**Laboratory 3 – Feedback Control System : PID**

**Introduction:**

 We introduce the idea of PID feedback control with a simple example.

The feedback control system we implement today consists of:

Fig 1

**PLANT:**  An LED shining light on to an LDR (light-dependent-resistor) placed facing it directly on the breadboard. Brightness of the LED is controlled using analog output voltage from the Arduino.

**CONTROLLER:**  A PID feedback algorithm you will implement in the Arduino

 **Setpoint:** Keep the LDR at a steady value in the presence of disturbances like blocking light path or shining a flash light on it. [detailed actuation circuit below]

**General background for Feedback control systems**

Fig 1 indicates a general feedback control system. GP is the physical system (‘plant’) to be controlled by a control signal *u* such that it’s measured output *y* is in a setpoint state *r.* Due to noise, *y* strays away from the desired output value. Therefore, *y* is fed back to the input and compared to the desired value *r.* The difference *(r-y)* is computed as the error signal *e*. A control function represented by GCis calculated based on *e* and a correction signal *u* is applied to the plant to bring *y* back to its desired value. This is done continuously: the signals *y, r, e, u*  are functions of time.

/15

/3

/1

/1

/4

A.1

A.2

P

PI

/6

PID

/5

Tuning

In this lab we will work on a particular form of control function GC called the ‘PID’ controller – a Proportional Integral Derivative controller, as shown in Fig 2



Fig 2: PID controller

Here the three terms P, I, D are calculated as:

The constants *KP KI  KD* are not known initially – they need to be tuned
for optimal response of the controller.

**Part A: Setup plant and control hardware\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Plant:** As mentioned in the introduction, our plant consists of an LDR (light-dependent-resistor) – a resistor whose value changes as a function of light falling upon it. A typical circuit for this setup is also shown in Fig 3: the LDR forms a voltage divider between 5V and GND. The goal of our controller is to keep *y(t)* steady at ***yRef = 2.5V***

The correction signal is applied by adjusting the light output of the LED

R

y(t)

GND

+5V

LDR

Fig 3: **PLANT:**

LED whose light intensity is adjusted by applying an analog voltage from Arduino

LDR whose resistance changes as a function of incident light

220Ω

Current limit

From Arduino

GND

LED and LDR must

face each other on b.b.

**Part A1**

Connect the LED portion of the plant in Fig 3, with the current limiting 220Ω resistor.

Write a small program that:

1. Sets the LED light level to an arbitrary value of brightness using analogWrite(..)

Recall work done in a prior lab session on how to obtain a smooth analog output level voltage from the Arduino by filtering it’s PWM output. By examining the light output of the LED visually (off to full brightness), determine an approximate working range of value in analogWrite(pin,value) that you will be able to work with. The LED starts turning on at ~ 1.6V and will be fully ON at full brightness ~ 5V – this varies slightly from one LED to another, and the current supplied.
You have to determine the approximate (min, max) range of value – this will be the working range of your control output for the rest of this experiment. Set to value be ½(max-min) for testing the next part A2

***Solution: Hardware demo – should be able to demonstrate analogWrite(..) functionality***

**Part A2**

Each LDR has its own steady state resistance value, so ‘R’ in the above diagram has been left unspecified. Connect the LDR part of the circuit picking a value of R such that *y(t)* is *approximately* ***yref*** V when the LED is shining at half the maximum level of brightness.

Write a small program that:

1. Reads the analog voltage value *y(t)* in volts using analogRead(..)
2. Use serial.println(..) to print the values obtained on the serial monitor

Note that by default, all arithmetic in an Arduino program is performed in integer form.

For example analogRead returns an integer between 0 and 1023 (corresponding to 0 - 5V)

To get a floating point number you must write a pair of instructions like:

Y\_raw = analogRead(1); // if Y is connected to analog pin 1

Y = Y\_raw \* 5.0/1023.0;//(note the decimal)-convert Y\_raw to floating
 //point number between 0.0 to 5.0

***Solution: Hardware demo – should be able to determine appropriate R needed to balance the LDR resistance and produce desired value of y=2.5V Depending on LDR, student may have to choose a slightly different yRef because R should not exceed ~ 1MΩ***

To simplify things, we will implement the PID controller one block at a time. Please write a *single* program: add the P, I and D blocks of code to it incrementally with neatly arranged functions. Demonstrate the operation of your program to a TA at each stage.

