PH 315
Marshall Colville

Supplemental Experiment
Process Control with a Microcontroller
PWM output, PID control, and hardware implementation

I. Purpose:
Familiarize the students with the Arduino microcontroller (uC) and IDE. Using the Arduino microcontroller and associated IDE, construct a circuit and write the associated program to control a model system that maintains a steady state on a sensor.

Experimental section 1: Understanding the basics of PWM control to achieve a analog output from a digital device. Using the onboard ADC to monitor an analog sensor. Use the Arduino's PWM output to see how the system gets a proportional output from a PWM signal.

Experimental section 2: Building on section 1, adding a sensor to control the PWM output using a simple control loop. Adding PID control to the control loop in order to reduce overshoot, correct steady state errors, and maintain steady state at the desire value.

II. Theoretical Considerations:

Assumed prior knowledge:

1. This experiment assumes that the student is familiar with the basic principle of pulse width modulation (PWM). If not please review http://en.wikipedia.org/wiki/Pulse-width_modulation
2. This experiment assumes that the student understands basic electronic devices including wiring and implementation of LEDs, resistors, transistor switches, and photoresistors.

II.1 The Arduino

Arduino is a fully open source package of hardware and software that was designed for rapid prototyping by students and hobbyists. On it's most basic level the Arduino is an implementation of an Atmel microcontroller loaded with the appropriate firmware bootloader to interface with the Arduino interactive development environment (IDE). There are many Arduino and compatible boards available with varying levels of onboard supporting hardware and at a range of prices. For this experiment we will be working with the Arduino Uno. Please take a moment to familiarize yourself with the specifications found here:
http://arduino.cc/en/Main/ArduinoBoardUno
In this experiment the Arduino will be powered by the USB connection to the computer. For more power intensive applications an external DC power supply may be required.

The Arduino software is a program that allows the user to write in code that will determine the Arduino's behavior. This lab uses Arduino 1.0, which can be downloaded here:
http://arduino.cc/en/Guide/HomePage
Please take a moment to download and install the software and associated drivers for your operating system according to the instructions found in the link.

Finally, spend a few moments familiarizing yourself with the Arduino website. There is a large support community available through both playground and forums on the product site. Don't worry about not understanding what they're talking about right now, just browse around and see what the board is capable of and what people are doing with them. Finally, take at these two pages:
http://arduino.cc/en/Reference/HomePage
http://arduino.cc/en/Reference/Libraries
The first page will be your first reference for the language used to write programs for the Arduino. The second page is a repository of libraries, which are groups of functions that you may want to use in your programs.

II.2 The IDE

The Arduino IDE allows the user to write code in a language that can be logically followed by a human. The uC cannot understand this textual code, so the IDE also contains a compiler to convert the programing language into machine code that the uC can use. When the upload button in the IDE is clicked this happens automatically.

