DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEEE3075 / Mechatronics Laboratory

Lab 4 DC Motor with PID and Servo Motor

Name: Jason Samuel Pangestu

Student ID: 16522575

Table of Content

Introduction

DC Motor is one of the most popular actuators that are used for industrial applications. In this report, a DC Motor is controlled by Arduino Uno R3 Microcontroller and the whole system will be designed and simulated in Tinkercad Software. In addition, the modification of the DC motor or also known as servo motor is investigated in this project and simulated in Tinkercad software. In the simulation software, a DC motor with an encoder model and Hobby servo motor is used in the simulations. These can be seen from the figure below. Next, a motor driver IC is introduced as the bridge between a microcontroller and DC Motor. This driver is needed since most DC Motors requires 5V- 12V to operate while microcontrollers have an operating voltage of 3.3V or 5.5V [1]. The motor driver IC that is used in this project was L293D. This driver is capable of drive bidirectional currents up to 600mA and voltages from 4.5V to 36V [2]. This driver is also able to control the speed and direction of DC Motor. Furthermore, PID control is also implemented in this project. The desired speed is set and compared to the actual speed. For the servo, the Arduino will control the servo rotation by changing the pulse width. Thus, the servo shaft can be rotated to the desired position. Another control by using two push buttons is also implemented in this project. This will enable the user to rotate the servo in a clockwise or counterclockwise direction with the desired increment. In this reports, basic configuration and theory aspect of the motor driver and PID controller will be provided in the method section while testing and simulation will be shown in analysis and results section. The same method is also applied to the servomotor control part.

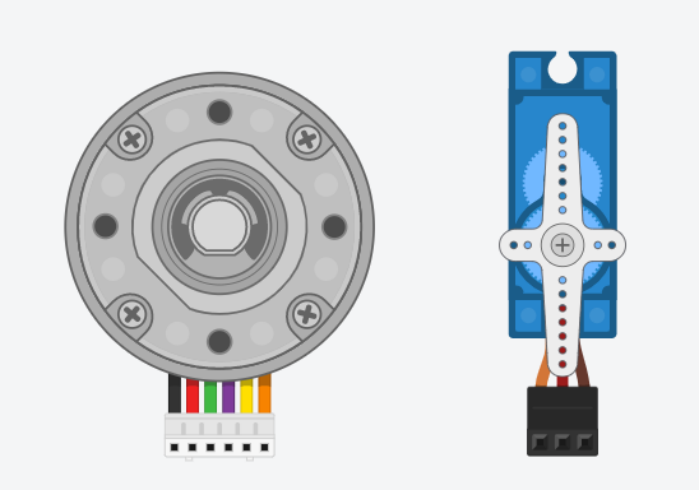

Figure 1 Left: DC Motor with Encoder. Right: Servo Motor

Methods

a. DC Motor PID Control

1. L293D Motor Driver

This motor driver is composed of 16 pins (Figure 2). Pin 1 and Pin 9 are 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 are pin 4, 5, 12, and 13. For the input pins, there are pin 2 and pin 7 at the left side and pin 15 and pin 10 at the right side. These input pins will receive the signal from devices such as a 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 the motor power supply (4.5 to 36V). For pin 16, it will be connected to 5V to drive internal logic circuitry.

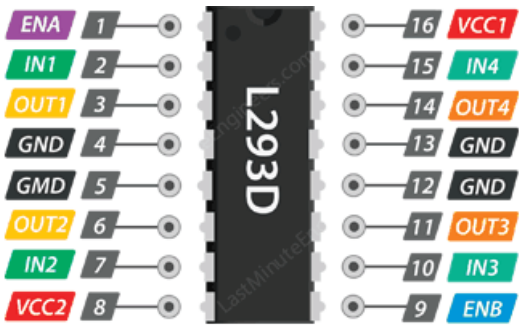

Figure 2 L294D Pinout Diagram [3]

2. Controlling Motor Direction

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

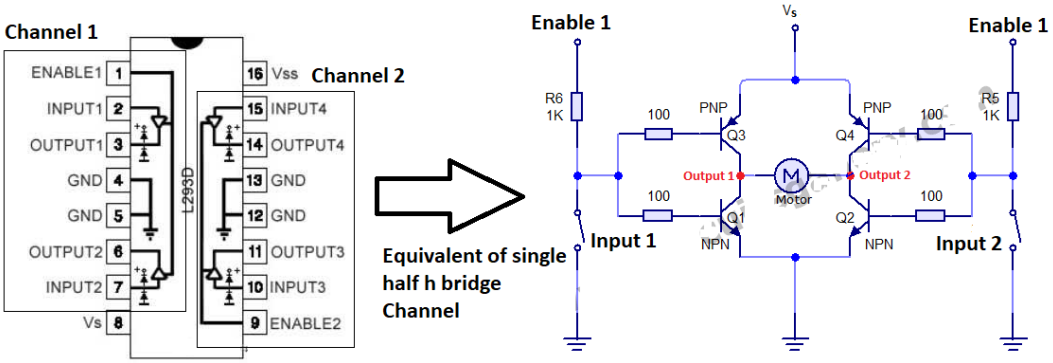

Figure 3 Single Half H-Bridge Representation [4]

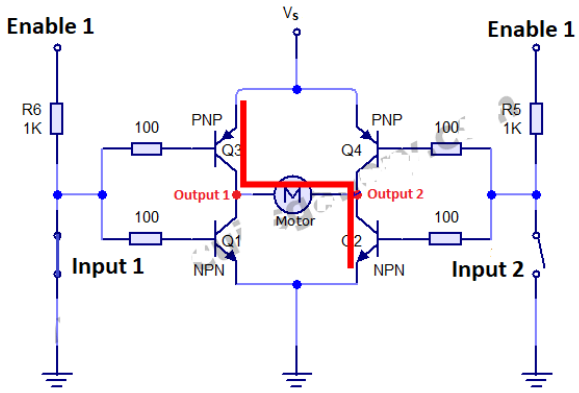

Figure 4 Single Half H-Bridge Representation when pin 2 is HIGH and pin 7 is LOW [4]

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 are rotating in clockwise direction. In this case, pin 2 is HIGH, pin 7 is LOW, and Enable pin is HIGH.

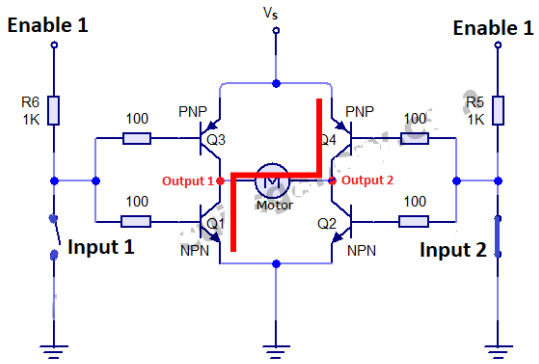

Figure 5 Single Half H-Bridge Representation when pin 2 is LOW and pin 7 is HIGH [4]

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 is HIGH. By using this technique, changing the motor direction can be applied immediately. In addition, when both pin 2 and 7 were low there are no rotation. It is also the same when both pin 2 and 7 are HIGH. For addition information, it is 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 is 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.

