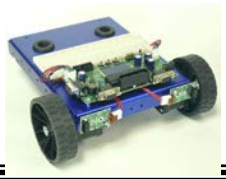

Lab 1

Getting to Know Your Robot: Locomotion and Odometry

(Demonstration due in class on **Thursday**)

(Code and Memo due in Moodle drop box by midnight on **Sunday at midnight**)

Read this entire lab procedure before coming to lab.

****

Read this entire lab procedure and complete the reading in Part 1 before coming to lab.

Purpose: The purpose of this lab is to confirm that you have all of the necessary software installed on your computer to program the CeenBoT. The secondary goal is to get the robot moving and to examine problems with raw odometry for pose estimation.

Objectives: At the conclusion of this lab, the student should be able to:

- Install the necessary software to program the CEENBot
- Describe the primary components of the CEENBoT including the actuators, effectors, and sensors.
- Program the CEENBoT primary control board microcontroller in C by using the CEENBoT Application Programming (API) suite of functions.
- Control the CEENBoT's stepper motors and use differential drive to move the robot in a circle and square.
- Apply the UMBARK test to identify Type A and Type B odometry errors and then calculate a tuning factor to compensate for the systematic errors.

Equipment: CEENBoT platform, '324 v2.21.
AVR In-System Programmer (ISP)
Masking Tape
Ruler

Software:

CEENBoT API Library	http://www.ceenbotinc.com/tools/capi324v221-v1.09.002R.zip
CEENBoT Programming	http://www.ceenbotinc.com/tools/CEENBoT-API-Prog-Fund(Rev1.02).pdf
CEENBoT API Reference	http://www.ceenbotinc.com/tools/CEENBoT-API_ProgReference(Rev1.08).pdf
WinAVR GCC toolchain	http://sourceforge.net/projects/winavr/files/WinAVR/20100110/
AVR Studio 6.2	http://www.atmel.com/tools/atmelstudio.aspx

Theory:

Locomotion refers to moving a robot from place to place. *Odometry* is a means of implementing dead reckoning to determine a robot's position based upon the robot's prior position and the current heading and velocity. The advantages of this method are that it is self-contained, it always provides an estimate of position, and positions can be found anywhere along curved paths. The disadvantages are that the position error grows and require accurate measurement of wheel velocities over time.

There are several types of odometry error including systematic from unequal wheel diameters, misalignment of wheels, finite encoder resolution and finite encoder sampling rate. There is also non-systematic odometry error such as travel over uneven floors, unexpected objects on the floor and wheel slippage. Lastly, there are odometry errors such as imprecise measurements, inaccurate control models and immeasurable physical characteristics. These inaccuracies in dead reckoning cause a robot traversing a square path to yield a result similar to Figure 1.

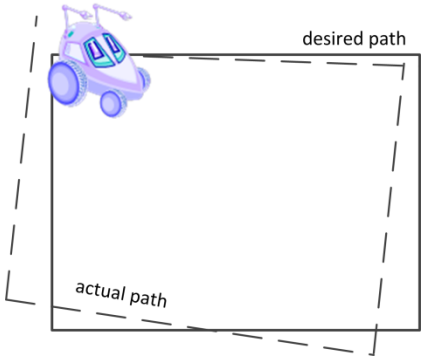

Figure 1: Robot Odometry Error

Thus, dead reckoning is most appropriate for short distances because of the error accumulation. For accuracy, an encoder or some other feedback sensor should be used to periodically null out or reset the accumulation error.

Most mobile robots contain wheels with an actuator. The actuator is typically a motor with an encoder to feedback information about how much and in what direction the motor shaft was rotated. This allows the robot to estimate its position relative to s a start point. The CEENBoT actually has stepper motors and they operate a little differently. 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. The 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). Even with 'locomotive precision' made 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.

