West Virginia University
College of Engineering and Mineral Resources

Lane Department of Computer Science and Electrical Engineering

Computer Engineering 313
Spring 2010

Laboratory #6 (Week 8)

Goals

This lab introduces the PID controller and discusses how to apply the PID controller in regards to the micromouse application.

Background

PID Controller

Proportional, Integral, Derivative Controller (PID Controller) is a control loop feedback mechanism used in many control systems. PID controllers are one of the most widely used controllers today since it is fairly easy to tune the PID parameters without having much knowledge of control theory. A PID controller calculates the "error" value as the difference between a measured process variable and a desired set point [2]. To minimize the error, the controller adjusts the process control inputs and the controller is tuned according to the nature of the system.

A PID controller consists of three separate parameters: proportional, integral, and derivative. The proportional parameter (K_p) determines the controller's reaction to the current error, the integral parameter (K_i) determines the reaction based on the sum of the recent errors, and the derivative parameter (K_d) determines the reaction based on the rate at which the error has been changing [2]. K_p, K_d, and K_i, are the gains for each term. The weighted sum of these three parameters is used to adjust the process such as the position of a micromouse in a maze. *Figure 1* provides a block diagram of a PID controller.

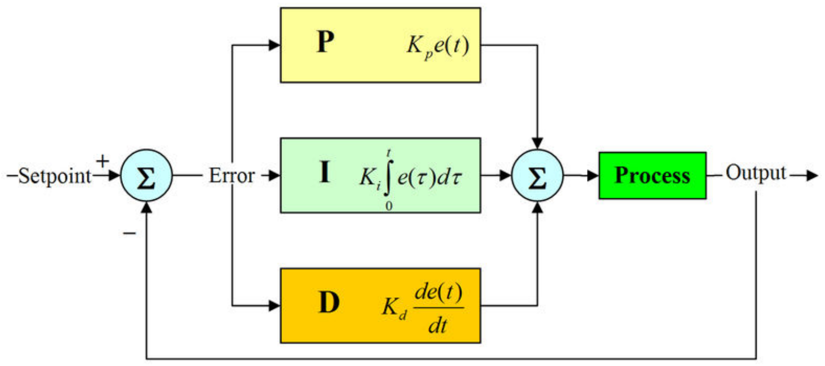


Figure 1: PID Controller Block Diagram [2]

Proportional

The proportional parameter makes a change to the output that is proportional to the current error value. Essentially, you are multiplying a scalar (K_p) values times the error to adjust. So, a high proportional gain results in a large change to the output, and a small gain results in a small output response.

$$P_{\text{out}} = K_p\, e(t)$$

where

P_{out}: Proportional term of output

K_p: Proportional gain, a tuning parameter

e: Error $= SP - PV$

t: Time or instantaneous time (the present)

Figure 2: Proportional Term [2]

Integral

The integral parameter takes into consideration both the magnitude of the error and the duration of the error (by summing the error over time). The integral term with the proportional term increases the movement to the desired state (setpoint) and eliminates the steady-state error that is caused with only a proportional only controller [2]. However, since the integral term takes into consideration past errors, it does invoke the possibility to make the current adjustment overshoot the desired state.

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

where

I_{out}: Integral term of output

K_i: Integral gain, a tuning parameter

e: Error $= SP - PV$

t: Time or instantaneous time (the present)

τ: a dummy integration variable

Figure 3: Integral Term [2]

Derivative

The derivative parameter involves using the rate of change of the error to reach the destination state. One can take the slope of the error and multiply it by K_d. This term decreases the rate of change of the controller output to reduce overshooting the setpoint.

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

where

D_{out}: Derivative term of output

K_d: Derivative gain, a tuning parameter

e: Error $= SP - PV$

t: Time or instantaneous time (the present)

Figure 4: Derivative Term [2]

The PID controller output then takes each term and sums them together as shown in *Figure 5*.

$$\mathrm{u(t)} = \mathrm{MV(t)} = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{d}{dt} e(t)$$

Figure 5: PID Controller Equation

So, the PID controller can be modified for use in different systems by changing the gains of the different terms described above. *Table 1* captures the effects of increasing each of these parameters.

Table 1: PID Controller – Effects of Increasing Parameters

Effects of *increasing* parameters				
Parameter	**Rise time**	**Overshoot**	**Settling time**	**Error at equilibrium**
K_p	Decrease	Increase	Small change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Indefinite (small decrease or increase)[1]	Decrease	Increase	None

It is also important to note that there are several variations of PID controllers. Often times, systems only need to implement a few of the PID parameters to suitably correct the error. So, a PID controller may be called a PI, PD, P, or I controller when one of the respective control actions is not used in the current system. There are also several ways to tune the parameters of a PID controller such as mathematical models, software tools, and manually. Given that each group only has a limited amount of time to finish this lab and the project, you will implement the PID algorithm and tune it by trial and error.

