Speed Control Tutorial

Introduction: Speed or velocity is the quantity of motion that takes place with respect to time. The particular manufacturing application determines the type of speed that is required to perform the job. One example of a high-speed application is an automated insertion machine that makes traverse movement to place parts. In some applications, it is important to have good speed regulation. It is the ability to maintain a constant speed under varying load conditions. A machine tool spindle that requires constant speed under varying load conditions due to changes in the cutting action is an example of a speed-regulated application.

Speed Control of DC motor

The block diagram of speed control is illustrated in Figure 1. The controller calculates PWM command by comparing the speed reference and measurement from sensor such as tachometer or encoder. The best way

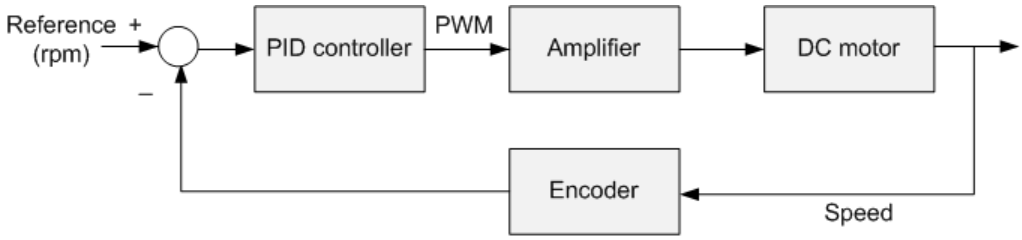


Figure 1: Block diagram of speed control of DC motor

to measure speed is to fit an optical encoder. The sensor consists of optical sensors and the disc attached to the rotational shaft. The output is swings to +5v when the light is blocked, and about 0.5 volts when light is allowed to pass through the slots in the disc. The frequency of the output waveform is given by,

$$f_{out} = N \times Rpm/60 \tag{1}$$

Whereas f_{out} is frequency of output waveform, Rpm is speed in revolutions per minutes and N is number of slots at disc. The speed of DC motor in rpm is given by, $Rpm = f_{out} \times 60/N$

For PID controller algorithm, the program subtracts the current speed from the setting speed and this results in the error value. The current controller output can be evaluated by feeding the error signal to the following discrete PID equation

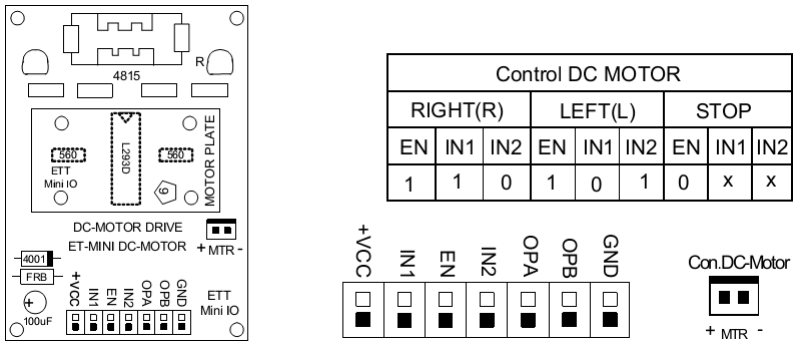
$$u_n = K_p e_n + K_i \sum_{i=0}^{n} e_i \Delta T + K_d \left(\frac{e_n - e_{n-1}}{\Delta T} \right) \tag{2}$$

where K_p, K_i and K_d are the proportional, integral, and derivative gains, respectively, e_i and $e_i - 1$ are the current and recent error values, and Δt is sampling time. The amplitude and sign of u_n imply the speed and the direction, respectively. To eliminate summation term, velocity algorithm is introduced which is the incremental form of the PID. The controller output u_n is obtained by computing time differences of controller output and adding the increments.

$$u_n = K_p(e_n - e_{n-1}) + K_i e_n \Delta T + \frac{K_d(e_n - 2e_{n-1} + e_{n-2})}{\Delta T} + u_{n-1} \tag{3}$$

Hardware Summery

ET-MINI DC-motor consists of DC motor driver circuit by using L293D, and two channels of opto input sensor. The module can control the speed and direction of DC motor and detect the revolution of rotation. Figure 2 shows the module and input port of ET-MINI DC-MOTOR.



