FARMINGDALE STATE COLLEGE

STATE UNIVERSITY OF NEW YORK

Computer Engineering Technology - BS

EET 452W – DESIGN PROJECT - 23935

LAWN MOWER (PATH MEMORZING)

Done by -

1. Chandra Swaroop Reddy Vuppuluri

2. Ponelarajan Elavarasan

TABLE OF CONTENTS –

INTRODUCTION :-

Lawn mowers are machines that use one or more spinning blades to cut grass to an equal height. The height of the cut grass may be predetermined by the mower's construction, but it is usually changeable by the operator, often by a single master lever or a lever or nut and bolt on each of the machine's wheels. The blades can be operated by hand, with wheels physically attached to the cutting blades so that as the mower is pushed forward, the blades spin, or the equipment can be powered by a battery-powered or plug-in electric motor. A tiny internal combustion engine is the most popular self-contained power source for lawn mowers. Smaller mowers frequently lack propulsion, relying on human strength to move across a surface; "walk-behind" mowers are self-propelled, requiring just a human to walk behind and direct them. Larger lawn mowers are often self-propelled "walk-behind" kinds, or "ride-on" mowers, which allow the user to ride on the mower and steer it. A robotic lawn mower is meant to run totally on its own or, less typically, with the assistance of a human operator through remote control.

MOTIVATION :–

There are about 70 million households in the US living in the sub-urban area. They need to mow their lawns biweekly, weekly, or monthly. The average lawn mowing time is 1-2 hours and the cost is around $20-$50 per hour, if you are hiring someone to do it for you. Automating this task allows individuals to better spend their time and allow the automated lawn mower to take over this mundane task. The main driving force behind this initiative is eliminating the chore of mowing your lawn. The user is relieved of this physically taxing and time-consuming duty by designing a lawn mower that handles it automatically.

The proposed design aids folks who would otherwise be unable to mow their lawn due to physical restrictions. The autonomous lawn mower gives the user additional free time even in the absence of a physical restriction. This independence is offered in a worry-free environment with minimal user input. Business owners that have relatively small grassy areas that need light mowing are another customer segment that needs this product. Currently, they pay a sizable quantity of money to have a modest amount of grass cut. They could quickly, efficiently, and effortlessly cut that grass with the lawnmower. This product may also be marketed to tech-savvy individuals who would love to prefer to operate a lawnmower remotely.

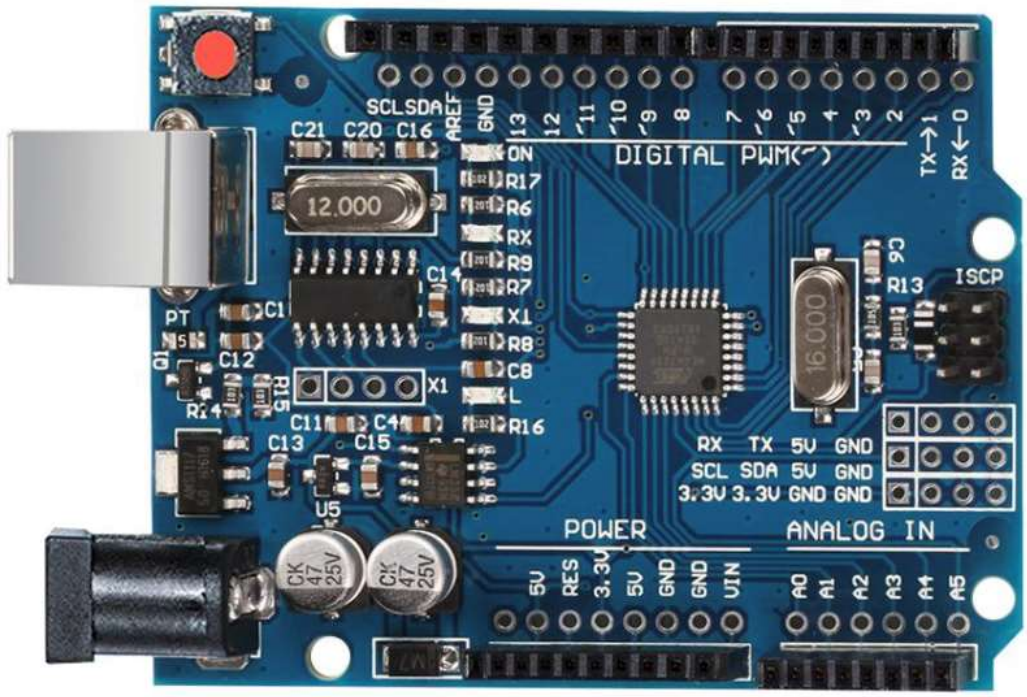
TECHNICAL CONTENT -

A) COMPONENTS -

1. **Arduino UNO**

Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

- Operating Voltage - 5V
- Input Voltage - 7-12V
- DC Current per I/O Pin - 20mA
- DC Current for 3.3V - 50mA
- Flash Memory - 32KB
- Clock Speed - 16 MHz



2. **IR Remote and Receiver -**

Infrared (IR) transmitters and receivers can be found in a wide variety of devices, but they are most commonly found in consumer electronics. This technology works by one component flashing an infrared light in a specific pattern, which another component picks up and converts into an instruction.

- Operating Voltage - 5V
- Output form - Digital Output
- VCC - 3.3V to 5V (External Voltage), GND - External Ground

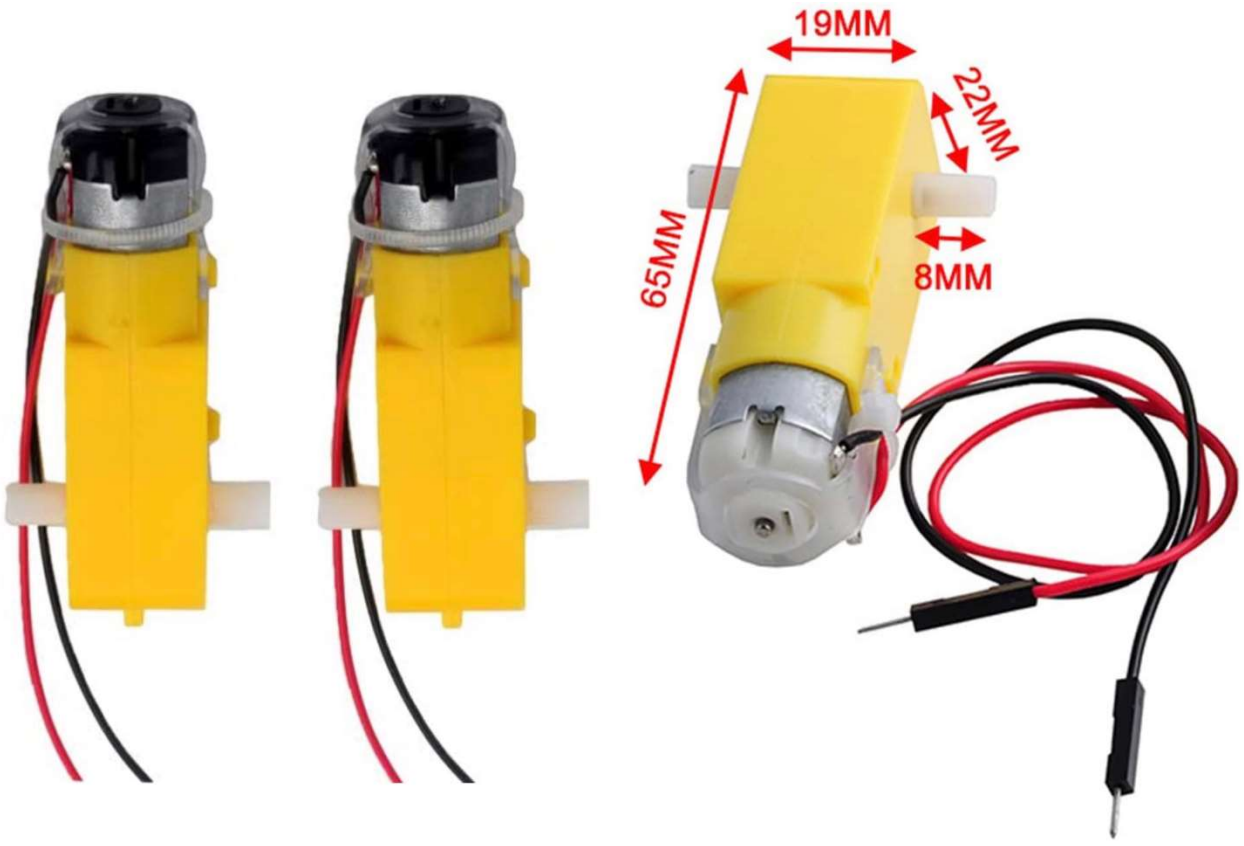
3. **9V Battery**

The 9V battery is used to power the DC geared motor, Arduino board, L239D IC. Overall, its used to power the path memorizing robot.