Micromouse

Due to the quality of the stepper motors, the robot does not have much trouble going in a straight line or even making a near perfect 90 degree or 180 degree turn. The stepper motors will rotate exactly the number of steps you need them to rotate. However, for our application, we saw the need to have some sort of feedback system in place to help find the center of the maze and help the robot go in a straight line (detect error's in its path) while it's moving through the maze. Using the distance measuring sensors as a feedback, the robot can keep track of its position and maintain a straight orientation in the maze to avoid from hitting the walls. To maintain the robot's orientation straight with reference to the wall is accomplished using a controller such as a PID controller. [1]

Examining our micromouse robot, we have two proximity sensors mounted on the left and right front of the robot. These sensors can be used to determine whether or not the robot is too close to the right wall or too close to the left wall. This calculation is the error for our controller and can generate the necessary signal to correct the error [1]. However, looking to the proximity sensor datasheet shows that it takes almost 55 milliseconds to complete the distance measuring operation. This time slows down our ability to fix the error in our control system and increases our chances to run into the wall especially if the micromouse is moving fast. As a result, we tailor the parameters of the PID controller to keep from hitting the walls while keeping in mind the time to perform the distance measuring operation.

A couple other things that you will want to consider in regards to the control problem are the stepper motor dynamics and single sensor readings. You will need to be careful when adjusting your stepper motor speed as stepper motors operate best within a certain range. If you go outside of this range, the stepper motor may become jerky or not respond correctly. In regards to the sensors, since there is only one sensor on each side of the robot, we are prone to sensor reading errors.

Let's take a look at an example scenario.

Suppose the robot begins to move towards the left wall instead of going straight through the maze. Based on the sensor readings, an error signal should be generated. Using this error, the controller should generate a control signal which increases the speed of the left wheel and decreases the speed of the right wheel which will make the robot turn slightly in the desire direction to straighten its path. The amount of the increase and decrease of each wheel is controlled by the magnitude of the control signal [1]. *Figure 6* provides an illustration of how to correct the robot's movement.

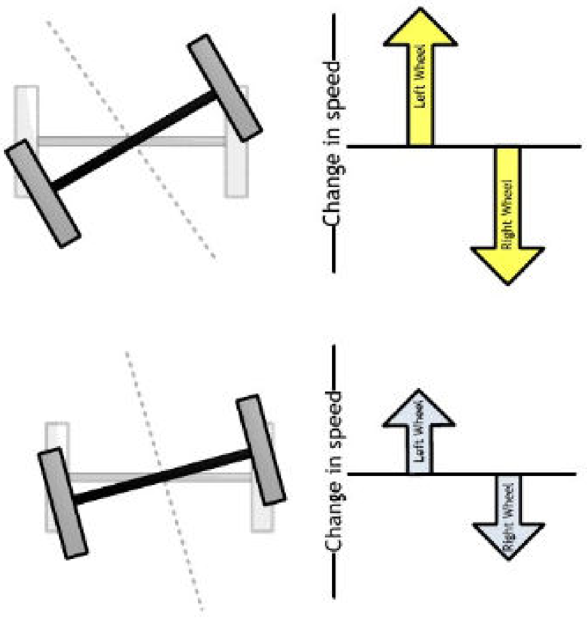


Figure 6: Adjusting Speed when in Error to Straighten Micromouse [1]

Now, pulling all of this together, we have the system as shown in *Figure 7.*

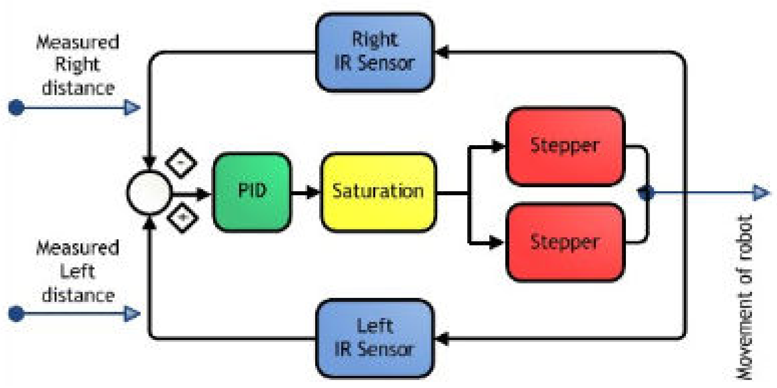


Figure 7: Micromouse Control Loop Diagram [1]

Procedure

Your task is to develop a control loop inside of a timer interrupt subroutine that is executed every 55 milliseconds. The 55 millisecond control loop is limited by the proximity sensors as discussed above. The subroutine should read the sensor data from all three sensors and can optionally convert the data into centimeters. The subroutine should use the distances to make the robot go straight with respect to the walls. For the purposes of this experiment, you will just have to keep the robot between two walls (a straight line traversing 13 cells) – you will not have to worry about turning and your program should be organized similar the flowchart in *Figure 8*.