Control DC MOTOR								
RIGHT(R)			LEFT(L)			STOP		
EN	IN1	IN2	EN	IN1	IN2	EN	IN1	IN2
1	1	0	1	0	1	0	x	x

Figure 2: ET-MINI DC-MOTOR module, reproduced from `http://www.ett.co.th`

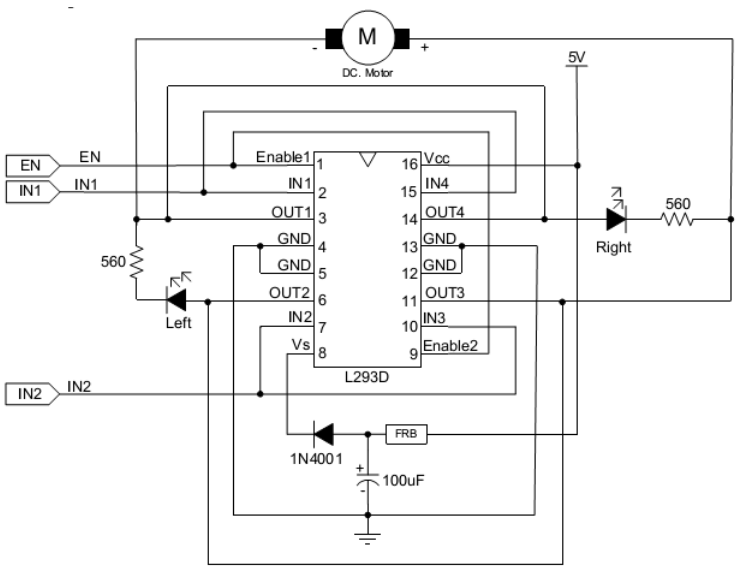


Figure 3: Schematic diagram of ET-MINI DC-MOTOR, reproduced from `http://www.ett.co.th`

Components

1. Arduino Uno R3
2. ET-MINI-DC-MOTOR board
3. Jumper

Arduino pin numbers

- Pin 2 as OpA
- Pin 3 as EN
- Pin 4 as OpB

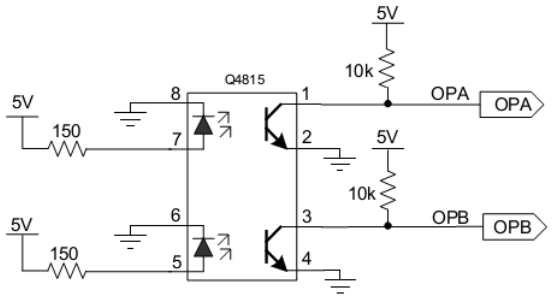


Figure 4: Schematic diagram of Opto Sensor, reproduced from `http://www.ett.co.th`

- Pin 7 as IN1
- Pin 8 as IN2

You can connect Arduino to ET-MINI DC-MOTOR board directly. The signal from Opto sensor is connected to Pin 2 and Pin 4. These pins can be configured as external interrupt to capture the rising edge of signal which we can find the period of signal and obtain the speed of motor. The Arduino board can control the speed of motor by providing PWM command to EN pin of Motor board.

Code : This program is used to control the speed of DC motor by using PID controller. The reference speed, PID controller gain are adjusted by RS232 communication. Press 'W' and 'S' to increase or decrease the reference speed.