4. **Two Geared DC Motor**

This is a TT DC Gearbox Motor with a gear ratio of 1:48, and it comes with 2 x 200mm wires with breadboard-friendly 2.54mm (0.1”) male connectors. Perfect for plugging into a breadboard or terminal blocks

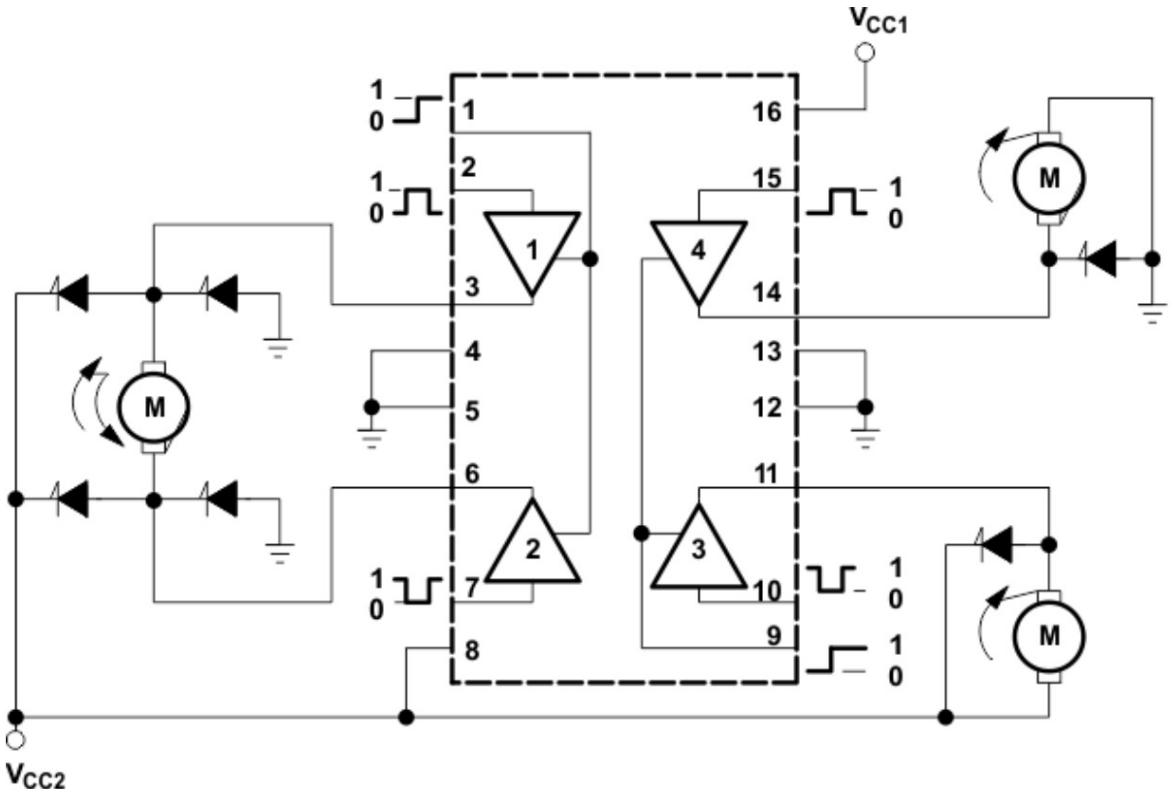
- Rated Voltage: 3~6V
- Min. Operating Speed (6V): 200+/- 10% RPM
- Torque: 0.15Nm ~0.60Nm
- Body Dimensions: 70 x 22 x 18mm
- Noise: <65dB



5. **Motor Driver (L293D IC)**

The L293D device is a quadruple high-current half-H drivers. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. It is designed to drive inductive loads such as relays,

solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.



6. Robot Wheels

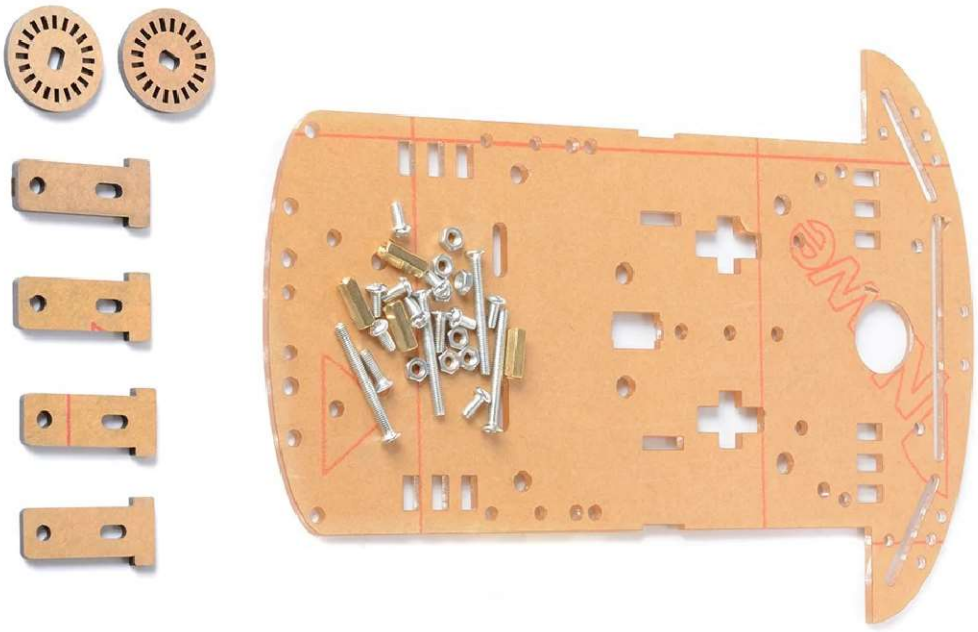
This is a TT DC Gearbox Motor with a gear ratio of 1:48, and it comes with 2 x 200mm 28AWG wires with breadboard-friendly 2.54mm(0.1”) male connectors. Perfect for plugging into a breadboard or terminal blocks

- Rated Voltage: 3~6V
- Continuous No-Load Current: 150mA +/- 10%
- Min. Operating Speed (6V): 200+/- 10% RPM
- Torque: 0.15Nm ~0.60Nm
- Gear Ratio: 1:48

7. Robot Chassis

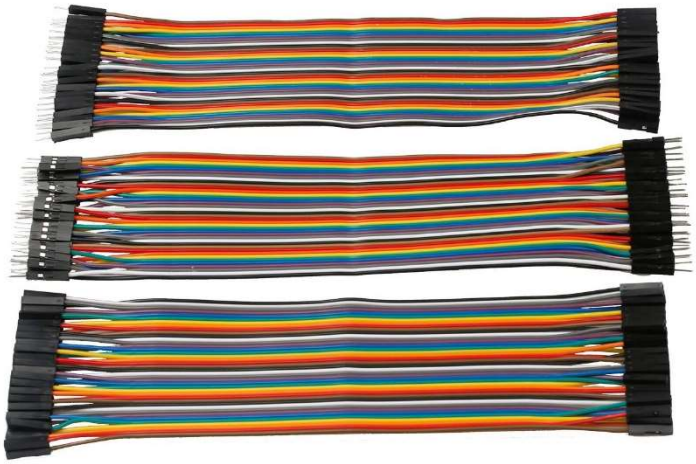
This is the base of the lawn mover where we assemble all the components and fix them

- Simple mechanical structure, it is easy to install.
- All necessary screw and nut.



