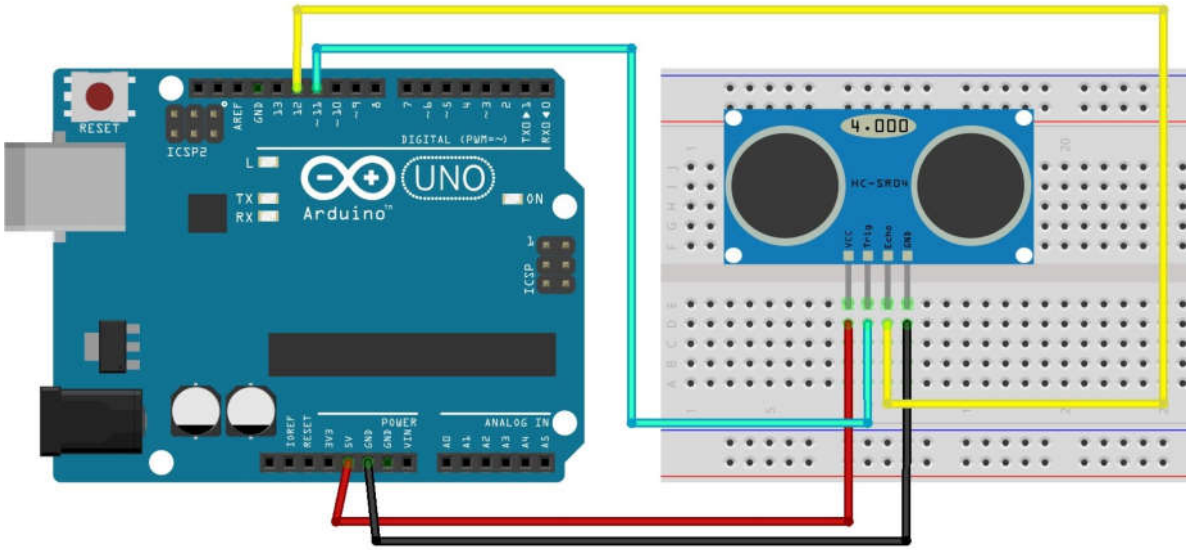

Figure2.2.3: Connection diagram of ultrasonic sensor

2.2.3 Motors and motor drive:

it gives the robot ability to traverse and turn toward the desired goal.

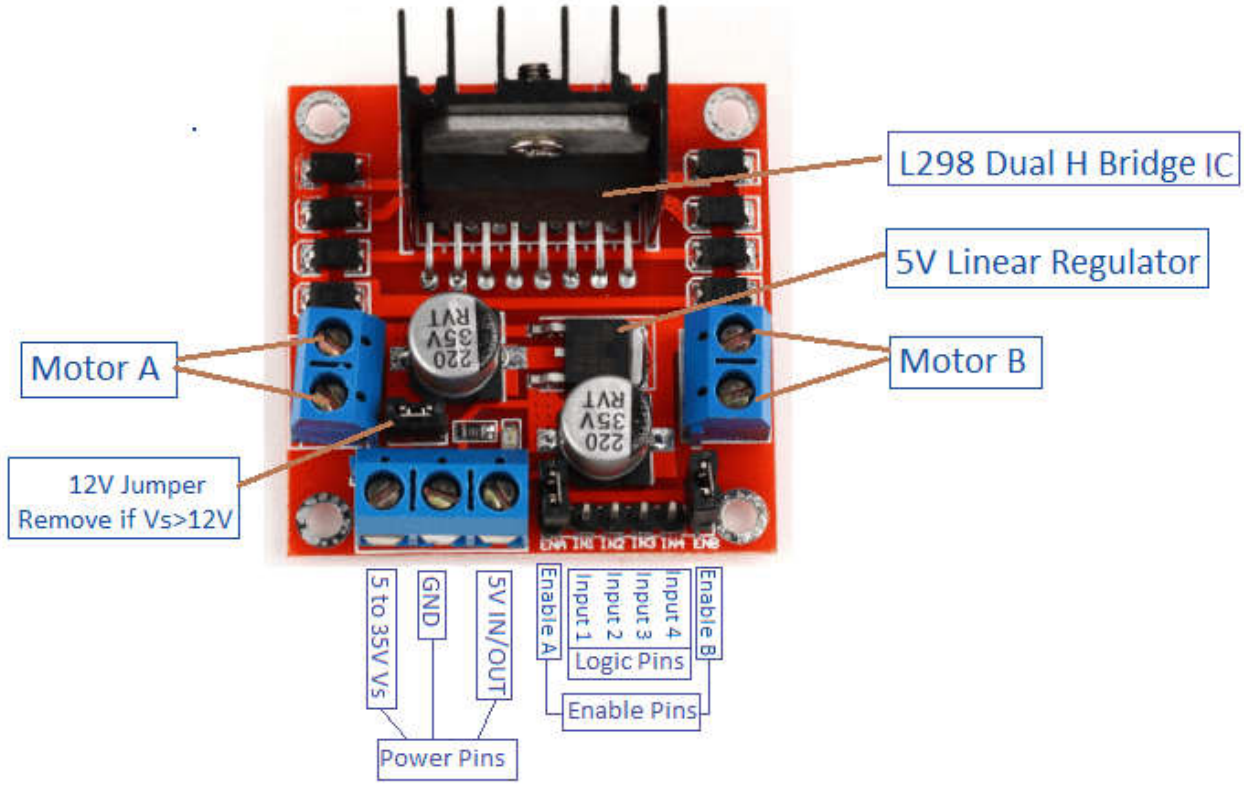

Figure 2.2.4: motor drive (L298 dual H Bridge IC)

Example 2.2.2: this example shows that how to control motors by using joystick.

