PID control with the Arduino

PID principles

The Arduino PID code library includes a mathematical function PID()

PID(&Input, &Output, &Setpoint, Kp, Ki, Kd, Direction)

Input; the analogue value read into the Arduino
Output, the pulse width modulation output to a pin
Setpoint; the value that Output should work towards, the ideal value/

The 3 constants **Kp, Ki, Kd** are the Tuning Parameters. These affect how the formula attempts to calculate the output:

Kp: Determines how aggressively the PID reacts to the current amount of error (Proportional)
Ki: Determines how aggressively the PID reacts to error over time (Integral)
Kd: Determines how aggressively the PID reacts to the change in error (Derivative)

The **Direction** can be either DIRECT or REVERSE and determines which direction the output will move when faced with a given error. DIRECT is most common.

The & at the beginning of the values of Input, Output and Setpoint means that these values are passed to the function PID by reference and not by value (the default). As these values are passed by reference the function PID() directly changes their value within the rest of the program.

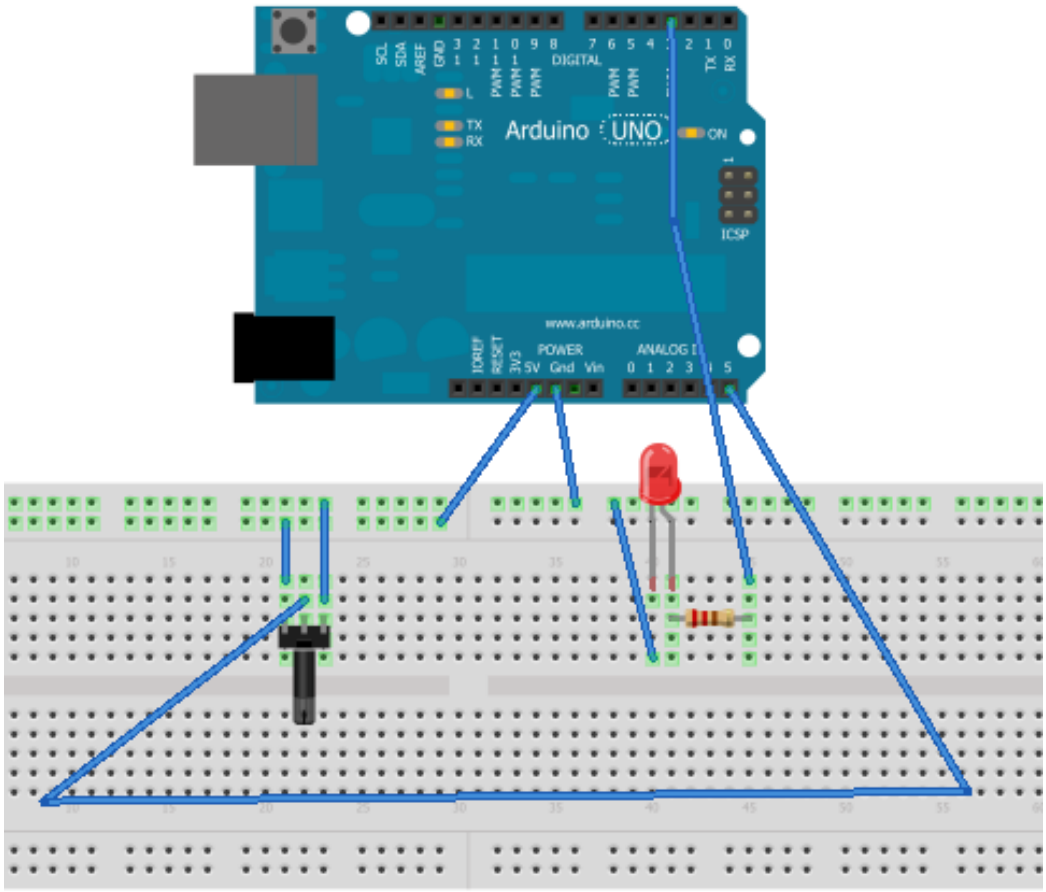
Code set up

PID control on the Arduino requires the PID library. This consists of a C++ and .h file together with some optional examples. The library is hosted at https://github.com/br3ttb/Arduino-PID-Library/zipball/master and needs to be extracted to the folder where Arduino keeps the sketchbook files. This can be found from File_Preferences in the Arduino program.

The PID library does not work as well with the Shrimp Arduinos as with the regular Arduino models. The programs compile and run with reasonable results but are not running the calculations as the Arduinos do. This could be due to the cheaper chip in the Shrimp model having difficulty with the maths involved in the PID processing.

Here a potentiometer is connected for input on A5 and a LED is connected for output on pin 3. The output pin needs to be a pulse width modulation pin so we can set it to different brightness levels rather than turning it on and off.

Wiring



Linear control

The code below does not use any PID control but does use the same circuit build as will be used to investigate PID.

The value of the potentiometer is read, converted to the range 0 to 255 and output to the LED.

The result of the program is that turning the potentiometer should brighten and dim the LED. Input and output values are written to the serial monitor to check the program.

```
const int analogInPin = A5;  // Analog input pin
const int analogOutPin = 3; // pwm output pin
int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM
void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}
```

```
void loop() {
  // read the analog in value:
  sensorValue = analogRead(analogInPin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  // change the analog out value:
  analogWrite(analogOutPin, outputValue);
  // print the results to the serial monitor:
  Serial.print("sensor = " );
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);
  // wait 2 milliseconds before the next loop
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(2);
}
```

PID control

In a PID system we do not want the input to directly control the output. The output should remain close to the set point regardless of any changes to the input. Instead of a light being affected by a dial a heater would maintain a constant room temperature regardless of the weather that day.

The following is a modification of the PID_Basic example:

```
#include <PID_v1.h>
//Define Variables we'll be connecting to
double Setpoint, Input, Output;
//Specify the links and initial tuning parameters
PID myPID(&Input, &Output, &Setpoint,2,5,1, DIRECT);
void setup()
{
  //initialize the variables we're linked to
  Input = analogRead(A5); //pot on A5
  Setpoint = 200;
  Serial.begin(9600);
  myPID.SetMode(AUTOMATIC);
  myPID.SetSampleTime(200);
}
void loop()
{
  Input = analogRead(A5);
  Serial.print("Input = " );
```

```
  Serial.println(Input);// what are we reading
  myPID.Compute();
  analogWrite(3,Output); //LED on 3 with Output brightness
  Serial.print("Output = " );
  Serial.println(Output);  //what has PID computed?
}
```

The serial monitor window shows the system maintaining an output of around 255 despite a fluctuating input. The set point value here is 200 not 255.