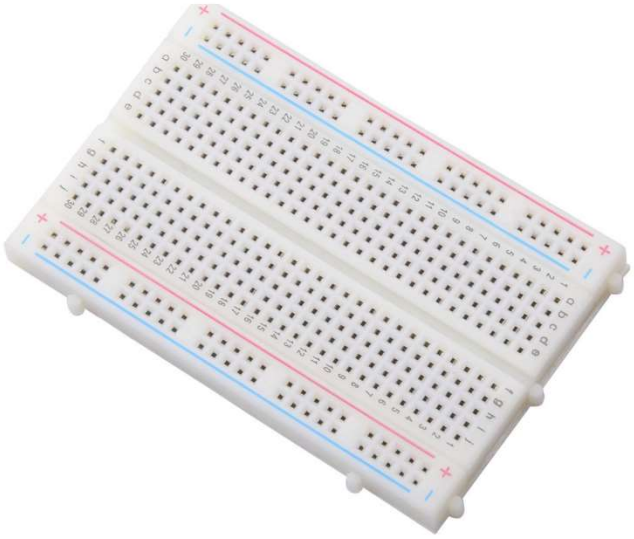
8. Connecting Wires

Connector Type: Standard 2.54mm Pitch 1Pin-1Pin Dupont Housing Connector, with brass nickel plated terminals, provides excellent electrical conductivity and oxidation resistance.
Cable length: 20cm (7.9 inch) / Cable material: 12-core pure copper wire



9. Bread Board

- 400 tie-points,2 power lanes
- Breadboard Material:ABS
- 300V/3-5A,Horizontal: 1-30



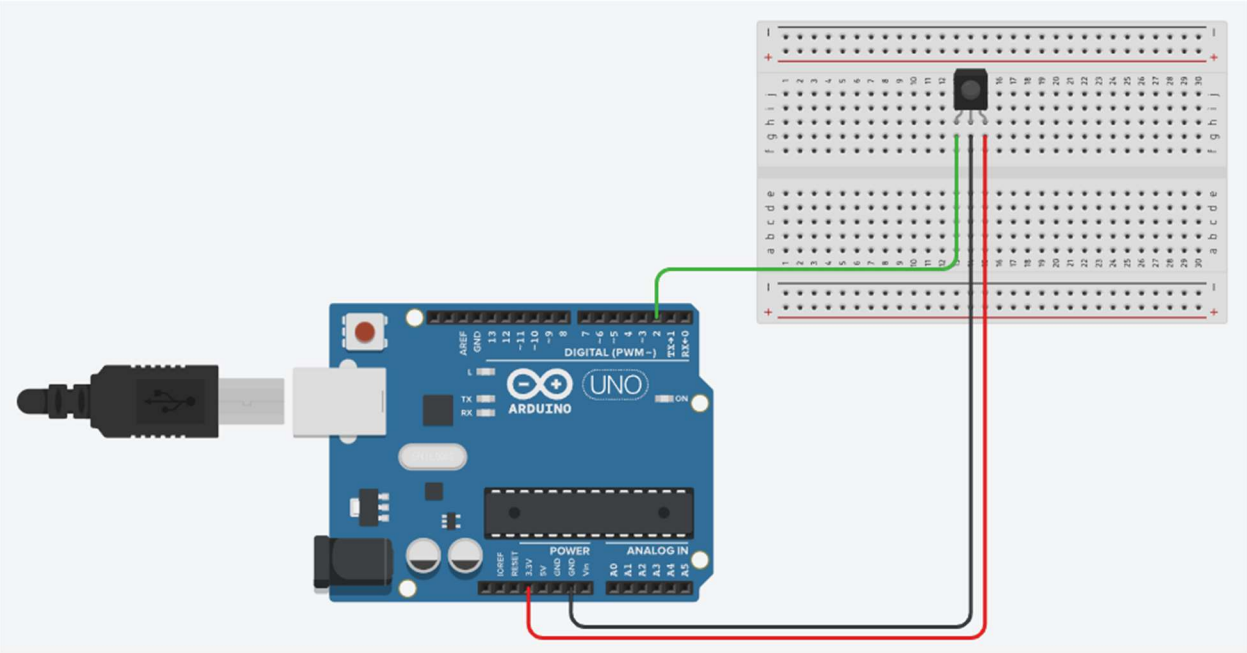
B) IMPLEMENTATION :-

CONFIGURE IR REMOTE –

We all use IR remotes on a regular basis, such as when we watch TV or turn on our air conditioner. The remote features an IR transmitter that sends out Infrared radiation of varying frequencies depending on which button is pressed on the remote. The IR receiver then receives and decodes this signal to determine which button you have pressed. This project will be built using the same concept.

Each button now has its own frequency to identify it from the others. That means, before utilizing an IR remote, you must understand the frequency of each button. To determine the frequencies of each button, we will construct a circuit with an IR remote, a sensor, and an Arduino. Connect the sensor's left terminal to Arduino's digital pin 2, the middle terminal to GND, and the right terminal to the +3.3V pin.

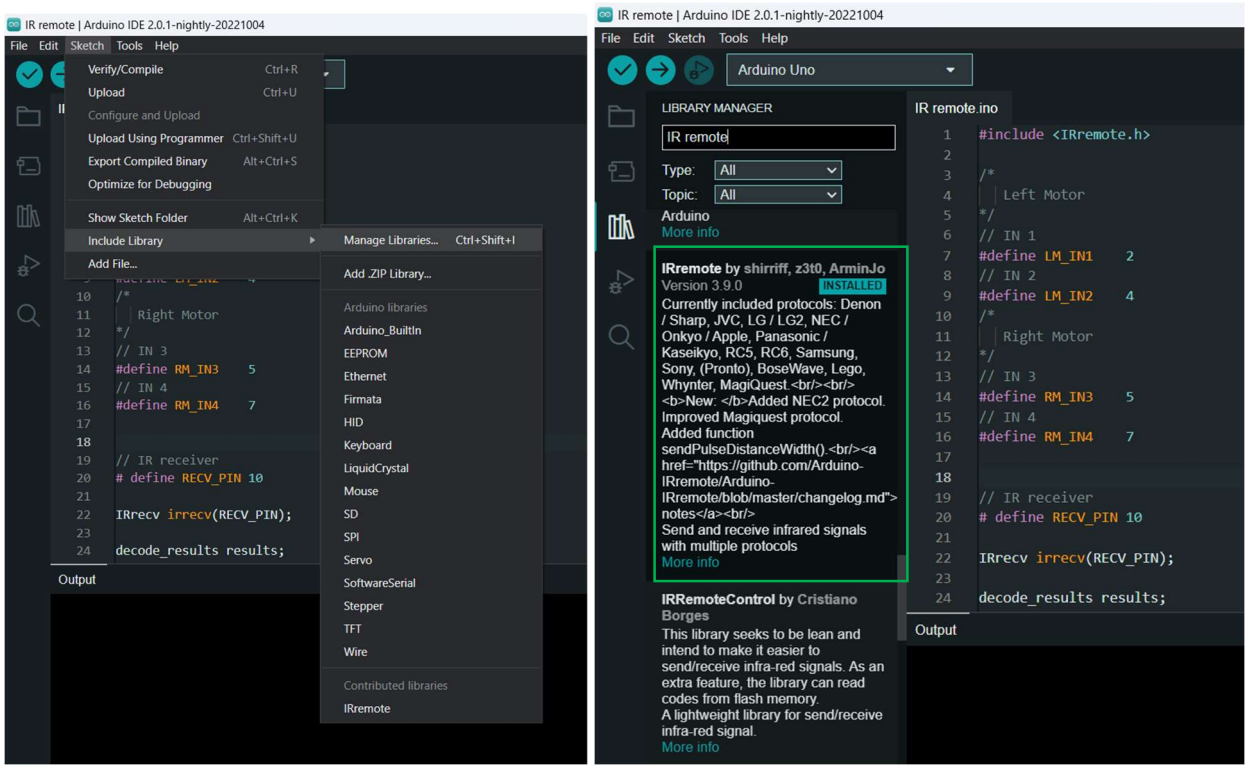
Circuit Connection –



Download a Library for IR Remote –

To download the library file for IR remote, open ARDUINO IDE Software on your pc, go to SKECTH and click on MANAGE LIBRARIES. Type IR REMOTE and search for IRremote by shirriff, z3t0, Version 3.9.0. Then download and install the library.