Part 1 – Software Installation

1. Read and complete the tasks in the *CEENBoT API Getting Started Manual* available at http://www.ceenbotinc.com/tools/CEENBoT-API_GettingStarted(Rev1.07).pdf. This document explains the process of writing programs for the CEENBoT using the API. The API discusses program structure, compiling, linking and flashing the CEENBoT.
2. Read the *Quick Guide to Programming Fundamentals,* to quickly learn how to use features that you may need right away such as motion, delays and IR sensors. This document is available at the following link: http://www.ceenbotinc.com/tools/CEENBoT-API-Prog-Fund(Rev1.02).pdf
3. Finally, you should always have the Programmer's Reference Manual available as you work and it can be downloaded from: http://www.ceenbotinc.com/tools/CEENBoT-API_ProgReference(Rev1.08).pdf

Things to watch out for:

- Make sure you follow all of the steps explicitly
- Make sure you have the GCC toolchain installed and on page 12 for linker options -Wl,-u,vfprintf should be a lower case L after the W
- Also the libm.a and libprintf_flt.a are not in the same directory as libcapi324v221.a so you have to add the `C:\WinAVR-20100110\avr\lib\avr6` directory to Project Options - >Library and then add them to the list under libcapi324v221.a and then remove that directory
- On page 12 the use AVR is always checked but there may be nothing in the avr-gcc: or make: boxes so you have to navigate to the correct directory.
- If AVR Studio does not recognize the AVR ISP MKII programmer, confirm in device manager that the driver was installed under Jungo and not a printer or some other USB device. If this happens, uninstall the driver and let the computer search until it reinstalls the correct driver.

****

LAB PROCEDURE

****

Part 2 – Square Path

Now that we have code to control the robot, let's measure how well odometry performs. As you may recall from the theory, the robot relies on odometry to determine how far it has traveled, and how far it has turned. If odometry readings are perfect, then the robot movement should be accurate and repeatable.

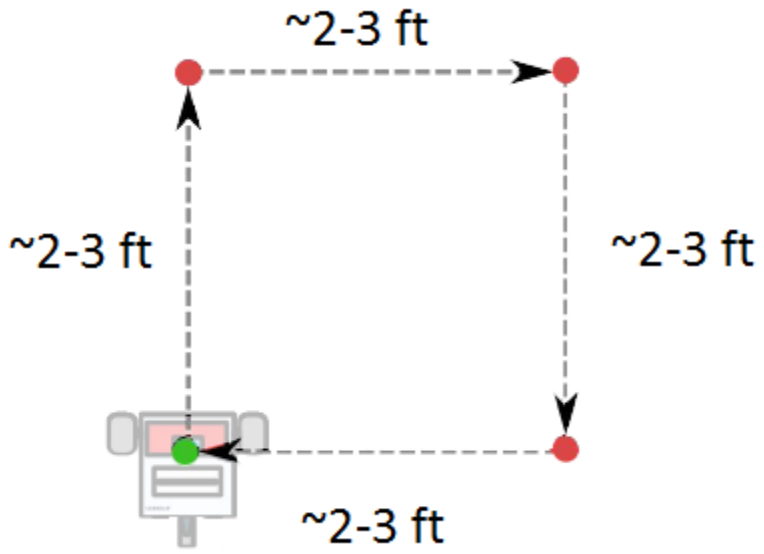

Figure 2: Square Robot Motion

1. Write a program to move the robot in a square path with sides between 2 and 3 feet (see Figure 2).
2. Place masking tape on the ground where the robot will start.
3. Run the *"Square"* program 5 times with the robot traversing the path in a clockwise direction.
4. After each run, find the difference in the final x and y position from the original start position. Typically the y position is the axis direction of forward motion and the x axis is the lateral translation axis.
5. Run the *"Square"* program 5 more times with the robot traversing the path in the counterclockwise direction.
6. After each run, find the difference in the final x and y position from the original start position.
7. If this data is plotted on an x-y scatter plot, there will be two distinct quadrants, one for the cw runs and one for the ccw runs (see Figure 3).
8. You should include the table of data for the 10 square traversals (cw, ccw) and the graph similar to Figure 3 for your data in the lab memo submission.