Code 2.2.2:

```
1.  //Joystick Pins
2.  int x_key = A0;
3.  int y_key = A1;
4.  int x_pos;
5.  int y_pos;
6.
7.  //Motor Pins
8.  int EN_A = 11; //Enable pin for first motor
9.  int IN1 = 9; //control pin for first motor
10. int IN2 = 8; //control pin for first motor
11. int IN3 = 7; //control pin for second motor
12. int IN4 = 6; //control pin for second motor
13. int EN_B = 10; //Enable pin for second motor
14. //Initializing variables to store data
15. int motor_speed;
```

```
int motor_speed1;

void setup ( ) {
Serial.begin (9600); //Starting the serial communication at 9600 baud rate
//Initializing the motor pins as output
pinMode(EN_A, OUTPUT);
pinMode(IN1, OUTPUT);
pinMode(IN2, OUTPUT);
pinMode(IN3, OUTPUT);
pinMode(IN4, OUTPUT);
pinMode(EN_B, OUTPUT);

//Initializng the joystick pins as input
pinMode (x_key, INPUT) ;
pinMode (y_key, INPUT) ;
}

void loop () {
x_pos = analogRead (x_key) ; //Reading the horizontal movement value
y_pos = analogRead (y_key) ; //Reading the vertical movement value

if (x_pos < 400){ //Rotating the left motor in clockwise direction
motor_speed = map(x_pos, 400, 0, 0, 255); //Mapping the values to 0-255 to move the motor
digitalWrite(IN1, LOW);
digitalWrite(IN2, HIGH);
analogWrite(EN_A, motor_speed);
}

else if (x_pos>400 && x_pos <600){ //Motors will not move when the joystick will be at center
digitalWrite(IN1, LOW);
digitalWrite(IN2, LOW);
}

else if (x_pos > 600){ //Rotating the left motor in anticlockwise direction
motor_speed = map(x_pos, 600, 1023, 0, 255);
digitalWrite(IN1, HIGH);
digitalWrite(IN2, LOW);
analogWrite(EN_A, motor_speed);
}

if (y_pos < 400){ //Rotating the right motor in clockwise direction
motor_speed1 = map(y_pos, 400, 0, 0, 255);
digitalWrite(IN3, LOW);
digitalWrite(IN4, HIGH);
analogWrite(EN_B, motor_speed1);
}

else if (y_pos>400 && y_pos <600){
digitalWrite(IN3, LOW);
digitalWrite(IN4, LOW);
}

else if (y_pos > 600){ //Rotating the right motor in anticlockwise direction
motor_speed1 = map(y_pos, 600, 1023, 0, 255);
digitalWrite(IN3, HIGH);
digitalWrite(IN4, LOW);
analogWrite(EN_B, motor_speed1);
}
}
```

Connection diagram 2.2.2:

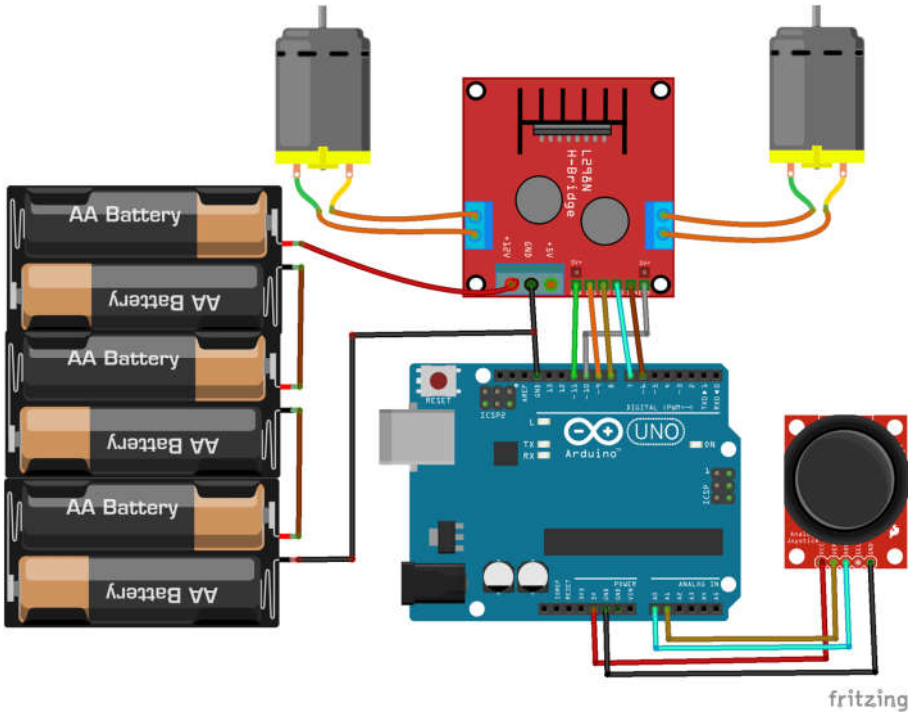

Figure 2.2.5: Controlling two motors by joystick

2.2.4 Encoder (speed sensor + ticks): this combination tells revolutions per second we have a disk that has 20 ticks, it acts as counter by the time of 19 tick is counter one more and you have completed a full revolution or translated 22 cm away.

Example 2.2.3: this example shows that how measure the speed of the motor

Code 2.2.3:

```
#include
<LiquidCrystal.h>
                    LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
                    int sensor = 11;
                    unsigned long start_time = 0;
                    unsigned long end_time = 0;
                    int steps=0;
                    float steps_old=0;
                    float temp=0;
                    float rps=0;


                    void setup()
                    {
```

```
 Serial.begin(9600);
 lcd.begin(16, 2);
 pinMode(sensor,INPUT_PULLUP);
 lcd.setCursor(0,0);
 lcd.print(" STEPS - 0");
 lcd.setCursor(0,1);
 lcd.print(" RPS   - 0.00");
}

void loop()
{
 start_time=millis();
 end_time=start_time+1000;
 while(millis()<end_time)
 {
  if(digitalRead(sensor))
  {
   steps=steps+1;
   while(digitalRead(sensor));
  }
  lcd.setCursor(9,0);
  lcd.print(steps);
  lcd.print("   ");
 }
   temp=steps-steps_old;
   steps_old=steps;
   rps=(temp/20);
   lcd.setCursor(9,1);
   lcd.print(rps);
   lcd.print("   ");
}
```

Connection diagram 2.2.3:

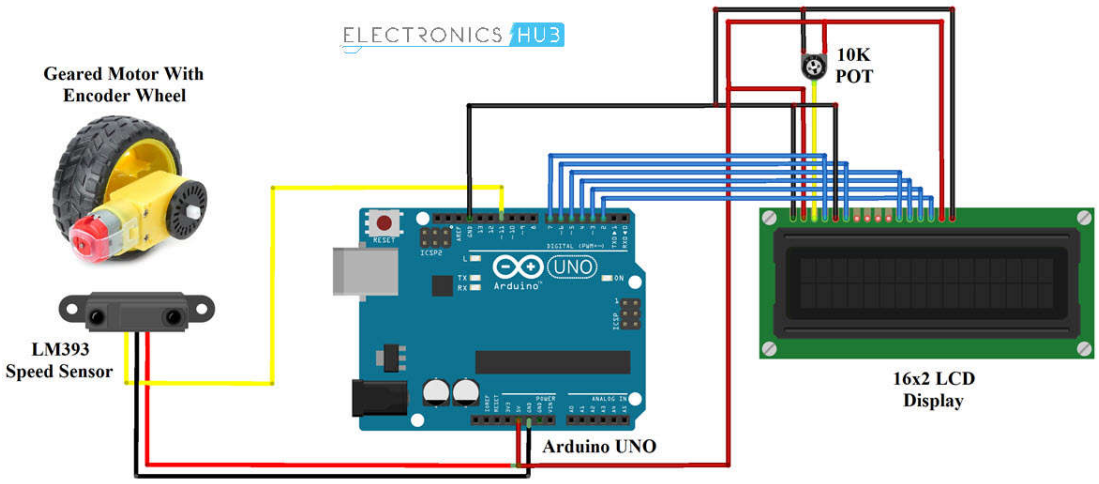

Figure2.2.6: using encoder as indicator of speed

Chapter 3:

Implementations

3.1 Software:

The following pages explain robots code. It's written in (C, C++) programming language, compiled in Arduino's official software (Arduino IDE).

3.1.1 Declaration and initializing:

inline declaration and initialization of variables according to real Arduino pins.

```
#include<math.h>
const int trigf = 6;
const int trigl = 7;
const int trigb = 8;
const int trigr = 9;
const int echof = 2;
const int echol = 3;
const int echob = 4;
const int echor = 5;
int sensorr = 12;
int sensorl = 13;
int motorr = 0;
int motorsr = 10;
int motorl = 1;
int motorsl = 11;
long durationf, durationl, durationb, durationr;
long distancef, distancel, distanceb, distancer;
float pi = 3.14, thetarad, ds = 13.5, s, revrot, stepsrot,
speedl = 70, speedr = 90, x1, y1, x2 = 87, y2 = 153, x, y,
steps, r, rev, lengthrob = 23, widthrob = 14, sumf = 0, sumb
= 0, suml = 0, sumr = 0, avgf, avgb, avgl, avgr;
int stepsr = 0, stepsrold = 0, stepsl = 0,
stepslold = 0, stepsrrot = 0, stepslrot = 0 ;
int a = 0;
```

3.1.2 Function declaration for encoder sensors:

Those functions are used for encoder sensors to measure the distance that the robot has taken.

```
int findstepl()
{ if (digitalRead(sensorl)){
    stepsl = stepsl + 1;
    while (digitalRead(sensorl)) {}}
    return 0;}
int findstepr()
{ if (digitalRead(sensorr)){
    stepsr = stepsr + 1;
    while (digitalRead(sensorr)) {}}
  return 0;}
int findsteplrot()
{ if (digitalRead(sensorl)){
    stepslrot = stepslrot + 1;
    while (digitalRead(sensorl)) {} }
  return 0;}
int findsteprrot()
{ if (digitalRead(sensorr))
  {
    stepsrrot = stepsrrot + 1;
    while (digitalRead(sensorr)) {}}
  return 0;}
```

3.1.3 Defining pins: the pins can be output or input.

```
void setup() {
  pinMode(sensorr, INPUT_PULLUP);
  pinMode(sensorl, INPUT_PULLUP);
  pinMode(motorr, OUTPUT);
  pinMode(motorl, OUTPUT);
  pinMode(motorsl, OUTPUT);
  pinMode(motorsr, OUTPUT);
  pinMode(trigf, OUTPUT);
  pinMode(trigl, OUTPUT);
  pinMode(trigb, OUTPUT);
  pinMode(trigr, OUTPUT);
  pinMode(echof, INPUT);
  pinMode(echol, INPUT);
  pinMode(echob, INPUT);
  pinMode(echor, INPUT);
  Serial.begin(9600);
```

3.1.4 Ultrasonic sensors:

We have four ultrasonic sensors to measure the distance from the frames to find its location (x1, y1) and measure the angle it takes to be in line with the desired goal, and the distance between the goal and the robot (x, y).

```
for (int i = 0; i < 10; i++) {
  digitalWrite(trigf, LOW);
  delayMicroseconds(2);
  digitalWrite(trigf, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigf, LOW);
  durationf = pulseIn(echof, HIGH);
  digitalWrite(trigl, LOW);
  delayMicroseconds(2);
  digitalWrite(trigl, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigl, LOW);
  durationl = pulseIn(echol, HIGH);
  digitalWrite(trigb, LOW);
  delayMicroseconds(2);
  digitalWrite(trigb, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigb, LOW);
  durationb = pulseIn(echob, HIGH);
  digitalWrite(trigr, LOW);
  delayMicroseconds(2);
  digitalWrite(trigr, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigr, LOW);
  durationr = pulseIn(echor, HIGH);
```

3.1.5 Calculation:

1. robot location calculation.
2. distance between the robot and goal calculation.
3. determining the angle between the goal and the robot.
4. Change the distance and angle to steps.

```
  sumf = sumf + distancef;
  suml = suml + distancel;
  sumb = sumb + distanceb;
  sumr = sumr + distancer;
}
avgf = sumf / 10;
avgl = suml / 10;
avgb = sumb / 10;
avgr = sumr / 10;
if(avgb<=avgf){
yl = avgb + (lengthrob / 2);

}
if(avgf<avgb){
  yl=183-(avgf+(lengthrob/2));
  }
  if(avgl<=avgr){
    xl = avgl + (widthrob / 2);
    }
    if(avgr<avgl){
     xl= 121-(avgr+(widthrob / 2));

    }
   x = abs(xl - x2);
   y = abs(yl - y2);
   Serial.println(xl);
   Serial.println(yl);
   if (y2 > yl) {
     thetarad = atan(x / y);
   }
   if (y2 == yl) {
     thetarad = pi / 2;
   }
   if (y2 < yl) {
     thetarad = pi - atan(x / y);
   }

   s = ds * thetarad;
   revrot = s / 23;
   stepsrot = revrot * 20;
   r = sqrt( pow(x, 2) + pow(y, 2));
   rev = r / 24;
   steps = rev * 20;
 }
```

