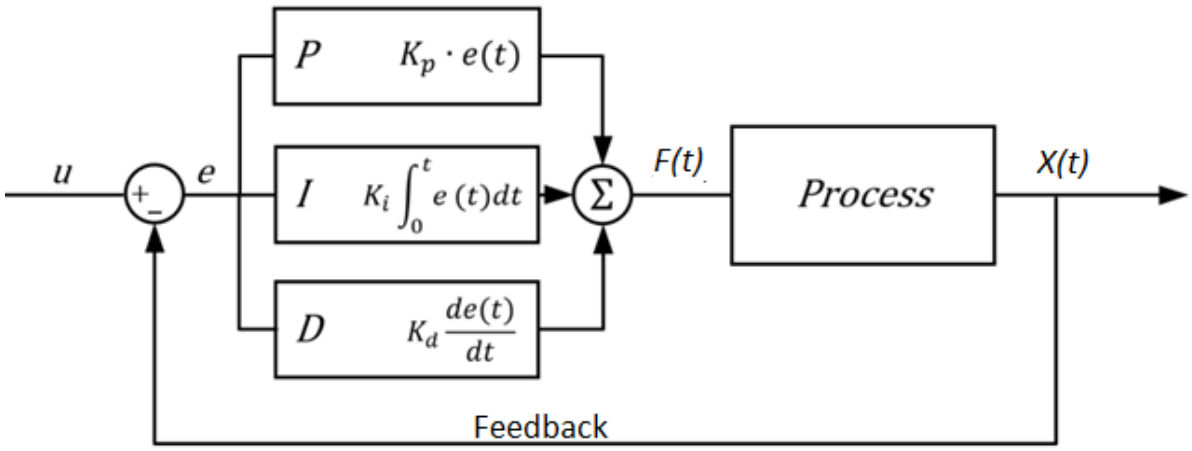
Mectron/Sfwr Eng 4AA4 - Lab 10
PID Controller

Part 1: PID Controller Introduction

In this lab you are required to write code to create a real time task to be run on MyRIO to control the position of the shaft on Quanser SRV02 Plant. The task will get a "set point" input from users for the desired motor shaft position, will get the "feedback" signals proportional to the shaft positions from the potentiometer installed on the motor assembly. Then the realtime task will calculate the "process errors" between the "set point" and the "feedback" signals. The realtime task will implement a PID controller, which, based on the process errors, will calculate values for the output signals. The output signals will drive the motor to bring its shaft to the desired position. See Figure 1 for the block diagram for PID control.



u: Set point, or target position degree
x(t): potentiometer reading, feedback degree
f(t): the volt value output to the DC motor

Figure 1: Block Diagram of PID Control

In time domain, a PID controller can be represented by the following differential equation:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

Using Euler's approximation method:

$$\frac{dx}{dt} \approx \frac{x(k) - x(k-1)}{T}$$

For proportional part:

$$u(k) = K_p e(k)$$

For integral part:

$$u(t) = K_i \int_0^t e(\eta) d\eta$$

Take derivative on both sides:

$$\frac{du(t)}{dt} = K_i e(t)$$

Using Euler's approximation:

$$\frac{u(k) - u(k-1)}{T} = K_i e(k)$$

That is:

$$u(k) = u(k-1) + K_i T e(k)$$

This can be further simplified by considering that:

$$u(k-1) = u(k-2) + K_i T e(k-1)$$

Substituting it into earlier equation:

$$u(k) = u(k-2) + K_i T (e(k-1) + e(k))$$

As the initial value $u(0) = 0$, then the contribution from the integral component can be written as:

$$u(k) = K_i T \sum_{i=1}^{i=k} e(k)$$

The derivative part approximates to:

$$u(k) = \frac{k_d}{T}[e(k) - e(k-1)]$$

So the PID controller approximates to:

$$u(k) = K_p e(t) + K_i T \sum_{i=1}^{i=k} e(k) + \frac{k_d}{T}[e(k) - e(k-1)]$$

Part 2: PID Controller Implementation

In order to implement the above differential equation numerically, you need to:

