Sierra Brown

04/07/2014

**Lab #2**

Background

Feedback control is a method used to compare a performed action with the original action goal, and compute the difference between the two in order to achieve more optimal results with the next adjustment. The negative feedback controller (servo) performs this process with the goal of calculating the difference, which can also be called the comparator. However, negative feedback control on its own is unable to create a high performance system due to long lags. Unlike in robotics, the nerves in our bodies transmit the signals relatively slowly, which results would result in an excess of error signals reaching to movement point when the comparator is actually at zero. Since we do have nerves, not wires or optical cables, feedback control and feedforward control are used in conjunction.

Central pattern generators automate a large portion of repetitive movements. Though they do not require sensory information, they perform much better with sensory feedback. These CPGS therefore operate on a combination of feedback control and learned feedforward control. Skill learning continues to be necessary since feedback control has a low upper bound on performance and can only perform corrections after an error has already been made. The cerebellum will perform functions of learned feedforward control and compensate for this.

While we have established that negative feedback control will not produce perfect movements due to lags, gains, and a level of non-elimnable error at the end of the process, we will explore how exactly it is that the feedback lag values affect performance.

Matlab Implementation

1. Main 3 components involved in a PID controller (high level meaning and simplest equation forms)

The PID controller works to calculate the difference between the set point (desired action) and the actual action. The next step is to calculate the manipulated variable which will use the comparator to designate the next action. There are three calculations which go into calculating the manipulated variable.

Proportional Gain

* Pout = Kpe(t)
* The signal emitted is the value of the proportional term, which will be proportional to the current error value. In order to change the magnitude, Kp can be manipulated.
* When Kp is large, Pout is large also, resulting in a high magnitude change for the error value and an unstable system.
* When Kp is small, the opposite is true, resulting in a low magnitude change for the error value and an unresponsive system.
* Kp should account for the majority of the output, as is logical since it is largely dependent on the error value in a proportional sense.

Integral Term

* The integral term is proportional to magnitude and duration of the error value
* The integral corresponds to the sum of instantaneous error over time and calculates accumulated error
* Ki is the integral gain and is multiplied by the integral

Derivative Term

* The slope of error over time and multiplying this rate of change by the derivative gain Kd
* This equation is able to predict system behavior and thereby help it reach the correct conclusion faster and improve stability

In order to achieve optimal behavior, gains should not be in excess in order to prevent extreme oscillations, but also must be responsive enough to take appropriate action for error signals and be accurate. The greater time lag which is present, the more unstable the system will be, since change will still be occurring while the new signal is being sent to the system, and may be inappropriate for the new position, thereby overcompensating.

Simulation in Matlab Overview

Proportional gain (Kp), Integral gain (Ki), and Derivative gain (Kd)

* These must be initialized in order to determine the gain of the system (how responsive it will be to the error signal. Having these in correct balance will determine the efficiency of the system, and if in fact it works at all. For example, if proportional gain does not have the biggest affect the system will fall out of balance.

Tau ()

* Tau will set the lag time for the system. For humans, this could try to simulate real lag time, which would be much slower, or for robotics the lag time of a wire.

History and PV

* The history is set to determine the value of actions, which is key to calculating the error value
* The function ones(tau,1) creates an array of tau 1’s (in the example case an array of size 25, of all ones).
* By mutliplying the history (action value) by ones(tau,1) the length of the history since the last action, an array the length of the lag (25) is created filled with the history value (2) Ex. [2,2,2…(25x)]
* This will be especially important for calculating the integral component of the MV since it takes into account past actions over time.

Set Point

* SP (set point) is set to initialize the ultimate goal of the system, which is key to calculating the error value

TimeEnd

* Since a feedback control system may reach a very tiny error but is not going to reach the perfect setpoint value in many cases, it is important to set a timeEnd value or the operation could continue infinitely.