Turning Operation:

We have designed such that it will turn to right if the target was on right hand. And it will turn left if the target was located in left hand.

```
void loop() {
  while (x2 >= x1) {
    if (stepslrot < stepsrot) {
      digitalWrite(motorl, HIGH);
      analogWrite(motorsl, 80);
      findsteplrot();
    }
    if (stepslrot >= stepsrot) {
      digitalWrite(motorl, LOW);
      analogWrite(motorsl, 0);
      goto label;
    }
    delay(1);
  }
  while (x1 > x2) {
    if (stepsrrot < stepsrot) {
      digitalWrite(motorr, HIGH);
      analogWrite(motorsr, 80);
      findsteprrot();
    }
    if (stepsrrot >= stepsrot) {
      digitalWrite(motorr, LOW);
      analogWrite(motorsr, 0);
      goto label;
    }
    delay(1);
  }
label: delay(1000);
```

3.1.7 Traverse operation:

The robot starts to move to the goal after it giant the proper orientation.

```
while (stepsr < steps || stepsl < steps) {
  stepsrold = stepsr;
  stepsrold = stepsrold + 3;
  stepslold = stepsl;
  stepslold = stepslold + 3;

  while (stepsr < stepsrold) {
    digitalWrite(motorr, HIGH);
    analogWrite(motorsr, speedr);
    findstepr();
    delay(1);
    if (stepsr >= stepsrold)
    { digitalWrite(motorr, LOW);
      analogWrite(motorsr, 0);
      goto label0; }
    delay(1);}
label0: delay(1);
  while (stepsl < stepslold) {
    digitalWrite(motorl, HIGH);
    analogWrite(motorsl, speedl);
    findstepl();
    delay(1);
    if (stepsl >= stepslold)
    { digitalWrite(motorl, LOW);
      analogWrite(motorsl, 0);
      goto label1; }
    delay(1); }
label1: delay(25); }delay(5000);}
```

*You can see the total code in appendix (last page).

3.2 Hardware:

3.2.1 Connection diagram of the system:

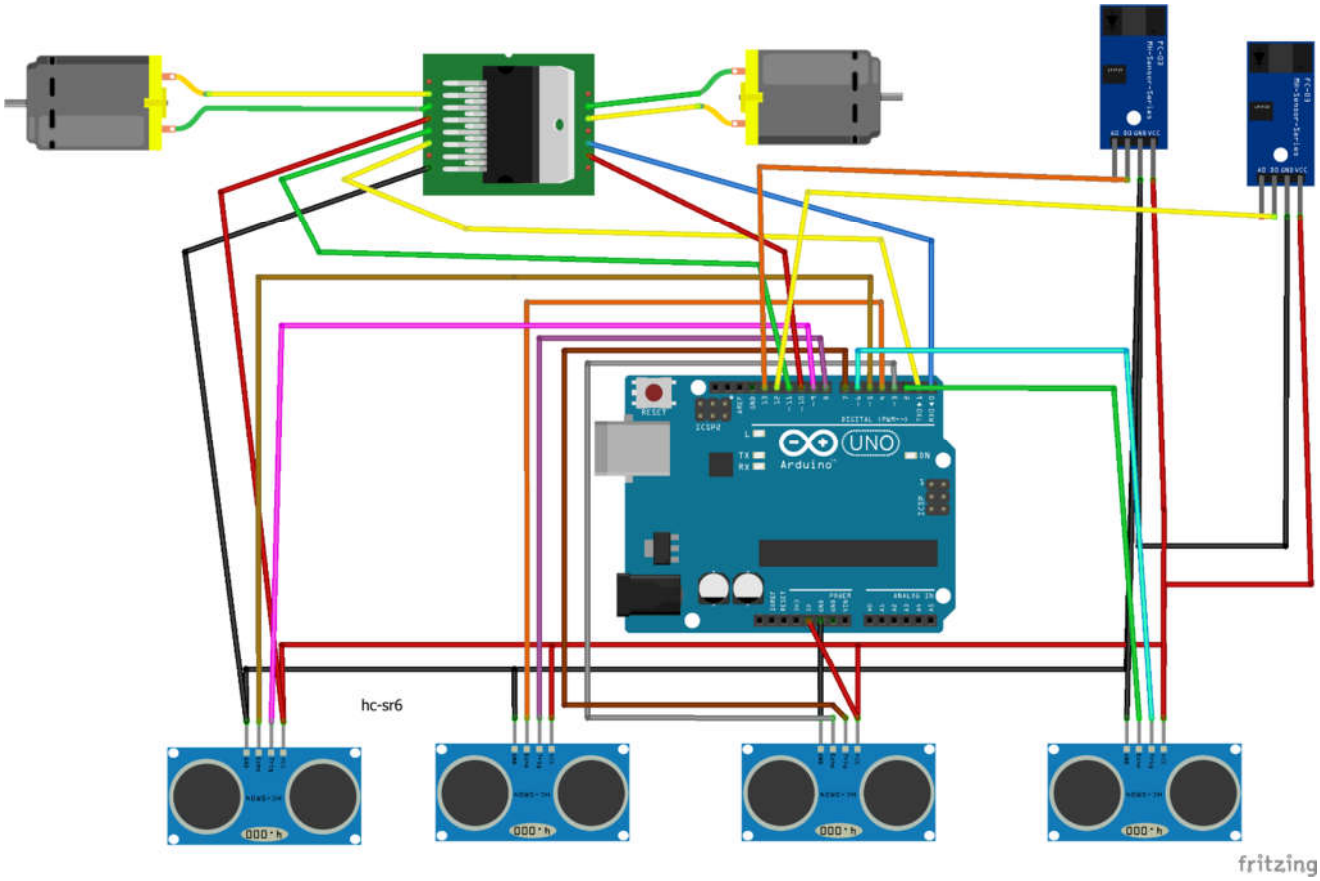

Figure 3.2.1.1: How parts connected between each other and to microprocessor.

3.3 The robot and frame:

The robot frame dimensions are (180X120) cm.

