DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEEE3075 / Mechatronics Laboratory

Lab 3 Arduino DC Motor Control

Name: Jason Samuel Pangestu

Student ID: 16522575

Table of Content

Figure List

Introduction

A DC Motor was one of the primary motors that were used for robotic system application. In this report, a DC Motor will be controlled by Arduino Uno R3 Microcontroller and the whole system will be designed and simulated in Tinkercad Software. In the simulation software, a DC motor with encoder model will be used [3]. Next, a motor driver IC was introduced as the bridge between microcontroller and DC Motor. This driver was needed since most DC Motors requires 5V- 12V to operate while microcontrollers has operating voltage of 3.3V or 5.5V [1]. The motor driver IC that will be used in this project was L293D. This driver was capable of drive bidirectional currents up to 600mA and voltages from 4.5V to 36V [2]. This driver also was able to control the speed and direction of the DC Motor. In addition, potentiometer will be used in this lab to control the speed manually and push button will be used to change the direction of DC motor rotations along with blue LED and red LED that indicated clockwise and counter clockwise direction respectively. Furthermore, P control was also implemented in this project. The desired speed will be set and compared to the actual speed. In addition, disturbance such as resistor in series with the motor power line was implemented. The PWM and motor current response will be investigated for different resistors loads. In this reports, basic configuration and theory aspect of the motor driver and p controller will be provided in the method section while testing and simulation will be shown in analysis and results section.

Figure 1 DC Motor with Encoder [3]

Methods

1. L293D Motor Driver

This motor driver was composed of 16 pins (Figure 2). Pin 1 and Pin 9 were enable A and B respectively. Enable A is used to enable all the pins at the left side and Enable B is used to enable all the pins at the right side. Meanwhile, the ground pins were pin 4, 5, 12, and 13. For the input pins there were pin 2 and pin 7 at the left side and pin 15 and pin 10 at the right side. These input pins will receive signal from devices such as microcontroller. For the output Pins (Pin 3 and 6 at the left side, Pin 11 and 14 at the right side), it will connect to the motor terminals. For pin 8, it will be connected to motor power supply (4.5 to 36V). For pin 16, it will be connected to 5V to drive internal logic circuitry.

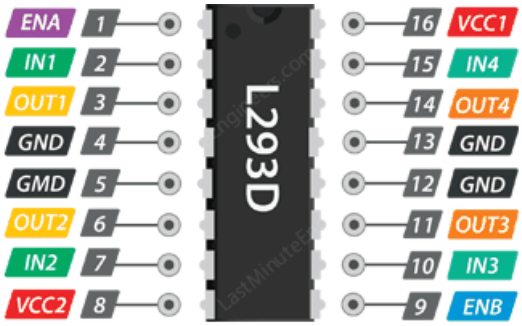

Figure 2 L294D Pinout Diagram [4]

2. Controlling Motor Direction

For controlling motor direction, single half h bridge channel was represented.

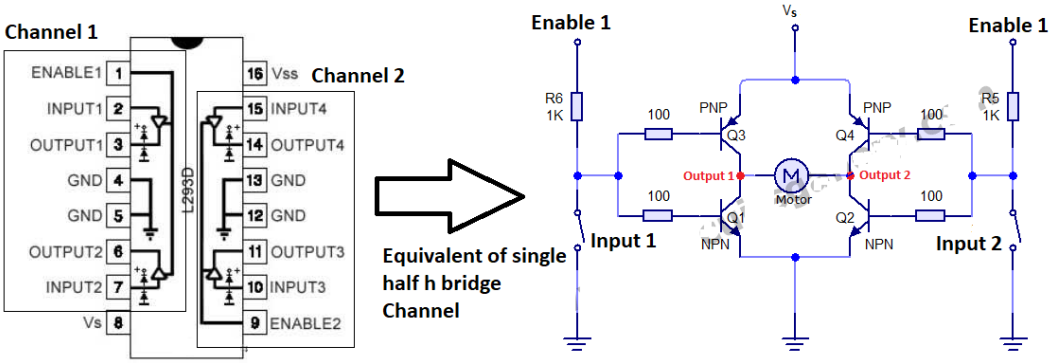

Figure 3 Single Half H-Bridge Representation [5]

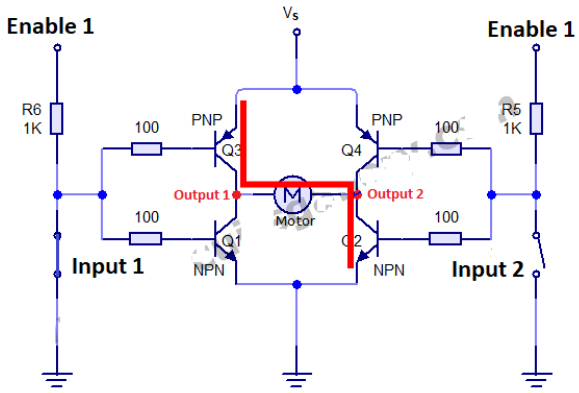

Figure 4 Single Half H-Bridge Representation when pin 2 is HIGH and pin 7 is LOW [5]

According to figure 4, when this condition occurred, the current will flow from Q3 and Q2. Thus, the motor between output 1 and output 2 were rotating in clockwise direction. In this case, pin 2 was HIGH, pin 7 was LOW, and Enable pin was HIGH.

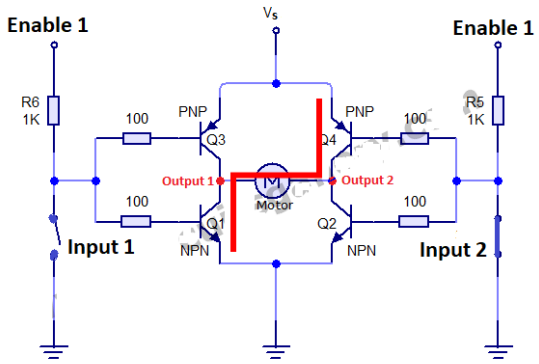

Figure 5 Single Half H-Bridge Representation when pin 2 is LOW and pin 7 is HIGH [5]

According to figure 5, when this condition occurred, the current will flow from Q4 and Q1. Thus, the motor between output 1 and output 2 were rotating in counter clockwise direction. In this case, pin 2 is LOW , pin 7 is HIGH, and Enable pin was HIGH. By using this technique, changing the motor direction can be applied immediately. In addition, when both pin 2 and 7 were low there were no rotation. It was also the same when both pin 2 and 7 were HIGH. For addition information, it was also applied the same technique to pin 15 and 10.

3. Controlling Motor Speed

DC Motor speed can be controlled by changing the input voltage. PWM (Pulse Width Modulation) technique was used since it can control the input voltages by changing its pulse width. According to figure 6, different duty cycles will generate different average voltage. For example, 50% duty cycle will generate 6V of average voltage out of 12V. The higher the duty cycle, the higher the average voltage. This technique can be applied to enable pin (pin 1 or pin 9), therefore it will control the speed of the motor by varying the pulse width.

