Date: 12/09/2014
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: Lab2 - Go to goal - Odometry - Motion Control

Introduction
The purpose of this lab is to program the robot to respond to an input value of an angle by turning the robot to that angle and to reach a specific goal located into a x-y plane dimensioned in feet using - Turn and then forward approach.

The Robot's Program
The created code is consisted of 6 functions including main. One additional library was included besides the ones required to work with the CeenBot which is the math.h. The standard C math library saved a good amount of lines of code with its atan2() and hypot() functions, as well as the pi constant, programed as M_PI. The other functions that were created can be divided into two different categories: Motional and Cognitive. Motional functions are moveForward(), sTurnLeft() and sTurnRight() since they are directly related to the motion of the robot. Cognitive functions are not literally cognitive, but performs the proper calculations that allows the robot perform its required tasks successfully. These functions are: go2Angle() and go2Point().

Starting with the motional functions, which are the simplest, moveForward() takes two inputs which are the number of steps and the speed. Both sTurnLeft() and sTurnRight() take also two inputs: angle and speed. Within the function the value of the angle will be converted into steps so the robot can turn within the necessary amount of steps that covers the given angle. The two functions do not take in consideration negative angles. The negative angles are handled by go2Angle() function where negative angles are converted into positive before being inserted in sTurnRight(). If the angle is neither positive or negative, therefore zero, the robot does nothing. go2Angle() also takes an integer angle as its input. Also if angles are bigger than 180 degrees, those angles are divided by two and then the turn functions are called twice. This solution was implemented because of a mysterious issue that when angles that are usually bigger than 180 makes the robot stuck in a spinning loop. My personal guess is that either a funky data type is influencing the values in the memory stack of the microcontroller or we are, so far, being completely unable to detect some sort of logic error in the code.

Continuing to cover Cognitive functions, go2Point() receives an (x,y) coordinates in feet that are then converted to inches. Δx and Δy are then calculated based on the current position of the robot (x_o,y_o). The angle ϑ is then calculated using the function atan2(Δy,Δx) that is then multiplied by $\frac{180}{\pi}$ in order to obtain the angle in degrees. $\Delta\vartheta$ is then calculated by subtracting the desired angle from the current ϑ_o. The distance from the current point to the desired is then calculated by taking the magnitude using the function hypot(Δx, Δy). With the distance in hands, the number of steps necessary to cover that distance is then calculated. Having the

values of angle and steps, these just need to be plugged in go2Angle($\Delta\vartheta$) and moveForward(steps, 200) respectively. In the end x_o, y_o and ϑ_o are updated.
In the while loop of the main function, a setup interface was created using the buttons and the LCD display where the user can choose to either insert a value of angles for the robot to spin to it or a position in the x-y plane for the robot to move to. This can been easily seen in Figure 1.

Testing the Program

The program was tested and developed in a linear manner. First the moveForward() function was created and tested, then go2Angle() had some issues with variable types where memory space was "over saved," making necessary to upgrade a type short to int. The next function to be implemented was go2Point() where some issues occurred with mistyping variable names that took some hours to figure out. When both go2Angle() and go2Point() were working with predefined values, the interface was created and properly adjusted to work in a satisfactory manner. After all of this was finished, we took some time to modularize the turn routines from go2Angle into sTurnLeft() and sTurnRight() to save memory space and to easily use in future projects.

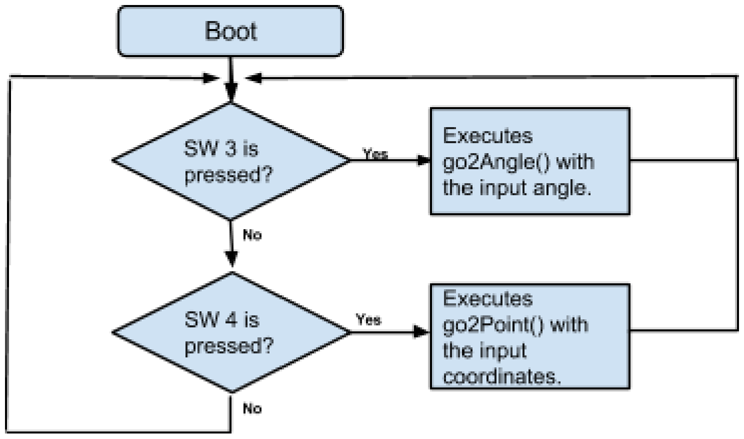
Boot
SW 3 is pressed?
Yes
Executes go2Angle() with the input angle.
No
SW 4 is pressed?
Yes
Executes go2Point() with the input coordinates.
No

Figure 1 - Robot program flowchart.

Data Collection and Interpretation

To test the quality of our robot, we tested a few target data points to see how the odometry of our robot was. There results can be seen below in Table 1.

Theoretical X (in), Y (in), and Θ (degrees)	Found X (in), Y (in), and Θ (degrees)
(12, 12, 45)	(5.5, 6.75, 50.82)
(0, 12, 90)	(1, 9, 83.65)
(-12, 0, 180)	(-9, -1, 186.34)
(0, -12, 270)	(-2.5, -7, 250.34)

Table 1 - The collected data from our test runs.

Discussion of Questions

To calculate the turn angle, we utilized previous knowledge from the last lab. We knew the amount of steps needed to turn 90 degrees. To get what angle we needed, we made a ratio of the desired angle over 90 degrees and multiplied that by the number of steps needed for 90 degrees.

The calculate the odometry to move to the goal position, we knew the distance each step travels. Therefor, by dividing the target distance over that distance per step, we get the number of steps our motors need to travel.

For the go to goal behavior, our distance traveled seemed to be consistently below the target distance. This is not a bad result to find, as this means that with minor tweaking, the error could be significantly accounted for.

For the go to angle behavior, our accuracy seemed to be consistently within 5 or so degrees of the target, with the exception of the last target angle. This off angle did not appear to be systematic, however, requiring an alternative to just tweaking to improve the accuracy.

To improve our accuracy, there are two manners. One way would be continue the tweek the robots steps till better odometry is achieved. This would be a drawn out and iterative process that could take a very long time and still may not produce much better results. The second way would be to add some external feedback to the robot. That would however mean adding sensors of some kind and integrating it into our motion loop.

We utilized the turn then go to goal approach. The pro to this approach is that it is simple to program the robot. Also, the robot can easily deal with large direction changes. The downside is that the robot cannot make progress while adjusting to the appropriate angle if that angle is a small one.

Conclusions

All in all, this lab wasn’t terribly difficult. We were able to complete all the tasks with minimal trouble. However, this lab did further emphasize the limits in accuracy that an open loop system has, as a sizable portion of this lab was spent not on writing the programs but on tweaking the values to attempt to get accurate reactions from the robot.

We look forward to working on closed loop systems in the future.