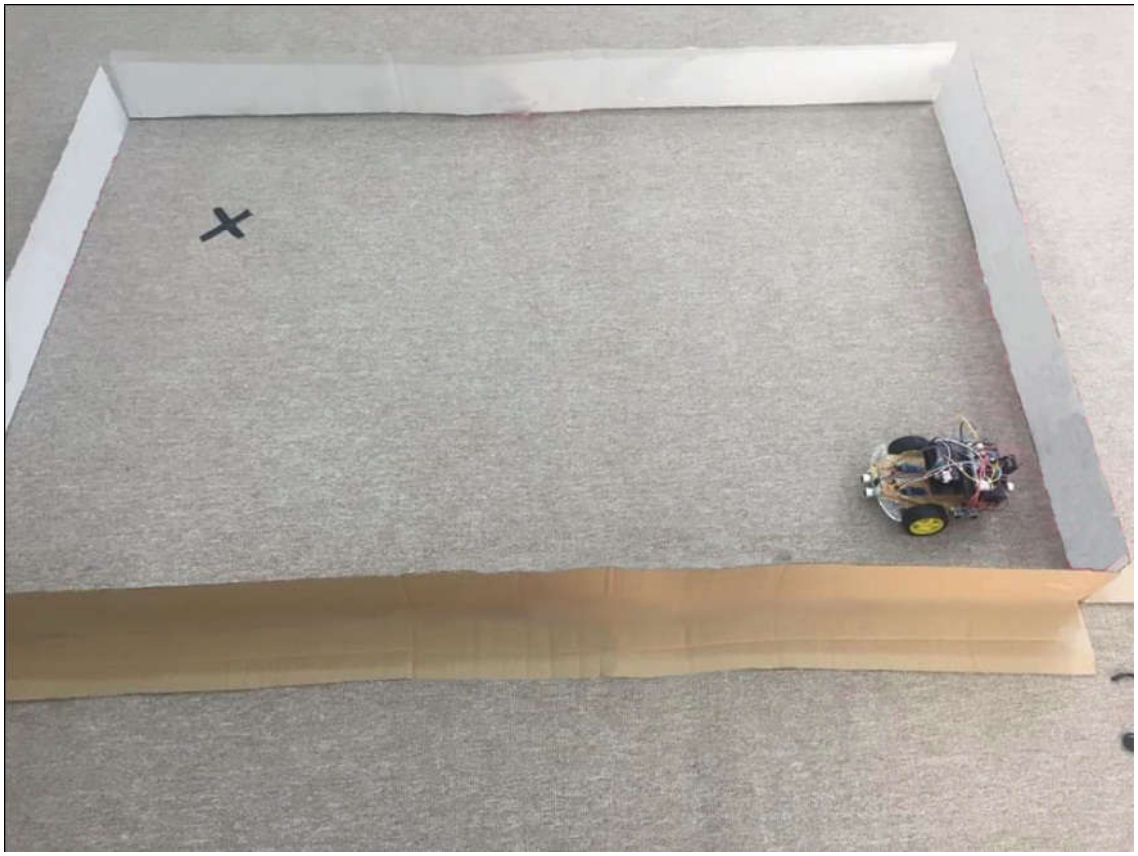

Figure 3.3.1: The robot and frame image.

Chapter 4:

Results and Conclusions

4.1 Results:

The following pictures are taken from practical operations of the robot, they depict the way robot traverse to the point which is indicated by black cross. The cross is within the closed area so it's reachable. It takes the shortest path in order to get there.

4.1.1 Test NO #1:

Error= 5cm

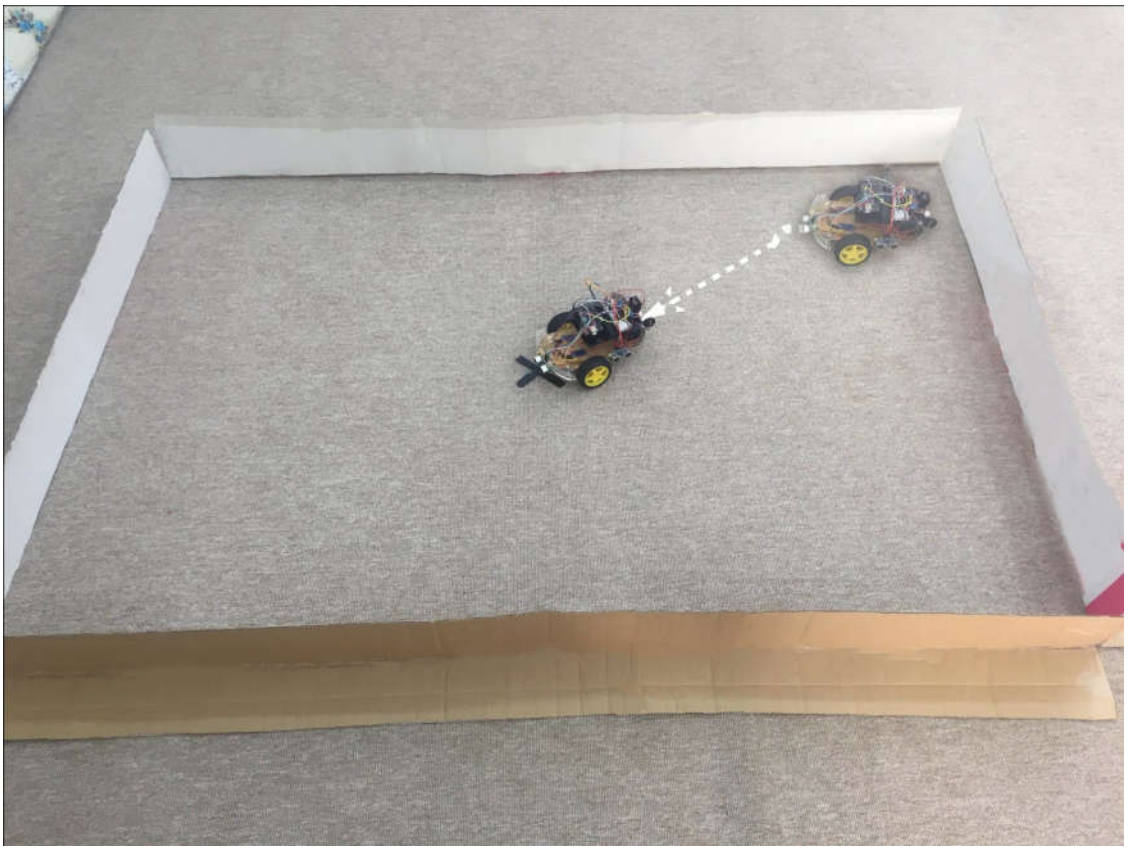

Figure 4.1.1: The Goal point is (60,90).