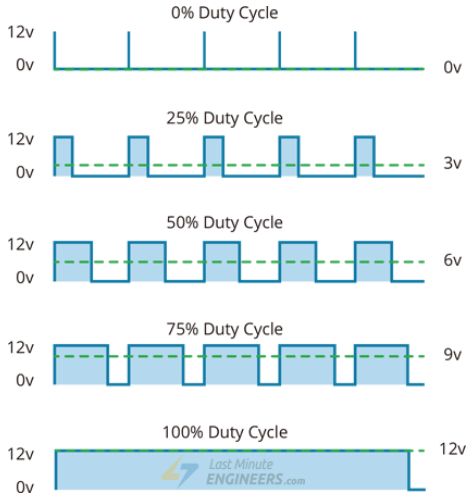

Figure 6 PWM with Different Duty Cycle [4]

4. Proportional Control

Proportional control was a type of linear feedback control system that the output change was proportional to the input signal generated by error [6]. Below was the general equation for proportional controller.

$$u(t) = u_{bias} + k_c\, e(t) \qquad \textit{Eqn1}$$

Where $u(t)$ = controller output, u_{bias} was the controller bias, $e(t)$ was the controller error ($e(t) = Set\ Value - Actual\ Value$), and k_c was controller gain [7]. For P controller in this application, the set value can have the same value with actual value, hence error will be zero. If u_{bias} was zero, then the controller result will be zero also. This will make no sense since there will be 0 PWM value so the output which was the DC motor will stop moving. In order to tackle that, bias value need to be existed in this application.

In this project, the set value was the desired speed and actual value was the actual speed. The method used to obtain the actual speed was by considering the encoder of DC motor. Since in the simulation software, the DC motor with encoder specification was not mentioned and actual speed was only provided in terms of display on the DC motor, the parameters that were used to calculate Actual speed in RPM was generated by trial and error until the the desired value was obtained. In addition, 2 output from the DC motor encoder will also be considered and can determine whether the motor was rotating in clockwise or counter clockwise direction (As shown in figure 8). Next, the actual speed will be inserted in the proportional control (closed loop) as in figure 7. The control process will lead to the PWM value and will directly control the speed of motor.

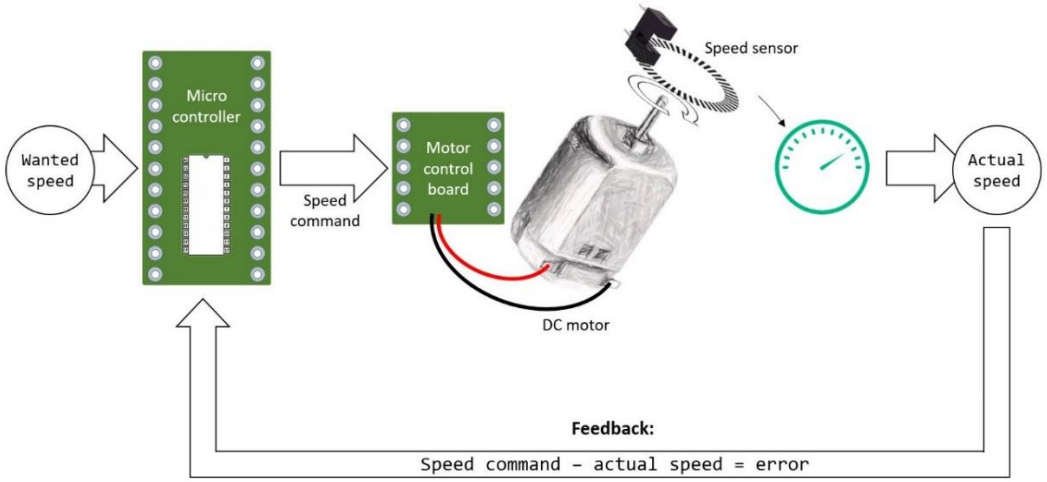

Figure 7 Closed Loop Feedback System for DC Motor Control [8]

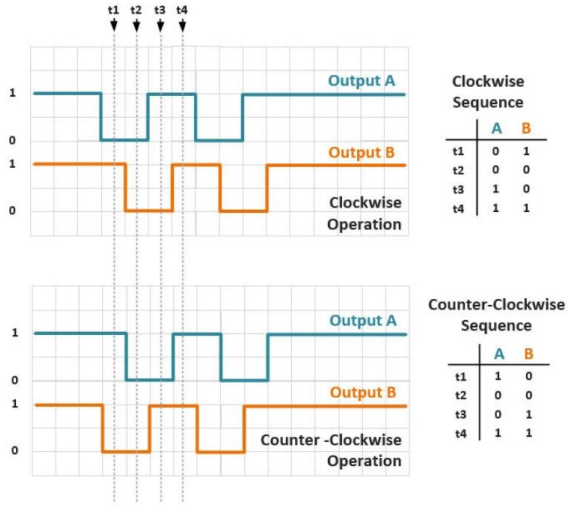

Figure 8 Encoder Output A and B [9]