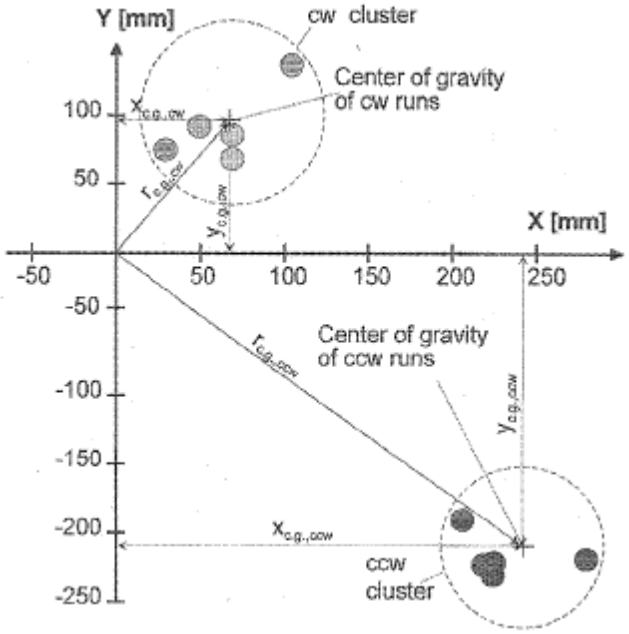

Figure 3: Odometery error on an uncalibrated vehicle

Part 3 - UMBmark method

1. The University of Michigan Benchmark (UMBark) method is used to quantitatively measure odometry error. This method is used to identify systematic errors and then calibrate the system using a tuning factor.
2. The first step is to use the data collected in part 3 to calculate the center of gravity for the cw and ccw runs. These values are found from

$$x_{c.g.,cw/ccw} = \frac{1}{n}\sum_{i=1}^{n} ex_{i,cw/ccw} \quad (1)$$

$$y_{c.g.,cw/ccw} = \frac{1}{n}\sum_{i=1}^{n} ey_{i,cw/ccw} \quad (2)$$

where n is the number of times the robot runs clockwise or counterclockwise runs, ($ex_{i,cw/ccw}$, $ey_{i,cw/cw}$) is the offset of the final position in Cartesian coordinates from the initial position in Cartesian coordinates.

3. The center of gravity for each run is also shown in Figure 3. The distribution of the readings within each quadrant is a result of nonsystematic errors. The asymmetry in the centers of gravity between the cw and ccw runs is due to two types of systematic errors. Type A errors are due to an increase or decrease in robot rotation for both cw and ccw runs. Type B errors are due to an increase or decrease in robot rotation for the cw run and the opposite effect happens for the ccw run. Type A errors occur because of uncertainty in the wheelbase measurement while Type Be errors because of unequal wheel diameters. A single numeric value that expresses the odometric accuracy for systematic errors can be found from

$$E_{max,syst} = \max(r_{c.g.,cw}; r_{c.g.,ccw}) \quad (3)$$

where

$$r_{c.g.,cw} = \sqrt{(x_{c.g.,cw})^2 + (y_{c.g.,cw})^2} \quad (4)$$

and

$$r_{c.g.,ccw} = \sqrt{(x_{c.g.,ccw})^2 + (y_{c.g.,ccw})^2}. \quad (5)$$

4. The radius, r, for each center of gravity is also shown on the Figure 3. Find the radii for your data.

5. From this calibration, there are two constants that are used in the basic odometry computation. The constants are α and β given by the following formulas assuming the square path has a side of length D, and the robot has a wheel base, B,

$$\alpha = \text{average}\left(\frac{x_{c.g.,cw}+x_{c.g.,ccw}}{-4D}, \frac{y_{c.g.,cw}-y_{c.g.,ccw}}{-4D}\right) \quad (6)$$

$$\beta = \text{average}\left(\frac{x_{c.g.,cw}-x_{c.g.,ccw}}{-4D}, \frac{y_{c.g.,cw}+y_{c.g.,ccw}}{-4D}\right) \quad (7)$$

