Lab 01
Getting to Know Your Robot – Let's Get Moving!

Read this entire lab procedure before starting the 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 robot. 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:

- Describe the primary components of the robot including the actuators, effectors, and sensors.
- Program the robot to move to a given angle or goal position
- Program the robot to move in a prescribed motion (circle, square, figure eight)
- Write properly commented, modular code
- Write a technical memo to report the results of an experiment

****

Equipment: Base Robot
Microcontroller (Arduino Mega 2560)
Masking Tape
Ruler
3 LEDs
Various Wires

Software: Arduino IDE:
https://www.arduino.cc/en/Main/Software

References: Arduino Reference, Getting Started
https://www.arduino.cc/en/Guide/HomePage
AccelStepper library
http://www.airspayce.com/mikem/arduino/AccelStepper/index.html
How to install libraries in Arduino software
https://www.arduino.cc/en/Guide/Libraries
A4988 Step Stick Driver
http://howtomechatronics.com/tutorials/arduino/how-to-control-stepper-motor-with-a4988-driver-and-arduino/

Other: MATLAB Support Package for Arduino
http://www.mathworks.com/help/supportpkg/arduino/examples.html
Simulink Support Package for Arduino
http://www.mathworks.com/help/supportpkg/arduinoio/examples.html
SimuLink and Arduino Getting Started Guide:
http://makerzone.mathworks.com/resources/install-support-for-arduino/?s_eid=PRP_5807
Alternative Arduino IDEs
http://playground.arduino.cc/Code/Eclipse
http://playground.arduino.cc/Main/DevelopmentTools
http://www.intorobotics.com/alternatives-standard-arduino-ide-one-choose/
https://kig.re/2014/08/02/arduino-ide-alternatives.html
https://learn.sparkfun.com/tutorials/alternative-arduino-interfaces

Pre-Lab:

- Read this entire lab procedure before the lab recitation.
- Read Olson paper, *A Primer on Odometry and Motor Control*.
- Review AccelStepper Library (see link in references)
- Write pseudocode for the functions: Go to Angle, Go to Goal, Circle, Figure Eight, Square

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. Thus, dead reckoning is most appropriate for short distances because of the error accumulation.

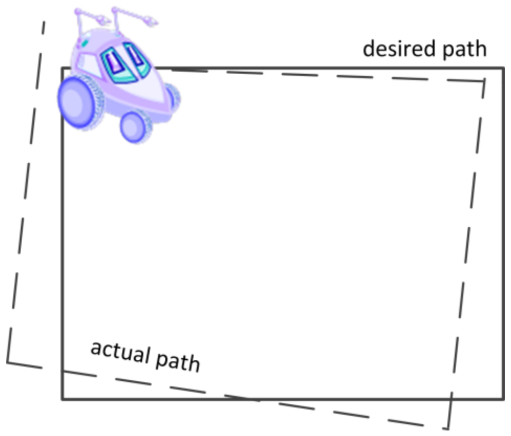

Figure 1: Robot Odometry Error

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 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. 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.

For accuracy, an encoder or some other feedback sensor should be used to periodically null out or reset the accumulation error. Note that when you have a robot with no sensor feedback (encoders), dead reckoning is the only method for estimating the robot's position. Without the encoders there is no way to correct the position.

In this lab, you will use odometry concepts to implement the go-to-angle and go-to-goal behaviors on the robot. Remember you will never be able to correct for **all** odometry error and must learn to implement the most ideal solution considering this variable. In order to use feedback control to implement these behaviors it is necessary to have sensors to measure the robot's actual position to compare to the desired position in order to calculate the error input for the controller. Figure 2 demonstrates the necessary feedback control system.

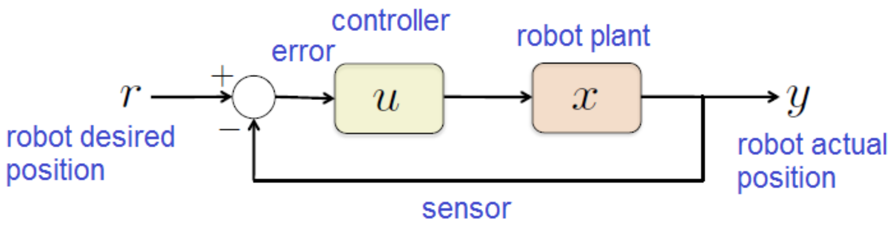

Figure 2: Robot Motion Feedback Control System

You can estimate the measured angle or goal by using velocity and time or number of steps to estimate position using forward kinematics. Figure 2 shows the dynamic model for implementing the Go-To-Goal behavior.