Analysis and Results

1. Overall Hardware Schematic of DC Motor Drive System in Tinkercad

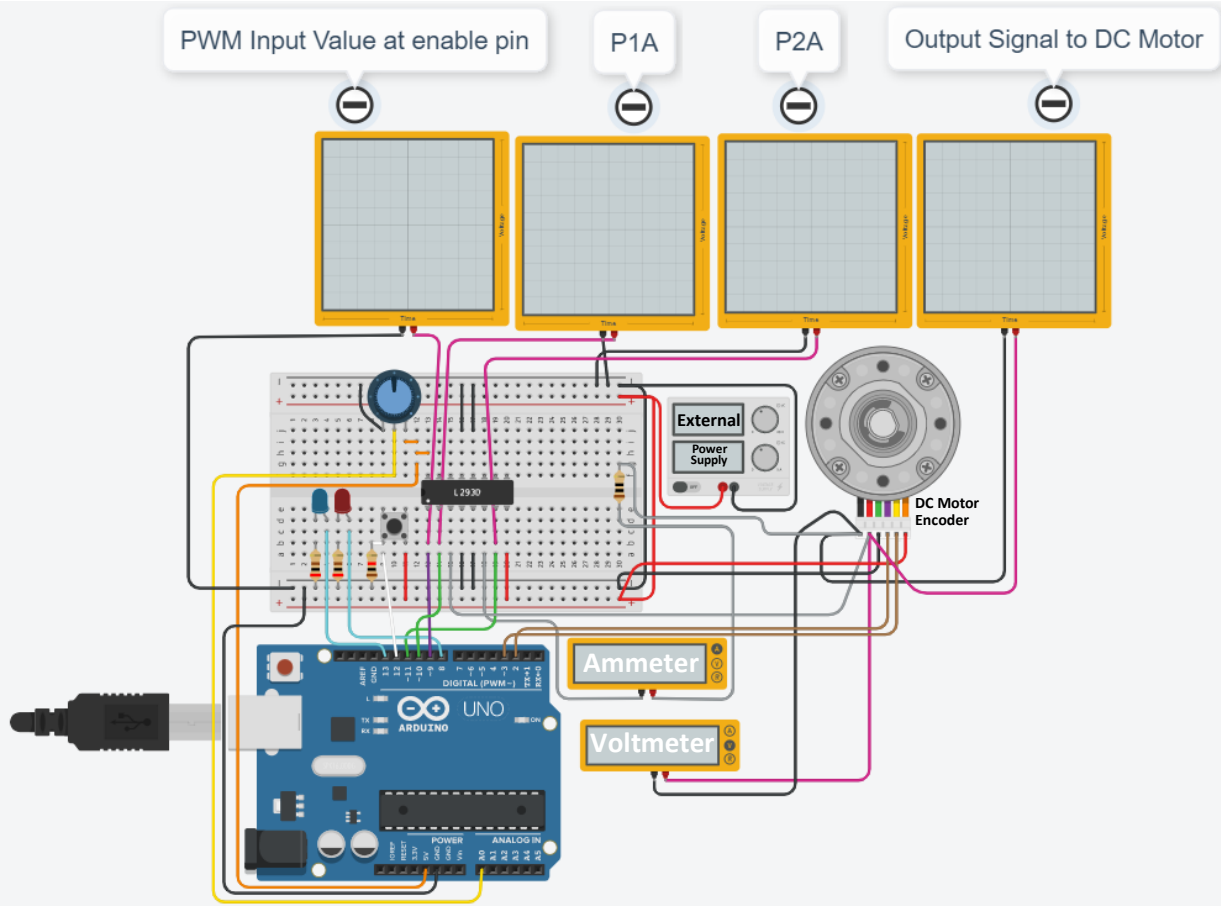

Figure 9 Overall Hardware Schematic of DC Motor Drive System

From figure 9, it can be seen that the Arduino Uno R3 was placed at the left bottom. On the Arduino, the yellow cable connected A0 analogue to the blue potentiometer at the top. This port will receive analogue signal from potentiometer (10k ohm). The Arduino will also provide power to the potentiometer and the L293D (pin 16 → to drive internal logic circuitry). The orange cable will represent these connections. Next, ground from Arduino will connect to the ground line including th external power supply ground connection. Then, pin 13 and 8 from Arduino will connect to the blue and red LED circuits respectively via blue cables. The colour blue and red indicated clockwise and counter clockwise direction of the dc motor. In the led circuits, 200ohm was used as the resistors values. Meanwhile, pin 12 was connected to the push button along with the resistors (1k ohm). The push button will send signal to the Arduino pin whether it was LOW or HIGH (this mechanism will be used to change the DC motor rotation). For pin 10 and 11, it will be connected to Output pin 2 and 7 of L293D respectively. The green cable will send either HIGH or LOW signal from Arduino to the L293D.

Next, pin 9 arduino (Able to send PWM signal) will be connected to pin 1 of L293D via purple cable. This pin 1 will enable all the pins on that side and deliver the PWM input signal from the Arduino. For Arduino pin 2 and pin 3, it will be connected to DC Motor encoder of Channel B (purple) and Channel A (Yellow) respectively by brown cables. These pins will act as an interrupt to receive encoder values from the DC motor.

For the L293D gate driver chip, which was located at the center, all the ground pins (pin 4,5,12, and 13) will be connected to the ground line. While pin 8 will be connected to the external power supply. This red cable will deliver supply voltage for the DC motor. For the output pins (pin 3 and 6), it will be connected to DC motor terminals by gray cable. Pin 6 will be connected to the ammeter, resistor, and motor terminal (black one). The resistor will be used later in proportional control as a disturbance to affect the motor current. Meanwhile, pin 3 will be connected to the motor terminal directly (red one). The rest of the circuit driver connections have been explained in the previous paragraph.

The external power supply was located at the center also and will supply 12V and 2A to the circuit. The red cable was the positive line and the black cable was the negative line. For the DC motor encoder, the motor terminal along with channel a and b were explained. The green one (encoder ground) will connect to the ground line and the orange one (encoder power) will connect to the power line in the circuit. In addition, voltmeter was set parallel to the motor terminal to check the supply voltage to the DC motor. At the top, the oscilloscope at the left will plot the PWM input at the enable pin. For 2 oscilloscopes at the centre, P1A and P2A were pin 2 and pin 7 at the L239D driver. These plots will show whether each of the pin is LOW or HIGH. For the oscilloscope at the right, it will plot the PWM input to the DC Motor.