Upload the code below to your arduino board. Now open the serial monitor and press any button on the IR remote. As you can see, the corresponding Hex code for the button is printed on the Serial Monitor. Note down the hexadecimal codes for all the buttons so that you can program Arduino board to perform a certain function or task when it receives a particular hexadecimal code.



Arduino Code –

```
#include <IRremote.h>
const int RECV_PIN = 2;
IRrecv irrecv(RECV_PIN);
decode_results results;
void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  irrecv.blink13(true);
}
void loop(){
  if (irrecv.decode(&results)){
       Serial.println(results.value, HEX);
       irrecv.resume();
  }
}
```

IR REMOTE :-

HEXADECIMAL CODE FOR ALL THE BUTTONS ON THE IR REMOTE :-

Power On/Off – 1FE48B7

Mode – 1FE58A7

Mute – 1FE7887

Pause/Play – 1FE807F

Rewind – 1FE40BF

Fast forward – 1FEC03F

EQ – 1FE20DF

Decrease Volume – 1FEA05F

Increase Volume – 1FE609F

Zero – 1FEE01F

RPT – 1FE10EF

U/SD – 1FE906F

One – 1FE50AF

Two – 1FED827

Three – 1FEF807

Four – 1FE30CF

Five – 1FEB04F

Six – 1FE708F

Seven – 1FE00FF

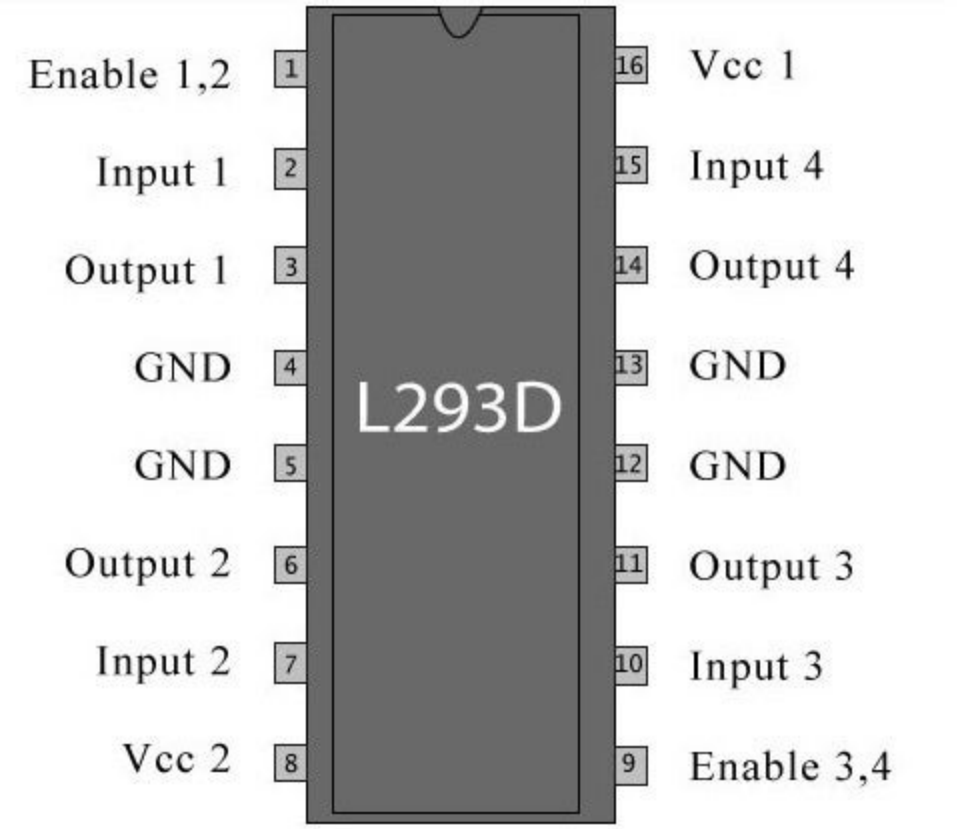
Eight – 1FEF00F

Nine – 1FE9867

Motor Driver (L293D IC) :-

The L293D is a 16 pin IC, with eight pins, on each side, dedicated to the controlling of a motor. There are 2 INPUT pins, 2 OUTPUT pins and 1 ENABLE pin for each motor. L293D consist of two H-bridge. H-bridge is the simplest circuit for controlling a low current rated motor.

Pin No.	**Pin Characteristics**
1 - Enable 1-2	When this is HIGH the left part of the IC will work and when it is low the left part won't work.
2 - INPUT 1	When this pin is HIGH the current will flow though output 1
3 - OUTPUT 1	This pin should be connected to one of the terminal of motor 4,5 - GND, ground pins
4,5 - GND	Ground pins
6 - OUTPUT 2	This pin should be connected to one of the terminal of motor
7 - INPUT 2	When this pin is HIGH the current will flow though output 2
8 - VCC2	This is the voltage which will be supplied to the motor.
9 - Enable 3-4	When this is HIGH the right part of the IC will work and when it is low the right part won't work.
10 - INPUT 3	When this pin is HIGH the current will flow though output 3
11 - OUTPUT 3	This pin should be connected to one of the terminal of motor
12,13 - GND	Ground pins
14 - OUTPUT 4	This pin should be connected to one of the terminal of motor
15 - INPUT 4	When this pin is HIGH the current will flow though output 4
16 - VCC1	This is the power source to the IC. So, this pin should be supplied with 5 V



CONFIGURE THE ROBOT -

Before we begin, let's look at the project's goal and how it works.

The robot is controlled manually by pressing the control buttons on the IR remote. We have used the following buttons for respective control.
2 -> Forward
4 -> Left
6 -> Right
7 -> Backward
5 -> Stop
1 -> Repeat
3 -> Delete
0 -> Reset

When you start the robot car for the first time after supplying power to Arduino, it automatically stores the sequence in its memory. The Repeat

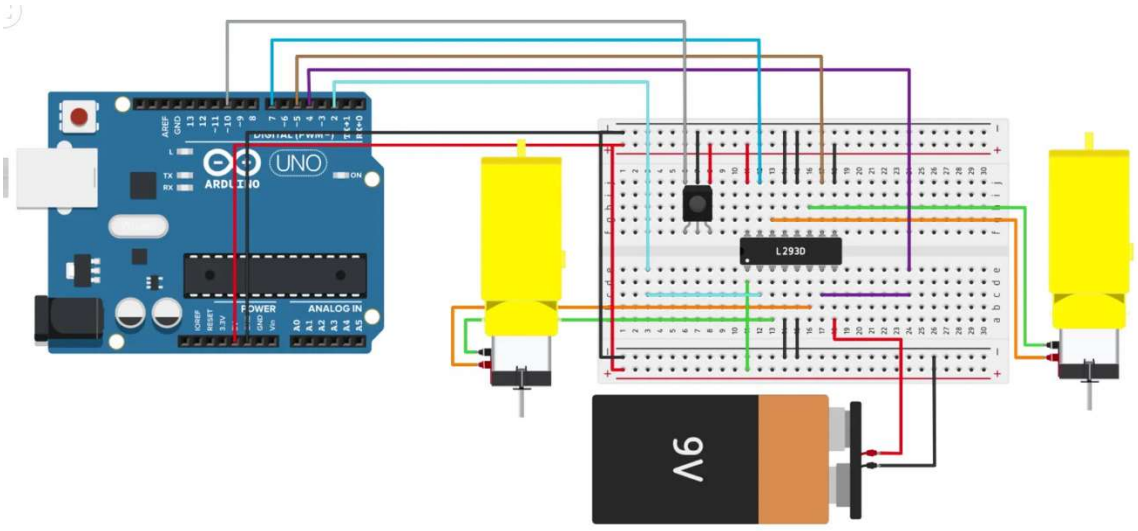
button on the IR remote has now been assigned to repeat this stored sequence. Additionally, you must first delete the previously stored sequence by pressing a button in order to store a new sequence.