6. Calculate α and β for your robot's data. Measure the wheel base, B, for your robot and use equations (8) - (12) to find the tuning factors for the wheel base, c_b, left wheel, c_l and right wheel, c_r.

$$E_d = \frac{D+B\sin(\frac{\beta}{2})}{D-B\sin(\frac{\beta}{2})} \quad (8)$$

$$c_b = \frac{\pi}{\pi-\alpha} \quad (9)$$

$$c_l = \frac{2}{E_d+1} \quad (10)$$

$$c_r = E_d c_l \quad (12)$$

These values could be used to scale the drive commands such as velocity or distance.

7. Use the base tuning factors for the forward distance, and the left and right tuning factors for the ninety degree turns. Comment in your memo on whether you observe any difference in the odometry error.

Part 4 – Circle and Figure 8

1. Write a *"Circle"* program to move the robot in a circle with a diameter between 2 and 3 feet (see Figure 4).
2. Try to tweak the tuning factors to get the robot to start and end at the same point. Comment on the results of this task in your memo.

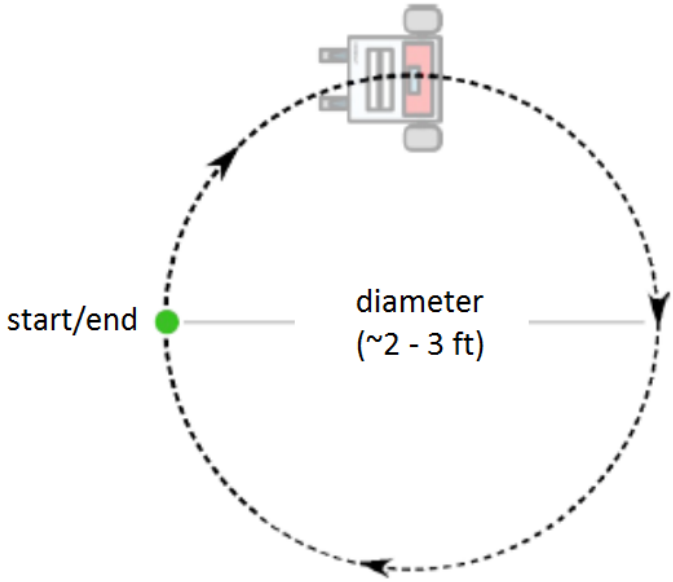

Figure 4: Circle Robot Motion

3. Finally, modify the *"Circle"* program to create a *"FigureEight"* program to move the robot in a figure eight using two circles (see Figure 5).

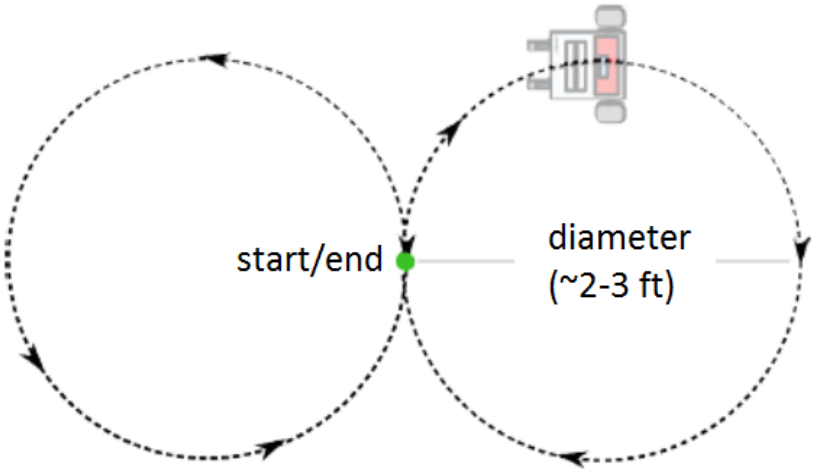

Figure 5: Figure Eight Robot Motion

4. Finally, modify the *"Circle"* program to create a *"FigureEight"* program to move the robot in a figure eight using two circles (see Figure 5).
5. Once again try to adjust the tuning factors so that the robot passes through the same center point of the figure 8 each time.