2. DC Motor Rotation Direction and Speed Control Simulation

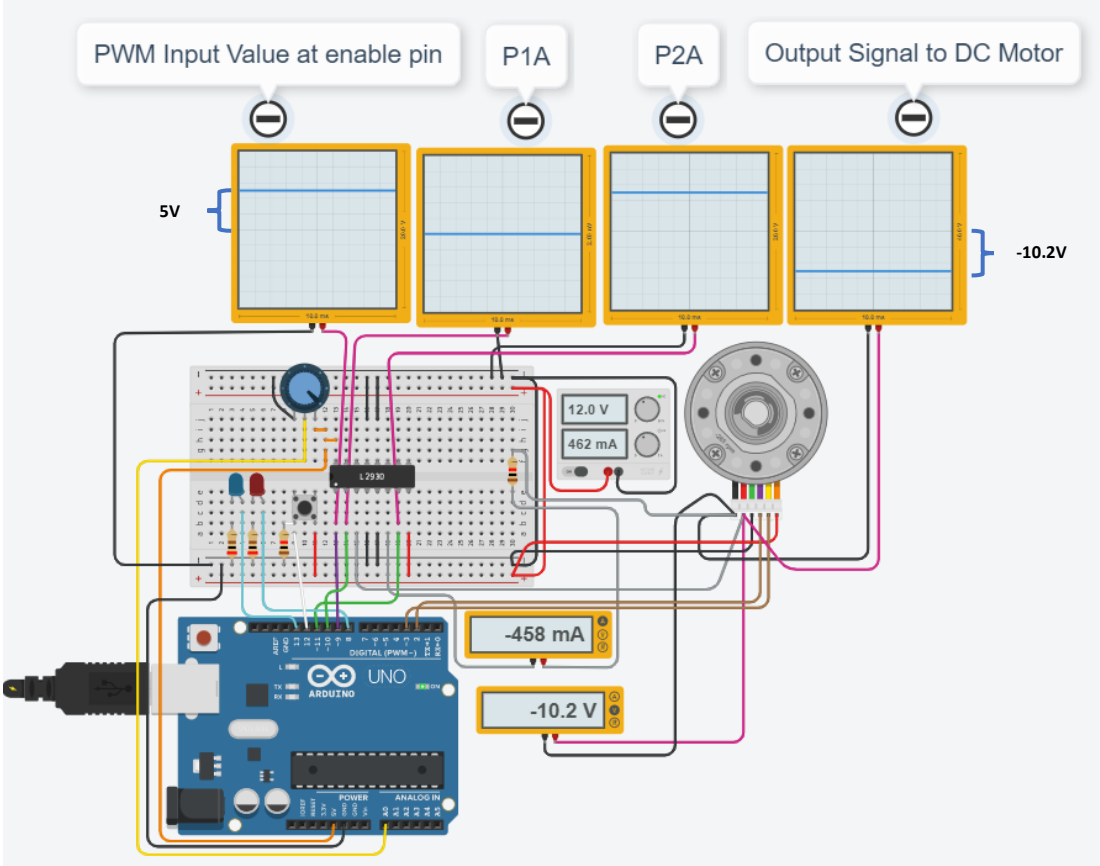

Figure 10 The Overall Circuit at Maximum Speed

From figure 10, the potentiometer was set to the maximum value, hence the DC motor will run on its maximum speed in this circuit. In this figure, the resistor beside the power supply was 0ohm and the push button was not pushed yet, hence the initial condition for the motor was set to rotate clockwise. The power supply was set to 12V and 2A for this circuit. The Maximum voltage and current input for this circuit before the DC motor was -10.2V and -458mA. These negative values only show the clockwise direction of the motor. From the oscilloscope, it can be seen that P1A and P2A were LOW and HIGH respectively. The PWM input value at the left showed around 5V and the output signal to DC motor showed for -10.2V. The max DC motor speed was -265rpm and the value was displayed on the DC motor.

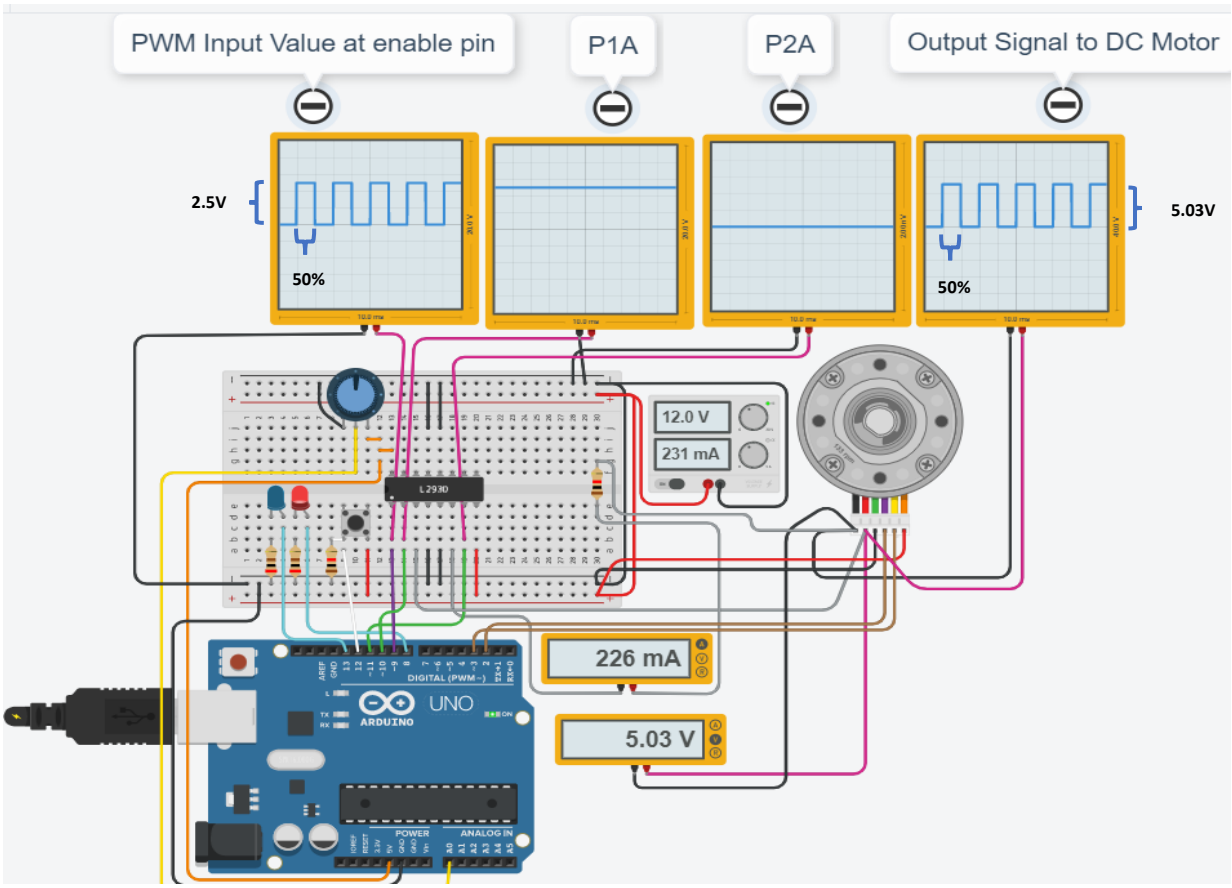

Figure 11 The Overall Circuit at Half Speed with Counter Clockwise Direction

From figure 11, it can be seen that the red LED was turned on, hence the motor rotated in counter clockwise direction. This was happened when the push button was pressed and released at the first time. For this case, the potentiometer was rotated halfway, hence the speed will be half of the maximum speed (132rpm). The voltage and current flowing to the DC motor were 5.03V and 226mA respectively. These positive signs showed that the rotation was in counter clockwise direction. For the oscilloscope, PWM input value at enable pin had duty cycle for 50%, hence it will generate average voltage of 2.5V. For the output signal to DC motor, it will also produce PWM voltage for 5.03V, half of voltages for the maximum speed. Meanwhile, P1A and P2A were HIGH and LOW respectively. This proved that the motor rotated in counter clockwise direction.

