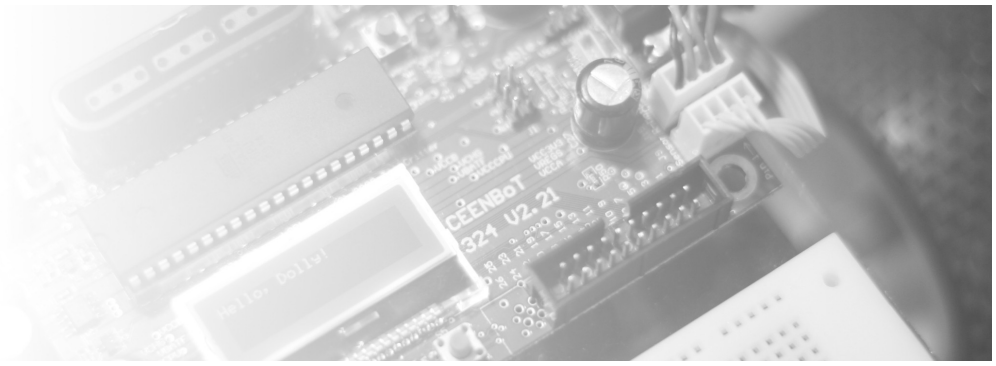
Mobile Robotics I: Lab 6

Wall Following with Feedback Control

CEENBoT™ Mobile Robotics Platform Laboratory Series
CEENBoT v2.21 – '324 Platform



Alisa N Gilmore, P.E., *Instructor, Course & Lab Developer*

The Peter Kiewit Institute of Information Science & Technology
Department of Computer & Electronics Engineering
University of Nebraska-Lincoln (Omaha Campus)

Rev 1.01

Purpose

The purpose of this lab is to implement a wall-following routine for the CEENBoT using an ultrasonic sensor positioned by a DC servo motor, and to incorporate Proportional and Derivative feedback control to improve wall-following performance. This will be incorporated as a Wall_Follow behavior in your existing BBC-structured program.

Lab Objectives

By following the directions in this lab, you are expected to achieve the following:

- Implement a practical feedback controlled "servo" behavior for wall-following in a behavior-based control structured program.
- Implement a Proportional and Derivative controller and tune the gains of both to achieve the best performance.
- Evaluate the performance of the implemented feedback controller in terms of overshoot, rise-time and settling time.
- Learn about RC Servo control using functions provided as part of the CEENBoT-API.

Requirements

Preliminary Readings

- Read about Feedback Control in course texts: Mataric, Chapter 10 and Jones, Chapter 2.
- Read about a practical implementation of PID controllers in: "*A PID Controller for LEGO Mindstorm Robots*" located here: http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html
- Read about the *RC Servo* control functions provided by the **`TINY`** subsystem module, covered in the *CEENBoT-API: Programmer's Reference*. (Chapter 11, on *Rev. 1.05*).

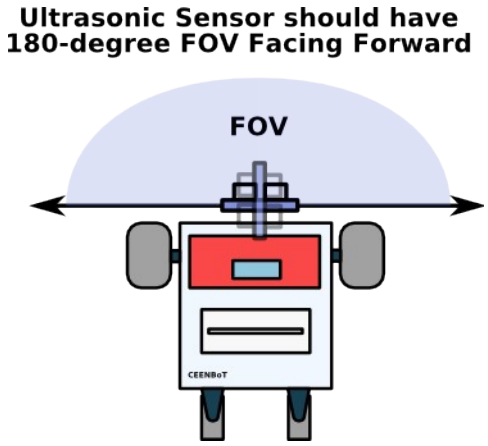
Required Equipment

- CEENBoT, platform '324 v2.21.
- Means to program your CEENBoT. (i.e., USB or Serial ISP programmer).

