

**MICROCONTROLLERS FOR MECHATRONICS – MECA442**

Experiment 5: Measuring the speed of rotation of a dc motor

**Jana Kabrit1, Oussama Al Ahmad Matar1**

**1** Rafik Hariri University, College of Engineering, Department of Mechanical and Mechatronics

*Abstract*

*The aim of this experiment is to design and simulate a circuit that will measure the speed of a DC motor with an integrated encoder in RPM (rotations per minute). The direction of rotation will be found using an Arduino Uno, and the results will be displayed on an LCD screen. The simulation will take place on Tinker CAD. The experiment was successful and the desired results were obtained.*

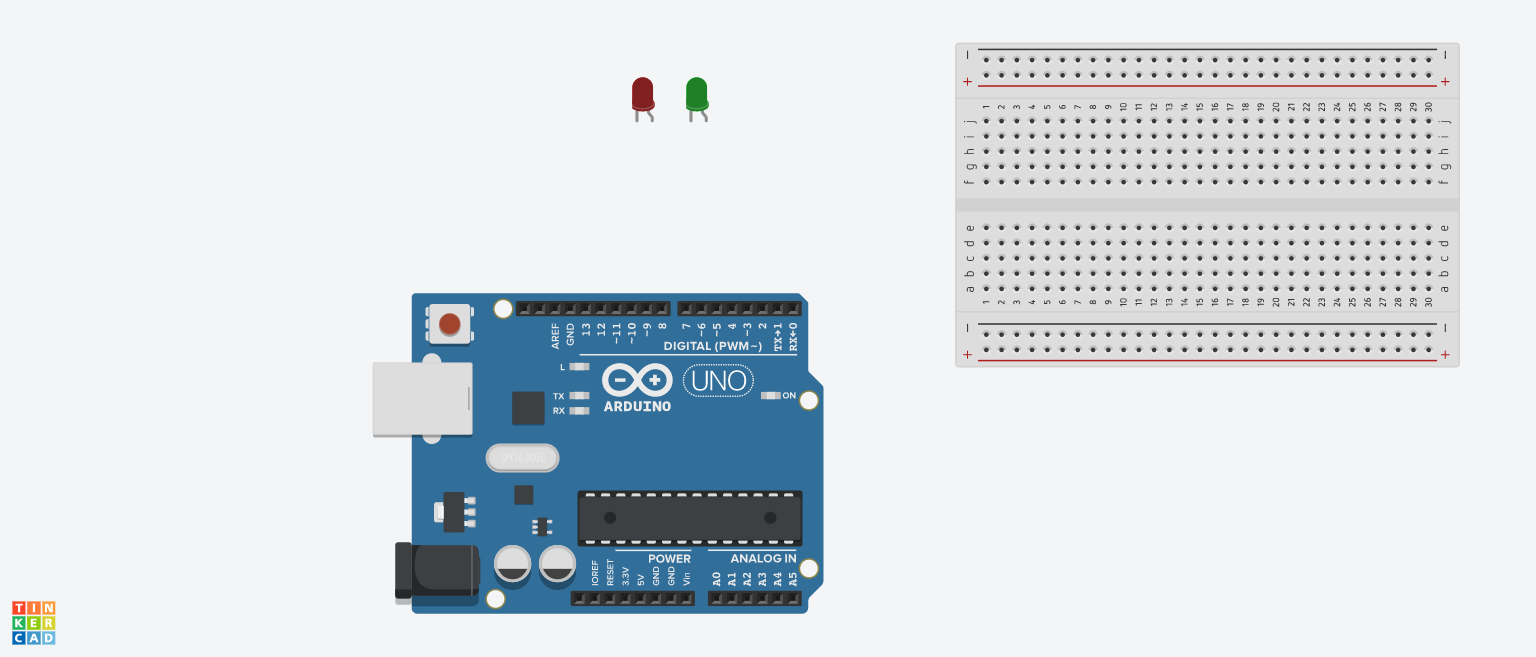
**Keywords:** Arduino Uno, DC Motor, Encoder, Speed, Rotation, LCD, Tinker CAD.