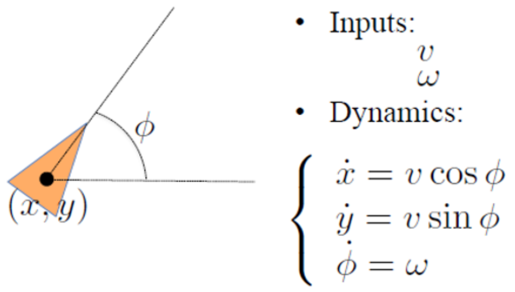

Figure 3: Unicycle robot dynamic model

In order to implement the Go-To-Goal behavior, the robot would calculate the angle to the goal point and use the Go-To-Angle behavior to turn first and then move forward to the goal. Therefore, the Go-To-Angle behavior should be implemented first. To calculate the desired turn angle, use the following formula.

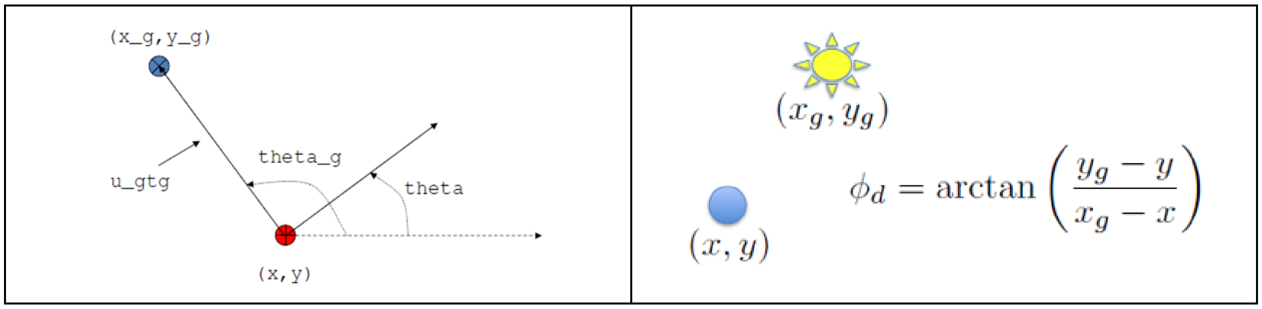

Figure 4: Go-To-Goal Notation

An alternative for implementing the Go-To-Goal behavior which may be a bit more difficult is to move the robot in a linear and angular motion at the same time as it converges on the goal point. In order to use this model, you will use the differential drive robot model to calculate the required motion given a constant linear velocity and variable angular velocity. This method would require the robot to incrementally calculate the vector to the goal position as it moves to converge on the position within some error. The Go-To-Goal behavior could be implemented by using a P controller to create the behavior, $\omega = K(\phi_d - \phi)$.

****

LAB PROCEDURE

****

Inventory

Your first task is to take an inventory of everything in your robot kit. The robot kit is shown in Figure 5, you must pack it back exactly like this before returning to the technician. You are responsible for returning the robot and all peripherals in the exact same condition that you received them in so if anything is missing, please notify your instructor or the ECE technician immediately. **Remember you are responsible to pay to replace any parts of the kit that are lost or damaged! You will receive an incomplete in this course and a hold on your business office account until this is done.**

- Robot
- Wall Adapter Power Supply
- USB Cable
- Energizer Battery Charger
- 2 Rechargeable Batteries
- Pocket Screwdriver
- Game controller (Wiimote or PlayStation)
- Arduino Mega 2560 or Arduino Uno Microcontroller
- Flashlight
- 2 Photoresistors and resistors
- 6 IR and sonar sensors (some combination)
- 2 nrF24L01 Wireless Transceivers
- IR Remote
- TSOP IR Receiver