Background

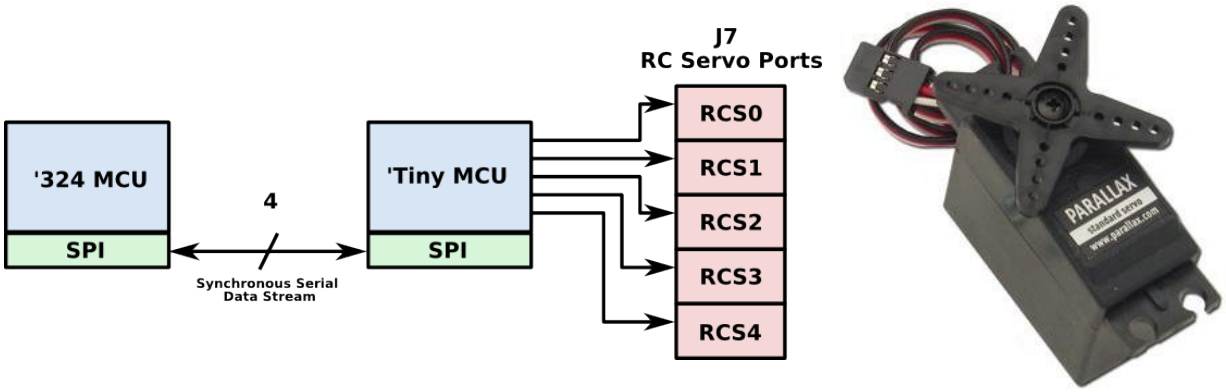
The RC Servos

The Parallax PING Ultrasonic Distance Sensor (#28015) will be used in this lab mounted on a Parallax Standard DC Servo Motor (#900-00005) on the front end of the robot. This standard servo motor is designed to hold any angular position from 90 degrees to the left of its center position to 90 degrees to the right of center. It contains a feedback system that will maintain a desired angular position, specified by the width of the pulse you send it every 20ms (see Servo documentation posted on BB). You will need to control the servo so that it faces 90 degrees to the left or to the right of the robot for left or right wall-following (you choose). First, observe the 180 degree range of your servo by slowly, manually turning the servo to locate its limits. If your ultrasonic sensor is not oriented to span an entire 180-degree field-of-view facing forward (e.g., it goes all the way to the left, but not to the right), then you will need to unscrew the center screw at the base of the RC servo and slowly and very carefully, detach this base and reposition it so that you can achieve a 180-degree field-of-view facing forward (see the figure below).



Note that the center position of the 180 degree range is achieved by sending a pulse of 1.5ms every 20ms to the servo. To achieve an angular position of up to 90 degrees to the left or 90 degrees to the right of this center position, you will increase or decrease the pulse width until you achieve your desired sensor position.

The figure below (next page) illustrates the control structure used with the RC servos on the CEENBoT controller board. The RC servo is pictured on the right. As can be seen, the CEENBoT includes up to 5 RC servo ports. These connectors are located next to the ON/OFF switch on the controller board and are labeled `J7`.



The control structure is similar to that for the IR sensors – they're under the control of the ATtiny. Therefore, to control the servos, *it is necessary* to talk to the 'Tiny. Controlling the servo is fairly straight forward – there are two functions to be aware of from the **`TINY`** subsystem module in the API. These are:

`ATTINY_set_RC_servos()` – Function allows you to control all 5 ports at once.
`ATTINY_set_RC_servo()` – Function allows you to control 1 servo at a time.

For example, suppose you have an RC servo on port 0 on J7. Then, you could issue:

```
ATTINY_set_RC_servo( RC_SERVO0, 750 );
```

Note: Please consult the *TINY* chapter in the *CEENBoT-API: Programmer's Reference* for details regarding usage of these functions. (Chapter 11, as of Rev. 1.05).

The Ultrasonic Sensor

Once appropriately positioned by the servo, the ultrasonic sensor will be triggered to gather distance-to-wall measurements. You should know by now how to determine the ultrasonic distance readings using the formulas outlined in the previous lab.

To recap, the Ultrasonic sensor will detect objects at a distance of 2cm (~0.8 inches) to 3 meters (3.3yards). The distance sensed by the Ultrasonic Sensor can be calculated from the time returned by the sensor, as follows:

The speed of sound in air at room temperature, 72 °F (22.2 °C), is 344.8 m/s. Multiplying by 100/10,000 to convert to cm/µs, then multiplying by time t in microseconds, gives distance in cm:

$$d_{\text{cm}} = 0.01724 \cdot t_{\mu\text{s}}$$

Conversion to inches: $d_{\text{in}} = d_{\text{cm}} \div 2.54$

Robot Programming

You are to add a wall-following behavior to your behavior-based control program. This behavior will be triggered when a wall is sensed to the left or right of the robot at a distance of 3 feet away. The Wall_Follow behavior is to be implemented using feedback control, as discussed in class. (Note: since you are using the ultrasonic sensor for wall-following, it will be necessary to disable the Sonar_Avoid behavior while Wall_Follow is active.)

To implement a feedback controller, you will calculate error as:

error = goal distance from wall – current distance from wall

Here, the "current distance from wall" is the feedback from the ultrasonic sensor. Calculate this error at a frequency that makes sense. To improve the performance of wall-following, you will integrate Proportional and Derivative (PD) control elements into your wall-following routine. The feedback controller will use the calculated error to control the speed of the motors and the direction of turns needed to minimize error. For Proportional control, the output to the motors is proportional to the error. For Derivative control, the time rate of change of error is calculated, scaled by a gain and added to the controller output. In this way, as error decreases over time, you will slow down your motors as you approach the goal distance in order to avoid overshoot. You will choose gains for the Proportional and Derivative control, K_p and K_d, by trial and error or by using the Ziegler-Nichols method outlined in literature. To get started, you should read the example of the Lego robot implementation of PID line-following in its entirety. It is accessible through a link posted on BB.

