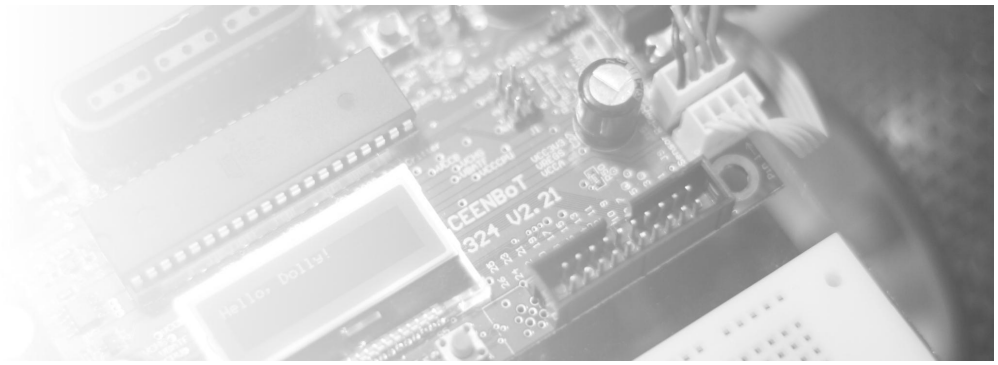
Mobile Robotics I: Lab 2

Dead Reckoning*: *Autonomous Locomotion Using Odometry

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



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

Rev 1.04

(Blank)

Purpose

In this laboratory you will become familiar with programming the CEENBoT mobile robotics platform and use it to explore *robot locomotion* using *odometry*. Specifically, you will examine the problems that arise when using raw odometry alone for position estimation and how it impacts the ability to program the robot for autonomous movement along a predetermined path.

Lab Objectives

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

- Learn how to program the CEENBoT's primary control board microcontroller in C using the CEENBoT Application Programming Interface (API) suite of functions.
- Become familiar with the physical aspects of the CEENBoT robotic platform, including characteristics of its frame and its effectors, controlling its movement, and its physical limitations.
- Learn to control the CEENBoT's stepper motors and use differential drive control to move the robot straight forward, at a right angle turn, in a circle, and along a path. Characterize the impact of speed and distance on the ability to achieve the expected robot path movements.
- Explore the problems with using odometry alone for robot locomotion. Calculate and characterize the errors observed when moving the robot along a desired path using odometry. Make refinements to try to improve robot path tracing performance.

Requirements

Preliminary Readings

- Read about *odometry* in the *Mataric* text: **`Chapter 19, section 19.1, pg. 225-227.`**
- You must have successfully completed and have read through the *CEENBoT-API – Getting Started* guide. This document guides through the procedure of writing CEENBoT programs that take advantage of the CEENBoT-API – the document discusses program structure, compiling, linking, and flashing of your CEENBoT.

Required Equipment

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

Background

The foundation of this laboratory exercise is in exploring the concept of *robotic locomotion* – in particular, you are to explore the a type of robotic locomotion using a type of navigational procedure known as *dead reckoning*. ***Dead reckoning*** refers to navigation by calculation. It is a method of determining the position of a plane or ship (or a robot) by making a graph of its course and speed from a previously known position.

Locomotion on the other hand is a generic term which refers to the act of making a robot move around in its environment in an intelligent manner. ***Odometry*** literally means "measuring the journey". Think about the trip odometer in a car. In the case of a robot, odometry consists of the robot measuring its wheel rotations.

Most mobile robots possess a set of wheels for movement. The motors that drive the wheels generally contain encoders that provide the robot with *feedback* information of how much and in what direction the motor shaft has rotated. This data allows the robot to estimate where it is (pose estimate) relative to a known starting point. Encoders count the number of motor rotations, and with a little math this count can be converted into a distance traveled. In the case of stepper motors, it's slightly different. Stepper motors operate by successively energizing the various coils of a motor in order to rotate a magnetic field through its different windings. This results in a shaft that rotates in discrete angular movements, called steps. In a stepper motor, a 360-degree wheel revolution is divided into a whole number of *steps*. Each step moves the motor some percentage of a revolution, either in a forward or reverse direction based on the stepping order.