4.1.2 Test NO #2:

error = 12cm

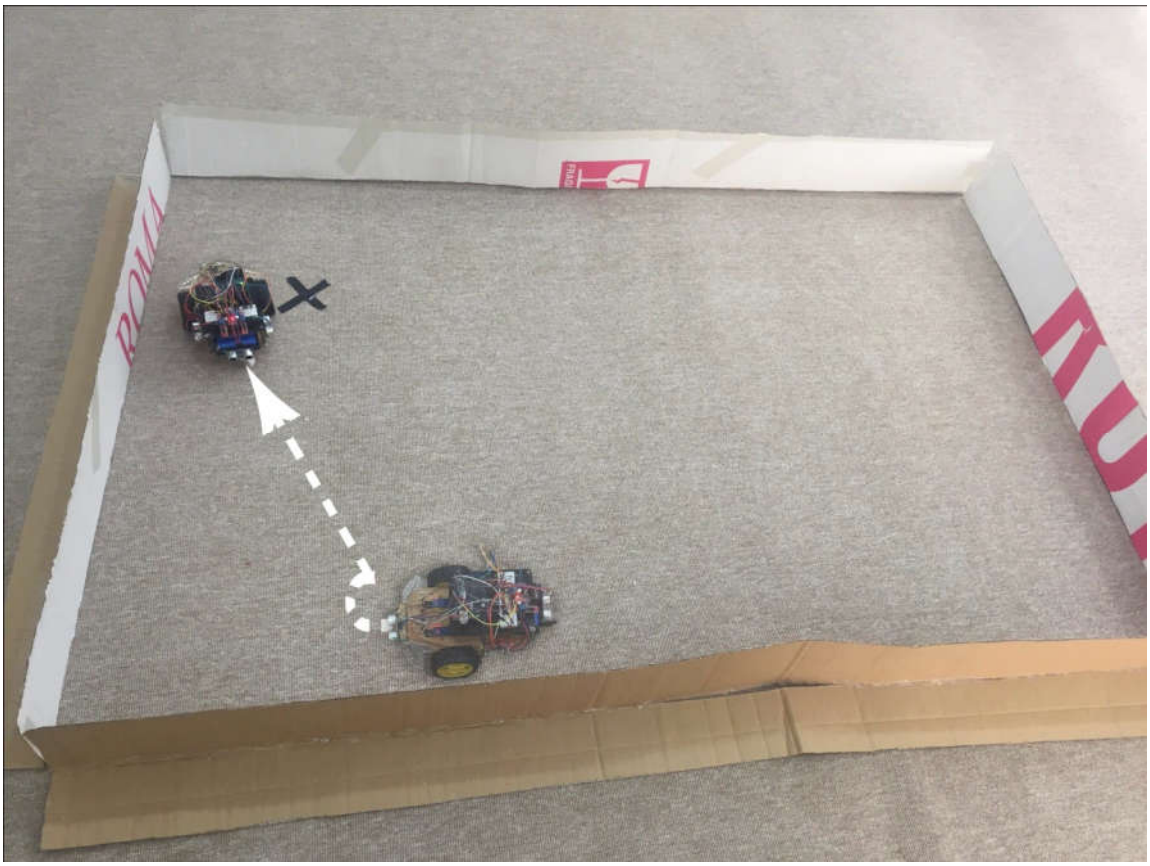

Figure 4.1.2: Goal point is (87,153).

4.1.3 Test NO #3:

Error =7cm

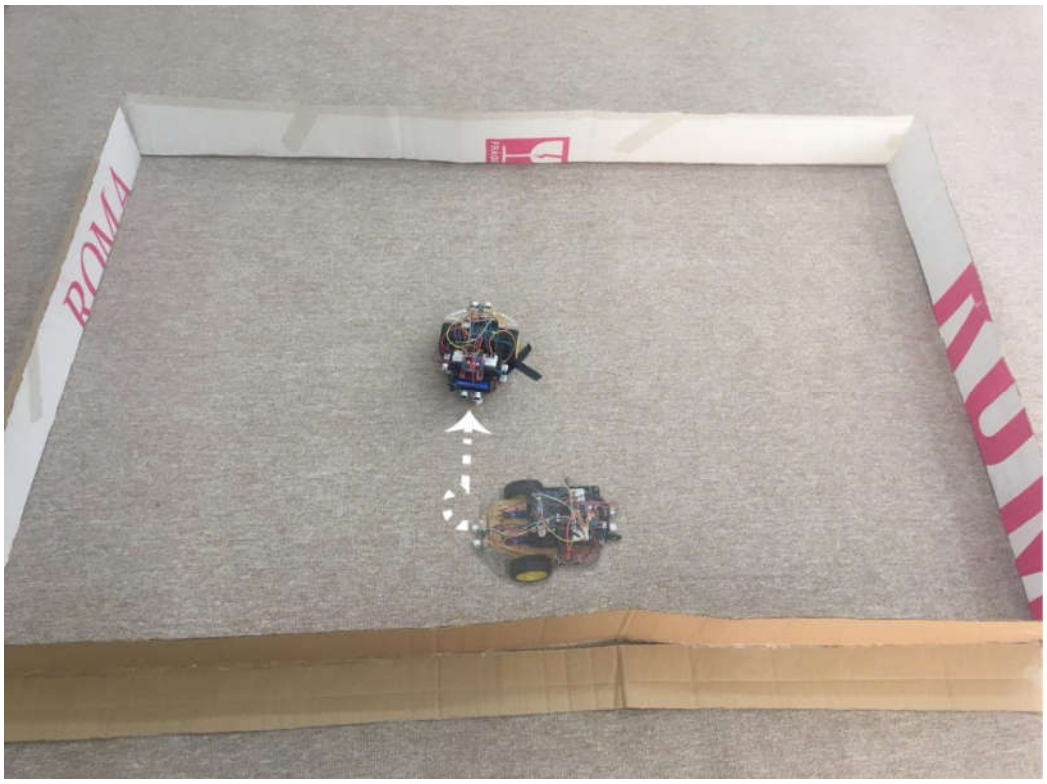

Figure 4.1.3: Goal point is (60,90).