- Initialize various variable. Assume $u(0) = e(0) = 0$ and set initial values for K_p, K_i, K_d.
- For each cycle:
  - Read the potentiometer from one of the analog input channels on connector C connected to Quanser Terminal Board. Note that -180 to +180 degree movement of the potentiometer corresponds to -5.0 to +5.0 volts. Therefore one degree of rotation corresponds to 5.0/180 volts. In the lab it was found that volts/degree = 5.0/176 gives more suitable results. For a target position of 90 degrees, it will be necessary to output 90 x 5.0/176 volts.

  - Calculate current error say $errorC$.
  - Calculate the proportional, integral and derivative contributions as follows:
    * Proportional component = $K_p * errorC$.
    * Integral component = Previous integral component + $K_i * T * errorC$. Where T is the integration time interval, here is your realtime task time period for each reading and integration cycles.
    * Derivative component = $\frac{k_d}{T}(errorC - errorP)$. Where errorP is the errorC in the previous cycle.
    * Calculate the PID output by adding together the three parts above.
    * Write the output to one of the analog output channels on connector C connected to Quanser Terminal Board. Please note, the unit of your errorC may be in "degree" of position. Depending on the units (or the scale) of the K_p, K_i and K_d, your PID output may be in unit of "degree" too. While the MyRIO function you will use to write the output to the analog channel to make the motor move may require you to supply a parameter in "volt".

- You need to ensure that the **set point**, $\boldsymbol{K_d}$**,** $\boldsymbol{K_i}$ and $\boldsymbol{K_p}$ can be passed on to the module as command line parameters, so that different values can be tried until a satisfactory control is achieved.

- The rated voltage of the motors used in this lab is 10 volts. But large input voltages may damage motors' parts. **You must limit your output signal within** ± 6 **volts in your code**, and ensure that the signal being output at any time does not violate these limits. If the calculated value of output signal is higher than 6V, for example, then limit it to 6V, and if the calculated output signal is lower than -6V, limit it to -6V.

- Due to the system error or control accuracy, an angel tolerance (0.5 degree) needs to be used in your code. Once the process result falls within this tolerance range around the set point position, you may consider the control system satisfied users' requirement. For example, if the target position angel is 90 degree, then if the feedback fall within the range from 89.5 to 90.5 degree, you can assume the system has done its job.

- `Analog Value Registers` contain the values read from analog input channels or the values for outputting to analog output channels. The values are given in bits/volt. You must scale and offset the values in the `Analog Value Registers` before using them. But these are all taken care of by channel scaling functions provided by MyRIO C programming library. The scaling values are assigned to channels by first declaring a struct of type "`MyRio_Aio`" say C0 and then using the built in function "`Aio_Scaling(&C0)`" to assign the pre-defined scaling factors to the channel C0. For more details about scaling weights please refer to the document: MyRIO_Shipping_Personality_Reference6.0.pdf.

- In the RT task that implements the PID controller, you can make it to have a period of 1ms (0.001 sec).

- Compile your code, run it with different values of K_d, K_p, K_i to get the best control. Start with $K_i = 0, K_d = 0$, and K_p to be the value you got from model simulation in Part 2 of Lab 9. Gradually change the K_p value to achieve the best control result. Too high K_p value will make the system unstable. Too low K_p value will make the system take too long time to set to the target position. So use your judgement carefully.

- Once you get a good control with a K_p, adjust the value of K_i. Observe how the K_i will affect your control system. You may need to slightly adjust K_p while you are trying to find the best value for K_i.
- Once you have your satisfied K_p and K_i, you can work on K_d. Observe how the K_d will affect your control system.
- Once you are satisfied with your application, demonstrate your work to your TA.

Score

- Finish DAC code, make the DC motor rotate. 25 points
- Finish ADC code, get the potentiometer voltage. 25 points
- Finish PID function code, including degree-volt conversion, PID error calculation, etc. 25 points
- Get the best K_p, K_i and K_d. 25 point