II.3 A Basic Sketch

The programming language is a combination of C and C## based on Wiring. It has an intuitive structure so that the user can easily understand what is desired. Within the Arduino community a program is called a *sketch*. Review the following code.

```
/*
 Fade

 This example shows how to fade an LED on pin 9 using the analogWrite() function. This
example code is in the public domain.

 */
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

void setup()  {   // declare pin 9 to be an output:
  pinMode(9, OUTPUT);
}

void loop()  {   // set the brightness of pin 9:
  analogWrite(9, brightness);  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;  // reverse the direction of the fading at the ends
of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

This is a basic program to fade an LED, and one of the example sketches. The first things to note are the *comments*. Comments are sections of text that are not a part of the code, but are added by the authors to let themselves and others know their intentions with various sections of code.

- There are two types of comments in the example, single line comments which are preceded by // and multiple line that are enclosed by /* */. Anything written after // on the same line or between /* */ will not be included as code. For single line comments the text may wrap to a lower line, however the comment is not ended until a return.

There are two required *functions* in any sketch, **setup** and **loop**.

- In this sketch the author starts be declaring the variables that will be called in the functions, brightness and fadeAmount. They are defined as integer values, and given initial values of 0 and 5, respectively. In the function these values will be changed.
- The function **setup** will be called once when the board is first powered up, and in this sketch simply defines the output pin (the one that powers the

LED) as pin 9. This tells the Arduino that the program will be writing values to the pin, and not expecting to receive values from that pin.

- Finally, the function **loop** will be called repeatedly until the board is powered down. The first command in this sketch is is analogWrite(9, brightness). This tells the Arduino to write a PWM output to pin 9 with a duty cycle equal to the value of the variable brightness *when the command executes*. The pin will continue to output this value until it is changed.
- It is important to note that the PWM generated with this command has a frequency of approximately 490 Hz, and the duty cycle can have a value of 0 to 255. Values outside this range will simply return the continuously off or continuously on state.
- To fade the pin in and out a simple arithmetic operation is called next, which adds the constant value of fadeAmount as defined to the current brightness. The next time this loop runs the brightness value will be +5 the previous value. After adding fadeAmount to brightness there is an if conditional.
- When the values inside parenthesis, in this case brightness == 0 || brightness == 255, are true, the commands inside the brackets will be called. The || symbol is the Boolean operator OR. Here when the brightness reaches a minimum or maximum allowed value the new value for fade amount is equal to the negative of the previous value. Finally the command delay is called for 30 ms.

A couple notes

i. It is important to note the difference between = and ==. The single = is the assignment operator, setting the value of the variable to the specified amount. The double == is a comparison operator which tests if two things are the same but doesn't assign a value.

ii. While a delay function is called the execution of the sketch is suspended for the specified amount of time, meaning that delays compound with the time taken to execute the sketch. If too many long delays are used the sketch may take too long to complete.

II.4 Libraries

One of the great benefits of using an open source platform like the Arduino is the number of other users sharing their work. The Arduino website has a long list of *libraries* available to be downloaded, used and modified:
http://arduino.cc/playground/Main/LibraryList
Go to the library page and locate the library titled PIDLibrary. Download v1.0.1 from GitHub here:
https://github.com/br3ttb/Arduino-PID-Library/zipball/master
Follow the directions on the Arduino website for installing the library.

A library is a collection of functions that can be called in your sketch. More information on the structure and creation of libraries can be found on the Arduino website. Refer to the PIDLibrary page for a list of these functions and how they are implemented.

II.5 Using Analog Inputs to Read Sensors

The Arduino Uno has 6 analog pins that are capable of reading analog inputs between 0 and 5 V. An onboard analog to digital converter (ADC) turns these voltages into a value between 0 and 1024. **The board cannot handle inputs greater than 5 V. Make sure that your circuit is designed such that the pin cannot be driven above 5 V or you will damage the board.**