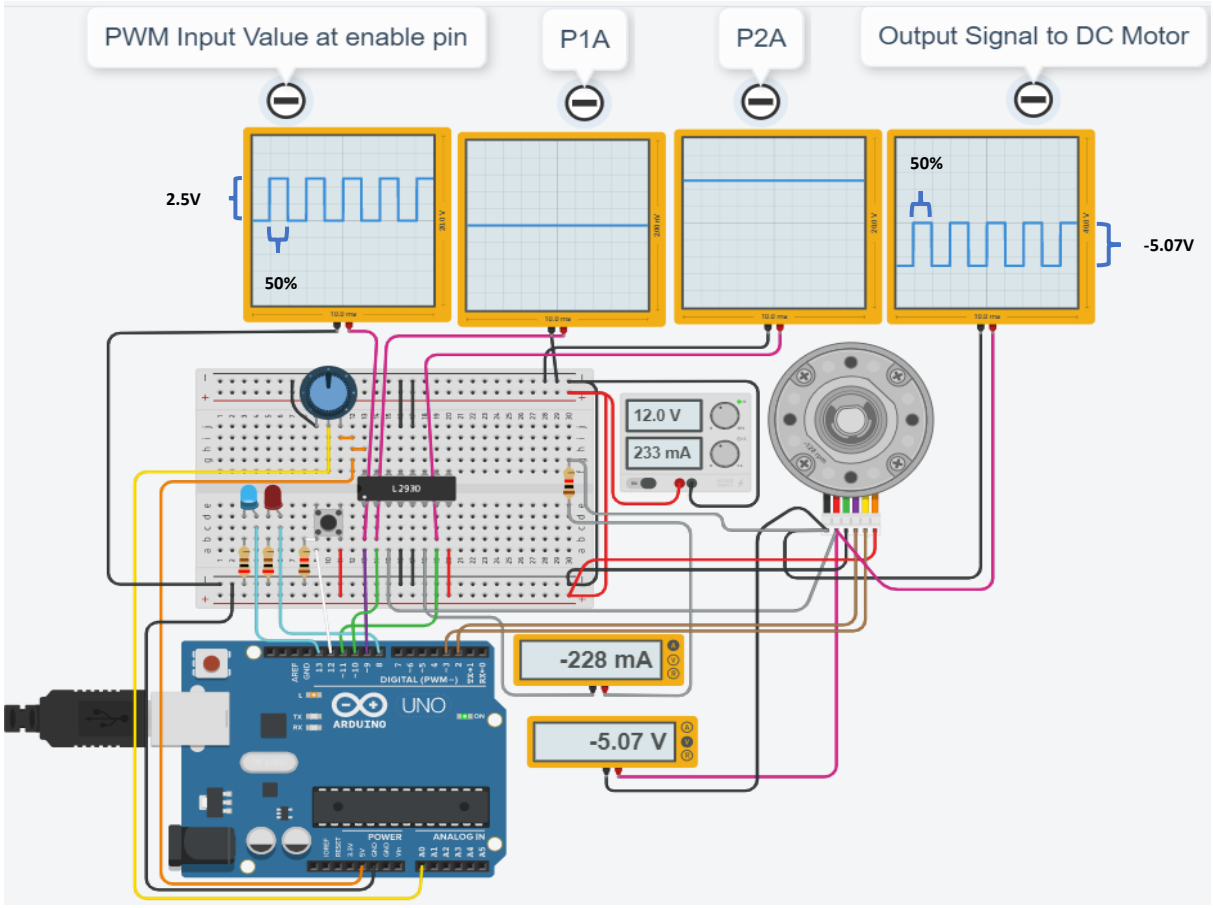

Figure 12 The Overall Circuit at Half Speed with Clockwise Direction

From figure 12, it can be seen that the blue LED was turned on, hence the motor rotated in clockwise direction. This was happened when the push button was pressed and released at the second time. For this case, the potentiometer was rotated halfway, hence the speed will be half of the maximum speed (-132rpm). The voltage and current flowing to the DC motor were -5.07V and -228mA respectively. These negative signs showed that the rotation was in clockwise direction. For the oscilloscope, PWM input value at enable pin had duty cycle for 50%, hence it will generate average voltage of 2.5V. For the output signal to DC motor, it will also produce PWM voltage for -5.07V, half of voltages for the maximum speed. Meanwhile, P1A and P2A were LOW and HIGH respectively. This proved that the motor rotated in clockwise direction. Next, the Arduino code was inserted after this page along with detail explanation in the comment line.

Code for DC Motor Rotation Direction and Speed Control Simulation

```
#define P1A 10                  // Pin 10 is defined as P1A (L293D input pin, pin number 2)
#define P2A 11                   // Pin 11 is defined as P2A (L293D input pin, pin number 7)
#define EN1 9                   // Pin 9 is defined as EN1 (L293D enable pin, pin number 1)
#define PushButton 12           // Pin 12 is defined as push button pin
#define BlueLed 13              // Pin 13 is defined for Blue LED
#define RedLed 8                // Pin 8 is defined for Red LED

// Initial set up for potentiometer
double Potentio = A0;           // Potentio variable is defined as A0 pin with double datatype (define the potentiometer)
int sensorValue = 0;            // sensorValue is set as integer (The analogue reading from potentiometer)
// Initial Set up for push button mechanism
bool pressed = false;           // pressed is set "false" initially
int rotDirection = 0;           // rotDirection variable is set to 0

void setup()
{
 Serial.begin(9600);                    // Initialize For serial monitor
 pinMode(P1A, OUTPUT);                  // P1A is defined as OUTPUT
 pinMode(P2A, OUTPUT);                  // P2A is defined as OUTPUT
 pinMode(EN1, OUTPUT);                  // EN1 is defined as OUTPUT
 pinMode(PushButton, INPUT);            // PushButton is defined as INPUT
 pinMode(BlueLed, OUTPUT);              // Blue LED is defined as OUTPUT
 pinMode(RedLed, OUTPUT);               // Red LED is defined as OUTPUT
 pinMode(Potentio, INPUT);              // Potentio is defined as INPUT

 // Initial Rotation direction --- CLOCKWISE -->
 // The initial configuration is set as clockwise hence P1A and P2A are Low and high respectively and
 // RedLED and Blue LED are LOW
 digitalWrite(P1A, LOW);
 digitalWrite(P2A, HIGH);
 digitalWrite(RedLed,LOW);
 digitalWrite(BlueLed,LOW);
 Serial.println("Arduino DC Motor Control"); // Print on the serial monitor
}

void loop()
{
// Potentiometer
sensorValue = analogRead(Potentio);
 // the analog read function will read the potentio value and stored it in sensorValue variable
double speedz = 0;
 // speedz was declared as 0 and has double datatype
speedz = map(sensorValue, 0, 1023, 0, 255);
 // the map function will convert the min and max value of the potentiometer (0-1023) to PWM value (0-255)
// 0-1023 Is because of arduino 10 bit ADC
// For the PWM Signal,  0 means signal of 0% duty cycle and 255 means signal of 100% duty cycle
analogWrite(EN1, speedz);                // the analogWrite will send the pwm signal in speedz variable to EN1 pin
Serial.println(speedz);                  // speedz is printed on the serial monitor

// Button mechanism
if (digitalRead(PushButton) == true)
// if we press the push button, pressed will have "true" value since we declared it as "false"
 {
   pressed = !pressed;
 }
while (digitalRead(PushButton) == true);    // This function enable us to push the button at period of time
                                            // while the push button is true (being pressed), it will do nothing
```

```
// Once we release the pushbutton . . .

// Change rotation direction to CCW
if (pressed == true  & rotDirection == 0)
// at this case, pressed has become "true" and rotDirection is initialized as 0 value in the beginning
 {
  digitalWrite(P1A, HIGH);                  // P1A became HIGH
  digitalWrite(P2A, LOW);                   // P2A became LOW
  digitalWrite(RedLed,HIGH);                // Turn on RedLed
  digitalWrite(BlueLed,LOW);                // Turn off BlueLED
  rotDirection = 1;                         // rotDirection is set to 1
 }

// Once we press again the pushbutton and release it,
// it will go through the first two function button mechanism and pressed value became false and rotdirection is 1

// Change rotation direction to CW
if (pressed == false & rotDirection == 1)   // at this case, pressed has became "false" and rotDirection is 1
 {
  digitalWrite(P1A, LOW);                   // P1A became LOW
  digitalWrite(P2A, HIGH);                  // P2A became HIGH
  digitalWrite(RedLed,LOW);                 // Turn off RedLEd
  digitalWrite(BlueLed,HIGH);               // Turn On BlueLED
  rotDirection = 0;                         //rotDirection is set to 0
 }
  delay(75);                                // 75 millisecond delay
}
```