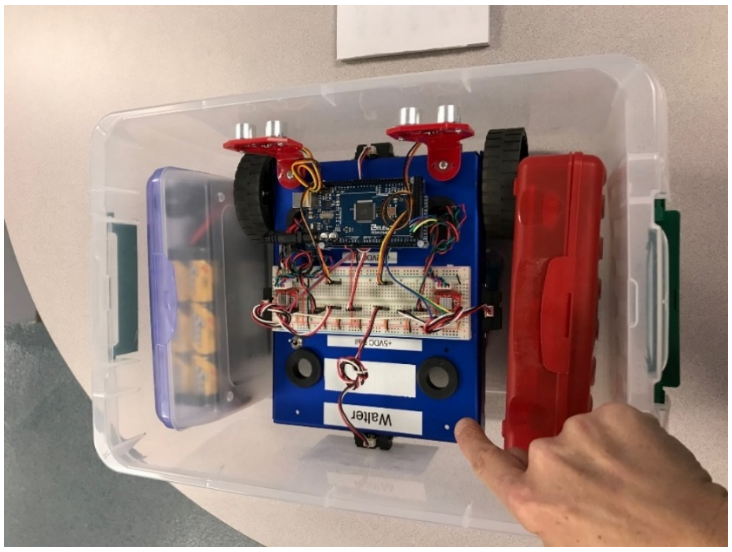

Figure 5: Robot Kit

The robot is shown in Figure 6, some of the key elements are highlighted included the stepper motor driver, Arduino MEGA 2560 microcontroller board, IR sensors, motor power, and sensor power busses on the breadboard.

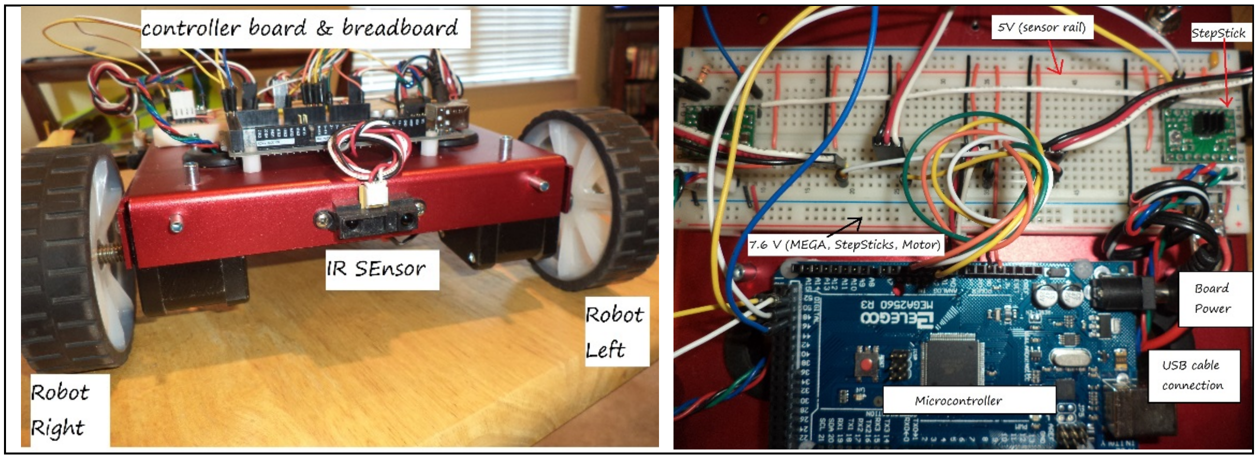

Figure 6: Robot Hardware

Part 1 – Software Installation

1. Navigate to the Arduino website, https://www.arduino.cc/, and click on Download.
2. Download the latest version of the Arduino IDE to your computer. Eventually you may want to change to a better IDE with a better debugger and I have provided some links to some options in the References above.
3. Go to the Moodle course website and download the Lab 01 files. The files are an Arduino sketch to help get the robot moving and the *AccelStepper* library.
4. Install the *AccelStepper* library zip folder by following the directions at the following link: https://www.arduino.cc/en/Guide/Libraries#toc2
5. Review the documentation and examples in the *AccelStepper* library folder to understand what it does. It is most important to understand the functions that
6. Open the *"RobotIntro.ino"* file and save and rename the file to "*YourRobotName-Lab01.ino*".
7. *"RobotIntro.ino"* is the beginning sample code to help you get the robot moving. It also shows the proper way to comment your code. You will need to modify the comments, add additional comments, create new functions and add and modify the code based upon the lab requirements.
8. Use the USB cord to connect the robot microcontroller to your computer and wait for the driver to install. If the driver does not install, go to the following link: https://www.arduino.cc/en/Guide/ArduinoMega2560
9. You can view the documentation about the library classes, functions, methods, and procedures in the folder where the library was installed.
10. In the Arduino IDE click Tools->Board->Arduino/Genuino Mega or Mega 2560
11. In the Arduino IDE, click Tools->Port->COM# (where board is connected, see Device Manager)

