Date: 12/07/2014
To: Dr. Carlotta A. Berry
From: Alexandre van der Ven de Freitas and Peter Richard Olejnik A.V.D.V.F, P.R.O.
Subject: Lab1

Introduction

The purpose of this lab is to set up all the software tools required by the CeenBoT and to make the robot move in a square, circle, and figure-eight paths, as well as to perform an odometry analysis for the square path.

Initial Setup

All the required software tools were successfully installed: Atmel Studio 6.2, AVR drivers, the required compiler and APIs. The biggest struggle with the initial setup was caused by a broken an ATmega 1284 microcontroller that was later on replaced by an ATmega 324P which seems to have less memory space and power, but should not pose a major problem.

The Robot's Program

The created code is basically consisted of 4 functions: one main function that has to necessarily be called CBOT_main() (or the program won't run), moveSquare(), moveCircle() and moveEight(). The names are very self explanatory. Within the main function we have a portion of code dedicated for setup and error detection, and the while loop where the tasks are in fact performed. For the setup portion the LCD subsystem is opened without error verification and the ATTINY subsystem is opened with error verification, so in case the ATTINY subsystem failed to open an error message would be displayed on the LCD screen. The ATTINY subsystem controls and technically accesses the ATtiny microcontroller, also embedded on the robot's motherboard, which receives inputs from the push-buttons and other sensors. In the while loop when one of each of the buttons is pressed the robot performs one of the tasks previously mentioned and modulated by their respective functions. Each of the functions is also consisted of an initial setup, in which opens the stepper subsystem and handles error if it happens, followed by the robot stepper motor motion functions themselves. Figure 1 explains in a high level how the program works.

Testing the Program

The program was debugged using the IDE provided and through test runs. It was first tested if the buttons were working and those worked right away. After that, the moveSquare() function was created and tested using a ruler to measure how far the robot is going in a straight line and if the robot comes back to its initial spot. For the moveCircle() function the diameter was measured and it was verified if the robot would come back to its initial spot. The same procedure was used to test moveEight() for both of the circular paths. All tests passed after iterative calibration.

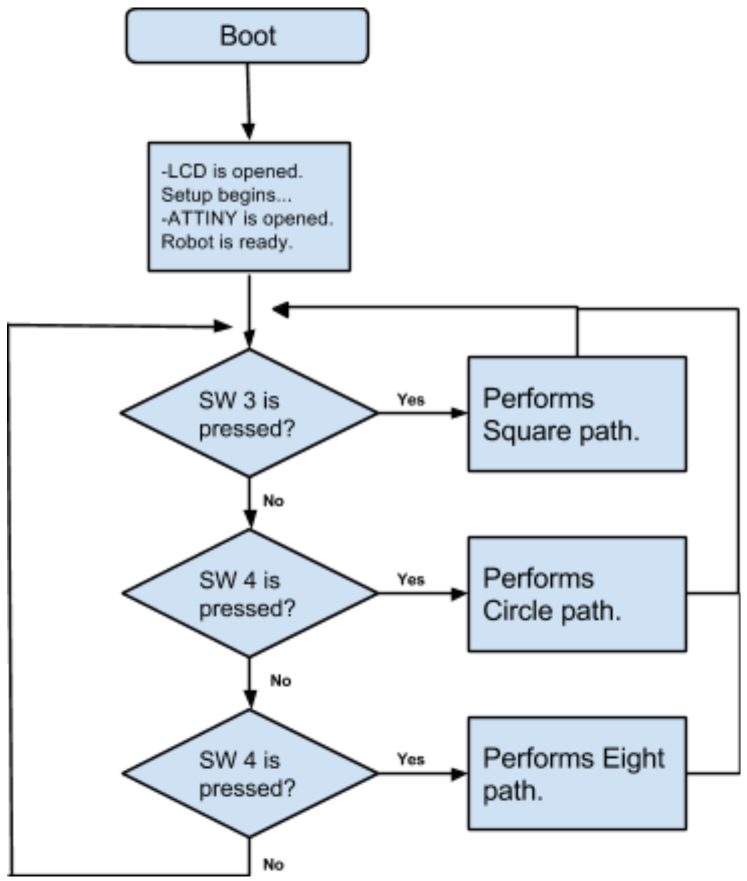

Figure 1 - Robot program flowchart.

Data Collection and Interpretation

The robot's square program was ran 5 times to collect all the clockwise data. The program was then slightly modified to have the robot run in a counter clockwise fashion. All the data collected can be seen below in Table 1.