Initialize the error

* Clear e; 3(1) = 0
* The error value should start off at zero before any action has been taken

Now that all of the necessary values are initialized the system can actually be run

For R = length(PV) + 1 – PV+1 : time End

* The simulation will run from the value of the time the starting action ended (in this case 25) until the end time specified (in this case 400), so it will run for 376 steps.

e(R+1 – tau) = SP – PV(R-tau)

* This will create an array which will represent error values which can be used in output calcuations

P = Kp \* e(end)

* This output signal will be proportional to the set gain multiplied by the last error signal received

I = Ki \* sum(e)

* This output signal will be proportional to the set gain for the integral times the sum of previous error values, therefore incorporating history into the equation

D = Kd \* (e(end) – e(end-1))

* This will calculate the change between the last two error values, and modulate the signal based on how fast change is occurring

MV = P + I + D

* This integrates the three signals to calculate the new ultimate signal

PV(R) = PV(R-1) + MV

* This changes the present location by adding the new motion signal in

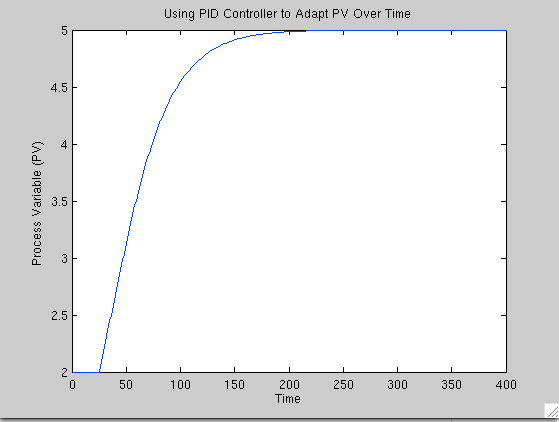
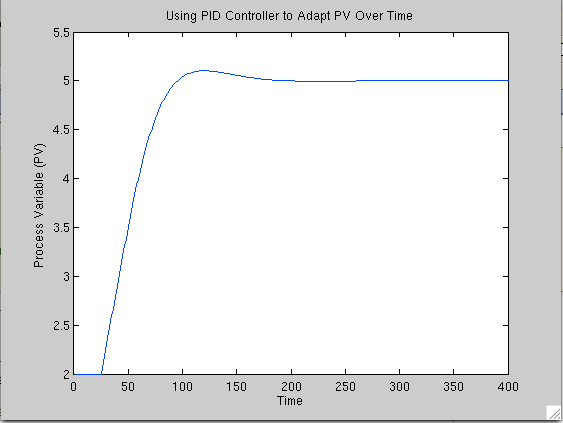
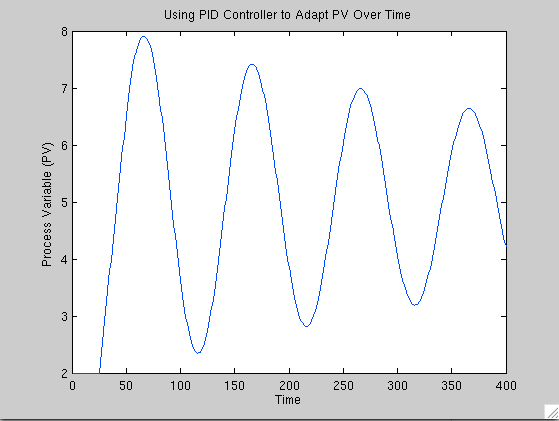
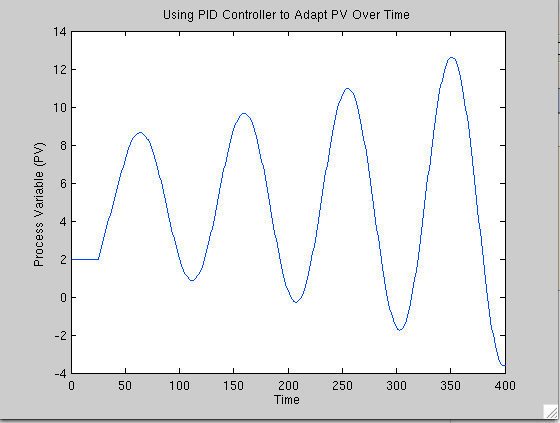
This process will repeat until for loop ends at the time end point

The rest of the matlab file simply plots the path of the process value against time with the title Using PID Contoller to Adapt PV Over Time

