German University in Cairo

Mechatronics Engineering (MCTR601)

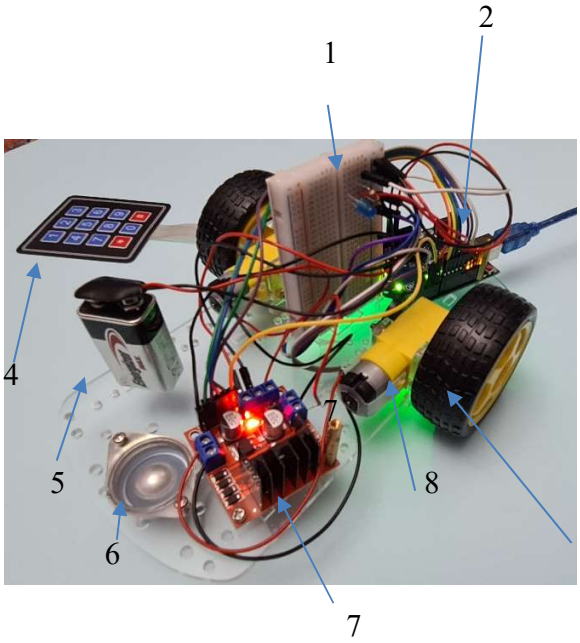
Coordinate Tracking Car
Team 7

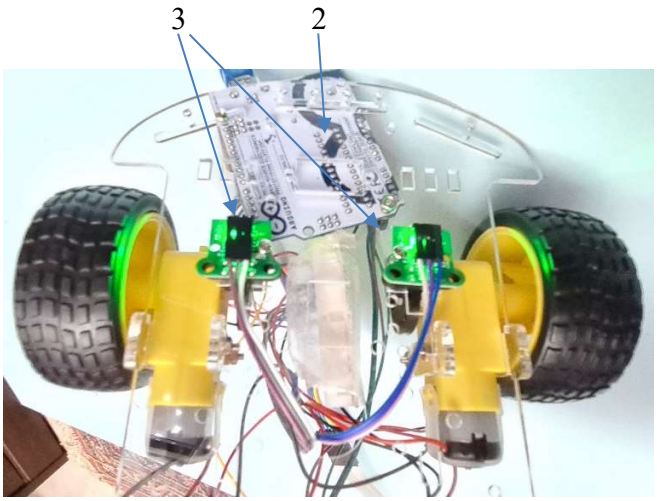
Name	ID	Lab Number
Ahmed Khaled Mady	46-798	55
Amro Hassan Abdellatif	46-3027	53
Mostafa Ahmed Kashaf	46-5456	53

Table of Contents

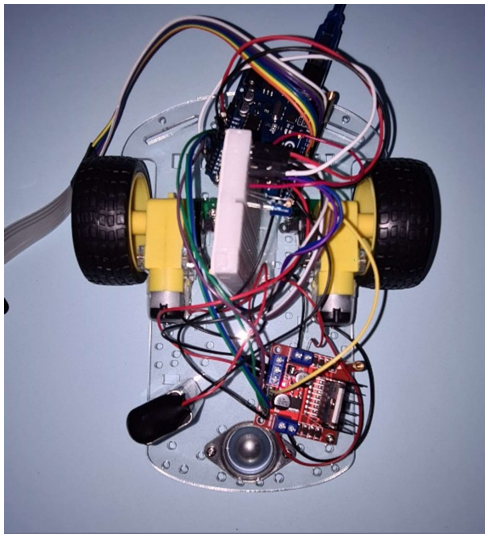
1. Project Description

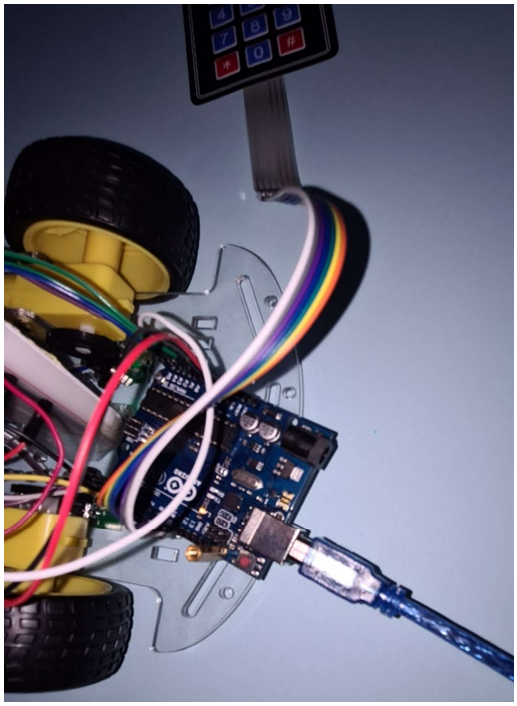
The micro-controlled car will be awaiting an input in the form of x-y coordinates through a keypad(3x4). The objective of the car is to reach the desired location. For the car to reach the location, an encoder that measures the number of revolutions required by the motor to reach its destination is used. The output of the encoder will be received by the microcontroller as x-y coordinates to maintain closed-loop feedback control. These x-y coordinates will be calculated using a certain algorithm to be able to know the required speed (v) and rotational torque (θ) of the car. Then the speed and rotational torque will be translated into angular velocity for each wheel (ω_L,ω_R) to accurately reach the destination. When the car reaches its destination, the car will stop.
The car uses a DC gear motor as an actuator and an Infrared motor speed sensor that will be fed back to Arduino's ATmega328p MCU to maintain a closed loop.
The car will move forward using 2 wheels that are controlled by the motor driver that is controlled by the Arduino development board PWM modes. During turns, left or right, the steering technique is made by increasing the speed of a wheel while keeping the other constant. The turn is made by the single caster wheel.
The car changes its's angle by measuring the arctan of the y coordinate/ x coordinate and multiplying it with the pulses needed by the motor to rotate. The car is assumed to always given positive x and y coordinates (1^{ST} Quadrant) so the right wheel is always given no PWM signal unlike the left wheel which will cause the rotation.
The x-y coordinates are then used to calculate the needed moving distance, or hypotenuse.