The arbitration priorities should be set in order of increasing priority as: Cruise, Wall_Follow, IR_Avoid. It should be easy to disable Light_Follow and Sonar_Avoid from your modular BBC program, by commenting out their function calls from the arbitration list.

Directions

Part I

1. Trigger the Ultrasonic Sensor as you did in the previous Lab to return a distance measurement at a time interval that you determine. It will be important to know *this frequency* at which new distance measurements will be read from the sensor, as you plan your control for wall-following. Timing is important. You should implement your control based on the timing of when new distance measurements are available, and how fast the motors can respond to new motion commands, which will be a function of the speed and acceleration/deceleration of the motors, and how often your behavior code will run.

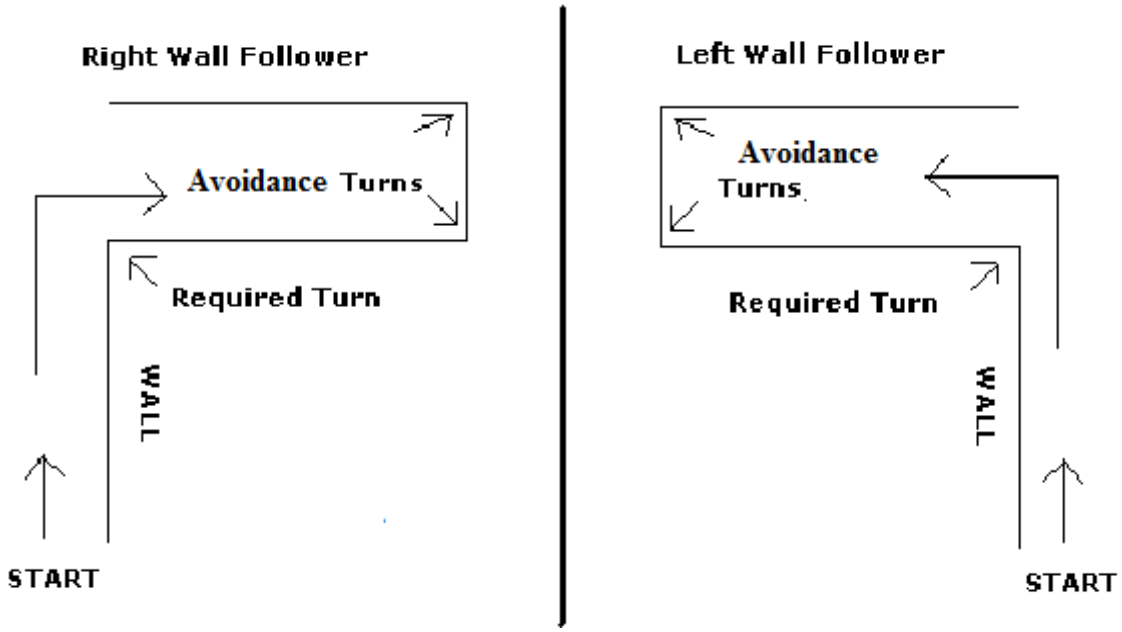
2. Step 1: You will begin with the CEENBoT at roughly 20 inches from the wall. You will need to write a behavior that will move the CEENBoT to roughly 10 inches from the wall, and once at the target distance (of 10 inches), continue to follow the wall, moving parallel to it, and maintaining a constant distance from the wall. Implement this as a Wall_Follow behavior. ***Note: this should NOT be written as a ballistic behavior, but as a servo behavior, based solely on the distance to wall measurement.***

3. Step 2: Observe your performance at wall-following without feedback control. For instance, you might implement a 3- level control: Keep straight, turn right, or turn left, based on error. How long (in distance) does it take to get the robot to within 10 inches of the wall? **Once at this distance, how well does it follow a straight path along the wall? In particular, how much oscillation (if any) do you observe, and at what frequency (oscillations-per-distance)? Do the oscillations ever die out, or are they always present? When do you notice the most overshoot? Describe the performance of this wall-following controller in terms of overshoot, rise time, and settling time.**