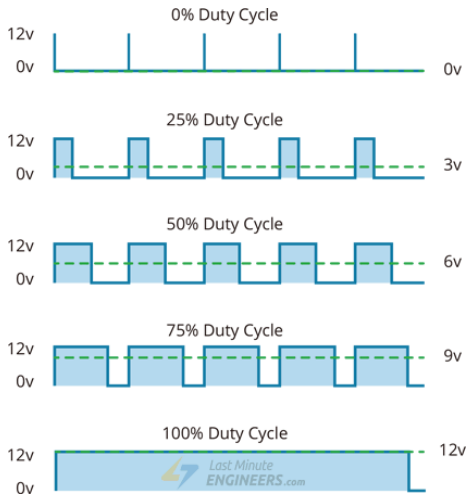

Figure 6 PWM with Different Duty Cycle [3]

4. PID Control

PID controller is the most common feedback and it is used in many control system applications. The terms P stands for Proportional, I stands for Integral, and D stands for derivatives. The figure below is the equation for PID Control.

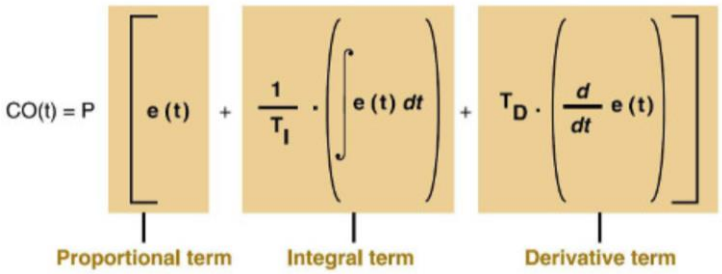

Figure 7 PID Control Equation [5]

For setting the PID gains, trial and error method are the most common method. Generally, the Ki and Kd are set to zero and Kp is increased until the system reaches to oscillating behaviour. Then, Ki and Kd values are tuned accordingly. Each of the parameters effect on the close loop system are shown in the figure below. If Kp and Ki are increased too much, overshoot will happen. Increasing Kd too much can cause the settling time to decrease.

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 8 Kp, Ki, and Kd Effect on Closed Loop System [6]

The figure 9 below shows the basic concept of a closed loop feedback system for DC Motor. The actual speed is subtracted with the desired speed resulting the error. This error is fed back to the control system to achieve the desired speed. Then, the speed is generated by the PWM voltage. The actual speed of the DC motor is detected by the speed sensor and is fed back again to the control system. While the last figure in this page shows the DC motor encoder 45RPM that is going to be used in the tinkercad simulation.

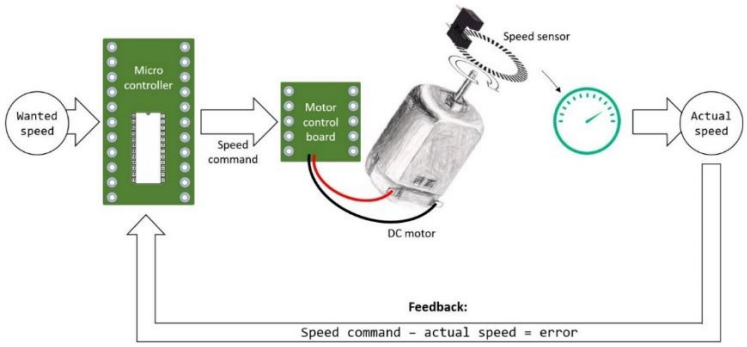

Figure 9 Closed Loop Feedback System for DC Motor Control [7]

Figure 10 DC Motor Encoder 45 RPM [8]

b. Servomotor Control

Servo Motors is a motor whose shaft position can be accurately controlled by varying the PWM signal voltage. It uses an internal "Servomechanism" and gearbox to provide positional feedback. It is commonly used in industrial and hobbyist applications. The most common hobbyist servo is the servo with 180 degrees range. In practice, the range is typically less than 180 degrees range, and it depends on the servo manufacturer. Another type is the 270 degrees servo motor and continuous rotation servo motors. For the continuous rotation servo motors, 1.5ms of pulse width will stop the servo rotation, 1ms will make the servo to rotate counterclockwise, and 2.0ms will make the servo to rotate clockwise,

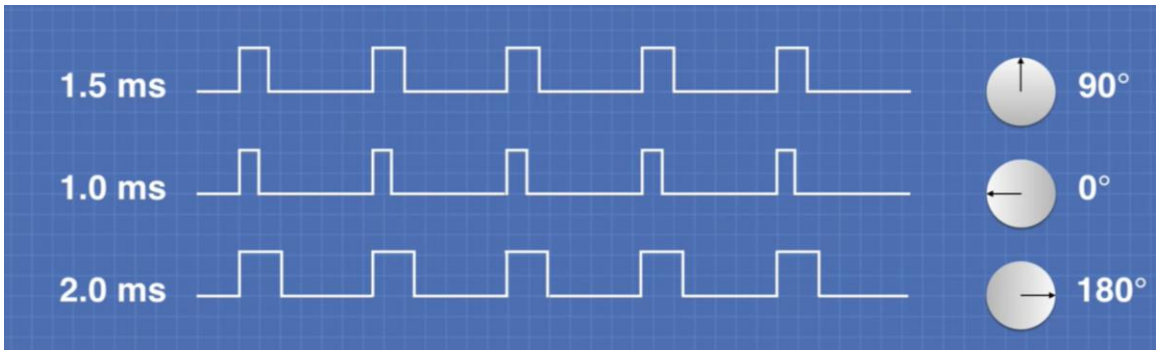

Figure 10 Pulse Width Modulation in Servo Motors [9]

The above figure shows the pulse width modulation variations in servo motors with 180 degrees range. 1.5ms will set the servo position to a 90-degree position. Meanwhile, 1.0ms of pulse width modulation will set the servo position to a 0-degree position. Finally, 2.0ms of pulse width modulation will set the servo position to 180-degree position. These are the common rules for controlling servo position with pulse width modulation. Nevertheless, these values can be different for each servo since it depends on the manufacturer.

Figure 11 Servo Motor

It is clearly illustrated at the above figure that the servo motor has a 3 pin connection. In this project, the servo motor pin 1 is the control signal, which denoted by orange cable. Servo motor pin 2 is the Power line, which is denoted by red cable. Lastly, the servo motor pin 3 is the ground which is denoted by a black cable. In practical applications, usually, a small size capacitor can be put in parallel to the control signal and ground. This approach can reduce the noise that occurs from the power supply, or it can eliminate the servo behaviour for jittering.