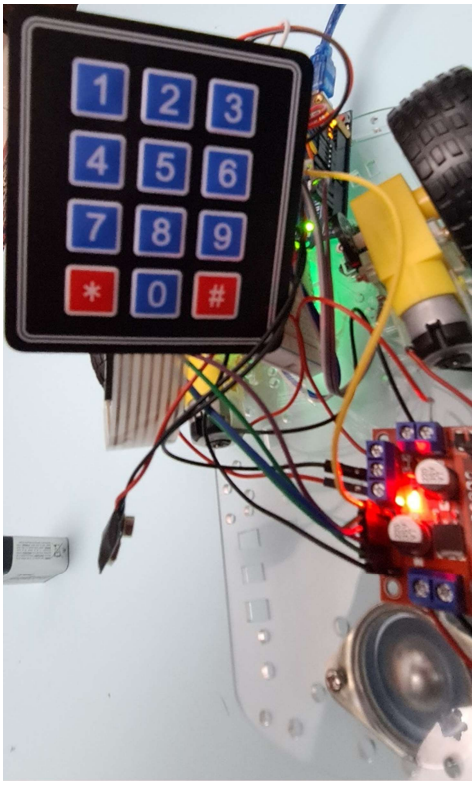


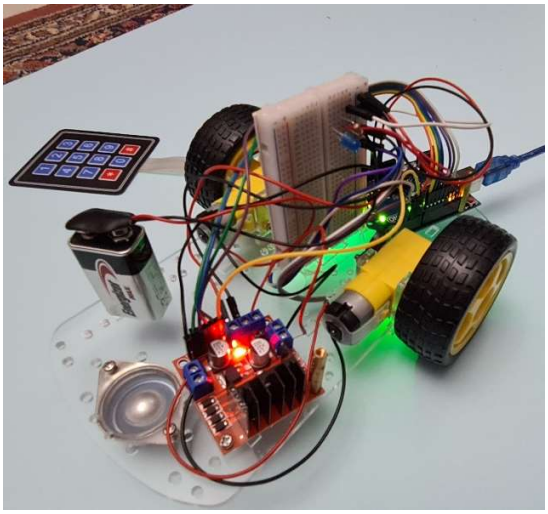


Part number	Name
1	Breadboard
2	Arduino UNO
3	LM393 IR Speed Sensor
4	Keypad
5	9V Battery
6	Caster Wheel
7	L298N Motor Driver
8	DC Gear Motor
9	Wheels






1 2 3
4 5 6
7 8 9
* 0 #



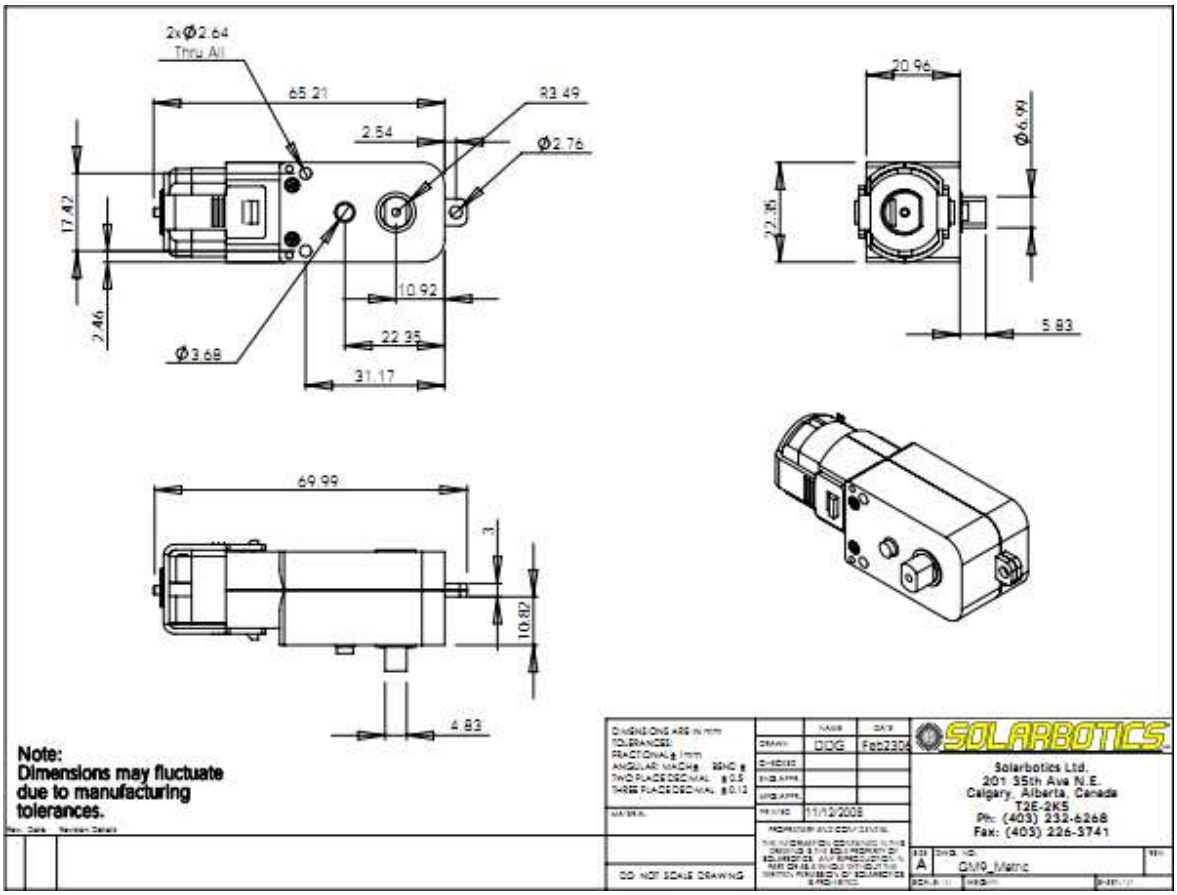
2. Methodology

2.1 Mechanical design

DC Gear Motor:

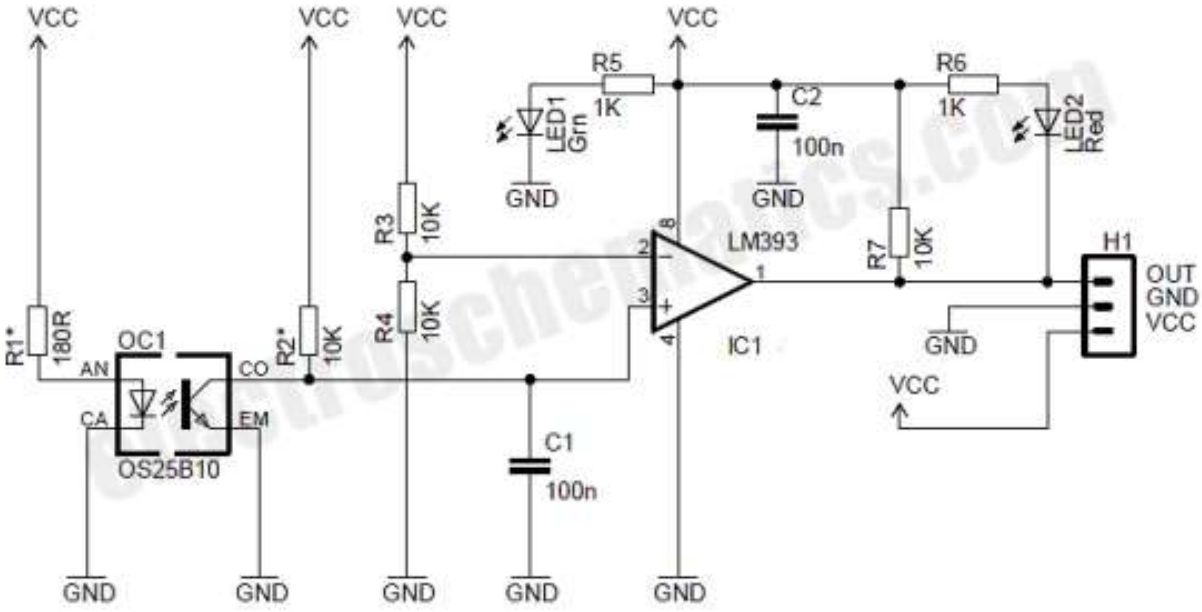
Specifications:

- Description: 90-degree
- Gear ratio: 143:1
- 3 V Operation:
  - No-load RPM: ~ 40
  - No-load current: ~ 50 mA
  - Stall current: ~ 400 mA
  - Stall torque: ~ 44 oz-in
- 6 V Operation:
  - No-load RPM: ~ 78
  - No-load current: ~ 52 mA
  - Stall current: ~ 700 mA
  - Stall torque: ~ 76 oz-in
- Weight: 33 g