1. INTRODUCTION

In this experiment, the speed and direction of rotation of a spinning DC motor will be displayed on an LCD screen. The DC motor used has an integrated encoder that sends a specific number of pulses each revolution which are then detected by the Arduino Uno. The direction of rotation is found by comparing the pulses of each of the two encoder channels and which started first, and the speed is calculated according to the number of pulses per a specific time interval, set to 200ms in our case. Both the speed and rotation are then displayed on an LCD screen that is connected to the Arduino directly and through a breadboard.

1. **MATERIALS AND METHODS**
   1. **Materials**
      1. **Simulated Electronics Components**

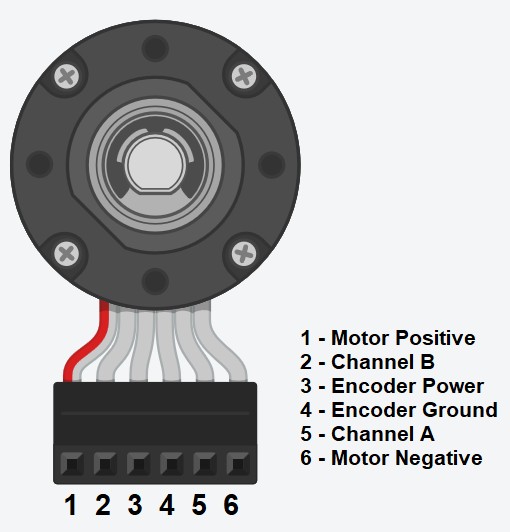
* Arduino Uno

The Arduino Uno (Figure 1) is a microcontroller board based on the ATmega328. It has 20 digital input/output pins (of which 6 can be used as PWM outputs and 6 can be used as analog inputs), a 16 MHz resonator, a USB connection, a power jack, an in-circuit system programming (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.[1] In this experiment, we need the Arduino so that we can program it to record the difference of distance every 0.1 seconds so we can calculate the speed.

**FIGURE 1:** ARDUINO UNO BOARD

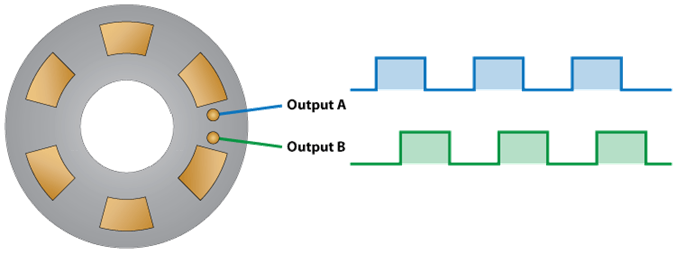
* DC Motor with Integrated Encoder:

**FIGURE 2:** DC MOTOR WITH ENCODER



A DC motor converts electrical energy into mechanical energy. You can use a DC motor with an integrated rotary encoder in order to measure the speed of the motor while it’s running. The pinout of the motor are shown in figure 2. Power is applied between pins 1 and 6 of the motor. This power can be drawn directly from a DC power supply or through a motor drive connected to a microcontroller in order to control the speed and the direction of rotation of the motor. The encoder power pin (pin 3) and ground (pin 4) should be connected to a voltage source. The output of the encoder are pins 2 and 5 (Channel A and B respectively).