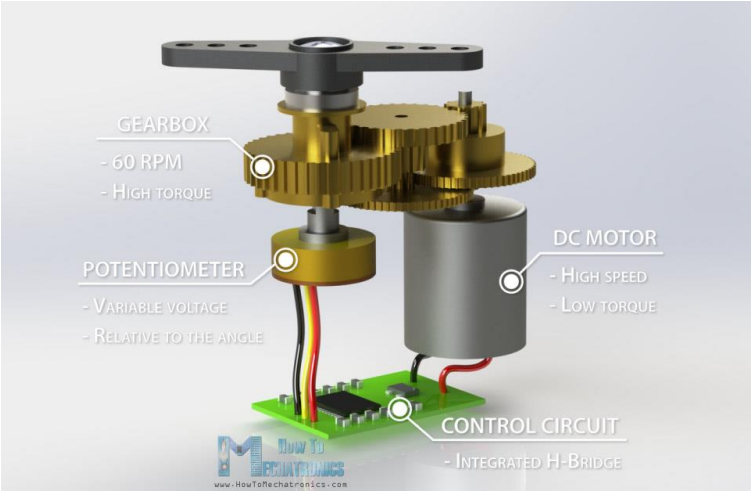

Figure 12 Servo Motor Inside Mechanism [10]

The inside mechanism of the hobby servo consists of a gearbox, DC motor, potentiometer, and control circuit. The above figure shows the inside mechanism of the servo motor. Each of these components has a specific function to operate the servo motor closed-loop system. The DC motor commonly generates high speed but has low value in torques. Next, the low torque is transformed by the gearbox to high torque. Meanwhile, the output RPM is also decreased after it through the gearbox. Besides, the potentiometer is attached to the system and moves as the servo motor rotates. This potentiometer gives the actual position of the servo shaft to the control circuit. This control circuit then compares the actual position of the servo shaft and the desired position of the servo shaft. The figure below shows the general concept of the servo motor closed-loop system.

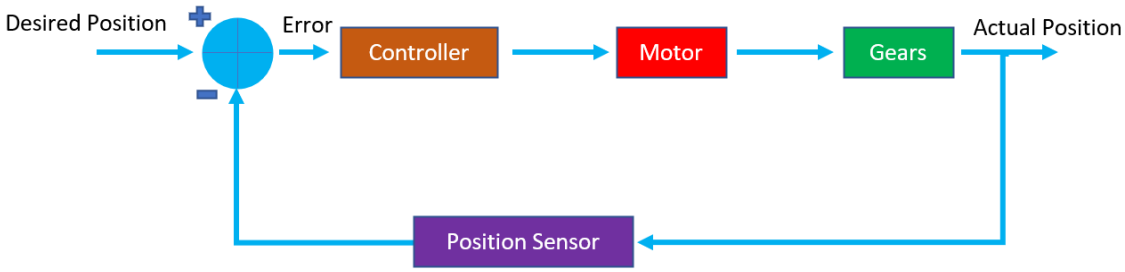

Figure 13 Servo Motor Closed-Loop System

Result and Analysis

5.0 DC Motor PID Control

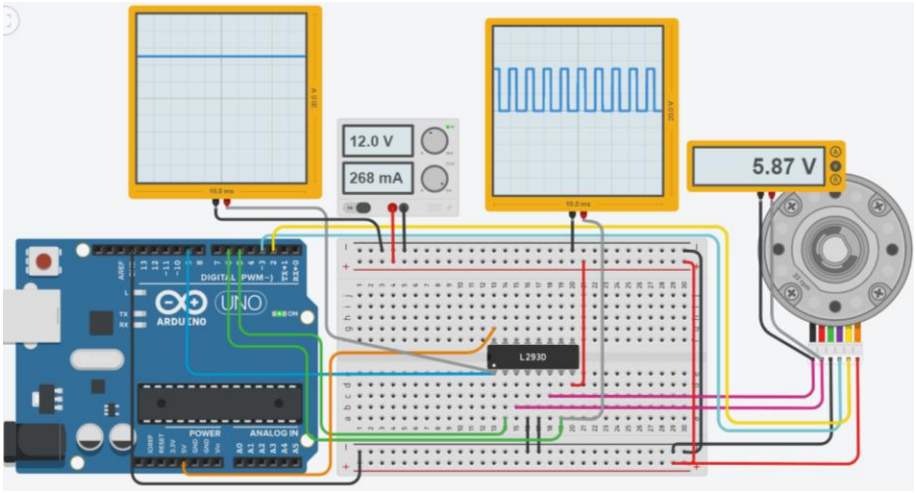

Figure 14 Schematic for DC Motor PID Control

From the above figure, it can be seen that the Arduino Uno R3 is placed at the bottom left. The Arduino provides power to the L293D via the orange cable (pin 16 → to drive internal logic circuitry). Next, the ground from Arduino connects to the ground line, including the external power supply ground connection. For pin 5 and 6, it is connected to output pin 2 and 7 of L293D respectively. The green cable sends an either HIGH or LOW signal from Arduino to the L293D. These pins are also able to send PWM signal. Next, pin 9 Arduino (Able to send PWM signal) is connected to pin 1 of L293D via purple cable. This pin 1 enables all the pins at the bottom side and delivers the PWM input signal from the Arduino. For Arduino pin 2 and pin 3, it is connected to DC Motor encoder of Channel A (Yellow) and Channel B (Purple) respectively by yellow and turquoise cable. These pins act as an interrupt to receive encoder values from the DC motor. For the L293D gate driver chip, it is located at the centre. All the ground pins from this chip (pin 4,5,12, and 13) are connected to the ground line.

Meanwhile, pin 8 is connected to the external power supply. This red cable delivers supply voltage for the DC motor. For the output pins (pin 3 and 6), it is connected to DC motor terminals by purple cables. Furthermore, the external power supply is located at the top and able to supply 12V and 4.8A to the circuit. The red cable is the positive line and the black cable is the negative line. For the DC motor encoder, a 45 RPM type is used. The motor terminal of this DC motor encoder along with channel a and b was 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, a voltmeter is set parallel to the motor terminal to check the supply voltage to the DC motor. At the top, the oscilloscope at the left displays the voltage at the enable pin. Meanwhile, the oscilloscope at the right displays the PWM value generated by pin 6 Arduino. This PWM value is generated to obtain the desired speed.