Define variables

```
volatile unsigned long lastTime;
volatile unsigned long duration, CapTime, LastCapTime;
volatile double errSum, lastErr, lastErr_2;
volatile double lastOutput;
volatile double dT;
volatile int count = 0;

double PV, Output;
double kp, ki, kd;
double ref;
```

The setup function

```
pinMode(encoder0PinA, INPUT);
pinMode(encoder0PinB, INPUT);
digitalWrite(encoder0PinA, HIGH);       //turn on pullup resistor
digitalWrite(encoder0PinB, HIGH);       //turn on pullup resistor

pinMode(MotorIN_1, OUTPUT);
pinMode(MotorIN_2, OUTPUT);
pinMode(MotorPWM, OUTPUT);
attachInterrupt(0, doEncoder, RISING);

CapTime = 0;
LastCapTime = 0;

//Controller Parameter
kp = 0.01;   ki = 0.00;  kd = 0.00;
```

```
Output = 0;
Serial.begin (9600);
```

Interrupt subroutine function

```
void doEncoder(){
  LastCapTime = CapTime;
  CapTime = millis();
}
```

The loop function

```
unsigned long now;
double Freq;
now = millis();
dT = (double)(now - lastTime) / 1000;  //Our Sampling time
if(dT>=0.01){         //Do control loop every 10ms
duration = CapTime - LastCapTime;  //Get the period in ms
  LastCapTime = CapTime;
  if(duration==0){
   Freq = 0;
  }
  else{
   Freq = 1000/(double)duration; //in Hz unit
  }
  PV = Freq*60/2;     //rpm unit
  compute(ref, PV);  // Find controller output
  sendCommand(Output);  //Send command to DC motor
  lastTime = now;
}
//Receive Command
byte val;
if(Serial.available()){
val = Serial.read();
switch (val) {
case 'A': ref = 0; break;
case 'W': ref += 100; break;
case 'S': ref -= 100; break;
case 'O': kp+=0.01; Serial.println(kp); break;
case 'L': kp-=0.01; Serial.println(kp); break;
case 'I': ki+=0.01; Serial.println(ki); break;
case 'K': ki-=0.01; Serial.println(ki); break;
case 'U': kd+=0.001; Serial.println(kd); break;
    case 'J': kd-=0.001; Serial.println(kd); break;
  }
}
// display setpoint and measured value
if(count>50){
Serial.print(ref);
Serial.print(' ');
Serial.print(PV);
Serial.print(' ');
Serial.print(Output);
Serial.println(' ');
count = 0;
}else{count++;}
```

The "sendCommand" function

```
void sendCommand(int cmd){
 if(cmd>0)  {
    analogWrite(MotorPWM, cmd);
    digitalWrite(MotorIN_1,LOW);
    digitalWrite(MotorIN_2,HIGH);
  }
  else if(cmd<0){
    analogWrite(MotorPWM, -cmd);
    digitalWrite(MotorIN_1,HIGH);
    digitalWrite(MotorIN_2,LOW);
  }
  else{
    analogWrite(MotorPWM, 0);
    digitalWrite(MotorIN_1,LOW);
    digitalWrite(MotorIN_2,LOW);
  }
}
```

The PID controller algorithm (velocity PID algorithm)

```
void compute(double Setpoint, double Measured)
{
   double error = Setpoint - Measured;

   /*Compute PID Output*/
   Output = kp*(error - lastErr) + ki*error*dT + kd*(error - 2*lastErr + lastErr_2) / dT
    + lastOutput;

   /*Max 255, Min -255*/
   if(Output>255){Output = 255;}
   else if(Output <-255){Output = -255;}
   else{}

   /*Remember some variables for next time*/
   lastOutput = Output;
   lastErr_2 = lastErr;
   lastErr = error;
}
```

Result and Conclusion:
After upload program to Arduino board, the reference speed, speed measurement and PWM command are shown. Set the reference speed as 2000 rpm and let's adjust PID parameters: K_p, K_i and K_d, Observe the effect to speed of motor. When you see the reference speed and resultant are the same, try to change the condition of system such as changing the supply of motor from 5V to 3.3V.

References

- "Industrial Control Electronics: Devices, Systems & Application" by - Terry Bartelt.
- `http://playground.arduino.cc/Code/PIDLibrary`
- `http://www.ett.co.th`