The rotary encoder can rotate infinitely. It tracks the turning of motor shaft to generate digital position and motion information. Inside we have a perimeter of copper connections. The more connectors there are the better the precision of the encoder is. In our TinkerCAD simulation, the motor used has 46 connections, which results in 46 pulses per revolution. The encoder has 2 connectors pins that will output the signals related to the direction of rotation and the speed. These pins should be connected to pullup resistor. If the pins are not touching the internal copper connectors, the voltage at the output is 5V since we have the pull up connected top that voltage. However, when the motor starts rotating clockwise, 2 square waves, phase shifted from each other will start appearing at the output shown in figure 3. [2]

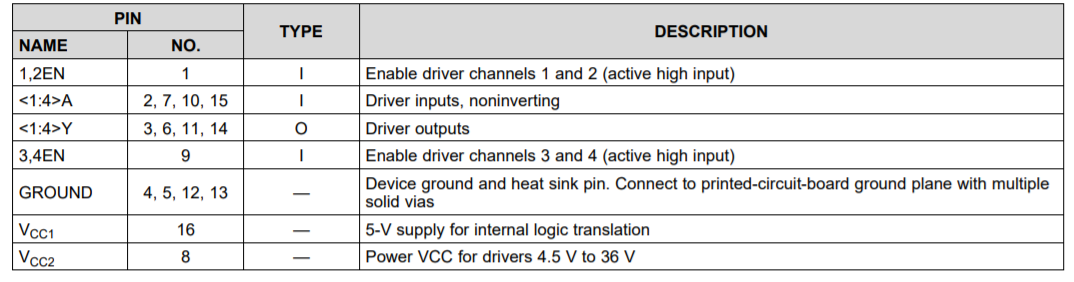
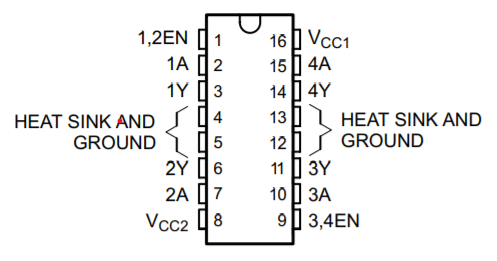


**FIGURE 3:** INTERNAL STRUCTURE OF A DC MOTOR WITH ENCODER

* L293D Motor Drive:

L293D is a dual H bridge motor drive. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5V to 36V. The L293D has two +V pins (pins 8 and 16). The pin +VCC2 (pin 8) provides the power for the motors, and +VCC1 (pin 16) for the chip's logic. As shown in figure 4 of the IC pinout. [3] The functions of each pin can be shown in table 1.

**FIGURE 4:** L293D IC



**TABLE 1:** L293D IC PINOUT FUNCTIONS

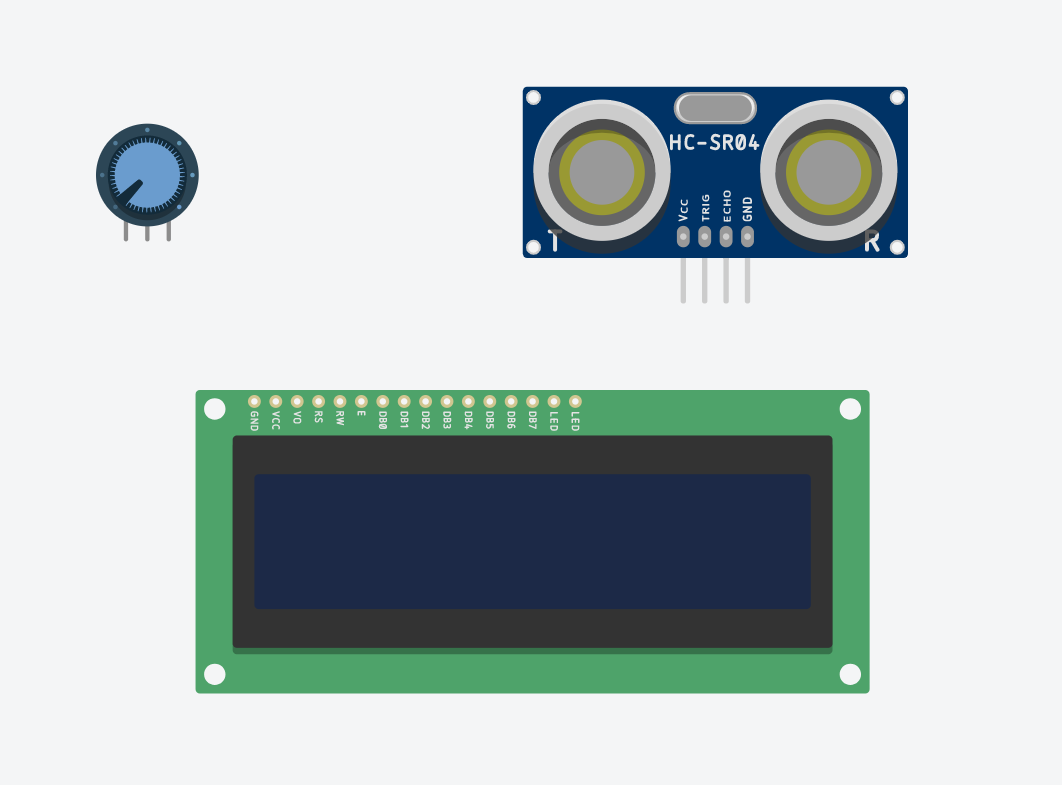
* LCD Screen:

These LCDs are available in many different sizes (16×2 1602, 20×4 2004, 16×1 etc.), but they all use the same HD44780 parallel interface LCD controller chip from Hitachi. This means you can easily swap them. The LCD has 16 connection pins, numbered 1-16 from left to right. If the display does not include a resistor, you will need to add one between 5 V and pin 15. It should be safe to use a 220Ω resistor, but this value might make your display a bit dim. In this experiment, we used a potentiometer to get the best brightness. The maximum current rating of the backlight can be checked from the and used this to select an appropriate resistor value.[4]



**FIGURE 3:** LIQUID CRYSTAL DISPLAY

* Potentiometer:

 The potentiometer, commonly referred to as a “pot”, is a three-terminal mechanically operated rotary analogue device which can be found and used in a large variety of electrical and electronic circuits. They are passive devices, meaning they do not require a power supply or additional circuitry in order to perform their basic linear or rotary position function.[5] In this experiment, we will use the potentiometer to get the best brightness of the LCD screen available.

**FIGURE 4:** POTENTIOMETER

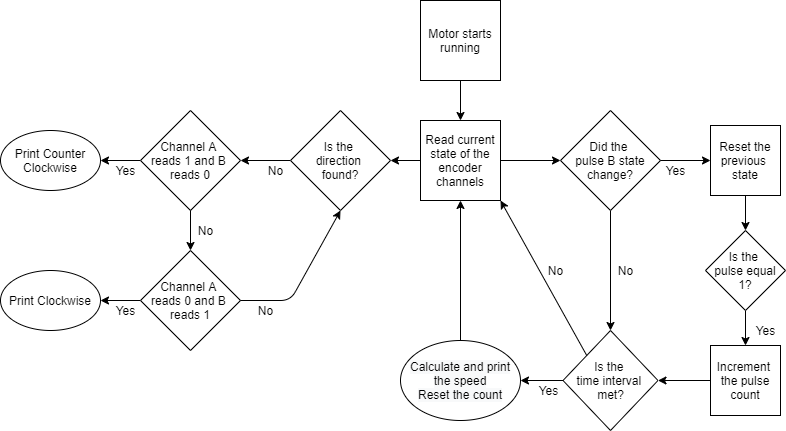
* + 1. **Code Components**
* const: A variable qualifier that modifies the behavior of the variable, making a variable "read-only" and constant.
* long int: Long variables are extended size variables for number storage, and store 32 bits (4 bytes) instead of the usual 16 bits (2 bytes).
* unsigned int: Same as int in that it stores a 2-byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535.
* void setup: The function is called whenever the program starts. It is used to initialize variables, pin modes, etc. It will only run once after each powerup or reset of the Arduino board.
* pinMode(): A function used to configure a specific pin to behave either as an input or an output, usually in the void setup.
* void loop: The function that holds the code inside and runs over and over as long as the board is turned on.
* millis(): A command that returns the number of milliseconds passed since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.
* LiquidCrystal lcd: Provides a set of endpoints to manage Arduino IoT Cloud Devices, Things, Properties and Timeseries. This API can be called just with any HTTP Client, or using one of these clients
* lcd.setCursor(): Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display
* lcd.print(): Displays what is in the argument on the LCD Screen
  1. **Methods**
     1. **Code Description**