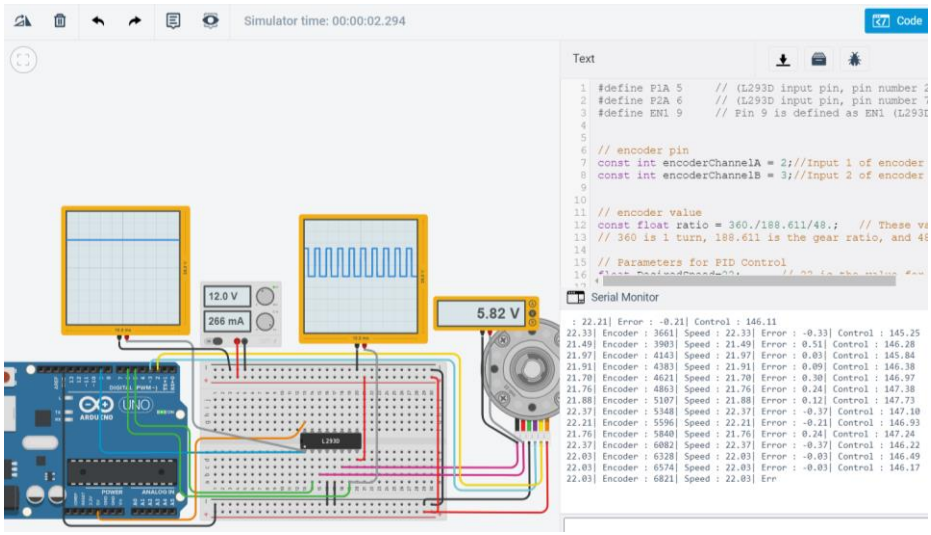

Figure 15 Simulation Picture of DC Motor PID Control

The above figure shows the simulation picture of the DC Motor PID Control. In this case, the desired speed is set to 22RPM and the max value of this DC motor is 45 RPM. It can be seen from the serial monitor that the encoder, speed, error, and control values are shown. The encoder shows the encoder value at the current time. Meanwhile, the speed represents the actual speed of DC Motor. The error shows the difference between the actual and desired speed. Meanwhile, the control shows the control variable from the PID. It is clearly illustrated that the actual speed is fluctuating around 22 RPM and the error is very small (below 1RPM). From the right oscilloscope, the pulse duration is around 0.5ms, and this is half of the PWM max value. This right oscilloscope measures the PWM output from Arduino digital pin 6. The max pulse duration generates maximum DC motor speed and half pulse duration generates half DC motor speed. In addition, it is showed that the operating voltage is around 5.8V. The enable pin also is in high condition or 5V of voltage.

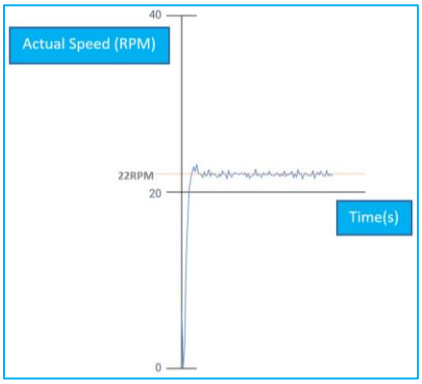

Figure 16 Actual Speed versus Time Graph

The above figure displays the actual speed of the DC motor vs time. The actual speed initially increases from 0RPM to around 22.5RPM. Then, it fluctuates around 22 RPM until the end time. In this case, the PID values (Kp = 1.29, Ki = 0.031, and Kd = 0.01) are valid for this process.

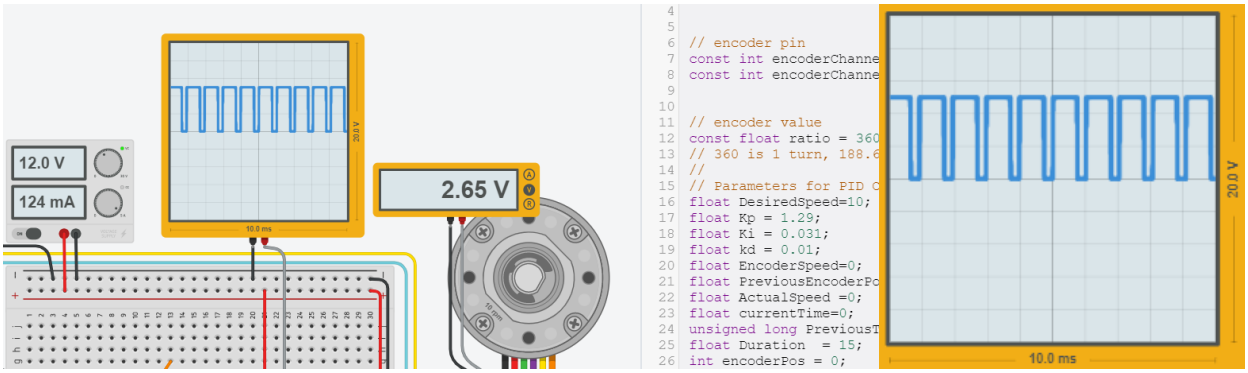

Figure 17 Left:Simulation Results when DesiredSpeed = 10RPM. Right: PWM Signal of Pin 6

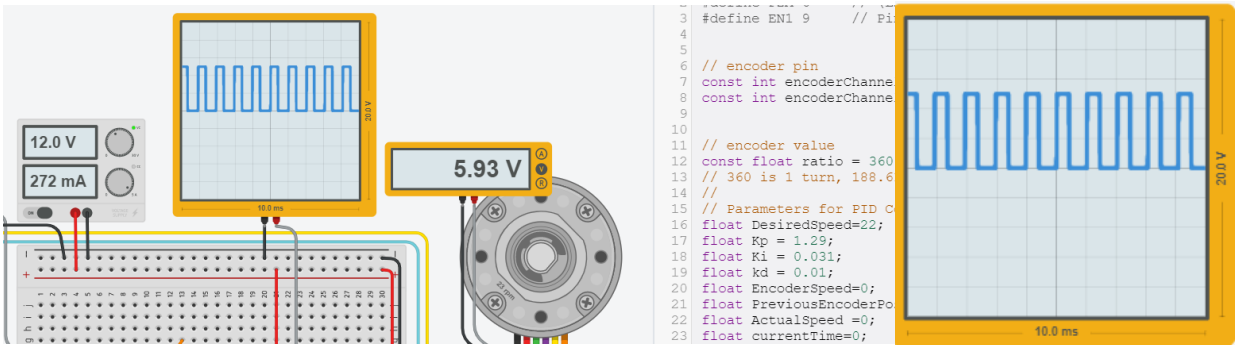

Figure 18 Left: Simulation Results when DesiredSpeed = 22RPM. Right: PWM Signal of Pin 6

