UG108: Praxis II
Asian Institute of Technology
Handout: PID controller

January 2013
Undergraduate Program
Instructor: Chaiyaporn Silawatchananai, Matthew N. Dailey

PID controller Tutorial

Introduction: For many application, a more precise level of control is required than that of an open-loop system. By adding a sensor that measures the condition of the load, the system can adjust itself. The correction is made by sending an output signal from the measuring device to the controller. As shown in Figure 1, the connection between the measurement device and controller is called the feedback loop. The comparison between command signal and measured signal produces an error signal that applied to the amplifier. If either the command signal is changed or the measurement feedback signal is changed due to load condition, the control will produce a different error signal. The result will be that the actuator will automatically be controlled to make the proper adjustment to reach the desired condition.

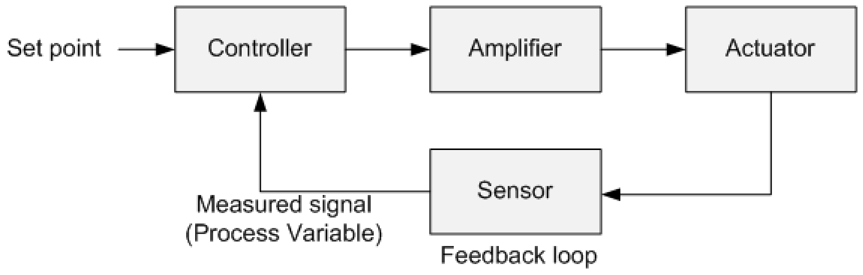


Figure 1: Block Diagram of a Closed-loop system, reproduced from Terry Bartelt, *Industrial Control Electronics: Devices, Systems & Application*

PID controller: stands for proportional-integral-derivative controller. It is one of the most popular controllers which widely used in industrial control system. The controller calculates an error value as the different between a measuring signal and the command signal or set point value. The controller consists of three terms: the *proportional*, *integral* and *derivative*.

- **Proportional term** produces an output value that is proportional to the current error value. The response can be adjusted multiplying the error with a constant K_p called the proportional gain.

$$P_{out} = K_p e_n \tag{1}$$

- **Integral term** is the sum of the instantaneous error over time and gives the accumulated value that should have been corrected previously. The accumulated error is then multiplied by the integral gain K_i and added to the controller output.

$$I_{out} = K_i \sum_{i=0}^{n} e_i \Delta T \tag{2}$$

- **Derivative term** slows the rate of change of the controller output. Derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability.

$$D_{out} = K_d \left(\frac{e_n - e_{n-1}}{\Delta T} \right) \tag{3}$$

The output of PID controller is the result of summing proportional(1), integral(2), and derivative(3) terms, which can be expressed as

$$u_n = K_p e_n + K_i \sum_{i=0}^{n} e_i \Delta T + K_d \left(\frac{e_n - e_{n-1}}{\Delta T} \right) \tag{4}$$

where e_n is the error between setpoint value and measurement value, u_n is the controller output. n is discrete time, ΔT is sampling time. For motor control, the controller output could be either input voltage or PWM duty cycle.

Trial and error tuning

There are several methods to find the parameter K_p, K_i and K_d. Trial and error tuning method can be used in case that the mathematical model is unknown. The method is very simple by adjusting parameters and monitors until the result is satisfied. There are three steps of tuning:

1. K_p is selected first in order to achieve the transient response desired while setting K_i and K_d to zero.
2. Then K_i is adjusted in order to satisfy any steady state error requirements. This will probably degrade the transient response due to the inclusion of a closed loop zero.
3. The transient response may be restored by selecting a suitable K_d. Several iterations of this process may be required before the system behaves as desired.

Position control of DC motors

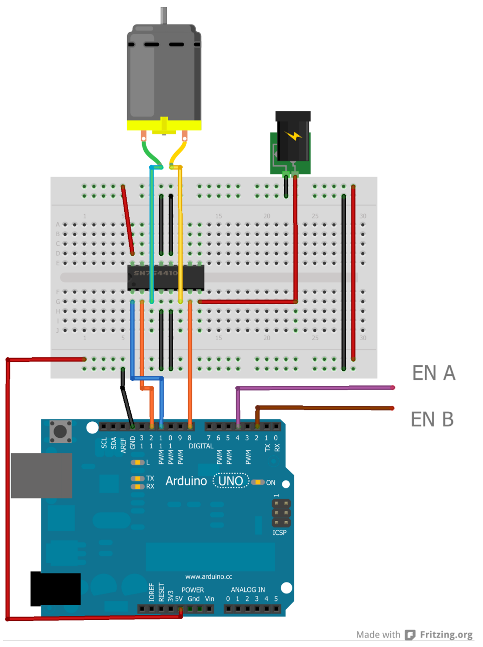
Write your program to control the position of the wheel by implementing PID controller in Arduino. As Figure 2, potentiometer is connected to Arduino for providing the reference position, which the maximum voltage value means to move for 180 degree and the minimum voltage value means to move for -180 degree. The position of wheel is measured by using optical encoder.