Part 2– Robot Motion

1. **WATCH THE ROBOT! DON'T LET IT DRIVE OF THE TABLE! USE A TEST STAND! A BOOK OR BOX!**
2. Remember to always insert a 5 second delay at the beginning of all your code to give you time to unplug the USB cable and put the robot on the floor. **IF THE ROBOT DOES NOT MOVE, PAY ATTENTION TO THE STEPPER PIN NUMBERS they may be different from your wiring.**
3. The robot uses the A4988 Stepper Motor Driver StepStick to drive the stepper motors, view the following link for more details on the code. http://howtomechatronics.com/tutorials/arduino/how-to-control-stepper-motor-with-a4988-driver-and-arduino/
4. In Arduino IDE, click Sketch->Verify/Compile (CTRL-R) or click check icon
5. In Arduino IDE, click Sketch->Upload (CTRL-U) or click right arrow icon

6. Use the toggle switch to turn on power to the robot motors. The switch turns on power to the board and robot motors when the switch is back toward the label and tail wheel.
7. **WATCH THE ROBOT! DON'T LET IT DRIVE OF THE TABLE! USE A TEST STAND! A BOOK OR BOX! IF THE ROBOT DOES NOT MOVE, PAY ATTENTION TO THE STEPPER PIN NUMBERS they may be different from your wiring.**
8. The robot should continue to move forward and backward one full rotation until you turn the power off.
9. Change the *stepTime* variable to a different value between 300 and 4000 and observe how this changes the robot behavior. ***Comment on this observation in your lab memo.***
10. Now comment out the first function, *move1()*, in the loop() function and uncomment the next second function, *move2()*, in the main loop.
11. Compile and upload the program to see the difference in the code and robot behavior. ***Comment on this observation in your lab memo.***
12. Next change the library functions to some of the other library function options listed in the comments and make observations about the change in robot behavior. The function options are *move()*, *moveTo()*, *stop()*, *run()*, *runSpeed()*, *runToPosition*I(), *runToNewPosition()* and *runSpeedToPosition()*. ***Comment on these observations in your lab memo.***
13. Now comment out the move2() function and uncomment the *move3()* function. Repeat steps 11 and 12. ***Comment on your observations in your lab memo.***
14. Next repeat steps 11 and 12 for the *move4()* and *move5()* functions.
15. Finally, this example code with header comment, block and in line comments and modularity with functions is how you should write your programs. It is very important to create functions and re-use code as much as possible for subsequent labs. In addition, if your code is not well commented it is impossible for you to understand later and for me to follow when grading it. Your code grade will be determined by how well your code is organized, commented and modularity.

Part 3 – Motion Functions

In this section you will create some frequently used robot motion functions. I have provided the template for these functions in the "*RobotIntro.ino*" file. You have the flexibility of modifying how the function works and what input it takes but the functions must be there in your final submission. You will turn on a different LED for each function in order to get in the habit of using LEDs to indicate the robot state.

1. Wire a red, green, and yellow LED to pins 14, 15, and 16 on the microcontroller. Each LED should be in series with a 220Ω resistor. The short leg or flag edge of the cap must go to ground.
2. A robot <u>forward</u> is when the wheels move in the same direction at same speed. Write a *forward()* function for the robot where the input is the distance or forward movement time or speed. You have the flexibility of how you want to write the function and can modify the template to meet your needs. Test your robot, how many steps does it take to move forward 2 feet? You can measure the diameter of your wheel as a starting point to estimate this value. ***State this value in your lab memo.***
3. A robot <u>reverse</u> is when the wheels move in the same direction at same speed. Write a *reverse()* function for the robot where the input is the distance or reverse time or speed. You have the flexibility of how you want to write the function and can modify the template to meet your needs.
4. A robot <u>stop</u> is obviously when the robot stops moving. Write a *stop()* function.
5. A robot <u>spin</u> is when the wheels move in opposite directions at the same speed. Write a *spin()* function for the robot where the input is the direction of the spin or spin time or speed. You have the flexibility of how you want to write the function and can modify the template to meet your needs. Test your robot, how many steps does it take to spin the robot 90 degrees? ***State this value in your lab memo.***