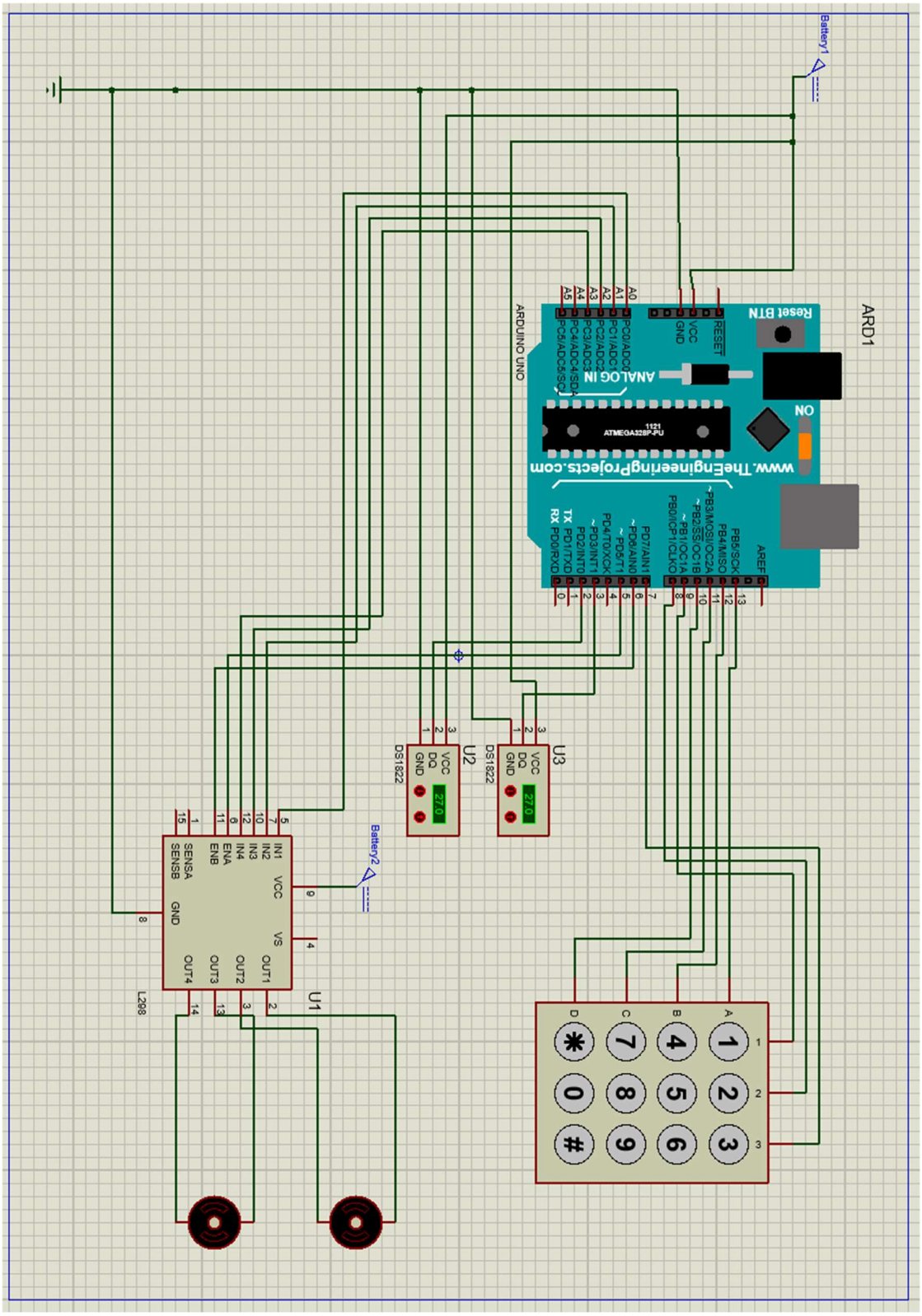


LM393 IR Speed Sensor

Note that one side of the optical sensor holds s a light emitting diode, and a photo transistor on the other side. If the visual path is not blocked, phototransistor would conduct, but when something blocked the light falling on the photo transistor it wouldn't conduct. So, an encoder disc (with a proper encoder resolution) can be placed in the slot of the optical sensor to count rotation rate of a connected wheel/motor. The encoder disc (also known as index wheel) in the above image have multiple slots/holes.



2.2 Electrical design

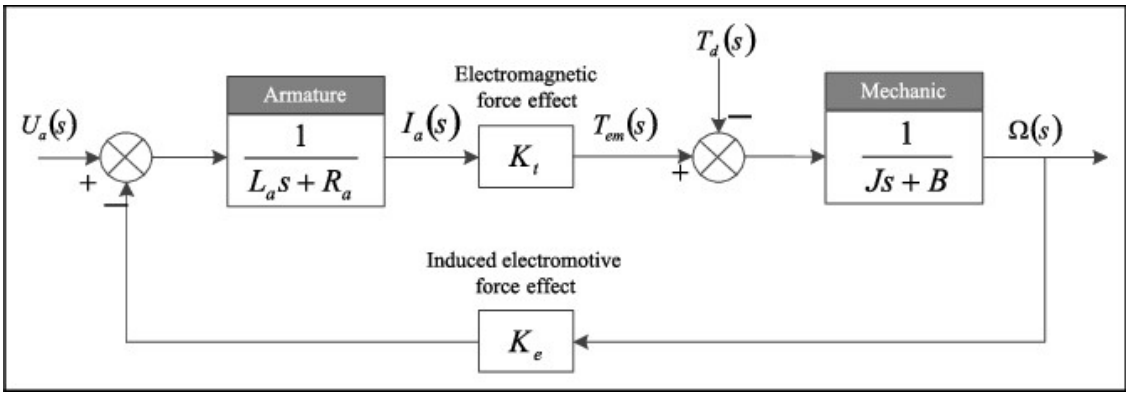


2.3 Control

1.3.1. Modeling

a. DC MOTOR:
From the datasheet of the DC geared motor, the following parameters were obtained:
(J) Moment of inertia of the rotor 0.1 kg.m^2
(B) Motor viscous friction constant 0.3550510258 N.m.s
(Ke) Electromotive force constant 0.25 V/rad/sec
(Kt) Motor torque constant 0.01 N.m/Amp
(R) Electric resistance 0.4222474487 Ohm
(L) Electric inductance 0.05 H
Transfer Function:

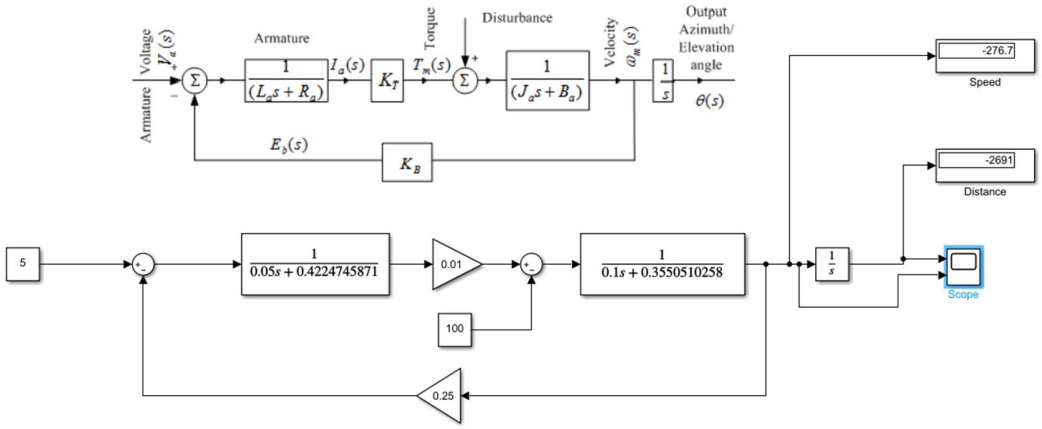
1.3.2. Analysis



Note that this transfer function outputs angular velocity (ω), but our purposes are revolutions, so an integrator was added at the output.
On MATLAB, the transfer function was generated

```
L_a = 0.05;
J_m = 0.1;
B = 0.3550510258;
R_a = 0.4224744871;
K_e = 0.25;
K_t = 0.01;
Armature= tf([1],[L_a R_a]);
A = series(K_t,Armature);
Mechanical = tf([1],[J_m B]);
M = series(A,Mechanical);
W = feedback(M,K_e);
G = series(W,tf([1],[1 0])
```

SIMULINK DIAGRAM:



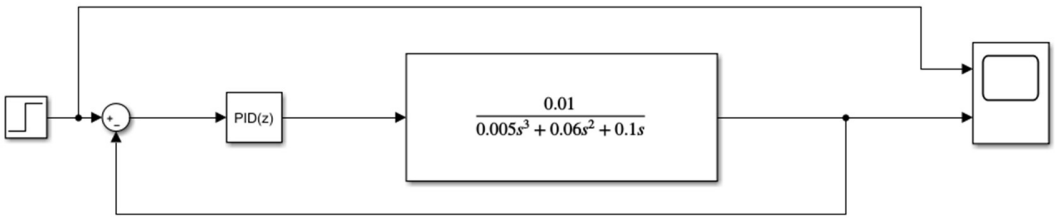
Plant:
G =

$$\frac{0.01}{0.005\,s^3 + 0.06\,s^2 + 0.1525\,s}$$

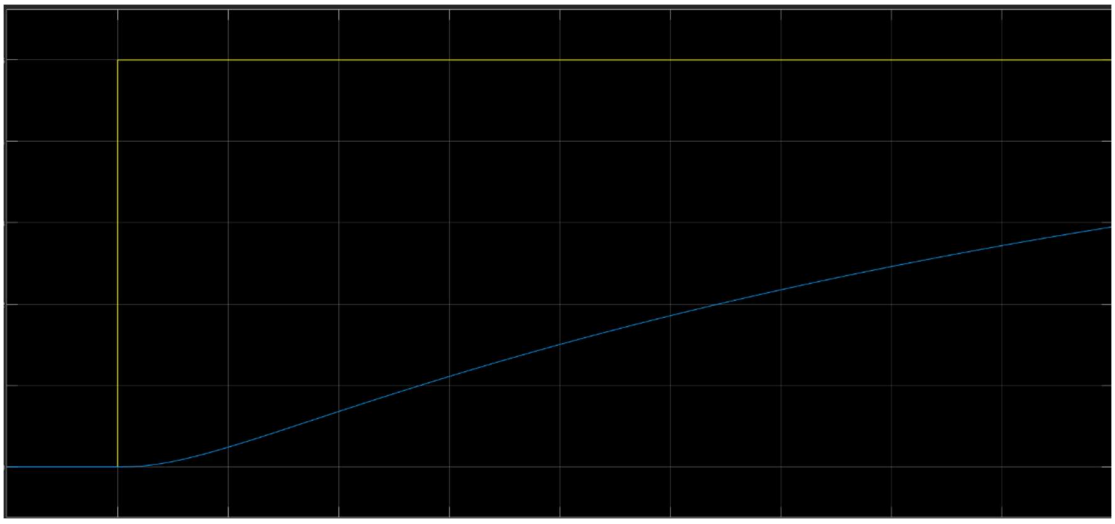
Continuous-time transfer function.

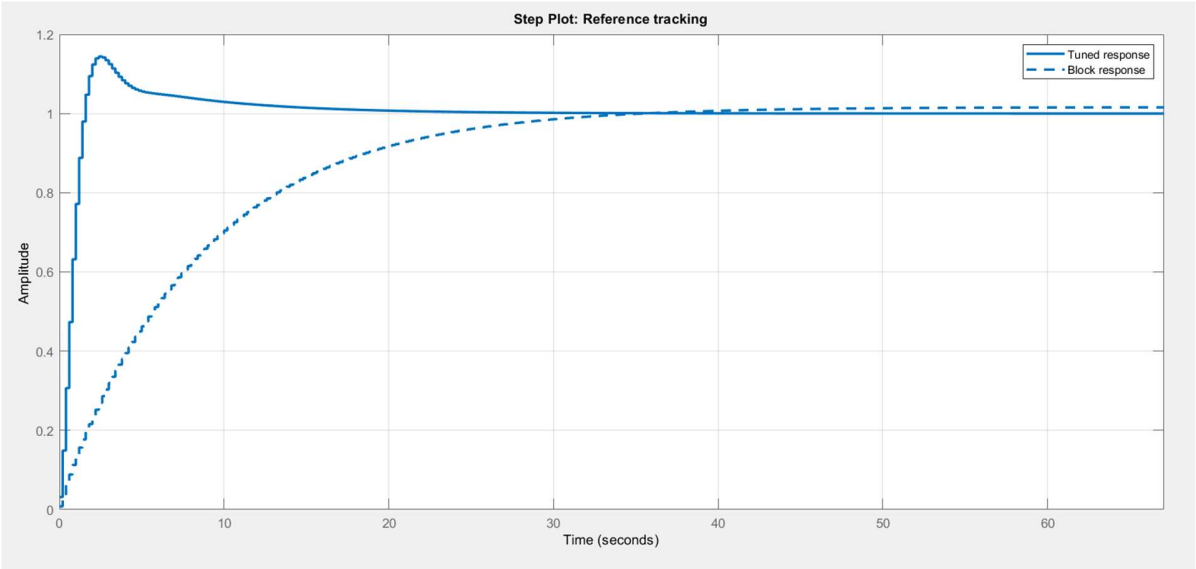
1.3.3. Controller Design

To design our controller, we modeled our system on SIMULINK:

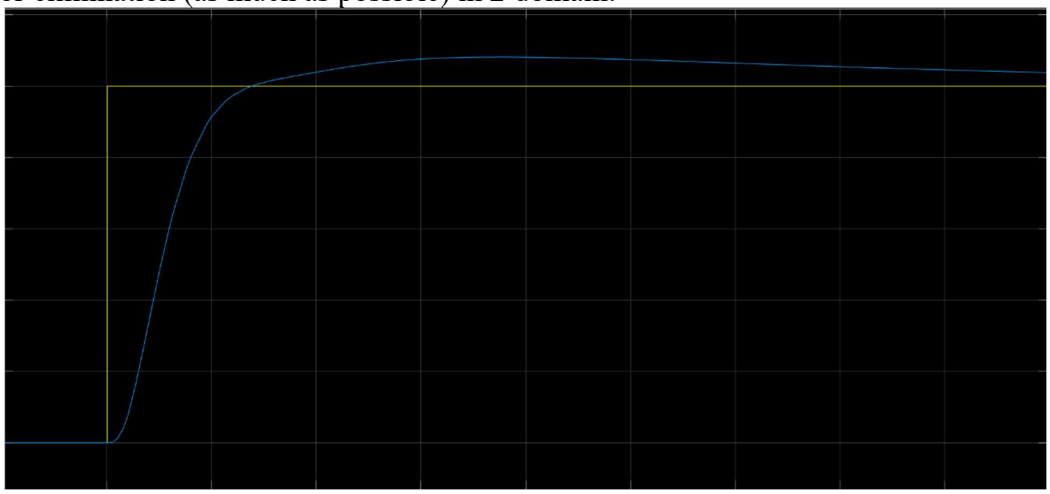


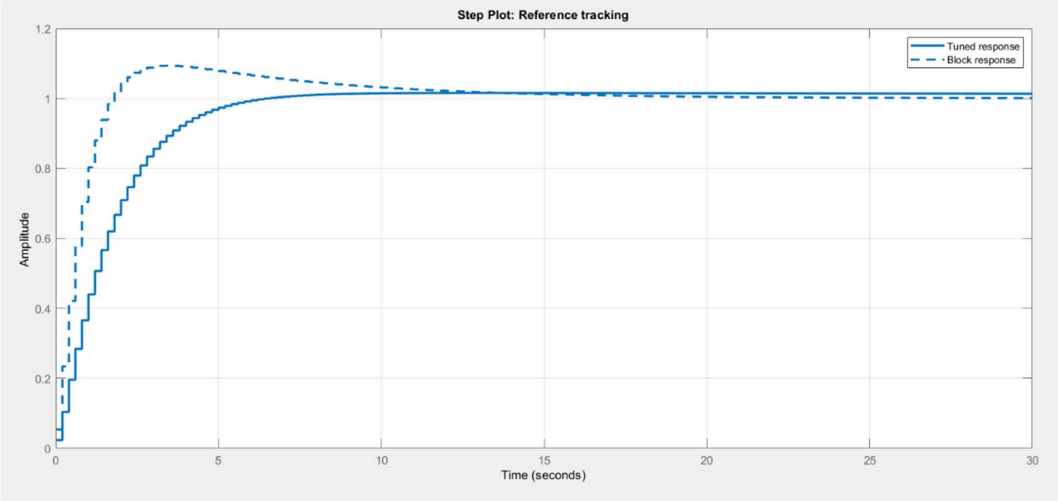
Running the simulation with reference tracking without tuning our controller gives:

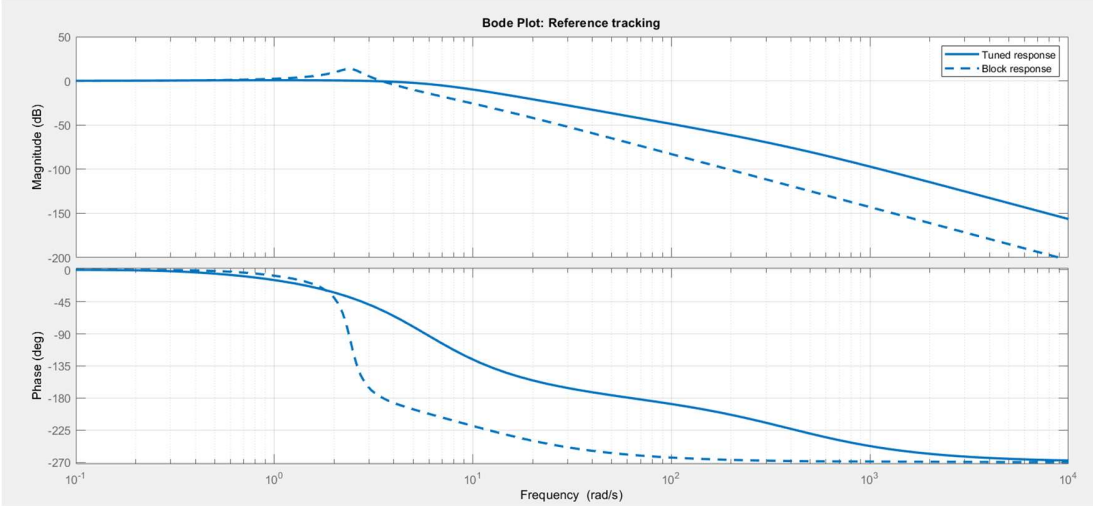




After tuning and optimizing the rise time, settle time, overshoot, stability, and steady-error elimination (as much as possible) in z-domain:







Control Law with a PID Compensator:
$K_P + K_I \frac{1}{z-1} + K_D \frac{z-1}{z}$ with
Kp=13.83;
Kd=7.3;
Ki=2;