An IR receiver is required to control your robot car with the IR remote. This infrared receiver will decode the received signal for the pressed button and send it to Arduino. Arduino serves as a controller for your Robot Car, and the program written in it provides the necessary logic to control the Motors. However, we all know that an Arduino cannot provide enough current to a DC motor. That is why you will also require a Motor driver here. You are free to use any motor driver you want. In this case, I'm using an L293D Motor driver IC to power this Robot car.

In addition to all of this, you can quickly reset Arduino by pressing the play button if the robot ever experiences a problem while it is in operation. Please be aware that you are free to substitute any button of your choosing for any of the operations listed below in the CODE section.

Circuit Diagram for the robot –

Connect your Arduino board and DC motors to the motor driver IC. I'm using an L293D integrated circuit, and the schematic is shown above. If you're using L293D, connect it as shown in the schematic. And now that we've completed the circuit connections, we can move on to the Arduino code:



Arduino Code – Normal IR Remote Control Car :-

```
#include <IRremote.h>

//Left Motor
#define LM_IN1    2
#define LM_IN2    4

//Right Motor
#define RM_IN3    5
#define RM_IN4    7

// IR receiver
# define RECV_PIN 10

IRrecv irrecv(RECV_PIN);

decode_results results;

//HEX codes for buttons
#define FWD      0x1FED827 // forward (2)
#define LFT      0x1FE30CF // left (4)
#define RGT      0x1FE708F // right (6)
#define BWD      0x1FEF00F // backward (8)
#define STOP     0x1FEB04F // stop (5)
#define RESET    0x1FEE01F // Resets the Arduino Board (0)

unsigned long int value=0;


/ ************Arduino Reset Pin***************/
#define RESET_PIN A0

void setup() {
  for (int i = 2; i <= 7; i++) {
    pinMode(i, OUTPUT);
  }
```

```
  Serial.begin(9600);
  Serial.println("Enabling IRin");
  irrecv.enableIRIn();
  Serial.println("Enabled IRin");

}

void loop() {
  if (irrecv.decode(&results)) {
    value = results.value;
    Serial.println(value, HEX);
    irrecv.resume();
    delay(200);
  }
  delay(100);
  check_Inst(value);
  value=0;
}

void check_Inst(long int value) {

  switch (value) {
    case FWD:
      go_Forward();
      break;
    case LFT:
      go_Left();
      break;
    case RGT:
      go_Right();
      break;
    case BWD:
      go_Backward();
      break;
    case STOP:
      go_Stop();
      break;
```

```
   case RESET:
     pinMode(RESET_PIN,OUTPUT);
     digitalWrite(RESET_PIN,HIGH);
     break;

   default:
     value = 0;
  }
}

void go_Forward() {
  movement_Inst_Fwd();
  delay(10);
}

void go_Left() {
  movement_Inst_Lft();
  delay(10);
}

void go_Right() {
  movement_Inst_Rgt();
  delay(10);
}

void go_Backward(){
  movement_Inst_Bwd();
  delay(10);
}

void go_Stop(){
  movement_Inst_Stp();
  delay(10);
}

/* These movement instruction are repeated several times in the code */
```

```
void movement_Inst_Fwd(void){
  Serial.println("Going_Forward");
  digitalWrite(LM_IN1,HIGH);
  digitalWrite(LM_IN2,LOW);
  digitalWrite(RM_IN3,HIGH);
  digitalWrite(RM_IN4,LOW);
  }

void movement_Inst_Lft(void){
  Serial.println("Going_Left");
  digitalWrite(LM_IN1,LOW);
  digitalWrite(LM_IN2,LOW);
  digitalWrite(RM_IN3,HIGH);
  digitalWrite(RM_IN4,LOW);
  delay(500);
  digitalWrite(LM_IN1,LOW);
  digitalWrite(LM_IN2,LOW);
  digitalWrite(RM_IN3,LOW);
  digitalWrite(RM_IN4,LOW);
  delay(500);
  }

void movement_Inst_Rgt(void){
  Serial.println("Going_Right");
  digitalWrite(LM_IN1,HIGH);
  digitalWrite(LM_IN2,LOW);
  digitalWrite(RM_IN3,LOW);
  digitalWrite(RM_IN4,LOW);
  delay(500);
  digitalWrite(LM_IN1,LOW);
  digitalWrite(LM_IN2,LOW);
  digitalWrite(RM_IN3,LOW);
  digitalWrite(RM_IN4,LOW);
  delay(500);

  }
```

```
void movement_Inst_Bwd(void){
  Serial.println("Going_Backward");
  digitalWrite(LM_IN1,LOW);
  digitalWrite(LM_IN2,HIGH);
  digitalWrite(RM_IN3,LOW);
  digitalWrite(RM_IN4,HIGH);
  }

void movement_Inst_Stp(void){
  Serial.println("Stopping");
  digitalWrite(LM_IN1,LOW);
  digitalWrite(LM_IN2,LOW);
  digitalWrite(RM_IN3,LOW);
  digitalWrite(RM_IN4,LOW);
  }
```

Arduino Code – EEPROM Path Memorizing (Permanent) :-

An EEPROM (electrically erasable programmable read-only memory) allows you to store small quantities of data indefinitely. This is extremely useful for keeping user preferences or storing tiny data sets when essential data must be retained even if the power is switched off. Almost all Arduino microcontroller boards include EEPROM memory built into the core chip, eliminating the need for additional hardware for tiny data retention and retrieval.