6. A robot pivot is when one wheel is stationary and the other wheel moves forward or backward. Write a *pivot()* function for the robot where the input is either the direction or which wheel should move forward or backward or pivot time or speed. You have the flexibility of how you want to write the function and can modify the template to meet your needs. Test your robot, how many steps does it take to pivot the robot 90 degrees? ***State this value in your lab memo.***
7. A robot turn is when the wheels move in the same direction at different speeds. Write a *turn()* function for the robot where the input is the direction of the turn or turn time or velocity differential. You have the flexibility of how you want to write the function and can modify the template to meet your needs.
8. Figure 7 gives an example of the robot motions.

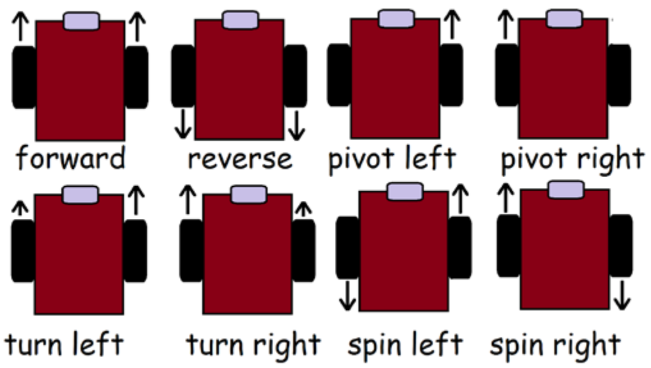

Figure 7: Robot Motion Summary

Hint: Use runSpeed() or runSpeedToPosition() for the turn function. You will need to call them in a loop() to continuous increment the steps. You will also need to setSpeed() right before the run function.

Part 4 – Circle and Figure 8

1. Write a *"moveCircle()"* function to move the robot in a circle with the diameter and direction as the input (see Figure 8). You should call the *moveCircle()* function from the *loop()* function. Turn on the RED LED when the robot is moving in a circle.
2. Try to adjust the code so that the robot will start and end at the same point. ***Comment on the results of this task in your memo.***

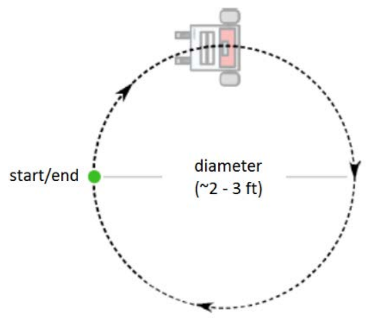

Figure 8: Circle Robot Motion

3. Next, create a *"moveFigure8()"* function to move the robot in a figure eight using two circles (see Figure 9).
4. Try to adjust the code so that the robot passes through the same center point of the figure 8 each time. You should call the *moveCircle()* function twice to create the *moveFigure8()* function. This means you may need to modify the *moveCircle()* function pass it a direction variable. You should call the *moveFigure8()* function from the *loop()* function. Turn on the RED LED and YELLOW LED when the robot is moving in a Figure 9.

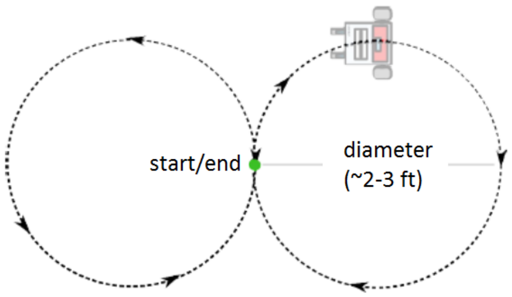

Figure 9: Figure Eight Robot Motion

Part 5 – Encoders

1. Your robot has been outfitted with encoders to enable feedback to correct for odometry error between the desired number of steps and actual number of steps. This can be used to move the robot to a given angle or position.
2. Open the "*RobotEncoders.ino*" file, read the code and comments to fully understand what it does.
3. Compile and upload the program to the robot to confirm that the program works the way you think it does.
4. **WATCH THE ROBOT! DON'T LET IT DRIVE OF THE TABLE! USE A TEST STAND! A BOOK OR BOX!**
5. In the main *loop()* function vary the move direction and amount to determine how the encoder values change. Record the average encoder ticks for a quarter, half, full, and two rotations. ***Comment on the results in your lab menu.***