*Hint:* A useful function to keep track of time in the program is millis() Look up its definition and usage examples in the Arduino language reference.

*Hint:* While it may have been useful to print calculated values of *y(t)* earlier, the Serial.println() function consumes a lot of program execution time and can potentially introduce unpredictable errors when you are differentiating or integrating quantities as a function of time – so it’s probably a good idea not to use Serial.println() in the final feedback program below. You can observe the plant output *y(t)* directly on a DSO.

**Part B: Feedback implementation: P\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Write a program that controls *y(t)* to a set point ***yRef*** by using the control function
*u(t) = Kp e(t)*

Demonstrate its operation by connecting *y(t)* to the DSO. Set a long time base.

1. Introduce a momentary disturbance in the plant by (for example) putting a thick piece of paper in between the LED and LDR and removing it quickly. *y(t)* should return quickly to its reference setpoint.
2. Introduce a continuous asynchronous disturbance by placing a second LED next to the LDR. Drive the second LED from the logic pulse generator using 1 Hz pulse – is your controller able to compensate for this periodic disturbance?

**Part C: Integral feedback PI\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Add the integral **I** block to your program so that control function is now calculated as



Demonstrate the operation of your program and the (better) control of *Y(t)* observed on the DSO

**Part D: Feedback implementation PID\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Add the differential D block to your program so that control function is now calculated as



Demonstrate the operation of your program as in Part C with (hopefully perfect) control of *y(t)* observed on the DSO.

Use the space provided on page 4 to draw an algorithm or write pseudo-code that indicate how you will calculate the correction terms in the above parts using finite time differences between measurements.

***Solution to parts B,C,D on page 4***

***Solution: Combined pseudo-code program that implements P, I, D given below.***

 ***Students may write slightly different programs. Criteria for evaluation:***

1. ***Program must be well documented & all variables must be meaningfully named***
2. ***Demonstration of y(t) trace on DSO should exhibit PID control behavior:***

***Momentarily insert paper between LED and LDR (or shine light from cellphone on LDR) – this produces a delta function input to the plant.***

* 1. ***P control should exhibit oscillation around the setpoint***
	2. ***PI control should be better – y(t)overshoots the setpoint before returning close to the setpoint – there is a residual offset between y(t) and setpoint.***
	3. ***PID control should exhibit best behavior. Quick return of y(t) to the setpoint with little oscillation/overshoot and no offset from setpoint in steady state***

**Sample pseudo-code program: In this code Ki has units of [1/t] and Kd units of [t]**

----------------------------

previous\_error = 0

integral = 0

start:

 error = setpoint - measured\_value

 integral = integral + error\*dt

 derivative = (error - previous\_error)/dt

 output = Kp\*error + Ki\*integral + Kd\*derivative

 previous\_error = error

 wait(dt)

 goto start

---------------------------

 **Part E) Control parameter tuning**

Indicate the procedure you have used to determine control parameters *KP KI KD* in your program. As we discussed in the preparatory notes, Zeigler Nichols technique is useful to estimate these control parameters.
*(Getting the parameters from your neighbor, while useful, is not particularly educational)*

**PID controller coefficients may be tuned using Zeigler – Nichols method as described in accompanying handout. It requires introducing step function input to the plant in open loop mode and measure the delay in response of LDR voltage *y(t)* using DSO.**

**DSO Channel 1 → Arduino control signal to LED (goes from 0 to max)**

**DSO Channel 2 → y(t) output from LDR in open loop, rises with delay *τdead***

Approximate values of the controller co-efficients will be as per the following table:

Proportional gain *K0*is set by trial and error. All the other parameters are then:



Note that realistically,
very few students will get to Part E successfully!