Once the Arduino Uno turns on, it sets up the pins for the LCD display (in our case pins 1, 2, 4, 5, 6, and 7 as the RS, enable, D4, D5, D6, and D7 respectively) to be able to print the results on it, then sets the pins for the encoder channels A and B as pins 13 and 12 as digital pins respectively, and pins 9 and 10 for the motor drive to control the speed and direction of the motor as analog pins. It also creates the count integer that will measure how many times a pulse is detected from a channel. The motor is also turned on in the void setup to simulate measuring a motor that’s already on by setting an analog speed to either pins.

In the void loop, the current state of both channels A and B are consistently being read. It starts with an if statement that checks if the direction of the motor has already been found, it depends on which channels reads a pulse of 1 first. So if channel A was 1 and B was 0, then the motor is spinning counter clockwise, and if A was 0 while B is 1, the motor is spinning clockwise. Once the direction has been found, the if statement prints it on the LCD and stops running every loop.

To be able to measure the pulses, an if statement checks if the current state of channel B has changed compared to the previous state which is initially set to 0 (we could use any of the channels to measure pulses). Each time the pulse changes from 0 to 1 or vice versa, the previous state is reset and the count increments only if the current state is 1. Finally, once difference of the current millis with the previous millis (which is initially set to 0) equals the interval of 200ms, the speed is calculated and displayed on the LCD according to how many pulses were counted in this time interval and the pulse count is reset to 0 to measure again. Since our motor is known has 46 pulses per revolution (PPR), we can use the equation 1 to calculate the revolutions per minute.

A flowchart is shown describing the code process in Figure 6.