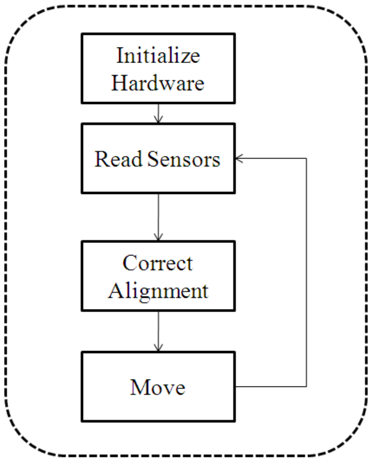


Figure 8: Program Flowchart

- **Correct Alignment Action**
  - Initially, both motors should be driven at the same speed
  - Once the robot is not centered, an error should be sent to the controller
  - Based on the magnitude of the error, the controller should generate a control signal to make the mouse turn slightly left or slightly left. You can make these adjustments by increasing the speed of one wheel which decreasing the speed of the other.

First, implement a Proportional controller (P controller) using your left sensor and right sensor readings. We can calculate the proportional error by performing the following calculation shown in *Figure 9*.

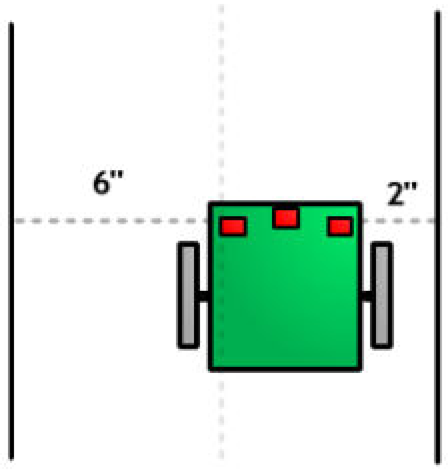
$$\boxed{Proportional\_error = \frac{Left\ distance - Right\ distance}{2}}$$

Figure 9: Proportional Error Calculation [1]

Using this equation, you notice that if the error is negative, the robot is moving towards the left, and if the error is positive, the robot is moving to the right.

Let's take a look at an example.

Looking at *Figure 10*, we note that the left sensor distance is 6" and the right sensor distance is 2", and thus we consider this an error. We can calculate the error by using our equation above.



$$Proportional\_error = \frac{6 - 2}{2} = 2''$$

Figure 10: Determining Error using Sensor Values [1]

For the proportional term, you can take this error and multiply it by the proportional gain (K_p). Demonstrate your results to your TA.

Second, now try to improve your controller by adding the derivative term to the control algorithm to create a PD controller. The derivative term takes into consideration the rate of change of the error and multiplies it times a by the derivative gain thus providing an idea of how fast the error is changing with respect to time. *Figure 11* provides the equation to calculate the derivative error. When finished, demonstrate your results to your TA.

$$\mathit{Derivative\ Error} = \frac{(\mathit{Proportional\_error} - \mathit{Previous\_Proportional\_error})}{\mathit{Sampling\ time}}$$

$$\mathit{Previous\_Proportional\_error} = \mathit{Proportional\_error}$$

Figure 11: Derivative Term Error

Finally if necessary implement the integral term to your control algorithm to create a PID controller. Remember the Integral term can help eliminate steady state error but can slow down the system response.
Demonstrate your results to your TA.

Lab Notebook

1. Include a brief description of what was accomplished in this lab. Be sure to include all functions with comments. Be sure to clearly indicate your PID code and highlight your values for the gains K_P, K_I, and K_D. (40 points)
2. What did you notice when you were using just a proportional controller? What are some advantages and disadvantages of a proportional controller? (5 points)
3. What did you notice when you added the derivative controller? What are some advantages and disadvantages of the proportional-derivative controller? (5 points)
4. Revisit your code from Lab #1 on initializing the wall map and distance values. Then, research the modified flood fill algorithm and describe how the algorithm works in your own words (5 points).
5. List any problems that you encountered in the lab and/or suggestions for improvement of the lab. If no problems were encountered or you have no suggestions, please state "NONE". (5 points)
6. For extra credit, design your controller in Matlab/Simulink. Turn in your completed code and a brief write-up on your design including all references used.

References

1. http://www.scribd.com/doc/16400768/Ishu-Pradhan-Implementing-PD-controller-in-a-Robot-Micro-mouse. A good portion of the text in this lab experiment handout was derived from this reference.
2. http://en.wikipedia.org/wiki/PID_controller