Demonstration

Finally, create a program to demonstrate all of your robot capabilities. You can use the three pushbuttons on the controller board to send the square, circle and figure eight without having to re-program the robot after each movement. You may also want to use the LCD or buzzer to indicate the current motion being executed.

Bring your robot fully charged to class on Thursday for the demonstration. Note that you always must re-flash the factory firmware and plug in the AC adapter in order for the robot to charge. Alternately, you can put the robot battery in the RC car battery charger. Note that this is a fast charger and will not last as long as the outlet charge.

More questions to answer in the lab memo

1. What are some sources of the odometry error?
2. How could you correct for this error?
3. How could you improve the three motions (*Square, Circle, FigureEight*) programs?

Memo Guidelines:

Please use the following checklist to insure that your memo meets the basic guidelines.

- ✓ Format
  - o Begins with Date, To , From, Subject
  - o Font no larger than 12 point font
  - o Spacing no larger than double space
  - o Includes handwritten initials of both partners at the top of the memo next to the names
  - o Written as a paragraph not bulleted list
  - o No longer than three pages of text
- ✓ Writing
  - o Memo is organized in a logical order
  - o Writing is direct, concise and to the point
  - o Written in first person from lab partners
  - o Correct grammar, no spelling errors
- ✓ Content
  - o Starts with a statement of purpose
  - o Discusses the strategy or pseudocode for implementing the robot paths (may include a flow chart)
  - o Discusses the tests and methods performed
  - o States the results including error analysis
  - o Shows data tables with error analysis and required plots or graphs
  - o Answers all questions posed in the lab procedure
  - o Clear statement of conclusions

Grading Rubric:

The lab is worth a total of 30 points and is graded by the following rubric.

Points	Demonstration	Code	Memo

10	Excellent work, the robot performs exactly as required	Properly commented with a header and function comments, easy to follow with modular components	Follows all guidelines and answers all questions posed
7.5	Performs most of the functionality with minor failures	Partial comments and/or not modular with objects	Does not answer some questions and/or has spelling, grammatical, content errors
5	Performs some of the functionality but with major failures or parts missing	No comments, not modular, not easy to follow	Multiple grammatical, format, content, spelling errors, questions not answered
0	Meets none of the design specifications or not submitted	Not submitted	Not submitted

Here is an excerpt from the RangeSensors.c program so that you can see an example of a proper heading and properly commented code.

```
//RangeSensors.c
//A demo application to test robot locomotion
//CEENBoT peripherals and effectors
//Carlotta A. Berry
//Dane Bennington
//Jose Santos
//August 9, 2011
	//main function to run the robot
	void CBOT_main(void)
	{
		ATopstat = ATTINY_open();//open the tiny microcontroller
		LEopstat = LED_open(); //open the LED module

		//keep the microcontroller running
		while(1)
		{
			btnValue=WaitButton();
			if (btnValue==1)
			{
				//Check Light Sensors
				//beep once
				SPKR_play_beep( 500,500,100);//500 Hz for 500 ms
				i = 0;
```

Submission Requirements:

You must submit you properly commented code as a zipped folder of the C code and the lab memo in a zipped folder by **11:59 pm on Sunday** to the Moodle Course Drop box. Your code should be modular with functions and classes in order to make it more readable. You should use the push buttons, buzzer and LCD to indicate the robot state during program execution.