4. Step 3: Add a *proportional controller* (P) to your routine. Calculate the error as the difference of the distance from the wall and the goal distance. Determine how you will control the speed and direction of the motors using this error (scaled by the gain K_p), for instance using a 'Turn' variable as described in class. Begin with a generic K_p value, and experiment with increasing and decreasing this value. Then, if needed, implement a trial for the *Ziegler-Nichols* method outlined in the 'LEGO document'. Does this K_p improve the wall-following performance? **Repeat step 2 and record performance changes with (P) controller added.**

5. Step 4: Now add the *derivative controller* (D) to your *proportional controller* (P). This is now considered PD-control. To incorporate the (D) component, calculate the *derivative* as *current error* minus *previous error*. The derivative term is then $K_d \cdot (\text{derivative})/dt$, however since "dt" remains constant, it is essentially absorbed by K_d, so your derivative controller reduces to $K_d \cdot (\text{derivative})$, where once again, the "derivative" (for our purposes) is simply – *current error* minus *previous error*. With a proper proportionality constant K_d, if you subtract in the correct order, then adding this term will have the effect of decreasing changes to motor speed and turn radius when the error grows smaller (closer to the set-point), and increasing motor changes when the error grows larger. Use a K_d value either based on the K_p value chosen in step 3, ,or one chosen by trial and error. **Repeat step 2 and record once again performance changes with the addition of the (D) controller.**

6. Step 5: Modify your code as needed so that your CEENBoT can successfully make a right turn (if you choose to do a *right-wall* following) or a left turn (if you choose to do *left-wall* following) when the wall bends at a 90-degree angle. The turns you need to make for the left or right wall-following scenarios are shown below:



Questions to answer in the RESULTS section of your Report

Answer all questions and present all data you were asked to record in the Directions section. In addition, discuss what hurdles you had to overcome to get it to work. What did you observe about servo-control and the P and D feedback controller implementation? How did specific values for K_p and K_d impact the robot's behavior in terms of time to reach the set-point (*rise time*), the amount of overshoot, and the settling time of the overshoot? Describe your approach and discuss your results to all parts of the lab exercises. In particular, describe in detail how you implemented the P and D controllers and how you controlled motor speed and direction. What did you learn? Append your C-code to the report. You will also demonstrate your results in class.

.

Deliverables

Demonstrations

You will demonstrate your robot's execution of these new capabilities during the designated class meeting or lab time.

C-code

Include a printout of your CEENBoT program. You may include *snippets* of your code and embed them in your lab report as you discuss how your program works, but a separate attachment of your entire source code must be included as part of your report.

Lab Report

The lab report should include all of the following:

- **Title Page** – Include *Course Number*, *Course Title*, *Instructor Name, Your Name*, *Lab Name*, & *Due Date*.

- **Overview** Section (a brief paragraph of what the lab was about, and the purpose behind it). Please also make sure you touch upon the following as well:

  1. Resources used: (human, text, online or otherwise).
  2. Time invested in this project.
  3. A high-level description of your robot and program.
  4. If this was a *team assignment*, state how tasks were delegated and split up among you.
  5. Known problems with your solution (e.g., the robot will not work on the *carpet*, or... it breaks if you make it go further than 5 ft (for whatever reason), etc.)

- **Background** Section (discuss some of the theory addressed in this particular lab).

- **Procedure** – Briefly describe the lab procedures for each lab – what were you asked to do? What did the process consist of? How *did you* approach it?

- **Discuss your Source Code** – Discuss and explain any relevant details behind the motivation and manner in which you have written your source code. You can *embed* [small] portions on of your code that are relevant to the discussion for clarity, but please ensure to keep a complete copy of your source code as a separate attachment (see *Source Code* section below). You may also have the reader *refer* to particular pages of your source code attachment instead.

- **Results –** Use this section to answer *questions* posted throughout your lab exercise. In particular those given out in the '*Directions*' section. In the '*Procedure'* section, feel free to refer the reader to *this* section for additional info – that is, there is no need to repeat information.

- **Conclusion** – Reiterate the objective of the lab, discuss what you have learned and any further comments you might have.

- **Source Code** – Attach your C-program also.

Grading

Laboratory Grading Rubric		Labs will be graded out of 30 points, unless otherwise	
Points	**Lab Report**	**Code**	**Demonstration**
10	Follows lab format, all sections are complete, thorough, concise. Answers all questions posed.	Properly commented, easy to follow with modular components.	Excellent work, the robot performs exactly as required.
7.5	Does not answer some of the questions or has spelling, grammatical, content errors.	Partial comments and/or not modular.	Performs most of the functionality with minor failures.
5	Multiple grammatical, format, content, spelling errors, and/or questions not answered.	No comments, not modular, not easy to follow.	Performs some of the functionality but with major failures or parts missing.
0	Not submitted or submitted late.	Not submitted or submitted late.	Meets none of the design specifications or not submitted.