Since many inexpensive sensors exist to convert environmental variables such as temperature, light, sound, humidity, pressure, etc, the ability to operate these pins effectively is crucial to students wishing to implement a uC in their experiments. One example is the photoresistor, which is a small device whose resistance decreases as illumination increases. By wiring in series with a discrete resistor and applying a voltage across the two elements a voltage divider is formed. The output of this circuit will be dependent on the illumination of the photoresistor. An example circuit is shown in fig. 1.

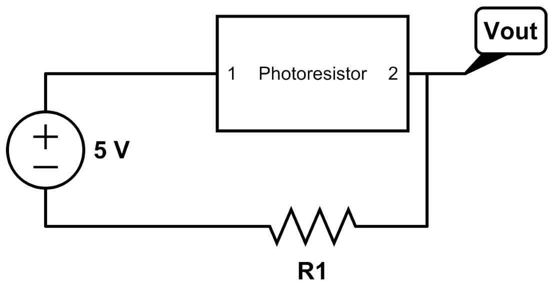

Fig 1 The simple photoresistor sensor circuit. The value of V_{out} will vary with the incident light on the photoresistor.

II.6 Implementing Control – Basic

Often in the sciences control over dynamic systems is crucial to obtaining good experimental results. One good example is temperature control. A given experiment may exhibit a strong dependence on the temperature at shich it was performed. The simplest way is to mimic the behavior of a bimetal thermostat. Namely, given a setpoint, toggle the output in response to direction of the input's displacement from the setpoint.

An improvement on this design is a trigger with hysteresis, as seen in the Schmitt trigger. By assigning a hysteresis to the setpoint and toggling the output based on the sensor state the frequency of switching the output is reduced, but oscillations about the setpoint are increased in magnitude.

II.7 Predictive Algorithms and the PID

Consider a volume of liquid that loses heat over time to the environment. A simple thermostat will produce a heat output until the sensor in the liquid reaches the setpoint. If the liquid is not a perfect conductor, however, there will be a temperature gradient between the sensor and the heat source. Once the liquid reaches an equilibrium it will be at a higher temperature than the threshold temperature. Similarly, heat lost through conduction and radiation will cause a reverse temperature gradient to form between the center and edges of the liquid body. Overall these gradients cause swings in the average temperature of the liquid. The addition of hysteresis, while increasing output stability, also increases the magnitude of these swings.

Ideally we want to create a device that has no overshoot, and can find a steady state output to maintain the setpoint. One such device is the proportional-integral-derivative controller (PID). As the name suggests this is a three-part process. The controller checks the difference between the setpoint and the sensor value to compute the error. The three components operate simultaneously to produce an output value that will correct the error with minimal overshoot and oscillatory behavior.

- The proportional function decides on an output magnitude based on the magnitude of the error. The farther the sensor value is from the setpoint, the greater the proportional response. This is multiplied by the proportional constant, K_p.
- The integral function performs an integration of the error over time. As small steady state errors of one direction are compounded in time, the integral function produces a larger output to bring the sensor value to the setpoint. This integral output is multiplied by the integral constant, K_i.
- The derivative function finds the change in the error with respect to time. This adjusts the output to match the rate of change of the system, so when the system responds quickly to a large output the output is decreased and when the system responds slowly the output is increased. This derivative output is multiplied by the derivative constant, K_d.

The values of the constants set the behavior of the system. Tuning a PID can be quite complex. For more information start here:
http://en.wikipedia.org/wiki/PID_controller#Loop_tuning
There are many good resources available for more advanced tuning techniques, depending on the degree of control required and the complexity of the process.

III. Experimental Considerations

III.1 Basic

Task 1:

- On a breadboard wire the + leg on an LED to Arduino pin 9, then run a 330 Ω resistor from the – leg to the Arduino ground.
- Open the Fade sketch from File>Examples>Basics>Fade in the Arduino IDE. Replace the equal to comparison operators(==) in the for() statement with <= and >=, then change the values of fadeAmount and the delay time to see how they effect the behavior of the LED.
- Connect the Arduino to the computer with a USB cable and click "Upload" from the Fade sketch. Qualitatively describe the behavior of the LED with respect to fadeAmount and delay().

Task 2: Do a web search for photoresistors and write a brief (one paragraph) summary of their characteristics including theory of operation, construction, and price. This does not need to be detailed, just enough to make it clear that you understand the devices.

Task 3: Rather than using temperature as in the case of the thermostat, build a light switch that turns a light on when the ambient light level is below a threshold value. To do this construct the circuit shown in fig. 2 on your breadboard. Type the following code into a new sketch and upload:

```
int photoresistorPin = 0;
int ledPin = 9;

void setup(){
  pinMode(ledPin, OUTPUT);
}

void loop(){
  int lightLevel = analogRead(photoresistorPin);
  if (lightLevel >= 600){
    digitalWrite(ledPin, LOW);
  }
  else{
    digitalWrite(ledPin, HIGH);
  }
}
```

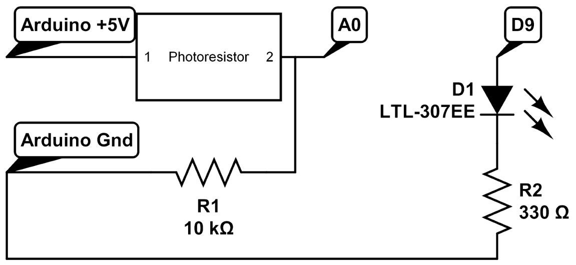

Fig 2 The light based light switch. Use the 5 V power and ground available on the Arduino to feed the voltage divider. The nodes A0 and D9 correspond to analog pin 0 and digital pin 9 on the Arduino Uno.

- If the light is on once the sketch is uploaded decrease the value on the right hand side of the comparison operator in increments of 50, uploading the new sketch after each change until the light is off.
- Now move your finger over the photoresistor. The LED should turn on when your finger is close enough. If not increase the sensor threshold value until you can switch the light on and off by covering the photoresistor with your finger.
- To see the instability around the threshold value hover your finger over the photoresistor at the distance where the light tends to switch. You should see fading behavior of the LED dependent on distance.
- Think back to the fade sketch. Why do you think the LED tends to fade in and out right near the switching point? In the current sketch the Arduino is only sending an on or off signal to the the LED, so why does it appear to receive a PWM signal?
- Now add a delay of 100 ms at the beginning of the loop function in your sketch and again find the threshold distance. Describe the behavior of the LED at the threshold.

Task 4: Add hysteresis to your thermostat. To do this the code must be expanded. While there are many ways to do this, one way is to replace the loop function in the previous sketch with the one below:

```
void loop(){
  int lightLevel = analogRead(photoresistorPin);
  if (lightLevel >= 625){
    digitalWrite(ledPin, LOW);
  }
  if(lightLevel <= 575){
    digitalWrite(ledPin, HIGH);
  }
}
```

You may have to adjust the switching values to suit the ambient light levels where you are doing your experiment.

- Repeat the procedure of covering the sensor and describe the behavior of the LED with respect to the distance of your hand. Has the instability (fading behavior) been removed?
- Change the switching values and see how they affect the LED's behavior.

Task 5: Do a quick web search on PID control. Drawing on your previous knowledge from PH 314 and PH 315, design an analog circuit that can achieve PID control. How would you go about tuning K_p K_i and K_d? (Hint: Think about the integrator, differentiator, and comparator circuits you have built using op amps).

Task 6: Upload the following sketch to your Arduino after installing the PID library:

```
#include <PID_v1.h>
```

```
double Setpoint, Input, Output;
PID myPID(&Input, &Output, &Setpoint, 20, 1, 1000, DIRECT);

void setup(){
  Input = analogRead(0);
  Setpoint = analogRead(1);
  myPID.SetMode(AUTOMATIC);
  myPID.SetSampleTime(1);
}