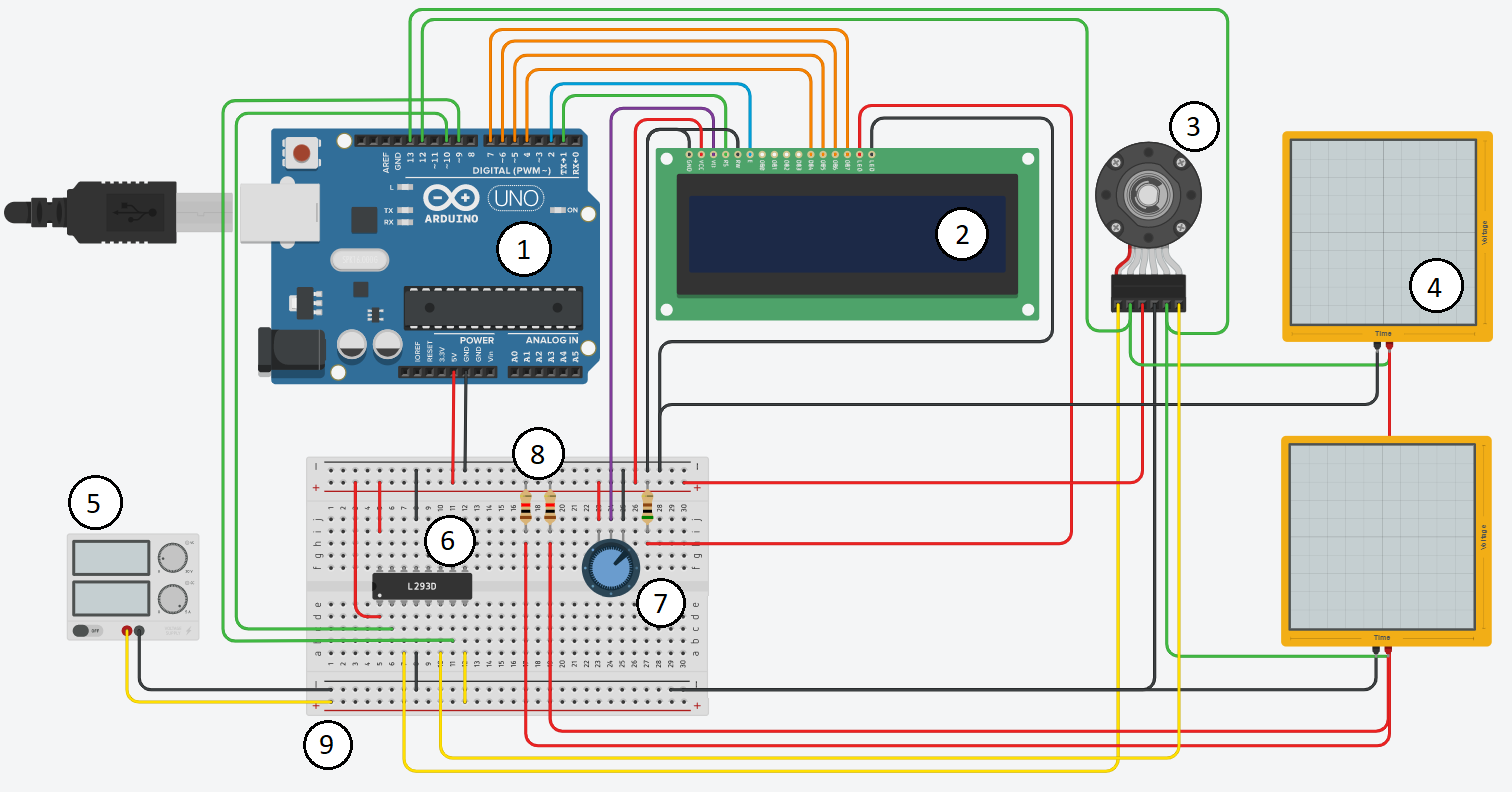
**FIGURE 5:** FLOWCHART

* + 1. **Connections on Tinker CAD**

1. First, we connected the 5V and ground pins of the Arduino Uno to the breadboard power and ground (red and black wires), and the power and ground pins of the encoder of the DC motor and power, enable, and ground of the L293D to the 5V and ground of the Arduino through the breadboard (red and black wires).
2. For the LCD, we connected the data pins D4, D5, D6, and D7 to the Arduino at pins 4, 5, 6, and 7 respectively to be able to print on the display (orange wires). The ground, R/W, and cathode pins are connected to the ground pin of the breadboard, and both the VCC and anode of the LCD are connected to the 5V of the breadboard, however, the anode is first connected to a 500-ohm resistor to avoid damaging the screen.
3. The RS pin of the LCD is connected to pin 1 (green wire), and the Enable to pin 2 (blue wire), and finally V0 is connected to a potentiometer (purple wire) which is connected to the 5V and ground of the breadboard.
4. To control the DC motor, first connect the input 1 and 2 of the L293D drive to pins 12 and 13 respectively (green wires), then connect a power supply to the breadboard that has a common ground with the Arduino. Output 1 and 2 of the motor drive to motor positive and negative pins (yellow wires) which draws power from the power supply through VCC2 of the drive.
5. To be able to measure the count of pulses, connect channel A and B of the DC motor encoder to pins 12 and 13 of the Arduino, and optionally to an oscilloscope for each channel to visualize the pulses with a 1ms time per division. Each oscilloscope is connected to 5V of the Arduino with a 1k-ohm resistor, and grounded to the Arduino as well.
6. **RESULTS AND DISCUSSION**
   1. **Simulation**

The circuit is implemented on Tinker CAD, adding the code to the Arduino Simulation, obtaining the circuit shown in Figure 7.