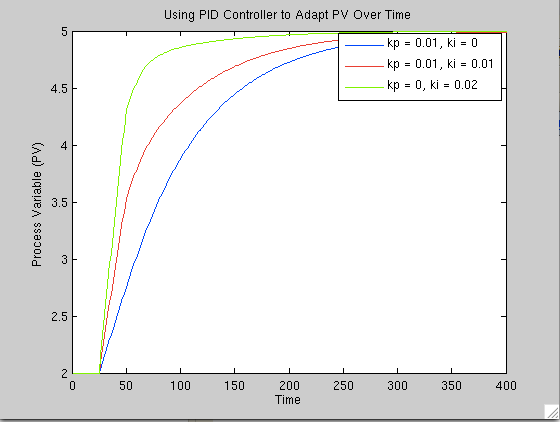
Results

Run the matlab code from class (or your own, improved code) to produce well-labeled plots of the following scenarios

Include these plots in your report

1. Using the proportional (P) component only, search through values of the feedback lag/delay (tau) and the gain (Kp) until you find values that lead to examples of the following types of system performance:
2. Overdamped system
   1. The processing variable exponentially transitions from the initial value to the set point vaule without any oscillations (no overshoot at all)
   2. Kp = .015
   3. 
3. Slightly underdamped system
   1. A slight, single overshoot of the PV passes the SP, after which the PV decays back towards the SP
   2. Kp = .02
   3. 
4. Underdamped system
   1. The PV overshoots and oscillates around the SP, but eventually stabilizes and gets nearer to the SP on each successive oscillation
   2. Kp = .06
   3. 
5. Unstable system
   1. The PV overshoots and oscillates around the SP, but each successive oscillation is greater in magnitude than the previous one
   2. Kp = .07
   3. 
6. Add in the **integral (I)** component
   1. Find a value set for Kp and Ki whereby adding in Ki makes the PV approach the SV faster

In an overdamped system, the inclusion of Ki makes PV approach the SV faster. I edited the code so that it would only include recent error values. The edited code and plot are below.



% Initialize time

time = 1

% Run simulation

for R = length(PV)+1:timeEnd

if time >= 50

time = time -1

end

e(R+1-tau) = SP - PV(R-tau);

P = Kp \* e(end);

I = Ki \* sum(e(end-time):e(end))

D = Kd \* (e(end) - e(end-1));

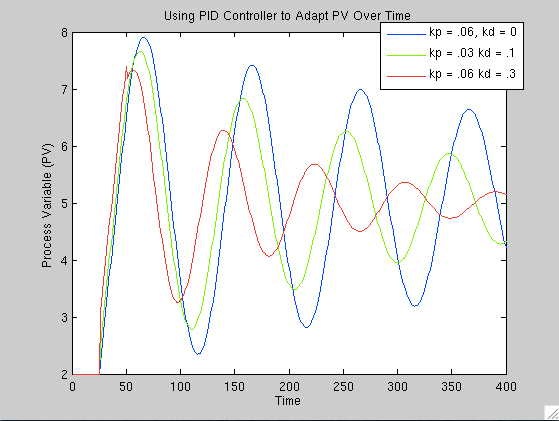
MV = P + I + D;

PV(R) = PV(R-1) + MV;

time = time + 1

end

1. Add in the **derivative (D) component**
   1. Find a value set for Kp and Kd whereby adding in Kd damps the oscillations produced from the proportional component alone



**Discussion**

The results show how feedback lag and gain are the driving forces behind regulatory feedback.

The proportional gain is directly related to the error signal and for the derivative and integral components to be able to enhance it, it is important that it is set to a reasonable value. A proportional gain that is too high will lead to intense oscillations that are difficult to dampen, while if it is too low it will be very unresponsive and therefore also difficult to control.

The integral gain is directly related to the history of the errors, and showed very clearly how difficult it is for the system to perform well if a long history of errors is included. By modifying the code and shortening the history, the system became more responsive. A too high integral gain will cause extreme overcorrection, while if too low it will not do a good job of speeding up process of reaching the set value.

The derivative gain calculates how fast the error value is changing. Since this is a way of predicting future behavior, it can stabilize the PV value, cause smaller oscillations, and ultimately result in less overall change making the PV reach the SV more quickly. If it is too big, however, it will cause unstable oscillations.

These circuit components drive feedback control in mammals. It shows that the faster response times are available, the better correction will function. It also shows how over time fine-tuning can occur and as the circuit balances out it can develop a system that performs correction in the most efficient way possible. Furthermore if any one component is not functioning correctly, it could impede the ability of correction. If perception of the error value were manipulated via visual manipulation, that would also be able to throw the system off balance. These equations also have interesting applications in the field of robotics, since in that field there is much lower lag time, they could work to enhance correction ability even better.