These above two figures show the comparison of the simulation results when the desired speed is equal to 10RPM and 22RPM. When the desired speed is 10RPM, the DC motor consumes around 2.65V. Meanwhile, the DC motor uses 5.93V to rotate at 22RPM speed. It is clearly illustrated that the PWM signal of pin 6 on both figures shows the voltage difference that applied to the DC motor. In addition, the same PID values are used for these two systems (when the desired speed is 10RPM and 22RPM). The following pages show the overall code for the DC Motor PID Control.

```
#define P1A 5    // (L293D input pin, pin number 2) , control direction of rotation
#define P2A 6   // (L293D input pin, pin number 7) , To control the PWM
#define EN1 9   // Pin 9 is defined as EN1 (L293D enable pin, pin number 1)
// encoder pin
const int encoderChannelA = 2;//Input 1 of encoder
const int encoderChannelB = 3;//Input 2 of encoder
// encoder value
const float ratio = 360./188.611/48.; // These values are obtained from the data sheet
// 360 is 1 turn, 188.611 is the gear ratio, and 48 is the motor shaft Countable Events per revolution
// Parameters for PID Control
float DesiredSpeed=22;          // 22 is the value for Desired Speed in RPM
float Kp = 1.29;                // proportional constant is 1.29
float Ki = 0.031;               // integral constant is 0.031
float kd = 0.01;                // derivative constant is 0.01
float EncoderSpeed=0;           // the speed of the encoder is declared and set as 0
float PreviousEncoderPos=0;     // PreviousEncoderPos is declared and set as 0
float ActualSpeed =0;           // ActualSpeed is declared as 0
float currentTime=0;            // currentTime variable is declared and set as 0
unsigned long PreviousTime   = 0; // PreviousTime variable is declared and set as 0
float Duration  = 15;           // Duration variable is declared and set as 15
int encoderPos = 0;             // Encoder pos is declared and set as 0
float error=1;                  // Error is declared and set as 1 initially
float previouserror = 0;        // Previouserror is declared and set as 0 initially
float SumError = 0;             // SumError is declared and set as 0 initially
// Initiate Variable for the Motor Motion
bool a; // a is the variable for setting the motor direction, in this case CCW
int b;  // b is the variable for controlling the PWM signal


 void doEncoderA() //Fuction to Counting rotation on encoder in pin A
 {
   encoderPos += (digitalRead(encoderChannelA)==digitalRead(encoderChannelB))?1:-1; // using ternary operator
   // encoderPos = 1 + encoderPos if true // encoderPos = -1 + encoderPos If false
 }
 void doEncoderB()//Fuction to Counting rotation on encoder in pin B
 {
   encoderPos += (digitalRead(encoderChannelA)==digitalRead(encoderChannelB))?-1:1;// using ternary operator
   // encoderPos = -1 + encoderPos if true // encoderPos = 1 + encoderPos If false
 }
 void setup()
 {
   Serial.begin(9600);
   // setup Serial Monitor to display information
   pinMode(encoderChannelA, INPUT_PULLUP);
   // EncoderchannelA was set as input_pullup, which the pin is effectively connected through an internal resistor to the VCC+ power line
   attachInterrupt(0, doEncoderA, CHANGE);
   // Interrupt number 0 means for Arduino pin number 2, it will cal the doencoderA function to calculate the encoder value
   // CHANGED is used to trigger the interrupt whenever the pin change the value
   pinMode(encoderChannelB, INPUT_PULLUP);
   // EncoderchannelB was set as input_pullup, which the pin is effectively connected through an internal resistor to the VCC+ power line
   attachInterrupt(1, doEncoderB, CHANGE);
   // Interrupt number 1 means for Arduino pin number 3,
   // CHANGED is used to trigger the interrupt whenever the pin change the value
   pinMode(P1A, OUTPUT);   // P1A is defined as output
   pinMode(EN1, OUTPUT);   // EN1 is defined as output
 }

  void loop()
  {
      for (int i = 0; i <= 100; i++)  {  // Iteration for 100 times
    currentTime = millis();       // Use Millis function to obtain the current time
    if(currentTime ==0)         // the first count is set to 1
    {currentTime =1;}
      Duration = (double)(currentTime - PreviousTime); // Calculate the duration time
    PreviousTime=currentTime;              // CurrentTime is set as previous time
    SumError += error * Duration;            // Total error is obtained from error times duration
     previouserror=error;                  // current error is set as the previous error
    EncoderSpeed=encoderPos-PreviousEncoderPos;    // The encoder speed is obtained from encoderPos - PreviousencoderPos
    PreviousEncoderPos=encoderPos;           // the current encoder pos is set as the previous encoderpos
    // Actual Speed Calculation
    ActualSpeed = float(EncoderSpeed)*ratio;
    ActualSpeed = ActualSpeed*1000*60/(360*Duration);//RPM
    error = DesiredSpeed - ActualSpeed;        // Error is obtained from the difference between Desired and actual speed
    float control = Kp*error+Ki*SumError+kd*(error-previouserror)*Duration;// Formula for PID
    digitalWrite(EN1, 255);          // send 255 PWM signal or 5volt to enable pin of L293D
    a = (control>=0)?HIGH:LOW;       // If control is greater than 0, a is set HIGH
    b =  min(abs(control), 255);     //  The control value is taken as absolute and limit to 255 value, cannot exceed this value
    analogWrite(P2A, a?(255 - b):b); // if a is HIGH the 255-b value will be sent to control the desired PWM value
    digitalWrite(P1A,a);             // send High or low value to Arduino pin a,in this case, HIGH is CCW


     Serial.print(ActualSpeed); // Display the actual speed in the graph_
    Serial.print("| Encoder : "); // Display the Encoder Value
    Serial.print(encoderPos);
    Serial.print("| Speed : "); // Display the Speed Value
    Serial.print(ActualSpeed);
    Serial.print("| Error : "); // Display the Error Value
    Serial.print(error);
    Serial.print("| Control : "); // Display the Control Value
    Serial.println(control);
      }
    Serial.print("Process Finished");
    while(1); // Function to stop the iteration from looping
      }
```

Figure 19 Code for PID Control

5.1 Generate the PWM signal so servomotor is at the centre

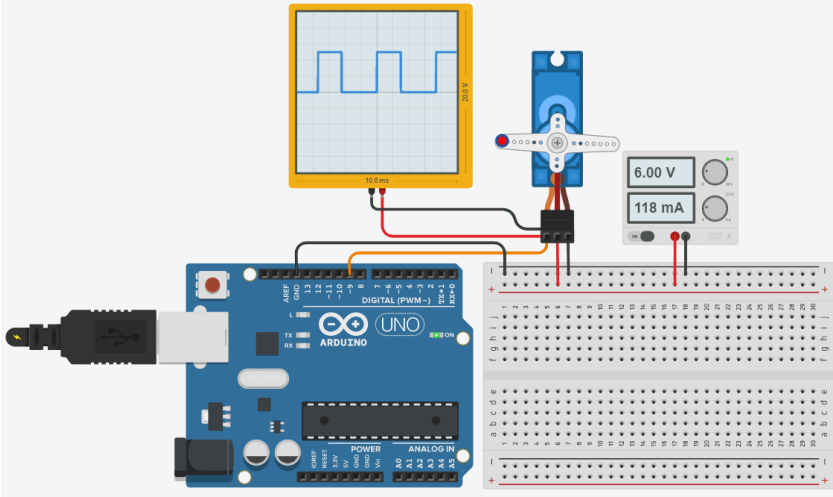

Figure 20 Servo Motor Position at the Center (Circuit)

```
// Task 5.1 ServoMotor at Center
void setup()
{
  pinMode(9,OUTPUT);            // The servo signal is connected to pin 9 in Arduino as an Output
}
void loop()
{
  int x = 1450;
  digitalWrite(9,HIGH);
  delayMicroseconds(x);
  digitalWrite(9,LOW);
  delayMicroseconds(20000-x);
}
```

Figure 21 Code for Task 5.1

The above figure (21), it can be seen that the servo is connected to the Arduino by the orange cable. The PWM signal is sent from Arduino digital pin 9 to the servo control signal port. The power supply uses a supply voltage of 6V and a supply current of 1A. Additionally, the Arduino ground is connected to the circuit ground line. To display the PWM value, the oscilloscope, which has 1ms time per division, is connected in parallel to the servo control signal and ground.