EEPROM max size - EEPROM does not have an endless storage capacity. It's usually rather little, so make sure you know the EEPROM size for your Arduino board. The EEPROM size of the Arduino Uno is 1024 bytes.

```
#include <IRremote.h>
#include <EEPROM.h>

//Left Motor
#define LM_IN1    2
#define LM_IN2    4

//Right Motor
#define RM_IN3    5
#define RM_IN4    7

// IR receiver
# define RECV_PIN 10

IRrecv irrecv(RECV_PIN);

decode_results results;

//HEX codes for buttons
#define FWD      0x1FED827 // forward (2)
#define LFT      0x1FE30CF // left (4)
#define RGT      0x1FE708F // right (6)
#define BWD      0x1FEF00F // backward (8)
#define STOP     0x1FEB04F // stop (5)
#define RPEAT    0x1FE50AF // repeat (1)
#define DEL      0x1FEF807 // delete (3)
#define PERST 0x1FE00FF //Temp memory to permanent memory(7)
#define PLAYEPROM 0x1FE9867// Execute the path stored in EEPROM(9)
#define RESET    0x1FEE01F // Resets the Arduino Board (0)


/************Global Variables and Arrays**************/

unsigned long int value = 0;
byte seq = 0;
byte seq_Array[50];
```

```
int fwd_Counter = -1;
int lft_Counter = -1;
int rgt_Counter = -1;
int bwd_Counter = -1;
int stp_Counter = -1;

unsigned long int current_Time0 = 0;// for FWD movement
unsigned long int current_Time1 = 0;// for LEFT movement
unsigned long int current_Time2 = 0;// for RIGHT movement
unsigned long int current_Time3 = 0;// for BWD movement
unsigned long int current_Time4 = 0;// for STOP

unsigned long int total_Fwd_Time[10];
unsigned long int total_Lft_Time[10];
unsigned long int total_Rgt_Time[10];
unsigned long int total_Bwd_Time[10];
unsigned long int total_Stp_Time[10];

/************Arduino Reset Pin**************/
#define RESET_PIN A0

void setup() {
 for (int i = 2; i <= 7; i++) {
  pinMode(i, OUTPUT);
 }
 Serial.begin(9600);
 Serial.println("Enabling IRin");
 irrecv.enableIRIn(); // Start the receiver
 Serial.println("Enabled IRin");
}

void loop() {
 if (irrecv.decode(&results)) {
  value = results.value;
  Serial.println(value, HEX);
  irrecv.resume();
  delay(200);
 }
```

```
  delay(100);
  check_Inst(value);
  value=0;

}

void check_Inst(long int value) {

  switch (value) {
    case FWD:
      go_Forward();
      delay(10);
      break;
    case LFT:
      go_Left();
      delay(10);
      break;
    case RGT:
      go_Right();
      delay(10);
      break;
    case BWD:
      go_Backward();
      delay(10);
      break;
    case STOP:
      go_Stop();
      delay(10);
      break;
    case RPEAT:
      go_In_Seq();
      delay(10);
      break;
    case DEL:
      del_From_Local_Mem();
      delay(10);
      break;
```

```
   case PERST:
    write_To_Permt_Mem();
    delay(10);
    break;
   case PLAYEPROM:
    Read_Permt_Mem();
    delay(10);
    break;
   case RESET:
    pinMode(RESET_PIN, OUTPUT);
    digitalWrite(RESET_PIN,HIGH);
    break;

    default:
    value = 0;
  }
}

void go_Forward() {
  movement_Inst_Fwd();

  current_Time0 = millis();
  int i = seq_Array[(seq - 1)];
  switch (i) {
   case 2:
    total_Lft_Time[lft_Counter + 1] = (current_Time0 - current_Time1);
    lft_Counter++;
    break;

   case 3:
    total_Rgt_Time[rgt_Counter + 1] = (current_Time0 - current_Time2);
    rgt_Counter++;
    break;

   case 4:
    total_Bwd_Time[bwd_Counter + 1] = (current_Time0 - current_Time3);
    bwd_Counter++;
    break;
```

```
   case 5:
    total_Stp_Time[stp_Counter + 1] = (current_Time0 - current_Time4);
    stp_Counter++;
    break;
  }

  seq_Array[seq] = 1;
  seq++;
}

void go_Left() {
  movement_Inst_Lft();

  current_Time1 = millis();
  int i = seq_Array[(seq - 1)];
  switch (i) {
   case 1:
    total_Fwd_Time[fwd_Counter + 1] = (current_Time1 - current_Time0);
    fwd_Counter++;
    break;

   case 3:
    total_Rgt_Time[rgt_Counter + 1] = (current_Time1 - current_Time2);
    rgt_Counter++;
    break;

   case 4:
    total_Bwd_Time[bwd_Counter + 1] = (current_Time1 - current_Time3);
    bwd_Counter++;
    break;

   case 5:
    total_Stp_Time[stp_Counter + 1] = (current_Time1 - current_Time4);
    stp_Counter++;
    break;
  }
```

```
  seq_Array[seq] = 2;
  seq++;
}

void go_Right() {
  movement_Inst_Rgt();

  current_Time2 = millis();
  int i = seq_Array[(seq - 1)];
  switch (i) {
    case 1:
      total_Fwd_Time[fwd_Counter + 1] = (current_Time2 - current_Time0);
      fwd_Counter++;
      break;

    case 2:
      total_Lft_Time[lft_Counter + 1] = (current_Time2 - current_Time1);
      lft_Counter++;
      break;

    case 4:
      total_Bwd_Time[bwd_Counter + 1] = (current_Time2 - current_Time3);
      bwd_Counter++;
      break;

    case 5:
      total_Stp_Time[stp_Counter + 1] = (current_Time2 - current_Time4);
      stp_Counter++;
      break;
  }

  seq_Array[seq] = 3;
  seq++;
}

void go_Backward() {
  movement_Inst_Bwd();
```

```
 current_Time3 = millis();
 int i = seq_Array[(seq - 1)];
 switch (i) {
  case 1:
   total_Fwd_Time[fwd_Counter + 1] = (current_Time3 - current_Time0);
   fwd_Counter++;
   break;

  case 2:
   total_Lft_Time[lft_Counter + 1] = (current_Time3 - current_Time1);
   lft_Counter++;
   break;

  case 3:
   total_Rgt_Time[rgt_Counter + 1] = (current_Time3 - current_Time2);
   rgt_Counter++;
   break;

  case 5:
   total_Stp_Time[stp_Counter + 1] = (current_Time3 - current_Time4);
   stp_Counter++;
   break;
 }
 seq_Array[seq] = 4;
 seq++;
}

void go_Stop() {
 movement_Inst_Stp();

 current_Time4 = millis();
 int i = seq_Array[(seq - 1)];
 switch (i) {
  case 1:
   total_Fwd_Time[fwd_Counter + 1] = (current_Time4 - current_Time0);
   fwd_Counter++;
   break;
```

```
   case 2:
    total_Lft_Time[lft_Counter + 1] = (current_Time4 - current_Time1);
    lft_Counter++;
    break;

   case 3:
    total_Rgt_Time[rgt_Counter + 1] = (current_Time4 - current_Time2);
    rgt_Counter++;
    break;

   case 4:
    total_Bwd_Time[bwd_Counter + 1] = (current_Time4 - current_Time3);
    bwd_Counter++;
    break;
  }

  seq_Array[seq] = 5;
  seq++;
}

void go_In_Seq(void) {
  value = 0;
  for (int i = 0; i < (seq + 1); i++) {
   int value1 = 0;
   value1 = seq_Array[i];
   switch (value1) {

    case 1:
     static int j = 0;
     go_Forward_Seq(j);
     j++;
     break;
    case 2:
     static int k = 0;
     go_Left_Seq(k);
     k++;
     break;
```

```
    case 3:
      static int l = 0;
      go_Right_Seq(l);
      l++;
      break;
    case 4:
      static int m = 0;
      go_Backward_Seq(m);
      m++;
      break;
    case 5:
      static int n = 0;
      go_Stop_Seq(n);
      n++;
      break;
    default:
      j = 0; k = 0; l = 0; m = 0; n = 0;
   }
  }
}


void del_From_Local_Mem() {
  fwd_Counter = -1;
  lft_Counter = -1;
  rgt_Counter = -1;
  bwd_Counter = - 1;
  stp_Counter = - 1;

  for (int i = 0; i < 10; i++) {
    total_Fwd_Time[i] = 0;
    total_Lft_Time[i] = 0;
    total_Rgt_Time[i] = 0;
    total_Bwd_Time[i] = 0;
    total_Stp_Time[i] = 0;
  }
```

```
 for (int i = 0; i < 50; i++) {
  seq_Array[i] = 0;
 }
 seq = 0;
}

/*********************************************************************
This function copy the data from the arrays to the EEPROM(permanent
memory)
*********************************************************************/
void write_To_Permt_Mem(){
 EEPROM.write(100,seq);

 for(int i=0; i<seq; i++){
 EEPROM.write(2*i,seq_Array[i]);
 }

 for(int i=1; i<seq+1; i++){
 if(seq_Array[i-1]==1){
  static byte a=0;
  EEPROM.write(2*i-1,(total_Fwd_Time[a])/1000
  a++;
  }
 else if(seq_Array[i-1]==2){
  static byte b=0;
  EEPROM.write(2*i-1,(total_Lft_Time[b])/1000);
  b++;
  }
 else if(seq_Array[i-1]==3){
  static byte c=0;
  EEPROM.write(2*i-1,(total_Rgt_Time[c])/1000);
  c++;
  }
 else if(seq_Array[i-1]==4){
  static byte d=0;
  EEPROM.write(2*i-1,(total_Bwd_Time[d])/1000);
  d++;
  }
```

```
 else if(seq_Array[i-1]==5){
  static byte e=0;
  EEPROM.write(2*i-1,(total_Stp_Time[e])/1000);
  e++;
  }
 }
}

/*********************************************************************
This function reads the stored sequence from the EEPROM(permanent
memory)
*********************************************************************/
void Read_Permt_Mem(){
 // Read from permanent memory
  byte x = EEPROM.read(100);
  for(int i=0; i<x+1; i++){
   byte r = EEPROM.read(2*i);
   switch(r){
    case 1:
     movement_Inst_Fwd();
     break;
    case 2:
     movement_Inst_Lft();
     break;
    case 3:
     movement_Inst_Rgt();
     break;
    case 4:
     movement_Inst_Bwd();
     break;
    case 5:
     movement_Inst_Stp();
     break;
    }
   delay((EEPROM.read(i+1))*1000);
  }
 }
```

```
/*****************************************************************************
These function moves the car in a direction for the time specified/stored in
the total_x_time array
*****************************************************************************/
void go_Forward_Seq(int j) {
  movement_Inst_Fwd();
  delay(total_Fwd_Time[j]);
}

void go_Left_Seq(int k) {
  movement_Inst_Lft();
  delay(total_Lft_Time[k]);
}

void go_Right_Seq(int l) {
  movement_Inst_Rgt();
  delay(total_Rgt_Time[l]);
}

void go_Backward_Seq(int m) {
  movement_Inst_Bwd();
  delay(total_Bwd_Time[m]);
}

void go_Stop_Seq(int n) {
  movement_Inst_Stp();
  delay(total_Stp_Time[n]);
}


/*****************************************************************************
These movement instructions are repeated(required) several times in the
code
*****************************************************************************/
void movement_Inst_Fwd(void) {
  Serial.println("Going_Forward");
  digitalWrite(LM_IN1, HIGH);
```

```
  digitalWrite(LM_IN2, LOW);
  digitalWrite(RM_IN3, HIGH);
  digitalWrite(RM_IN4, LOW);
}

void movement_Inst_Lft(void) {
  Serial.println("Going_Left");
  digitalWrite(LM_IN1, LOW);
  digitalWrite(LM_IN2, LOW);
  digitalWrite(RM_IN3, HIGH);
  digitalWrite(RM_IN4, LOW);
  delay(500);
  digitalWrite(LM_IN1, LOW);
  digitalWrite(LM_IN2, LOW);
  digitalWrite(RM_IN3, LOW);
  digitalWrite(RM_IN4, LOW);
  delay(500);
}

void movement_Inst_Rgt(void) {
  Serial.println("Going_Right");
  digitalWrite(LM_IN1, HIGH);
  digitalWrite(LM_IN2, LOW);
  digitalWrite(RM_IN3, LOW);
  digitalWrite(RM_IN4, LOW);
  delay(500);
  digitalWrite(LM_IN1, LOW);
  digitalWrite(LM_IN2, LOW);
  digitalWrite(RM_IN3, LOW);
  digitalWrite(RM_IN4, LOW);
  delay(500);
}

void movement_Inst_Bwd(void) {
  Serial.println("Going_Backward");
  digitalWrite(LM_IN1, LOW);
  digitalWrite(LM_IN2, HIGH);
```

```
  digitalWrite(RM_IN3, LOW);
  digitalWrite(RM_IN4, HIGH);
}

void movement_Inst_Stp(void) {
  Serial.println("Stopping");
  digitalWrite(LM_IN1, LOW);
  digitalWrite(LM_IN2, LOW);
  digitalWrite(RM_IN3, LOW);
  digitalWrite(RM_IN4, LOW);
}
```