Part 6 – Go To Angle Behavior

1. In the program for part 5, there is a template for a go to angle behavior, create a method to implement the Go-To-Angle behavior.
2. The user will input a given angle in degrees and the robot should turn the given angle. The robot should be able to take negative and positive angles to turn clockwise and counterclockwise. You may have to add a variable to the function to indicate whether the robot should turn left or right. Remember to use the right hand rule, positive angles turn the robot left.
3. Take measurements to estimate the accuracy of the Go-To-Angle behavior. ***Comment on the results in your lab memo.***
4. Your code should be modular with the *goToAngle()* written as a function that is called from the loop() function.
5. The reference frame for the robot is shown in Figure 10, where the robot's start position is (x,y,θ) (0,0,0). Note that the z axis is coming up out of the robot centered between the two wheels. The x axis is straight ahead and the y-axis is to the left. Positive turns are counterclockwise and negative turns are clockwise. Turn on the GREEN LED when the robot is executing the Go To Angle behavior.

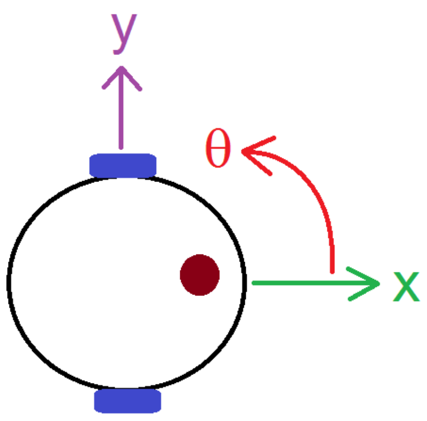

Figure 10: Robot Relative Reference Frame

Part 7 – Go To Goal Behavior

1. Create a method to implement the Go-To-Goal behavior.
2. The robot should move to the given goal position and stop within a certain error.
3. Take measurements to estimate the accuracy of the Go-To-Goal behavior.
4. Your code should be modular with the *goToGoal()* written as a function that is called from the *loop()* function when the button is pressed. The *goToGoal()* function should call the *goToAngle()* function first and then the robot should move forward as required.
5. Use the reference frame shown in Figure 8 for the *goToGoal()* behavior.
6. Turn on the GREEN LED and YELLOW LED when the robot executes the Go To Goal behavior.

Part 8 – Square Path

Now that you have functions to create robot motion, you will create a function to move the robot in a square.

1. Write a program to move the robot in a square path with sides specified in the function (see Figure 11).
2. You may want to compare encoder ticks and desired steps to stop the robot to make the square more accurate to try to correct for odometry. Alternately you could use encoder ticks to move robot. You can implement a proportional controller that compares the desired steps and encoder ticks in order to correct for odometry in the square function.

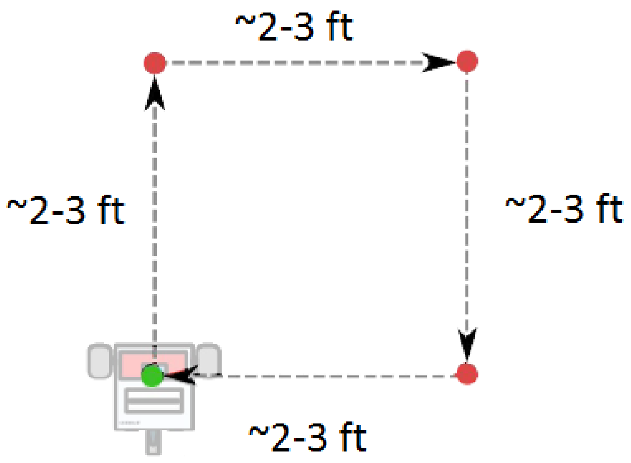

Figure 11: Square Robot Motion

3. The *moveSquare()* function should be called from the *loop()* function and the input can be the length of the sides. Turn on the RED, GREEN, and YELLOW LED when the robot executes the Square behavior.