$$13.83 + 2\ \frac{1}{z-1} + 7.3\ \frac{z-1}{z}$$

2.4 Programming

```
#include <math.h>
#include <Keypad.h>
//Keypad
const byte ROWS = 4;
const byte COLS = 3;
char hexaKeys[ROWS][COLS] =
{
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};
byte rowPins[ROWS] = {13, 12, 11, 10};
byte colPins[COLS] = {9, 8, 7};
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);
String inputString;
long inputInt;
//Flags
char inputTaken = 1;
char rotated = 1;
char xDone = 1;
char moved1 = 1;
char moved2 = 1;
//Encoder reading vatriables
unsigned int rpm1 = 0;
unsigned int rpm2 = 0;
int target;
float velocity1=0;                //Velocidad en [Km/h]
float velocity2=0;
volatile int pulses1 = 0;
volatile int pulses2 = 0;
static volatile unsigned long debounce1 = 0;
static volatile unsigned long debounce2 = 0;
//position variables
int x;
int y;
signed int theta;
int ref=0;
int dist=0;
//PID gains
double Kp=13.83;
double Kd=7.3;
double Ki=2;
//1st Motor
```

```
signed int e1=0;
float PID1_Kp=0;
float PID1_Kd=0;
float PID1_Ki=0;
int PrevE1=0;
int PWM1=0;
int PWMmap1=0;
float PID1=0;
//2nd Motor
signed int e2=0;
float PID2_Kp=0;
float PID2_Kd=0;
float PID2_Ki=0;
int PrevE2=0;
int PWM2=0;
int PWMmap2=0;
float PID2=0;
unsigned long prevTime = 0;
unsigned long timeold1 = 0;
unsigned long timeold2 = 0;
unsigned long time1 = 0;
unsigned int pulsesperturn = 20;
const int wheel_diameter = 64;
void setup() {
  Serial.begin(9600); // Configuración del puerto serie
  pinMode(14,OUTPUT);
  pinMode(15,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(16,OUTPUT);
  pinMode(17,OUTPUT);
  pinMode(18,OUTPUT);
  digitalWrite(14,LOW);
  digitalWrite(15,HIGH);
  digitalWrite(16,LOW);
  digitalWrite(17,HIGH);
  attachInterrupt(0, counter1, RISING);
  attachInterrupt(1, counter2, RISING);
  x=400;
  y=400;


}
void counter1(){
  if( digitalRead (2) && (micros()-debounce1 > 500) && digitalRead (2) )
      {
```

```
        debounce1 = micros();
        pulses1++;
        }
        else ;
        }
void counter2(){
  if( digitalRead (3) && (micros()-debounce2 > 500) && digitalRead (3) )
        {
        debounce2 = micros();
        pulses2++;
        }
        else ;
        }
void loop() {
  if(inputTaken)
  {
    char customKey = customKeypad.getKey();
    if(xDone){
    if (customKey =='0' ||customKey =='1'||customKey =='2'||customKey =='3'||customKey
=='4'
    || customKey =='5'||customKey =='6'||customKey =='7'||customKey =='8'||customKey
=='9') {     // only act on numeric keys
      inputString += customKey;
      //Serial.println(inputString);// append new character to input string
    } else if (customKey == '*') {
      if (inputString.length() > 0) {
        inputInt = inputString.toInt(); // YOU GOT AN INTEGER NUMBER
        //Serial.println(inputString);
        x = inputInt;
        Serial.print("x = ");
        Serial.println(x);
        inputString = "";                // clear input
        xDone = 0;
        delay(50);

      }
    }
    }
    else if(!xDone) {
    if (customKey =='0' ||customKey =='1'||customKey =='2'||customKey =='3'||customKey
=='4'
    || customKey =='5'||customKey =='6'||customKey =='7'||customKey =='8'||customKey
=='9') {     // only act on numeric keys
      inputString += customKey;
      //Serial.println(inputString);// append new character to input string
    } else if (customKey == '*') {
```

```
    if (inputString.length() > 0) {
      inputInt = inputString.toInt(); // YOU GOT AN INTEGER NUMBER
      //Serial.println(inputString);
      y = inputInt;
      Serial.print("y = ");
      Serial.println(y);
      inputString = "";              // clear input
      xDone = 1;
      delay(50);

   }
   if(y == 0){
     theta = -1;
   }
   else{
     theta = x/y;
   }
    inputTaken = 0;
    delay(5000);
   }

  }

  }
  //encoder readings
  time1 = millis();
  if (millis() - timeold1 >= 1000){  // Se actualiza cada segundo
     noInterrupts(); //Don't process interrupts during calculations // Desconectamos la
interrupción para que no actué en esta parte del programa.
     rpm1 = (60UL * 1000 / pulsesperturn )/ (millis() - timeold1)* pulses1; // Calculamos
las revoluciones por minuto
     velocity1 = rpm1 * 3.1416 * wheel_diameter * 60 / 1000000; // Cálculo de la
velocidad en [Km/h]
     timeold1 = millis(); // Almacenamos el tiempo actual.
     interrupts();
  }

  if (millis() - timeold2 >= 1000){  // Se actualiza cada segundo
     noInterrupts(); //Don't process interrupts during calculations // Desconectamos la
interrupción para que no actué en esta parte del programa.
     rpm2 = (60UL * 1000 / pulsesperturn )/ (millis() - timeold2)* pulses2; // Calculamos
las revoluciones por minuto
     velocity2 = rpm2 * 3.1416 * wheel_diameter * 60 / 1000000; // Cálculo de la
velocidad en [Km/h]
     timeold2 = millis(); // Almacenamos el tiempo actual.
     interrupts();
```

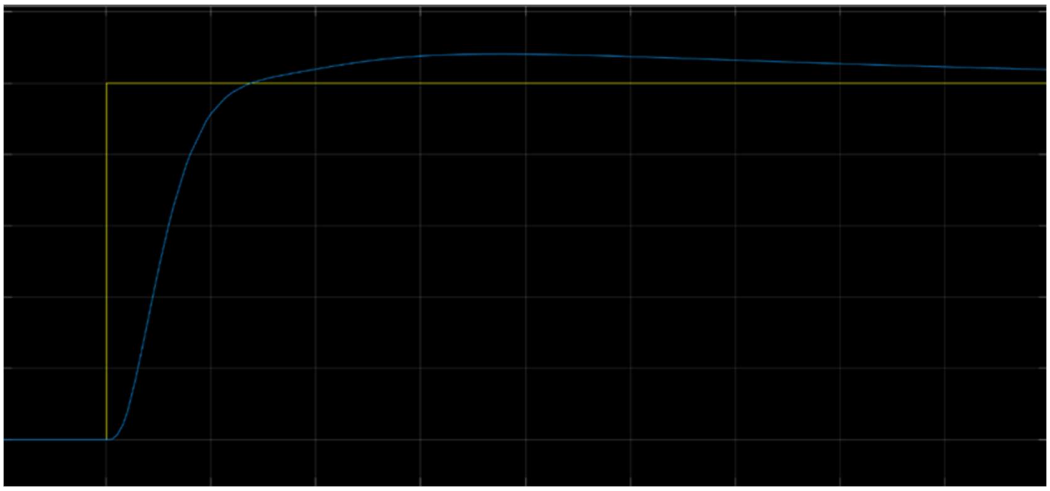
```
 }
 // Car rotation
 if(rotated & !inputTaken){
   Serial.println(theta);
   if(theta<0){
      target = (175/(4*3.1416))*(1.5708);
     }
     else{
       target = (175/(4*3.1416))*atan(theta);
     }
     while(pulses1 < target){
       //Serial.print(2);
       digitalWrite(14,LOW);
       digitalWrite(15,HIGH);
       analogWrite(5,160);
  }
       analogWrite(5,0);
       pulses1 = 0;
       pulses2 = 0;
       rotated = 0;
       delay(2500);
 }

// Moving in straight line
if(!rotated & !inputTaken){
 dist = sqrt((pow(x,2) + pow(y,2)));
 ref = (dist/(6.2832*32))*20;

 //PID1 readings
 e1 = ref - pulses1;
 PID1_Kp = Kp*e1;
 PID1_Ki = PID1_Ki +(Ki*e1);
 PID1_Kd  = Kd*(e1-PrevE1)/(time1-prevTime);
 PID1 = PID1_Kp + PID1_Ki + PID1_Kd;

   if(PID1 < 0){
   PWM1 = -1*PID1;
   digitalWrite(14,HIGH);
   digitalWrite(15,LOW);
   digitalWrite(16,LOW);
   digitalWrite(17,HIGH);
 }
 else{
   PWM1 = PID1;
   digitalWrite(14,LOW);
   digitalWrite(15,HIGH);
```

```
  digitalWrite(16,LOW);
  digitalWrite(17,HIGH);
  Serial.println(PID1);
 }

 //PID2 readings
 e2 = ref - pulses2;
 PID2_Kp = Kp*e2;
 PID2_Ki = PID2_Ki +(Ki*e2);
 PID2_Kd  = Kd*(e2-PrevE2)/(time1-prevTime);
 PID2 = PID2_Kp + PID2_Ki + PID2_Kd;
  if(PID2 < 0){
  PWM2 = -1*PID2;
  digitalWrite(14,LOW);
  digitalWrite(15,HIGH);
  digitalWrite(16,HIGH);
  digitalWrite(17,LOW);
 }
 else{
  PWM2 = PID2;
  digitalWrite(14,LOW);
  digitalWrite(15,HIGH);
  digitalWrite(16,LOW);
  digitalWrite(17,HIGH);
 }
 //Motor1
if(!(-1 < e1 <1)){
  analogWrite(5,PWM1);
 }
 else{
  analogWrite(5,0);
  moved1 = 0;
 }
 //Motor2
 if(!(-1 < e2 <1)){
  analogWrite(6,PWM2);
 }
 else{
  analogWrite(6,0);
  moved2 = 0;

 }
 prevTime = time1;
 PrevE1 = e1;
PrevE2 = e2;
 }
```

```
if((!moved1) & (!moved2) )
{
   delay(1000);
   digitalWrite(18,HIGH);
   inputTaken = 1;
   rotated = 1;
   xDone = 1;
   moved1 = 1;
   moved2 = 1;
   pulses1 = 0;
   pulses2 = 0;
}

}
```