3. Proportional Control Implementation and Simulation

For the control implementation, proportional control was used in this project to obtain the desired value while there were some disturbances. From equation 1 at the section method, Actual value was set to be the Actual speed value (denoted with *RPM* variable) and Set Value was set to be the desired speed value (denoted with speedz variable). These two values will generate the error. For the controller bias, it will be set to *speedz* variable since the desired speed value will always change according to the potentiometer value.

In this simulation, a load resistors (disturbances) were introduced and tested to the controller system. First, the desired speed was set to 137RPM by using the potentiometer. Next, the Kc value was set to 0.71 and will be tested to different circumstances. In this case, several tables will be built and composed of current and voltage measurement of the DC Motor, PWM value that will be sent to the enable pin (the value will be controlled by the controller output), and the actual speed (RPM variable). The measurement will be conducted for every 1 second and counts up until 8 seconds. The full simulation video will be uploaded in different file.

Time(s)	1	2	3	4	5	6	7	8
Current (mA)	420	243	282	220	240	238	240	239
Voltage (V)	9.3	3.3	6.3	5	5.3	5.25	5.30	5.32
PWM	234.27	78.07	159.01	126.35	137.71	134.16	134.87	135.58
Actual Speed (RPM)	219	109	148	139	139	141	140	139

Table 1.0 Simulation Results with No load condition (R = 0 Ohm)

Time(s)	1	2	3	4	5	6	7	8
Current (mA)	377	150	250	220	220	225	226	226
Voltage (V)	8.35	3.3	5.6	4.9	5	4.9	5.2	5.1
PWM	234.27	93.69	158.30	135.58	142.68	140.55	141.26	141.26
Actual Speed (RPM)	198	106	140	129	133	132	131	131

Table 2.0 Simulation Results with (R = 3 Ohm)

Time(s)	1	2	3	4	5	6	7	8
Current (mA)	341	155	230	210	208	212	212	208
Voltage (V)	7.6	3.3	5.12	4.6	4.6	4.67	4.67	4.7
PWM	234.27	106.47	159.01	144.10	146.94	146.23	146.23	146.94
Actual Speed (RPM)	180	106	127	123	124	123	124	124

Table 3.0 Simulation Results with (R = 6 Ohm)

Time(s)	1	2	3	4	5	6	7	8
Current (mA)	300	155	201	199	195	195	196	195
Voltage (V)	6.07	3.4	4.6	4.3	4.2	4.2	4.35	4.2
PWM	234.27	120.67	160.43	151.91	152.62	152.62	153.33	152.62
Actual Speed (RPM)	159	103	117	115	115	114	116	114

Table 4.0 Simulation Results with (R = 10 Ohm)

Time(s)	1	2	3	4	5	6	7	8
Current (mA)	140	112	115	116	115	X	X	X
Voltage (V)	3.2	2.45	2.55	2.56	2.55	X	X	X
PWM	234.27	181.02	185.99	187.41	186.7	X	X	X
Actual Speed (RPM)	75	68	66	67	66	X	X	X

Table 5.0 Simulation Results with (R = 50 Ohm)

From the above tables, it can be seen that the increase of disturbances will affect the whole system parameters. For additional information, The PWM for the first one for every table has the same value since rpm is set to 0 and the desired speed is always the same (137rpm) unless the potentiometer was rotated by the user. It was noticeable for all tables that the increase of resistance will lead to the decrease of voltage and current used of the DC Motor. This verified the ohms Law where resistor's resistance was inversely proportional with current that through it. This is happened since the resistance limit the current that goes into the DC Motor. From table 1, at first 4 parameters of the systems were oscillating. Investigating on the actual speed, the RPM first decreased to 109RPM, then it increased to 148RPM, next it decreased to 139RPM. And after that it will just have very small fluctuation around that value. The yellow boxes represented the small fluctuation between the previous and afterward data. The current and voltage flow in this area also had small fluctuation since the PWM value is already stable (did not oscillate). In addition, the error was obtained around 2rpm value for this no load condition. From table 2, table 3, and table 4, it had also the same pattern with table 1. At first, the system was oscillating then at t = 5 second it will become stable with small fluctuation. However, these tables had different errors. Comparing the actual speed (RPM) with the desired speed, table 1 had around 5rpm error, table 2 had around 13rpm error, and table 3 had around 22 rpm error. The Kc value that is set for this system still manage to deal with small disturbances. On the other hand, by using 50ohm load resistor (table 5), the error significantly large (around 69rpm). For the X values in table 5.0, it was not written since it was similar with the previous data (t = 5 second). Furthermore, the Kc was changed to 2 from 0.71 for this system and the error result was decreased for some amounts. The details can be seen in the simulations.