4.1.4 Test NO #4:

Error =15cm

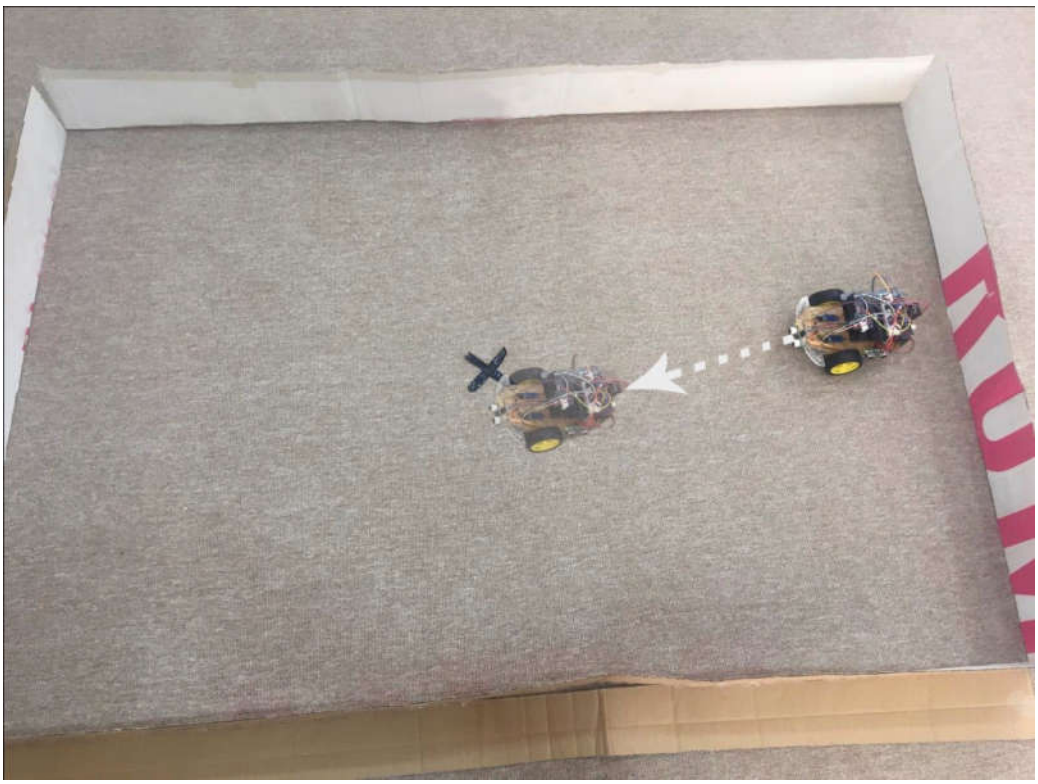

Figure 4.1.4: Goal point is (60, 90).

4.1.5 Test NO #5:

Error =15cm

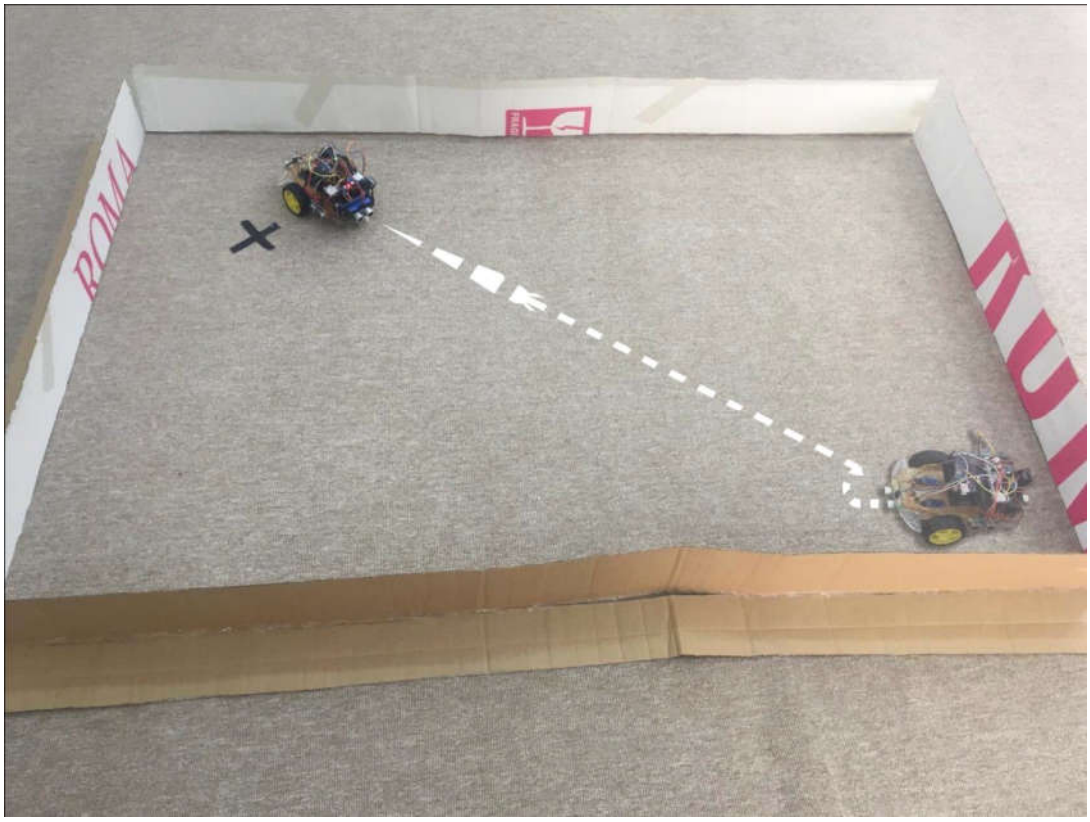

Figure 4.1.5: Goal point (87,153).

tests	error
NO1	5cm
NO2	12cm
NO3	7cm
NO4	15cm
NO5	15cm
Average error	10.8 cm

Table 4.1.1: Average of Errors

*The resulted errors are due to the encoder, sensor and motors deficiencies. Ground that robot explores on can also have effect for instance slip can be occurred on frictionless surfaces, they all lead to amount of errors.

4.2 Conclusions:

1- Our rover has ability to go to desired point within a predefined workspace.
2- It can be equipped further more to become an object avoidant rover.
3- The code needs to be maintained to check whether it's compatible with tools.
4- the concept behind this project is very important even if it isn’t that complex one.