Advantages :-

- You don't have to mow your lawn or hire someone to do so.
- Lawn mower is great in tight spaces
- The mower can be of any size and shape
- The length of the lawn can be adjusted by moving the blade into different positions
- The maintenance is every low and affordable
- Scheduled mowing but must look out for weather conditions

Disadvantages :-

- Cannot operate the lawn mower using your phone
- The mower has to be placed in the exact position and angle in order to follow the desired path.
- The lawn has to be dry for perfect length
- Cannot detect obstacles
- Long time of mowing as it is automated
- Initially the path for the robot must be set manually by driving the mower around the lawn.

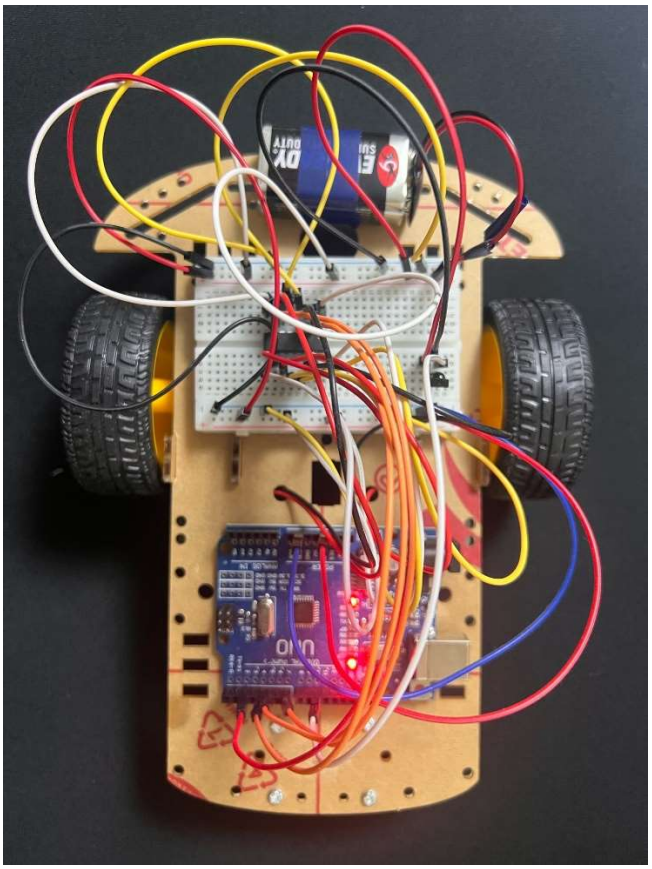
TIMELINE :-

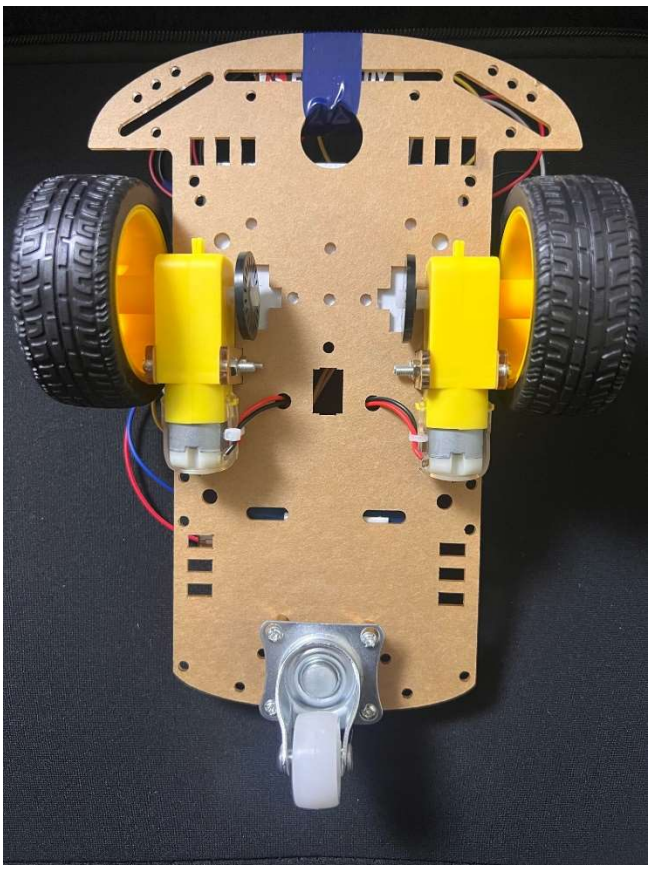

MILESTONE NO. 1
1. Reviews old project
1
AUGUST 2022
SEPTEMBER 2022
2
MILESTONE NO. 2
2. Collecting data from the internet
MILESTONE NO. 3
3. Sets deadline
3
SEPTEMBER 2022
OCTOBER 2022
4
MILESTONE NO. 4
4. Working with the Lawn Mower
MILESTONE NO. 5
5. Ordering the components
5
OCTOBER 2022
NOVEMBER 2022
6
MILESTONE NO. 6
6. Setting up the components
MILESTONE NO. 7
7.Vision to the final line
7
NOVEMBER 2022
NOVEMBER 2022
8
MILESTONE NO. 8
8. Testing the mower
MILESTONE NO. 9
9. Implementing the Code into the Arduino
9
NOVEMBER 2022
NOVEMBER 2022
10
MILESTONE NO. 10
10. debugging the code
MILESTONE NO. 11
11. Setting up for the Finals
11
DECEMBER 2022
DECEMBER 2022
12
MILESTONE NO. 12
12. Finally completing the project