void loop(){
  int lightLevel = analogRead(0);
  int setPoint = analogRead(1);
  Input = map(lightLevel, 0, 900, 0, 255);
  Setpoint = map(setPoint, 0, 1024, 0, 255);
  myPID.Compute();
  analogWrite(9, Output);
}
```

- This will require the addition of a potentiometer to adjust the setpoint. Install a 10 kΩ pot as a voltage divider between the 0 and 5 V, then run the center tap to Arduino analog pin 1. Put the potentiometer in the middle of its range.
- With the photoresistor located approximately 1 cm from the LED the LED should be slightly dimmed.
- Shine a flashlight on the photoresistor and describe the LED's behavior when the flashlight is turned on and off. The LED should quickly brighten then dim again when the light is turned off. When the light is turned back on the Led should shut off then fade back in as the system approaches steady state again. If this is not the case try altering the distance between the flashlight and the photoresistor.
- Describe the behavior of the LED in terms of the theoretical considerations for the PID.

III.2 Going a Step Farther

Using what you have learned so far, can you adapt the PID circuit and sketch to control temperature?

- Start by replacing the photoresistor with a TMP36 temperature sensor. Lookup example code on the Arduino website, this is a very common sensor in the community. For wiring help consult the product datasheet, available online.
- Now you will need a way to effect the state of the temperature probe. A small 5-12 V DC fan will work perfectly. The positive lead can be connected directly to the Arduino pin 9, and the negative lead grounded. If the 5 V output of the Arduino is not sufficient to drive the fan a transistor switch can be used as shown in fig. 3. **If you use this circuit make absolutely sure that the 12 V power supply remains isolated from the Arduino pins.**

Running more than 5 V into any of the Arduino pins other than V_{in} will destroy the board.

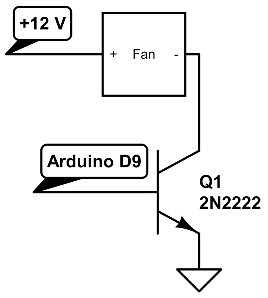

Fig 3 The transistor switch for driving a 12 V fan from the Arduino uC.

- Examine the PID sketch. Rename your variables so that you can keep track of your inputs and outputs. Notice that the inputs have been mapped from the analog input range to the PWM output range. This mapping needs to be retained in your sketch. Remember to use comments when you alter your code. Put a few words in there about what you did and why so that you do not forget. Finally since your fan will be cooling your sensor, you need to change the output direction. In the PID myPID () function replace DIRECT with REVERSE.
- Now, with your circuit built and your new sketch uploaded, place the breadboard on top of your computer tower. The heat generated will cause to the TMP36 to register a temperature above the room temperature. Point the fan at the sensor. Adjust the potentiometer until the fan reaches a steady state speed in the middle of its speed range. Now adjust the potentiometer up and down by small increments to see how the fan speed responds.
- Using what you have learned by adapting your PID from one process to another, describe another way in which a PID could be used to control a steady state system.

IV. List of equipment

The experiment was designed using the Arduino Uno, microcontroller, and the latest (03-2012) software version Arduino 1.0.

Here is a more complete equipment list:
Arduino Uno
Usb type A to type B cable
1x 5mm LED (red works, but yellow is better)
1x photoresistor (this one is the one used in this experiment http://www.sparkfun.com/products/9088 It should be the same one that comes in the getting started with Arduino kit.)
1x 330 Ohm resistor
1x 10 kOhm resistor

1x 10 kOhm potentiometer

For the final section, adapting PID to control temperature, you will need:

- 1x 12V PC case fan (this can come from an old PC or a 5V model can be purchased here: http://search.digikey.com/us/en/products/MB40100V2-000U-A99/259-1565-ND/2757776)
    - ➢ If you use a 12V fan you will need to supply 12V power, and build a transistor switch. This will requires a 1kOhm resistor between the Arduino digital pin and the transistor base to limit base current. The 2N2222 works well, or a 2N3904.
- 1x TMP36 temperature sensor (or an equivalent thermistor the TMP36: http://www.sparkfun.com/products/10988 Again, this should have come in the getting started with Arduino kit)
- 1x 0.1 uF capacitor

The circuits, with the exception of the 12V fan, are all designed to work using the Arduino's onboard power regulator and be powered completely by the USB connection to the PC. This means that the USB connection must be made over a powered USB receptacle on the computer. Any of the USB ports on the CPU case will work or a powered external USB hub. The circuits should be built on an unpowered breadboard to avoid signal noise.

Also, it should be possible to connect oscilloscope leads across the legs of the LED and monitor the output signal. This has not been tested and may produce some interesting results since the PWM output will be changing in time. If you choose to do this include pictures and a brief description of the signal in the final report.