**TABLE 1:** LEGEND



**FIGURE 6:** CIRCUIT SCHEMATIC ON TINKER CAD

|  |  |
| --- | --- |
| NUMBER | COMPONENT |
| 1 | Arduino Uno Board |
| 2 | 16x2 LCD Display |
|  |  |
| 3 | DC Motor with Integrated Encoder |
| 4 | Oscilloscope |
| 5 | Power Supply |
| 6 | L293D Motor Drive |
| 7 | Potentiometer |
| 8 | Resistors |
| 9 | Breadboard |

* 1. **Analysis**

After running the simulation, the circuit showed the LCD displaying the speed of the motor in RPM and its direction, whether clockwise or counterclockwise. After changing the voltage from the power supply which increases the speed of the motor, the LCD screen updates and displays the new speed that could be higher or lower according to the voltage set. The pulses of each of the encoder channels can be seen on the two oscilloscopes, so as the voltage increases, the speed of the motor increases, and the frequency of the square pulses increases on the oscilloscope. The knob of the potentiometer can also be turned to get a variance in the LCD screen brightness. The video of the running simulation can be seen through [this link](https://drive.google.com/file/d/1UmvieqSHkpzLykkT9E3F1BsO4ofog2qs/view?usp=sharing).

1. **CONCLUSION**

The objective of this experiment is to display the speed of a DC motor in RPM and the direction of rotation using an Arduino Uno and calculating the speed in the code. The millis function is used in the Arduino Uno to get the count of pulses every 200ms and calculate the speed according to the count. The direction of rotation was also found by comparing both channels of the encoder. The experiment was successful, updating the speed every 0.2 seconds according to the channel readings.

**REFERENCES**

[1] Pololu Robotics & Electronics. *Arduino Uno*, 2020

https://www.pololu.com/product/2191#:~:text=Overview,header%2C%20and%20a%20reset%20button.

[2] HowToMechatronics, *How Rotary Encoder Works and How To Use It with Arduino*

https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/

[3] Adafruit, *L293D*

https://learn.adafruit.com/adafruit-arduino-lesson-15-dc-motor-reversing/lm293d

[4] Maker Guides, *How to use a 16×2 character LCD with Arduino*, 2020

https://www.makerguides.com/character-lcd-arduino-tutorial/

[5] Electronics Tutorials, *Potentiometers*, 2020

https://www.electronics-tutorials.ws/resistor/potentiometer.html

[6] Arduino Official Website  
 https://www.arduino.cc

**Appendix**

* + - * Arduino Code:

#include <LiquidCrystal.h>

LiquidCrystal lcd(1, 2, 4, 5, 6, 7);

int count = 0, interval = 200;

const int encoderB = 12;

const int encoderA = 13;

unsigned long int previousmillis = 0;

int previousStateB = 0, currentStateB;

int previousStateA = 0, currentStateA;

float speedRPM = 0;

bool direction = false;

void setup() {

lcd.begin(16, 2);

pinMode(encoderB, INPUT\_PULLUP);

pinMode(9, OUTPUT);

pinMode(10, OUTPUT);

analogWrite(10, 0);

analogWrite(9,255);

}

void loop() {

currentStateB = digitalRead(encoderB);

currentStateA = digitalRead(encoderA);

if(direction == false){

if(currentStateA == 1 && currentStateB == 0){

lcd.setCursor(0, 1);

lcd.print("CounterClockwise");

direction = true;

}

if(currentStateA == 0 && currentStateB == 1){

lcd.setCursor(0, 1);

lcd.print("Clockwise");

direction = true;

}

}

if(currentStateB != previousStateB){

previousStateB = currentStateB;

if(currentStateB == 1){

count++;

}

}

if(millis() - previousmillis > interval){

previousmillis = millis();

// 6.52 = 5 \* 60 / 46

speedRPM = count \* 6.52;

count = 0;

lcd.setCursor(0, 0);

lcd.print("RPM Speed ");

lcd.print(speedRPM);

}

}