Code for Proportional Control on Overall System

```
#define P1A 10                  // Pin 10 is defined as P1A (L293D input pin, pin number 2)
#define P2A 11                  // Pin 11 is defined as P2A (L293D input pin, pin number 7)
#define EN1 9                   // Pin 9 is defined as EN1 (L293D enable pin, pin number 1)
#define PushButton 12           // Pin 12 is defined as push button pin
#define BlueLed 13              // Pin 13 is defined for Blue LED
#define RedLed 8                // Pin 8 is defined for Red LED

// Initial set up for potentiometer
double Potentio = A0;           // Potentio variable is defined as A0 pin with double datatype (define the potentiometer)
int sensorValue = 0;            // sensorValue is set as integer (The analogue reading from potentiometer)

// Initial Set up for push button mechanism
bool pressed = false;           // pressed is set "false" initially
int rotDirection = 0;           // rotDirection variable is set to 0

// Initial Set up for Encoder
const int encoderPinA = 3;      // Pin 3 interrupt in Arduino will receive encoder value from channel A
const int encoderPinB = 2;      // Pin 2 interrupt in Arduino will receive encoder value from channel B
volatile long encoderPos = 0; // Encoder value will be set as encoderPos variable and will be set as 0 for initial value
                                // Volatile means that this variable can change any time during the interrupt service routine
long previousMillis = 0;        // the previous count value was initialized to zero
long currentMillis = 0;         // the current count value was initialized to zero
int interval = 1000;            // The interval is set to 1000 millisecond / 1 second
                                // since the program was set to calculate rpm and clears encoder pulse counter every 1 second
float rpm = 0;                  // rpm value or the actual speed was initialized to 0
#define ENCODEROUTPUT 650
// the encoder output was set to 650 value since every turn of the motor the encoder outputs 663 times.
// This value was obtained by using trial and error since the exact Count per revolute was not given in this lab

void doEncoderA()           // Function to calculate the encoder Value
{
  encoderPos += (digitalRead(encoderPinA)==digitalRead(encoderPinB))?-1:1; // using ternary operator
  // encoderPos = -1 + encoderPos if true
  // encoderPos = 1 + enncoderPos If false
}
void doEncoderB()
{
  encoderPos += (digitalRead(encoderPinA)==digitalRead(encoderPinB))?1:-1; // using ternary operator
  // encoderPos = 1 + encoderPos if true
  // encoderPos = -1 + enncoderPos If false
}

// Initial Set up for Controller
float Kc = 0.71;      // Kc or proportional constant was set to 0.71 for this experiment, it was obtained using trial and error
float control = 0;    // The control value is the controller output for the proportional controller
float error = 0;      // The error was defined as 0 initially
float PWM;            // PWM value was set as float datatype, it is used to send PWM value to the enable pin


void setup()
{
  Serial.begin(9600);// setup Serial Monitor to display information



  // Encoder and Interrupt
```

```
  pinMode(encoderPinA, INPUT_PULLUP);
// EncoderPinA was set as input_pullup, which the pin is effectively connected through an internal resistor to the VCC+ line
  attachInterrupt(0, doEncoderA, RISING);
 // Interrupt number 0 means for Arduino pin number 2, it will cal the doencoderA function to calculate the encoder value
// RISING is used to trigger the interrupt when pin goes from low to high
  pinMode(encoderPinB, INPUT_PULLUP);
//  EncoderPinA was set as input_pullup, which the pin is effectively connected through an internal resistor to the VCC+ line
  attachInterrupt(1, doEncoderB, FALLING);
// Interrupt number 0 means for Arduino pin number 3
// FALLING is used to trigger the interrupt when pin goes from high to low
  previousMillis = millis();
// The previous count value was started to count
// millis function in here will return the current value of the arduino internal timer in milisecond

  // General Setup for other PINS
  pinMode(P1A, OUTPUT);              // P1A is defined as OUTPUT
  pinMode(P2A, OUTPUT);              // P2A is defined as OUTPUT
  pinMode(EN1, OUTPUT);              // EN1 is defined as OUTPUT
  pinMode(PushButton, INPUT);        // PushButton is defined as INPUT
  pinMode(BlueLed, OUTPUT);          // Blue LED is defined as OUTPUT
  pinMode(RedLed, OUTPUT);           // Red LED is defined as OUTPUT
  pinMode(Potentio, INPUT);          // Potentio is defined as INPUT

  // Initial Rotation direction --- CLOCKWISE -->
  // The initial configuration is set as clockwise hence P1A and P2A are Low and high respectively and
  // RedLED and Blue LED are LOW
  digitalWrite(P1A, LOW);
  digitalWrite(P2A, HIGH);
  digitalWrite(RedLed,LOW);
  digitalWrite(BlueLed,LOW);
  Serial.println("Arduino DC Motor Control"); // Print on the serial monitor
}

void loop()
{
// Generating desired speed (speedz) from potentiometer

float sensorValue = analogRead(Potentio);
// the analog read function will read the potentio value and stored it in sensorValue variable
float speedz;
// speedz was declared and has float datatype
speedz = map(sensorValue, 0, 1023, 0, 265);
// the map function will convert the min and max value of the potentiometer (0-1023) to max and min speed value (0-265)

// Proportional Control
error = speedz - abs(rpm);
// error is obtained from desired speed - actual speed
control = Kc*error + speedz;
// In this case the bias was not constant since the desired speed is always changing according to potentiometer rotation values
// Hence, bias was set as speedz variable
PWM = constrain(control, 0, 255);
// The control value will be converted to pwm value where  0 is the minimum value and 255 is the maximum value
analogWrite(EN1, PWM);
 // The current PWM value will be sent to enable pin 1 and this pwm value will be updated for each iteration by proportional
control


// RPM calculation and it will be updated for every second
```

```
 currentMillis = millis();                          // The previous count value was started to count
 if (currentMillis - previousMillis > interval)     // it will update for every 1 second
 {
  previousMillis = currentMillis;                   // The current count value was set to previous count value after 1 second
  Serial.print("Encoder ");
  Serial.print(encoderPos);                         // Display the encoder value
  Serial.print("| PWM ");
  Serial.print(PWM);                                // Display the PWM value
  // Calculating RPM
  rpm = abs((encoderPos * 60 / ENCODEROUTPUT)); // abs is used so the RPM value is not negative when the motor is rotating
counterclockwise or clockwise direction
  Serial.print("| speedz ");
  Serial.print(speedz);                             // Display the Desired speed value
  Serial.print("| RPM ");
  Serial.print(rpm);                                // Display the RPM value
  Serial.print("| error (speedz - Previous RPM) ");
  Serial.println(error);                            // Display the error value
   encoderPos = 0;                                  // The encoder value will be set back to 0
 }

// Button mechanism
if (digitalRead(PushButton) == true)                // if we press the push button, pressed will have "true" value since we
declared it as "false"
 {
  pressed = !pressed;
 }
while (digitalRead(PushButton) == true);
                                                    // This function enable us to push the button at period of time
                                                    // while the push button is true (being pressed), it will do nothing
// Once we release the pushbutton . . .
// Change rotation direction to CCW
if (pressed == true  & rotDirection == 0)   // at this case, pressed has became "true" and rotDirection is initialized as 0 value in
the beginning
 {
  digitalWrite(P1A, HIGH);              // P1A became HIGH
  digitalWrite(P2A, LOW);               // P2A became LOW
  digitalWrite(RedLed,HIGH);            // Turn on RedLed
  digitalWrite(BlueLed,LOW);            // Turn off BlueLED
  rotDirection = 1;                     // rotDirection is set to 1
 }

// Once we press again the pushbutton and release it, it will go through the first two function button mechanism and pressed
value became false and rotdirection is 1
// Change rotation direction to CW
if (pressed == false & rotDirection == 1)   // at this case, pressed has became "false" and rotDirection is 1
 {
  digitalWrite(P1A, LOW);               // P1A became LOW
  digitalWrite(P2A, HIGH);               // P2A became HIGH
  digitalWrite(RedLed,LOW);             // Turn off RedLEd
  digitalWrite(BlueLed,HIGH);           // Turn On BlueLED
  rotDirection = 0;                     //rotDirection is set to 0
 }
  delay(75);                            // 75 millisecond delay

 delay(50);// 50 millisecond delay
}//loop end
```