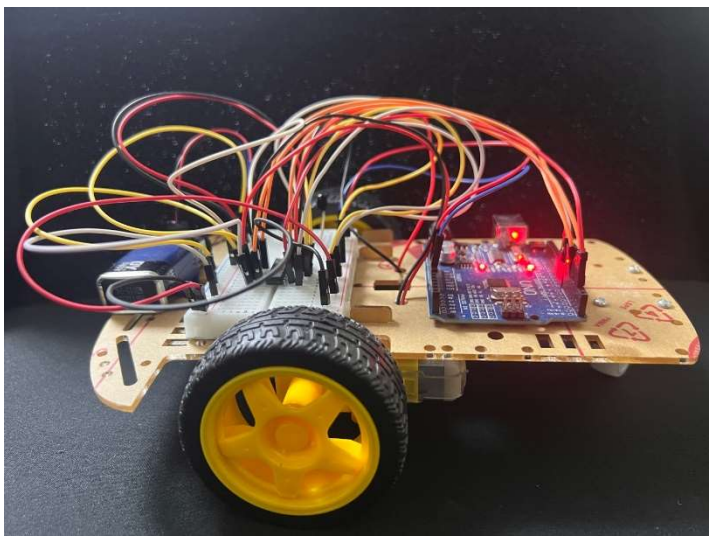
BUDGET :-

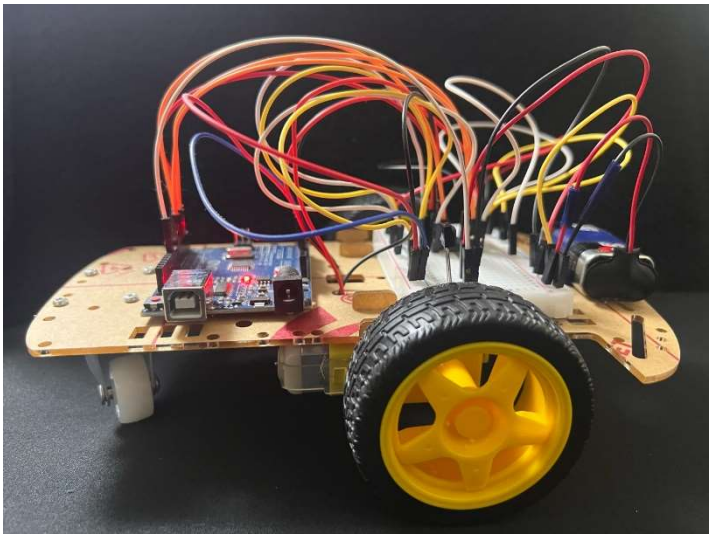
S. NO	COMPONENT NAME	DESCRIPTION	PRICE
1	ARDUINO UNO	Development Board	$ 49.99
2	IR REMOTE AND RECEIVER	Transmitter	$ 19.99
3	9V BATTERY	To power the DC motor	$ 14.99
4	TWO-GEARED DC MOTOR	TT DC Gearbox Motor	$ 34.99
5	MOTOR DRIVER (L293D IC)	Quadruple half-H drivers	$ 12.99
6	ROBOT WHEELS	TT DC Gearbox Motor	$ 19.99
7	ROBOT CHASSIS	The base of the lawn mover	$ 29.99
8	CONNECTING WIRES	Standard 2.54mm	$ 9.99
9	BREADBOARD	Breadboard, Material: ABS	$ 14.99
10	OTHER EXPENSES		$ 40.00
	TOTAL		$ 250

RESULTS :-









CONCLUSION :-

Grass cutting at schools, sports tracks, fields, factories, hotels, public centers, and so on was traditionally done with a cutlass. Although this approach requires hand cutting, it has several advantages. Using the manual cutting approach, precision in cutting level was also noticed. This work is concerned with the cutting of green (shrubs, grass, flowers, and tree leaves), as well as the design of the machine, its efficiency, stiffness, mode of operation, and material selection. The design allows for increased flexibility and interchangeability. The goal of this activity includes, but is not limited to, lowering the cost of cutting and beautifying the environment.

In this report, we have concluded that the proposed design is feasible. The biggest issues we found while working on the project is time and cost constraint. While moving forward into our research we have found it more difficult to finish in its entirely, as our project has a deep amount of complexity. Even with all the issues we have faced while working on the project, we were able to complete a Path Memorizing robot. The robot is built using wireless IR remote control using arduino, and we have used Arduino's inbuilt EEPROM to tore the path forever to its memory.

REFERENCES :-

https://www.reddit.com/r/mildlyinteresting/comments/6g2275/the_pattern_on_this_lawn_makes_it_look_3d/

https://www.brickhousesecurity.com/gps-trackers/gps-accuracy/

https://www.scag.com/pro-tip/lawn-striping-and-lawn-patterns/

https://www.lawnstarter.com/ny

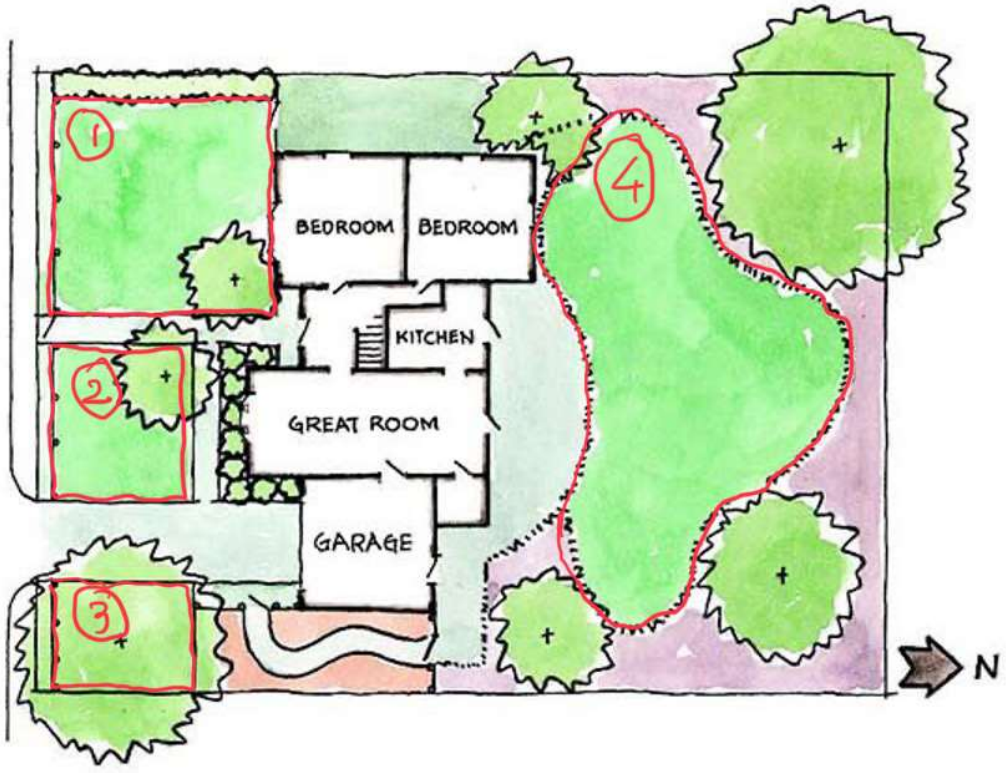
http://ceng786-hw1-e162702.blogspot.com/2013/10/tangent-bug-planning-algorithm-with.html

FUTURE PLANS :-

Memorizing Multiple Paths -

The additions of storing multiple paths will be a huge advantage because every lawn area won't be simple like a square. Some lawns which have uneven shapes can be mowed easily when we can have a robot that can store multiple paths, so that the lawn can be divided into parts and stored in the mower. By selecting a particular function by pressing a button on IR remote the corresponding path must be executed.

For example, as shown and mentioned in the below figure, we must be able to store multiple paths like path 1, 2, 3, 4 and when the mover is in lawn 3 and we give the command of executing path 3, the mower must execute the path 3.



Change the path execution method –

Instead of using path memorizing for the path execution, we can use Pixhawk & GPS module so that the mower can operate using GPS. The mission planner software helps in planning the route when the user gives 3 or more inputs

Specifications –

- 32-bit ARM Cortex M4 core with FPU
- 168 Mhz/256 KB RAM/ 2 MB Flash
- 32-bit failsafe co-processor
- 5x UART serial ports, 1 high-power capable, 2 with HW flow control
- 3.3V and 6.6V ADC inputs

Software –

Mission Planner - ArduPilot