References

Book(s):

[1] Introduction to autonomous mobile robots by Roland Siegwart and illah R. Nourbakhsh.

[2] Wheeled Mobile Robotics, from Fundamentals Towards Autonomous Systems by Gregor Klancar, Andrej Zdesar, Saso Blazic and Igor akrjanc.

3.4 Appendix:

```
#include<math.h>
const int trigf = 6;
const int trigl = 7;
const int trigb = 8;
const int trigr = 9;
const int echof = 2;
const int echol = 3;
const int echob = 4;
const int echor = 5;
int sensorr = 12;
int sensorl = 13;
int motorr = 0;
int motorsr = 10;
int motorl = 1;
int motorsl = 11;
long durationf, durationl, durationb, durationr;
long distancef, distancel, distanceb, distancer;
float pi = 3.14, thetarad, ds = 13.5, s, revrot, stepsrot,
speedl = 70, speedr = 90, x1, y1, x2 = 87, y2 = 153, x, y,
steps, r, rev, lengthrob = 23, widthrob = 14, sumf = 0, sumb
= 0, suml = 0, sumr = 0, avgf, avgb, avgl, avgr;
int stepsr = 0, stepsrold = 0, stepsl = 0,
stepslold = 0, stepsrrot = 0, stepslrot = 0 ;
int a = 0;
```

```
int findstepl()
{ if (digitalRead(sensorl)){
    stepsl = stepsl + 1;
    while (digitalRead(sensorl)) {}}
    return 0;}
int findstepr()
{ if (digitalRead(sensorr)){
    stepsr = stepsr + 1;
    while (digitalRead(sensorr)) {}}
  return 0;}
int findsteplrot()
{ if (digitalRead(sensorl)){
    stepslrot = stepslrot + 1;
    while (digitalRead(sensorl)) {} }
  return 0;}
int findsteprrot()
{ if (digitalRead(sensorr))
  {
    stepsrrot = stepsrrot + 1;
    while (digitalRead(sensorr)) {}}
  return 0;}
```

```
void setup() {
  pinMode(sensorr, INPUT_PULLUP);
  pinMode(sensorl, INPUT_PULLUP);
  pinMode(motorr, OUTPUT);
  pinMode(motorl, OUTPUT);
  pinMode(motorsl, OUTPUT);
  pinMode(motorsr, OUTPUT);
  pinMode(trigf, OUTPUT);
  pinMode(trigl, OUTPUT);
  pinMode(trigb, OUTPUT);
  pinMode(trigr, OUTPUT);
  pinMode(echof, INPUT);
  pinMode(echol, INPUT);
  pinMode(echob, INPUT);
  pinMode(echor, INPUT);
  Serial.begin(9600);
```

```
for (int i = 0; i < 10; i++) {
  digitalWrite(trigf, LOW);
  delayMicroseconds(2);
  digitalWrite(trigf, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigf, LOW);
  durationf = pulseIn(echof, HIGH);
  digitalWrite(trigl, LOW);
  delayMicroseconds(2);
  digitalWrite(trigl, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigl, LOW);
  durationl = pulseIn(echol, HIGH);
  digitalWrite(trigb, LOW);
  delayMicroseconds(2);
  digitalWrite(trigb, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigb, LOW);
  durationb = pulseIn(echob, HIGH);
  digitalWrite(trigr, LOW);
  delayMicroseconds(2);
  digitalWrite(trigr, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigr, LOW);
  durationr = pulseIn(echor, HIGH);
```

```
    sumf = sumf + distancef;
    suml = suml + distancel;
    sumb = sumb + distanceb;
    sumr = sumr + distancer;
  }
  avgf = sumf / 10;
  avgl = suml / 10;
  avgb = sumb / 10;
  avgr = sumr / 10;
  if(avgb<=avgf){
  yl = avgb + (lengthrob / 2);

  }
  if(avgf<avgb){
    yl=183-(avgf+(lengthrob/2));
    }
    if(avgl<=avgr){
      xl = avgl + (widthrob / 2);
      }
      if(avgr<avgl){
       xl= 121-(avgr+(widthrob / 2));

      }
```

```
  x = abs(x1 - x2);
  y = abs(y1 - y2);
  Serial.println(x1);
  Serial.println(y1);
  if (y2 > y1) {
    thetarad = atan(x / y);
  }
  if (y2 == y1) {
    thetarad = pi / 2;
  }
  if (y2 < y1) {
    thetarad = pi - atan(x / y);
  }

  s = ds * thetarad;
  revrot = s / 23;
  stepsrot = revrot * 20;
  r = sqrt( pow(x, 2) + pow(y, 2));
  rev = r / 24;
  steps = rev * 20;
}
```

```
void loop() {
  while (x2 >= x1) {
    if (stepslrot < stepsrot) {
      digitalWrite(motorl, HIGH);
      analogWrite(motorsl, 80);
      findsteplrot();
    }
    if (stepslrot >= stepsrot) {
      digitalWrite(motorl, LOW);
      analogWrite(motorsl, 0);
      goto label;
    }
    delay(1);
  }
  while (x1 > x2) {
    if (stepsrrot < stepsrot) {
      digitalWrite(motorr, HIGH);
      analogWrite(motorsr, 80);
      findsteprrot();
    }
    if (stepsrrot >= stepsrot) {
      digitalWrite(motorr, LOW);
      analogWrite(motorsr, 0);
      goto label;
    }
    delay(1);
  }
label: delay(1000);
```

```
    while (stepsr < steps || stepsl < steps) {
      stepsrold = stepsr;
      stepsrold = stepsrold + 3;
      stepslold = stepsl;
      stepslold = stepslold + 3;

      while (stepsr < stepsrold) {
        digitalWrite(motorr, HIGH);
        analogWrite(motorsr, speedr);
        findstepr();
        delay(1);
        if (stepsr >= stepsrold)
        { digitalWrite(motorr, LOW);
          analogWrite(motorsr, 0);
          goto label0; }
        delay(1);}
    label0: delay(1);
      while (stepsl < stepslold) {
        digitalWrite(motorl, HIGH);
        analogWrite(motorsl, speedl);
        findstepl();
        delay(1);
        if (stepsl >= stepslold)
        { digitalWrite(motorl, LOW);
          analogWrite(motorsl, 0);
          goto label1; }
        delay(1); }
    label1: delay(25); }delay(5000);}
```