It can be seen from figure 22 that the code is written to rotate the servo at the centre position. From trial and error, the pulse has a duration of 1.45 ms. Meanwhile, one period is last for 20ms. First, it generates a high signal for 1.45ms and generates a low signal for 18.55ms. It is clearly illustrated from figure 21 that the servo horn position (red dot) is at the centre or 90-degree position. Moreover, the oscilloscope also shows the PWM signal width for 1.45ms.

5.2 Generate the PWM signal so servomotor is at 90 degrees CW from centre

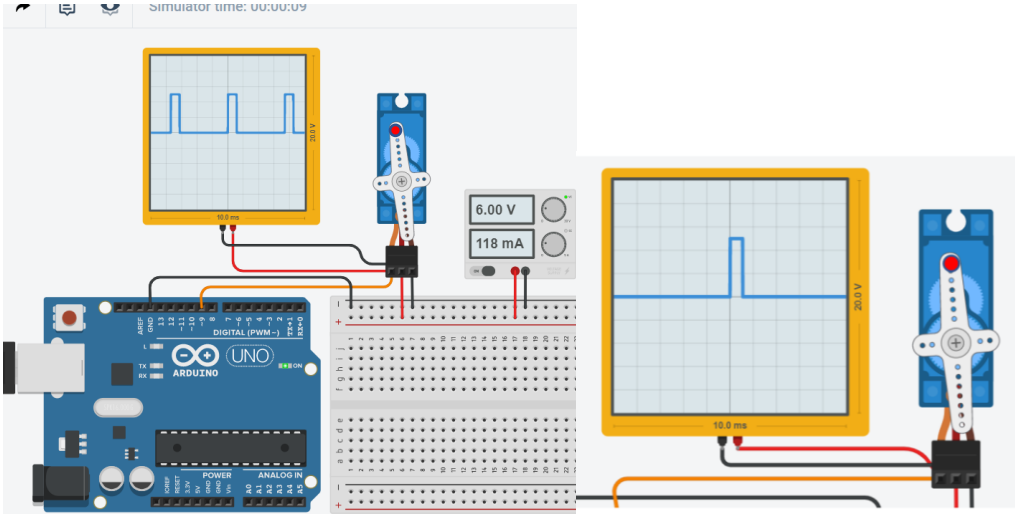

Figure 22 Left: Servo Motor Position at 90 Degrees CW from Center (Circuit). Right: Servo Motor Position at 0°

```
// Task 5.2
void setup()
{
  pinMode(9,OUTPUT);            // The servo signal is connected to pin 9 in Arduino as an Output
}
void loop()
{
  int x = 540;
  digitalWrite(9,HIGH);
  delayMicroseconds(x);
  digitalWrite(9,LOW);
  delayMicroseconds(20000-x);
}
```

Figure 23 Code for Task 5.2

The above figure 23, it can be seen that the servo is connected to the Arduino by the orange cable. The PWM signal is sent from Arduino digital pin 9 to the servo control signal port. The power supply uses a supply voltage of 6V and a supply current of 1A. Additionally, the Arduino ground is connected to the circuit ground line. To display the PWM value, the oscilloscope, which has 1ms time per division, is connected in parallel to the servo control signal and ground.

It can be seen from figure 24 that the code is written to rotate the servo 90 degrees clockwise from the centre position. From trial and error, the pulse has a duration of 0.54ms. Meanwhile, one period is last for 20ms. First, it generates a high signal for 0.54ms and generates a low signal for 19.46ms. It is clearly illustrated from figure 23 that the servo horn position (red dot) is at 90 degrees CW from a centre or 0-degree position. Moreover, the oscilloscope also shows the PWM signal width for 0.54ms. From figure 23 right picture, the servo is set to 0-degree position by using the servo library to control the output rotation angle directly. It can be seen that this servo position is the same as the servo position in the simulation.

5.3 Generate the PWM signal so servomotor is at 90 degrees CCW from centre

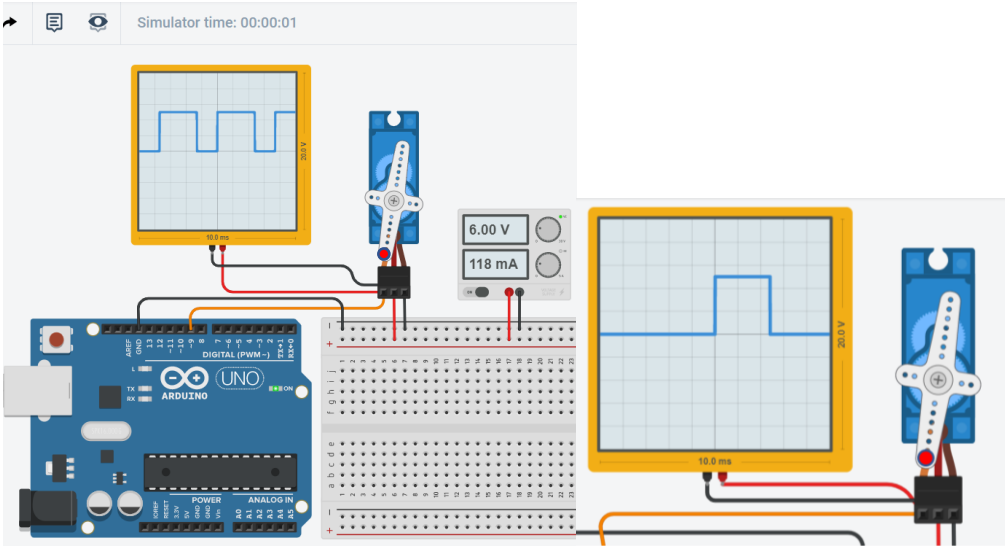

Figure 24 Servo Motor Position at 90 Degrees CCW from Center (Circuit). Right: Servo Motor Position at 180°

```
// Task 5.3
void setup()
{
  pinMode(9,OUTPUT);            // The servo signal is connected to pin 9 in Arduino as an Output
}
void loop()
{
  int x = 2350;
  digitalWrite(9,HIGH);
  delayMicroseconds(x);
  digitalWrite(9,LOW);
  delayMicroseconds(20000-x);
}
```

Figure 25 Code for Task 5.2

The above figure 25, it can be seen that the servo is connected to the Arduino by the orange cable. The PWM signal is sent from Arduino digital pin 9 to the servo control signal port. The power supply uses a supply voltage of 6V and a supply current of 1A. Additionally, the Arduino ground is connected to the circuit ground line. To display the PWM value, the oscilloscope, which has 1ms time per division, is connected in parallel to the servo control signal and ground.