Trial	CW Delta X (in)	CW Delta Y (in)	CCW Delta X (in)	CCW Delta Y (in)
1	2.5	3.5	-2.5	-1.5
2	-0.5	4.5	-4.5	-1.5
3	-2.0	4.0	-4.0	-0.5
4	3.0	3.5	-4.5	-2.0
5	0.0	3.5	-4.0	-2.0

Table 1 - Data collected from running the robot.

Following the collection of the data, it was plotted below on Figure 2. In addition to this, the UMBark method was ran. The center of gravities and radii computed from this method were also plotted.

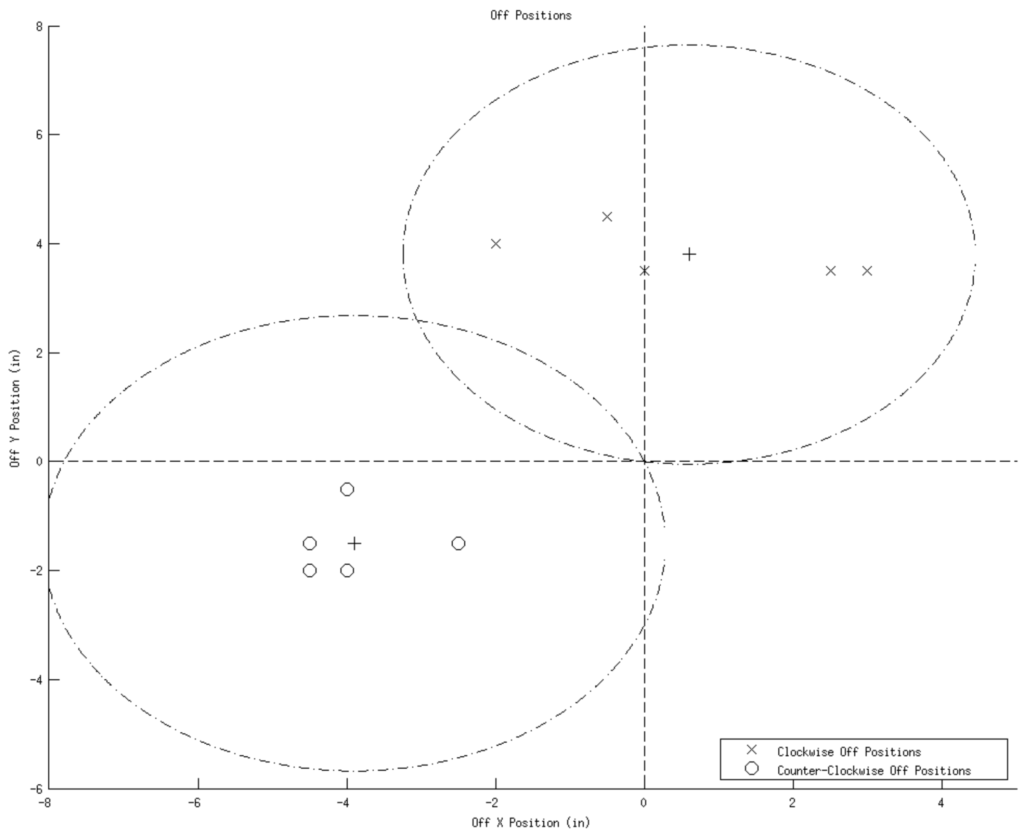

Figure 2 - The data points, as well as the points of interest on a single plot.

When the UMBark method was fully finished, it was found that the tuning factors for the wheelbase, left wheel, and right wheel were 0.9974, 1.0040, and 0.9960 respectively. This unfortunately indicates that our errors are not systematic and that they cannot be corrected simply by adding a factor.

Discussion of Questions
Sources of error in odometry can come from a few places. One such source is the imperfection of the open loop system. For instance, in our robot, slippage of the wheel on the ground or skipping of steps in the motor would cause errors. Also, imperfections in the mechanical design can cause errors as well.

Odometry errors can be corrected in a few ways. The simplest way to is to add correction factors to appropriate values. However, this is only valid for errors which are systematic. A second way is to add some sort of feedback and have the robot operate in a closed loop. This however, requires sensors of sort, which adds additional complexity to the robot.

The three motions that our robot performed in this lab cannot be improved by much at this point. As mentioned above, the errors our robot is experiencing at this point are not systematic. Because of this, a simple adjustment in a few values will not suffice. The robot's algorithm needs to be adjusted from an open-loop system to a closed loop system, with sensors for feedback.

Conclusions

Overall, this lab went well. Aside from some initial start-up issues, everything went smoothly and was straightforward. We were able to have the robot run in the three prescribed manners and collect the data that was needed.