Your CEENBoT '324 v2.21 platform uses ***stepper motors*** that are designed with 200 steps per wheel revolution. That means each wheel in the CEENBoT can step forwards or back with a minimum granularity of `360/200 = 1.8` degrees per step. This is a rather 'fine' granularity from a locomotive standpoint. The CEENBoT-API provides facilities for controlling these stepper motors. You can specify precisely how many steps each stepper should travel, along with direction and speed (in steps/sec).

Note: Remember to read the *Getting Started* manual for using the CEENBoT-API.

Note: After reading the *Getting Started* guide, make sure to keep a copy of the *Programmer's Reference* manual for the CEENBoT-API where all of the functions for controlling the steppers along with required parameters with descriptions of the same are provided.

Even with this 'locomotive precision' made inherently possible by the fine stepping distance of the stepper motors the fact remains that mechanical systems are not perfect. Wheels slip from the surface, and stepper motors might miss steps along the way. In addition the topology of the terrain may exacerbate this problem. As a result, odometry measurements *will* accumulate errors over time. Therefore, in this lab, you will program and use your CEENBoT to precisely observe first hand these aforementioned issues, comment on your findings and results achieved with this locomotion and position estimation method.

Directions

Part I – Dead Reckoning In a Straight-Line Pattern

For this first Part you will start with a simple task. Your goal is to write a C-language program to make the CEENBoT drive in a *straight* line. You must mark the start and top positions in some way (e.g., masking tape, etc) on tiled floor. It is helpful to use a lab partner to help.

1. For 4 different distances (`6 in`, `12 in`, `18 in`, `24 in`) write a program to make the robot drive in a STRAIGHT line. Mark the start and stop positions with masking tape on the tile floor. It is helpful to use a lab partner to help.

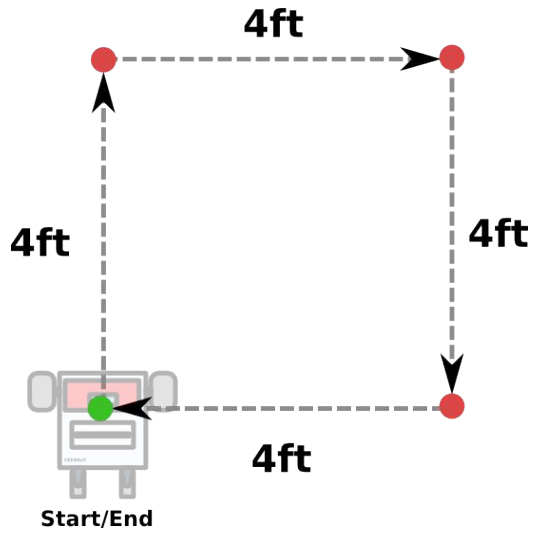
2. Drive the robot for each distance 5 times generating 20 data points and record the data on a table. Give the wheel speed used. Perform an error analysis by calculating the average error at each distance. Also, calculate the average distance traveled for each distance, and state the shortest and longest distances. Does the odometry error change based upon the distance driven? If you notice the robot consistently drifting to one side (e.g. left wheel always goes a little slower), how can you accommodate for this in the robot's program to make the robot move in more of a straight line?

3. Change speeds (slower or faster and state the speed you use in the write up), and repeat part 1 for two of the distances in Part 1 at which the largest errors occur. Does changing robot speed increase or decrease the average error at each speed? How does changing robot speed affect odometry error?

4. Move the robot to a carpeted floor and repeat part 1 and discuss the differences in the robot performance on a different surface. Make the same 20 point data table and error analysis notes.