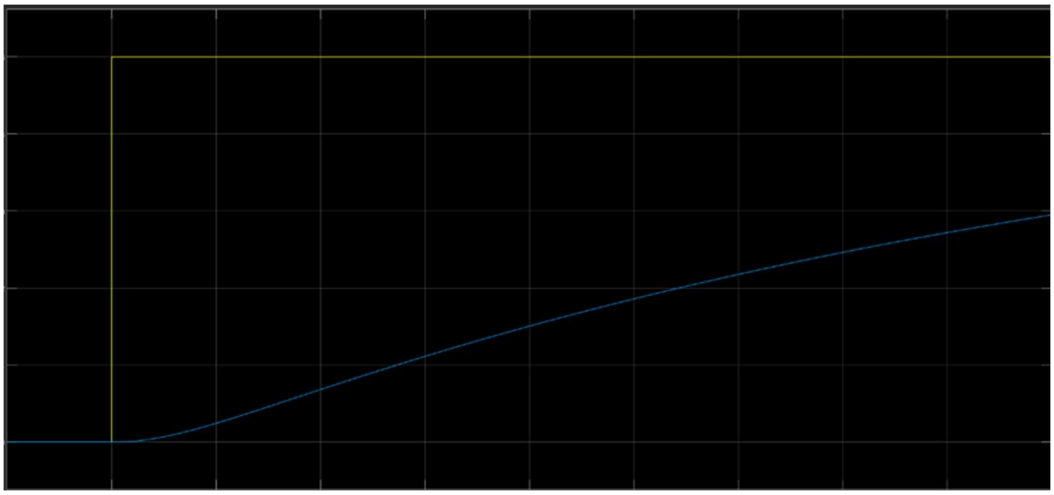
3. Design Evaluation

As in any design there are always flaw factors which were not taken into consideration and had a somewhat negative and unpredictable effect on the actual experimental performance. To start with the tires of the wheels were made of a smooth material and even though it did have some marks on it to create for the necessary friction, it was found that some floors didn't enough traction, causing the wheel to slip and the encoder to count rotations which did to contribute to the distance moved and hence missing the intended end position and sometimes ended up undershooting. So, the surface played a factor which we were unable to take into account.
Another problem was that even though both gear motors were identical, there was a difference in performance between each motor due to small differences in internal friction of the motors and inductance.
This problem did contribute some unwanted results but was overcome by having separate encoder readings for each wheel to compensate for different rotational speeds to ensure each wheel has rotated the intended number of turns.
A factor which added to the difference in performance in each motor was the fact that the weight of the components used (Arduino, breadboard and the l298N motor driver) were not evenly distributed over the whole car which was a flaw in our design. Proper positioning of these components would have evened the weight on each wheel hence reducing the noticeable difference in each wheel's rotational speed.



WITH PID:

NO PID:



4. Appendix

Keypad Library Arduino:
char getKey(): Returns the key that is pressed, if any. This function is non-blocking.
Creation:

Constructors:

Keypad(makeKeymap(userKeymap), row[], col[], rows, cols)

```
const byte rows = 4; //four rows
const byte cols = 3; //three columns
char keys[rows][cols] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'#','0','*'}
};
byte rowPins[rows] = {5, 4, 3, 2}; //connect to the row pinouts of the keypad
byte colPins[cols] = {8, 7, 6}; //connect to the column pinouts of the keypad
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, rows, cols );
```

Instantiates a Keypad object that uses pins 5, 4, 3, 2 as row pins, and 8, 7, 6 as column pins.
This keypad has 4 rows and 3 columns, resulting in 12 keys.

5. Sources

1. https://www.mathworks.com/matlabcentral/fileexchange/80482-mathematical-transfer-function-modelling-of-dc-motor
2. https://www.electroschematics.com/motor-speed-sensor-module-circuit/
3. https://nationalmaglab.org/education/magnet-academy/watch-play/interactive/dc-motor
4. https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/