Submission Requirements:

Software Design Plan

For each lab you will submit a software design plan which may be in the form of pseudocode, a flowchart, state machine or subsumption architecture. You will show this plan to the instructor during class, the plan will be graded and you will get feedback on implementation before designing the full system.

Demonstration:

Bring your robot fully charged to class every day! Plug it in overnight.

During the demonstration you will be graded on the following requirements:

- Ability to describe how the code works (commenting, organization, modularity)
- Robot move functions (forward, reverse, turn, pivot, spin, stop)
- Robot moves in a circle (RED LED)
- Robot moves in a figure eight (RED, YELLOW LED)
- Robot can move to angle given inputs such as -60° or 120° (GREEN LED)
- Robot can move to goal given inputs such as (12 inches, 6 inches) or (0 ft, 3 ft) (GREEN, YELLOW LED)
- Robot moves in a square with given side length (inches, feet) (RED, GREEN, YELLOW LED)

Code:

Use the following guidelines for creating your code.

- Code has a header comment with
- Program Name
    - Author Names
    - Date Created
    - Overall program description
    - Summary of key functions created with a description of each
- Code is modular with multiple functions called from the loop function. Functions have logical names that describe their functionality.
- Code is organized in a logical fashion with global and local variables with intuitive names.
- Code has block comments to introduce and describe each function and sufficient in line comments to describe how key parts of the code work.
- All functions and variables should have logical names that describe what they do or the value they hold. There should be no magic numbers or numeric values not related to a constant or variable to make it clear what it does.
- In general, someone unfamiliar with your project should be able to read your code and understand the functionality based upon your comments and naming convention.

Memo Guidelines:

Please use the following checklist to insure that your memo meets the basic guidelines. See sample lab memos in the assignment folder.

✓ Format
- Begins with Date, To , From, Subject
- Font no larger than 12 point font
- Spacing no larger than double space
- Written as a combination of sentences or paragraphs and only bulleted list, if necessary
- No longer than three pages of text

✓ Writing
- Memo is organized in a logical order
- Writing is direct, concise and to the point
- Written in first person from lab partners
- Correct grammar, no spelling errors

✓ Content
- Starts with a statement of purpose, objective, goals
- Summarizes at a high level the procedure implemented
- Discusses the strategy or pseudocode for implementing any functions created
    - includes pseudocode, flow chart, state diagram, or control architecture in the appendix
- Discusses the tests and methods performed and any comments or observatoins

- States the results and or data tables including error analysis, if required
- Shows any required plots or graphs, if required
- Answers all questions posed in the lab procedure
- Clear statement of conclusions

Additional questions to answer in the lab memo

1. What is the diameter and circumference of the robot wheels?
2. How many inches or feet can it move in a quarter, half, full, and two rotations?
3. How many encoder ticks do you get in a quarter, half, full and two rotations?
4. Compare the accuracy between encoder ticks and steps for a given distance (i.e. 2 feet).
5. Describe how you can use a proportional controller to move the robot a given distance and correct for odometry error.
6. How did you calculate the turn angle for the robot? Explain and show formula in memo.
7. What type of accuracy/error did you have in the go-to-angle behavior?
8. How did you calculate the move distance given the x and y position? Explain and show formula in memo.
9. What type of accuracy/error did you have in the go-to-goal behavior?
10. What could you do to improve the accuracy of the behaviors?
11. Did your team use the turn then forward approach for go-to-goal or move and turn at the same time? If so, what were the pros and cons of using your approach versus the other one?
12. What are some sources of the odometry error?
13. How could you correct for this error?
14. How could you improve the three motions (move*Square, moveCircle, moveFigure8*) functions?
15. Describe the method, pseudocode, flow chart, or state diagram.

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

Upload Details:

You must submit your properly commented Sketch code & memo to the Moodle DropBox by midnight on Sunday after the demonstration. Put the documents in a zip folder and then upload it to Moodle. Check the course calendar for the lab demonstration due date.