Part II – Dead Reckoning in a Square Pattern

1. Write a C-language program using the CEENBoT-API that will move the robot in a square pattern. Each square of the leg should be 4-feet long. This is illustrated in the figure below:



2. Measure how well your 'odometry' performs. Recall that your robot relies on 'relative' odometry assumptions to determine how far it has traveled, and how far it has turned. If odometry readings are perfect, then the robot should make a perfect square. Therefore perform the following:

   a. Measure the amount of *odometry error*:

   i. Mark the path of the square pattern with masking tape (your T.A. *may* have a setup for you).
   ii. Place a marker at each of the other three corners.
   iii. Run your program and let the CEENBoT run through the square pattern. You must place a mark at the point where the robot turns.
   iv. Measure the distance between each pair of points (the *correct* corner and the *actual* from your CEENBoT). Compute the average – this will be your *average error*:

   Distance Error (1st corner): _____ Distance Error (3rd corner): _____

   Distance Error(2nd corner): _____ Distance Error (4th corner): _____

$$e_{avg} = \frac{e_{d1}+e_{d2}+e_{d3}+e_{d4}}{4}$$

   v. Record the average error using the above formula and think or discuss the possible sources of error.

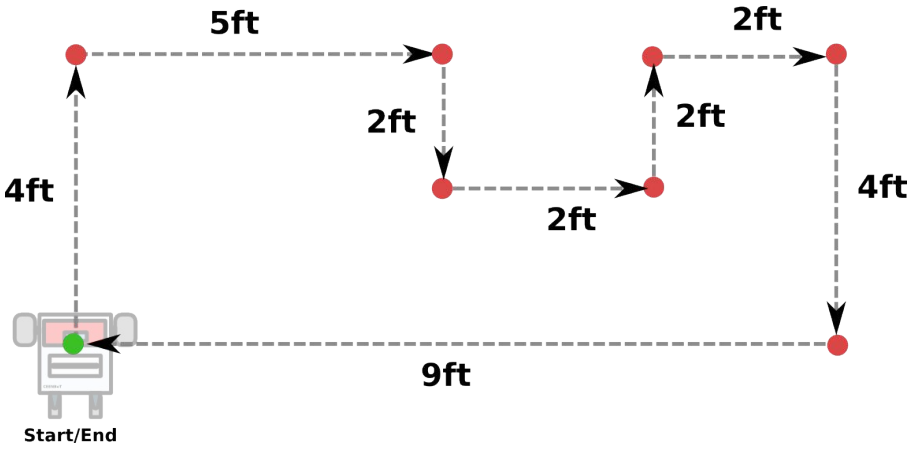
   Average Error: _____

b. How can you improve the performance (reduce the error) of the right turns? Experiment with different ways of making a turn to better keep on the path (Recall there are multiple ways of making a turn using differential drive control). Repeat the above error analysis. Were you able to improve your results – decrease your average error?

c. Does speed make a difference in making the turns better or worse ? Run the square at different speeds and find out! Record your findings.

Part III – Dead Reckoning along a Maze Path

1. Write a C-language program using the CEENBoT-API that will move the robot in the following pattern:



2. Repeat the procedure you performed in Part I – that is, determine the accuracy of your *odometry* results. Let your CEENBoT run the pattern and mark the location where your CEENBoT makes a turn. Then measure the distances from the *true* corner to the *actual* corner taken by your CEENBoT. These are the distance errors – record these below :

   Distance Error (1): _____ Distance Error(5): _____

   Distance Error (2): _____ Distance Error(6): _____

   Distance Error (3): _____ Distance Error(7): _____

   Distance Error (4): _____ Distance Error(8): _____

3. Determine the average error using:

$$e_{avg} = \frac{e_{d1}+e_{d2}+e_{d3}+e_{d4}+e_{d5}+e_{d6}+e_{d7}+e_{d8}}{8}$$

   Average Error = _____

4. Do you make it back to the same starting point? How far off does your 'BoT end up from the starting point? Where do you observe sources of error along the path? Is the error constant or cumulative? Be ready to demonstrate this maze in class.