Components required

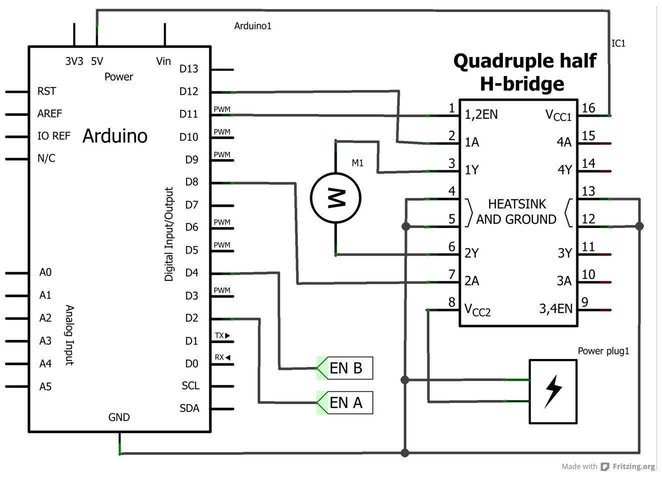
1. Arduino UNO R3
2. Proto-board
3. Jumper (Connecting wires)
4. DC motor
5. L293D
6. Optical Encoder
7. Potentiometer
8. 5V DC power supply

Arduino pin numbers

- Pin 2 for Encoder A
- Pin 4 for Encoder B
- Pin 8 for H-bridge input2A
- Pin 11 for H-bridge 1,2Enable input



(a) Diagram generated by Fritzing-0.7.11



(b) Schematics generated by Fritzing-0.7.11

Figure 2: Positon Control of DC motor with Arduino

- Pin 12 for H-bridge input1A

Code:

```
//Define Variable
int MotorPWM  = 11; // ENABLE to digital pin 11
int MotorIN_A = 8;
int MotorIN_B = 12;
int encoder0PinA = 2;
int encoder0PinB = 4;
int sensorPin = A0; // select the input pin for the potentiometer

/*working variables*/
double PV, Output, Setpoint;
double kp, ki, kd;
volatile unsigned long lastTime;
volatile double errSum, lastErr;
volatile int encoder0Pos = 0;
int sensorValue = 512;

void setup(){
pinMode(encoder0PinA, INPUT);
  digitalWrite(encoder0PinA, HIGH);       //turn on pullup resistor
  pinMode(encoder0PinB, INPUT);
  digitalWrite(encoder0PinB, HIGH);       //turn on pullup resistor

  attachInterrupt(0, doEncoder, CHANGE);  //encoder pin on interrupt 0 - pin2
  //Controller Parameter
  kp = 0.5;
  ki = 0.0;
  kd = 0.0;
  Serial.begin (9600);
  Serial.println("start");
}
void loop(){
sensorValue = analogRead(sensorPin);   //read the value
  Setpoint = (double)(sensorValue - 512)*0.35;  //Full scale setpoint +/- 180deg
  PV = (double)encoder0Pos*11.25;    //360 deg = 32 pulses
  compute(); // Find controller output

   //Send command to DC motor
  if(Output > 0)  {
    analogWrite(MotorPWM, Output);
    CC();
  }
  else if(Output<-0){
    analogWrite(MotorPWM, -Output);
    CCW();
  }
  else{
    STOP();
  }
  // display setpoint and measured value
  if(count>50)  {
    Serial.print(Setpoint);
    Serial.print(' ');
    Serial.print(PV);
    Serial.print(' ');
    Serial.println(Output);
    count= 0;
  }
  else{count+=1;}
}
```

Motor Control Function

```
void CC(){
  digitalWrite(MotorIN_A,HIGH);
  digitalWrite(MotorIN_B,LOW);
}
void CCW(){
  digitalWrite(MotorIN_A,LOW);
  digitalWrite(MotorIN_B,HIGH);
}
void STOP(){
  analogWrite(MotorPWM, 0);
  digitalWrite(MotorIN_A,LOW);
  digitalWrite(MotorIN_B,LOW);
}
```

Compute controller output by PID algorithm

```
void compute()
{
  /*How long since we last calculated*/
   unsigned long now = millis();
```

```
    double dT = (double)(now - lastTime);  //Our Sampling time

    /*Compute all the working error variables*/
    double error = Setpoint - PV;
    errSum += (error * dT);
    double dErr = (error - lastErr) / dT;

    /*Compute PID Output*/
    Output = kp * error + ki * errSum + kd * dErr;

    /*Max 100, Min -100*/
    if(Output>100){Output = 100;}
    else if(Output <-100){Output = -100;}
    else{}

    /*Remember some variables for next time*/
    lastErr = error;
    lastTime = now;
}
```

doEncoder function

```
  /* If pinA and pinB are both high or both low, it is spinning
   * forward. If they're different, it's going backward.*/
  if (digitalRead(encoder0PinA) == digitalRead(encoder0PinB)) {
    encoder0Pos++;
  }
  else {
    encoder0Pos--;
  }
}
```

Description:

In this tutorial we use interrupts to read a rotary encoder. It is a perfect job for interrupt because the interrupt service routine (doEncoder function) can be short and quick. When the Arduino sees a **change** on the A channel (connect to pin 2), it immediately skips to the "doEncoder" function which parses out both the low-to-high and the high-to-low edges, consequently counting twice as many transitions. Thus the resolution of encoder is increased twice. Since our encoder disc has the resolution 16 slot per revolution, the program can count for 32 pulses when the disc is rotated for one round.

In **loop** function, the reference position (Setpoint) is obtained from the knob position of potentiometer. Then, we get the position of the wheel by "doEncoder" function. After that, "compute" function finds the error by substracting setpoint value with measured value. The controller output is calculated by using expression 4. The result is used as the motor command which the sign +/- indicates the direction of rotation.

References

- `http://playground.arduino.cc/Code/PIDLibrary`
- `http://playground.arduino.cc/Main/RotaryEncoders`
- `http://multimechatronics.com/images/uploads/design/2012/Optical%20Encoder%20and%20the%20Arduino%202012.pdf`
- "Industrial Control Electronics: Devices, Systems & Application" by - Terry Bartelt.