Discussion and Conclusions

To sum up, a various kind of DC Motor can be controlled easily using motor driver, in this case L293D motor driver was used in these applications. This motor driver enabled us to control various DC Motors and microcontrollers regardless their voltage specification. The speed of the DC Motor was successfully controlled by adjusting the potentiometer values. Rotating in clockwise direction will lead the motor speed to become maximum and rotating in counter clockwise direction will lead the motor speed to become minimum. The driver also enabled us to control the DC Motor rotation direction. In the analysis and result section, the DC motor was successfully rotated in counter clockwise direction by pushing the push button, hence the red LED will be turned on. Meanwhile, the DC motor was successfully rotated in clockwise direction by pushing the push button again, hence the blue LED will be turned on. For the control part, several disturbances were introduced to the DC motor driver system. The results were the increase of the load will decrease the current supply to the DC motor along with the DC motor voltage. For Kc = 0.71, the systems first have some oscillation and it will fluctuate and became stable at particular value. However, the small errors were still existed, even after the proportional control was implemented. Although the P controller was easy to be tuned, it had drawback that it can cause offset error. To decrease the offset error the gain or Kc need to be increased but increasing the gain means overshoot will also be increased [10] (figure 13). Next, the system under 50ohm load was tested and generated around 70 rpm errors. Then, the error changed to around 50 rpm after Kc was changed to 2. However, the amount of error was still big enough and overshoot will be increased if the gain was increased. Furthermore, from figure 14, it can be seen that PV or process variable response in proportional controller was not good for load change(disturbances). It was different with PI and PID controller that were able to tackle this problem. Overall, the simulation process was worked along with the Kc values even though some errors were occurred. For recommendation, it will be better to use PID controller instead of using P only controller.

References

[1] Components101. “Introduction to Motor Driver: H-Bridge Topology and Direction control”. Internet: https://components101.com/articles/what-is-motor-driver-h-bridge-topology-and-direction-control [29 April 2020].

[2] Texas Instruments. “L293x Quadruple Half-H Drivers”. Internet: http://www.ti.com/lit/ds/symlink/l293.pdf [29 April 2020].

[3] Amazon. “Motor-Encoder-Speed-Gearmotor-Robotics”. Internet: https://www.amazon.com/Motor-Encoder-Speed-Gearmotor-Robotics/dp/B07GNDG2NC
[29 April 2020].

[4] Lastminuteengineers. “L293D DC Motor Arduino Tutorial”. Internet: https://lastminuteengineers.com/l293d-dc-motor-arduino-tutorial/ [29 April 2020].

[5] Engineergarage. “L293D Pin Description and Working”. Internet: https://www.engineersgarage.com/stm32/l293d-pin-description-and-working/ [29 April 2020].

[6] ScienceDirect. “Proportional Control”. Internet: https://www.sciencedirect.com/topics/engineering/proportional-control [29 April 2020].

[7] UNHAS. “Practical Process Control”. Internet: http://www.unhas.ac.id/rhiza/arsip/arsip-macam2/practicalprocesscontrol.pdf [29 April 2020].

[8] Medium. “An Introduction to PID Control with DC Motor”. Internet: https://medium.com/luos/an-introduction-to-pid-control-with-dc-motor-1fa3b26ec661 [29 April 2020].

[9] Instructables. “Rotary Encoder - Understand and Use It (Arduino/other MController)”. Internet: https://www.instructables.com/id/Rotary-Encoder-Understand-and-Use-It-Arduinoother-/ [29 April 2020].

[10] Control Tutorials for Matlab and Simulink. “Introduction: PID Controller Design”. http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID [29 April 2020]

[11] OptiControls. “Derivative Control Explained”. https://blog.opticontrols.com/archives/153 [29 April 2020]

Appendices

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Decrease
Kd	Small Change	Decrease	Decrease	No Change

Figure 13 Controller Parameter Effects on Close Loop System [10]

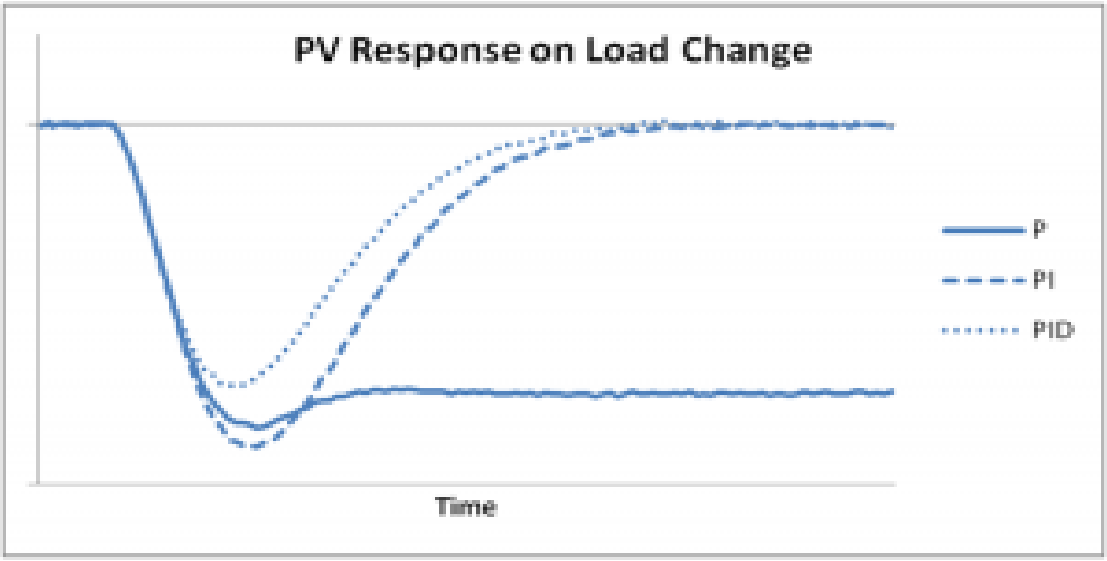

Figure 14 Comparison between P, PI, and PID Controller on Ability to Recover from Disturbance [11]