It can be seen from figure 26 that the code is written to rotate the servo 90 degrees counterclockwise from the centre position. From trial and error, the pulse has a duration of 2.35ms. Meanwhile, one period is last for 20ms. First, it generates a high signal for 2.35ms and generates a low signal for 17.65ms. It is clearly illustrated from figure 25 that the servo horn position (red dot) is at 90 degrees CCW from the centre or 180-degree position. Moreover, the oscilloscope also shows the PWM signal width for 2.35ms. From figure 25 right picture, the servo is set to 180-degree position by using the servo library to control the output rotation angle directly. It can be seen that this servo position is the same with the servo position in the simulation.

5.4 Servo Control CW and CCW with 2 Buttons

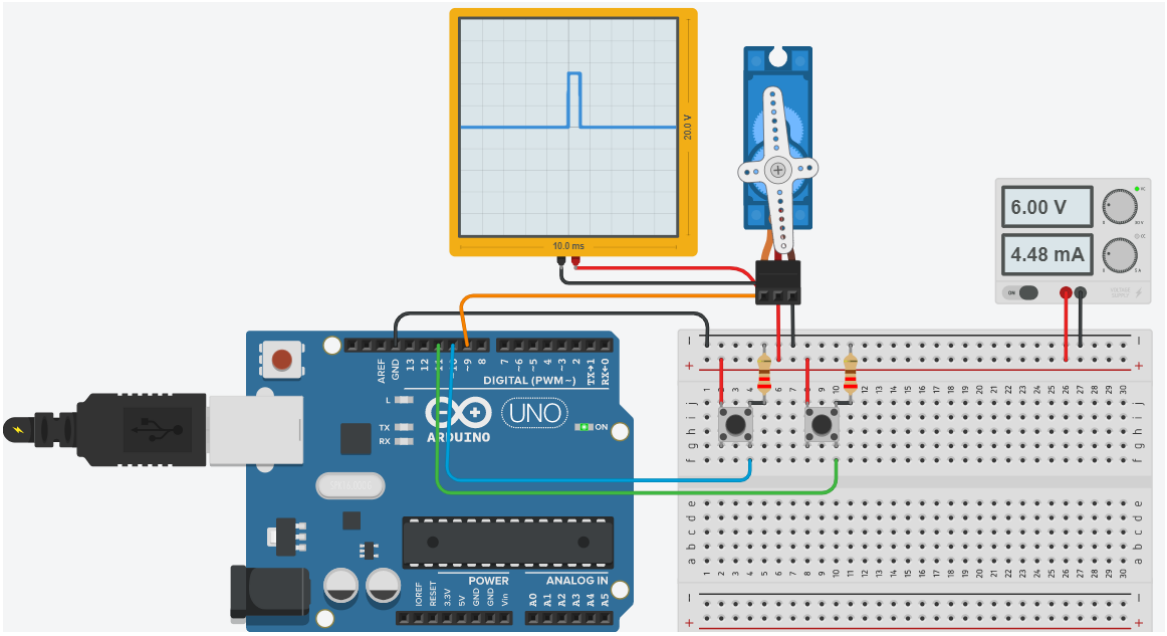

Figure 26 Servo Control CW and CCW with 2 Buttons (Circuit)

The above figure (27), it can be seen that the servo is connected to the Arduino by the orange cable. The PWM signal is sent from Arduino digital pin 9 to the servo control signal port. The power supply uses a supply voltage of 6V and a supply current of 1A. Additionally, the Arduino ground is connected to the circuit ground line. To display the PWM value, the oscilloscope, which has 1ms time per division, is connected in parallel to the servo control signal and ground. In this circuit, two buttons are introduced to control the servo rotation for 20 degrees. The left button is connected to the power line on one side while another side is connected to the 220ohm resistor and Arduino pin 10. The right button is connected to the power line on one side while another side is connected to the 220ohm resistor and Arduino pin 11. Moreover, these resistors are connected to the ground line.

For controlling the push button, several functions are placed in the code to make the push-button fully functional. The button mechanism is designed that pushing the button for a long time will not affect the result. Thus, the next command will be executed after an appropriate push button is released. Furthermore, the servo is controlled by using a servo object in Arduino. This can be used after calling the servo library in Arduino. By using this, the servo position in degree can be controlled directly with *write* command for servo object. The results of the simulations are displayed on the next page.

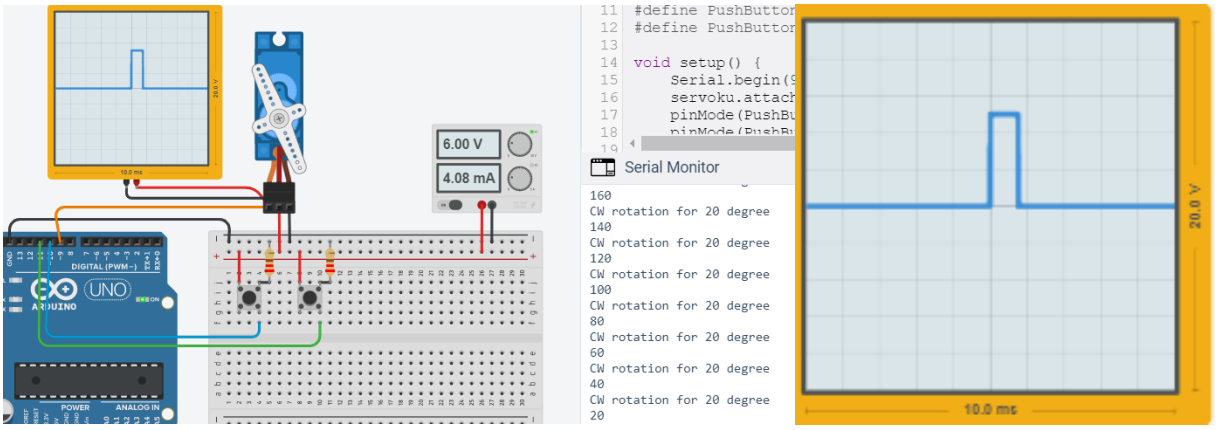

Figure 27 Left: Simulation Picture (1). Right: PWM Signal (1)

The above figure at the left shows the servo position at a 20-degree angle. It can be seen from the serial monitor that the servo starts at a 160-degree position. Then the right button is used to rotate the servo for 20 degree in a clockwise direction. The right button is clicked for several times until the servo position is at 20 degree-angle. It can also be seen that the oscilloscope shows the pulse width at almost 1ms.