(*Continued on next page*)

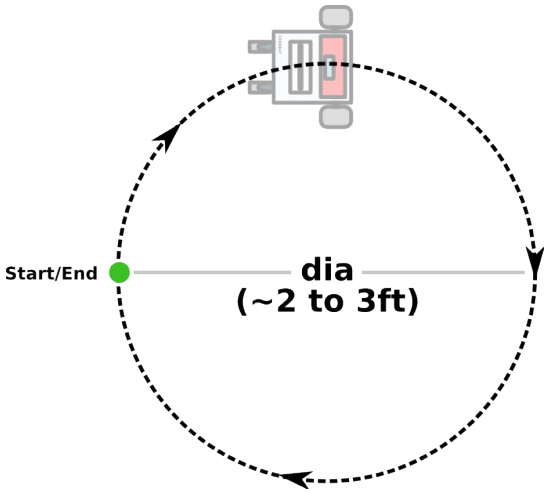
(*Continued*)

5. Could the average error be improved? Either through *differential drive*, or *speed control*? Does it make a difference? Describe how you control each motor's speed and direction to achieve your lowest error performance for making the 90 degree turn.

6. What are some sources of odometry error noticed in this lab? Name at least two. What did you do to overcome some of the error? What is needed to further correct for odometry error in robot locomotion (beyond this lab)?

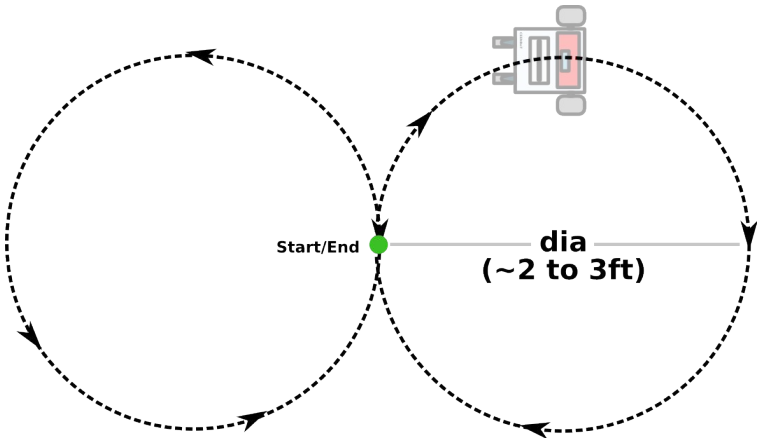
Part IV – Dead Reckoning in Circular Motion

For this part, your goal is to make the CEENBoT move in a circle with a diameter of your choosing (~2 to 3 ft should be good). Work to get the robot start and end at the same point. Comment on your successes and challenges in attempting this exercise.



Bonus

As a bonus, extend the algorithm program you wrote for circular motion in Part IV above and make the robot move in a figure "8" fashion as shown below. (Note the dimensions *you* used in your attempts). Can you get the robot to pass through the same center point of the figure 8 each time? (You'll earn additional [bonus] points if you do). Calculate the average error observed for the center point of your figure 8 in 4 runs.



Note: This is not as easy as it sounds. You need to look at the *Programmer's Reference* manual for the API and determine which functions will empower to you achieve this goal. For example, maybe you'll need to issue a command in *free-running* mode and then 'modulate' the speed of left and right motors accordingly to achieve the figure-eight pattern. As a tip – look at `STEPPER_run()` and `STEPPER_set_speed()` functions.

Deliverables

Demonstrations

Demonstration of your final results of Parts II, III, and IV of this Laboratory will occur next Wednesday during class time. Bring robots fully charged to class. See the T.A. for procedures on saving your code and re-flashing the CEENBoT for battery charging. Also consider using the on-board push-button switches to select and start programs so that you can easily run one demonstration after the other without a need to re-program your robot.

C-code

Include a printout of your CEENBoT program. 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

Write a lab report which contains 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: (human, text, online or otherwise).
  2. Time invested in this project.
  3. A '30,000ft', 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. Problems you encountered and solutions you used to overcome them (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 behind the topic discussed – *locomotion, dead reckoning,* etc).

- **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.

(*Continued on next page*)

- **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.

Evaluation

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.