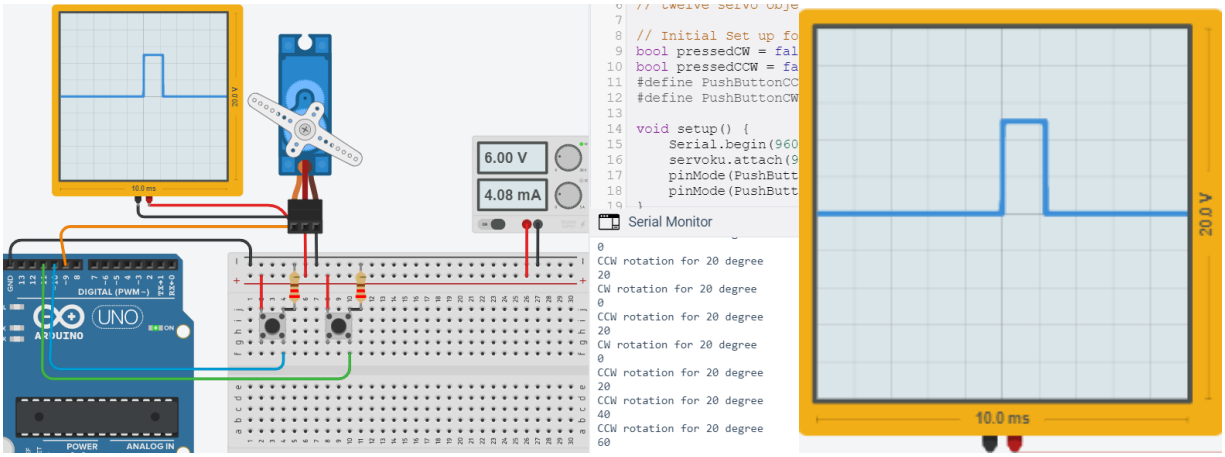

Figure 28 Left: Simulation Picture (2). Right: PWM Signal (2)

The above figure at the left shows the servo position at a 60-degree angle. It can be seen from the serial monitor that the servo starts at a 0-degree position. Then the right button is used to rotate the servo for 20 degrees in a counterclockwise direction. Next, the left button is clicked, so the servo rotates back to 0 degrees. Then, the right button is clicked again, so the servo rotates to a 20-degree position. These two processes are repeated, and then the right button is clicked twice to achieve a servo position of 60 degrees. It can also be seen that the oscilloscope shows the pulse width at around 1ms. The following page shows the overall code for this task.

ServoControl_2PushButtons_ | Arduino 1.8.12

File Edit Sketch Tools Help

ServoControl_2PushButtons_

```
#include <Servo.h>
int pos = 0;
Servo servoku;  initialize servoku as servo object

// Initial Set up for push button mechanism
bool pressedCW = false;    // pressedCW is set "false" initially
bool pressedCCW = false;   // pressedCCW is set "false" initially
#define PushButtonCCW 10   // Pin 10 is defined as push button pin to increment 20 degree in CCW direction
#define PushButtonCW 11    // Pin 11 is defined as push button pin to increment 20 degree in CW direction

void setup() {
    Serial.begin(9600);// setup Serial Monitor to display information
    servoku.attach(9);  // attaches the servo on pin 9 to the servo object
    pinMode(PushButtonCW, INPUT); // PushButtonCW is defined as INPUT
    pinMode(PushButtonCCW, INPUT); // PushButtonCCW is defined as INPUT
}
void loop()
{
  servoku.write(pos); // Write the servo position in degree (1-180)


// Left Button Mechanism
if (digitalRead(PushButtonCCW)==HIGH)  // if we press the push button, pressed will have "true" value since we declared it as "false"
{
 pressedCCW = !pressedCCW;
}
while(digitalRead(PushButtonCCW)==HIGH) // 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 . . .
if (pressedCCW == true && pos<180)      // If these conditions are true, the servo will rotate 20 degree in CCW direction
{
    pos = pos + 20;                     // Incrementing the pos value
    delay(30);                          // Delay for 30ms
    pressedCCW = !pressedCCW;           // Change the condition for pressedCCW
    Serial.println("CCW rotation for 20 degree"); // Print on the serial monitor
    Serial.println(pos); // Print on the serial monitor
}

// Right Button Mechanism
if (digitalRead(PushButtonCW)==HIGH)    // if we press the push button, pressed will have "true" value since we declared it as "false"
{
 pressedCW = !pressedCW;
}
while(digitalRead(PushButtonCW)==HIGH)  // 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 . . .
  if (pressedCW == true && pos>0)       // If these conditions are true, the servo will rotate 20 degree in CW direction
{
    pos = pos - 20;                     // Decrementing the pos value
    delay(30);                          // Delay for 30ms
    pressedCW = !pressedCW;             // Change the condition for pressedCW
    Serial.println("CW rotation for 20 degree"); // Print on the serial monitor
    Serial.println(pos);                         // Print on the serial monitor
}
```

Figure 29 Code for Task 5.4

Discussion and Conclusions

To sum up, various kind of DC Motor can be controlled easily using a motor driver, in this case, L293D motor driver is used in these applications. This motor driver enabled us to control various DC Motors and microcontrollers regardless of their voltage specification. In this case, several speed values in RPM are tested in the system, and all of them are able to achieve the desired speed with an error less than 1RPM. In addition, the settling time also happens in a short time. Thus, the PID controller constants are valid, and the PID controller method is verified. For the servo control, the PWM values are successfully set for servo position at 90 degrees or at the centre, servo position at 0 degrees or at 90 degrees CW from the centre, and servo position at 180 degrees or at 90 degrees CCW from the centre. Furthermore, the servomotor is controlled by 2 buttons to rotate in CW or CCW direction. The right push-button is used to control increment of 20 degrees in a clockwise direction, and the left push-button is used to control increment of 20 degrees in a counterclockwise direction. From the simulation results, the system is successfully performed.

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

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

[5] "Understanding PID control and loop tuning fundamentals", *Control Engineering*, 2020. [Online]. Available: https://www.controleng.com/articles/understanding-pid-control-and-loop-tuning-fundamentals/. [Accessed: 31- May- 2020].

[6] 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]

[7] 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].

[8] "45 RPM HD Premium Planetary Gear Motor w/Encoder", *Servocity.com*, 2020. [Online]. Available: https://www.servocity.com/45-rpm-hd-premium-planetary-gear-motor-w-encoder. [Accessed: 31- May- 2020].

[9] "Using Servo Motors with the Arduino | DroneBot Workshop", *DroneBot Workshop*, 2020. [Online]. Available: https://dronebotworkshop.com/servo-motors-with-arduino/. [Accessed: 27- May- 2020].

[10] "How Servo Motor Works & How To Control Servos using Arduino", *HowToMechatronics*, 2020. [Online]. Available: https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/. [Accessed: 27- May- 2020].