

**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/worksheet to report the results of an experiment

Equipment: Base Robot

Masking Tape

Ruler or Tape Measure

3 LEDs and 3 220Ω resistors

Various Wires

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

PlatformIO IDE: <https://docs.platformio.org/en/stable//integration/ide/vscode.html>

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



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
- Simulate Square and Figure Eight in Tinkercad or Dawson Virtual Robot Lab
- Upload software design plan to Gradescope and screen capture to Moodle DropBox.

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.

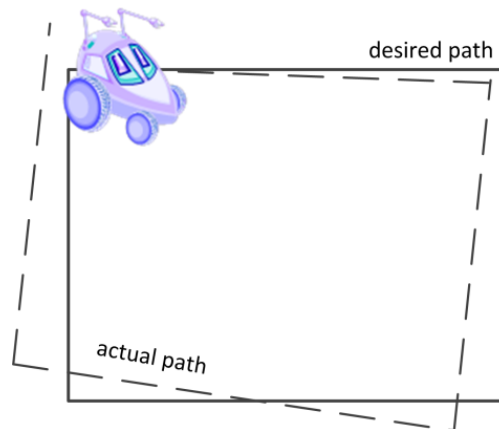


Figure 1: Robot Odometry Error

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 a start point. The MegaBot 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 MegaBot platform uses stepper motors that are designed with 200 steps per wheel revolution. That means each wheel in the MegaBot 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 proportional control, use the encoder to correct odometry error such as when the wheels slip or overshoot a goal position.

An inertia measurement unit (IMU) is used to get the position of a robot in three-dimensional space. The MPU6050 IMU includes an accelerometer and gyroscope. It is a six-axis IMU because it has six degrees of freedom. The output is three accelerations from the accelerometer and three angular velocities from the gyroscope. The chip uses I2C (inter-integrated circuit) protocol for communication.

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.

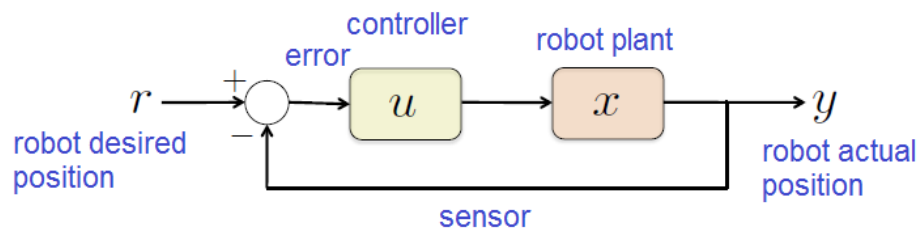


Figure 2: Robot Motion Feedback Control System

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.

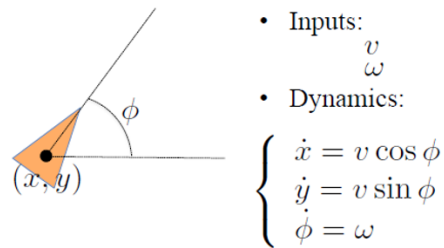


Figure 3: Unicycle robot dynamic model

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.

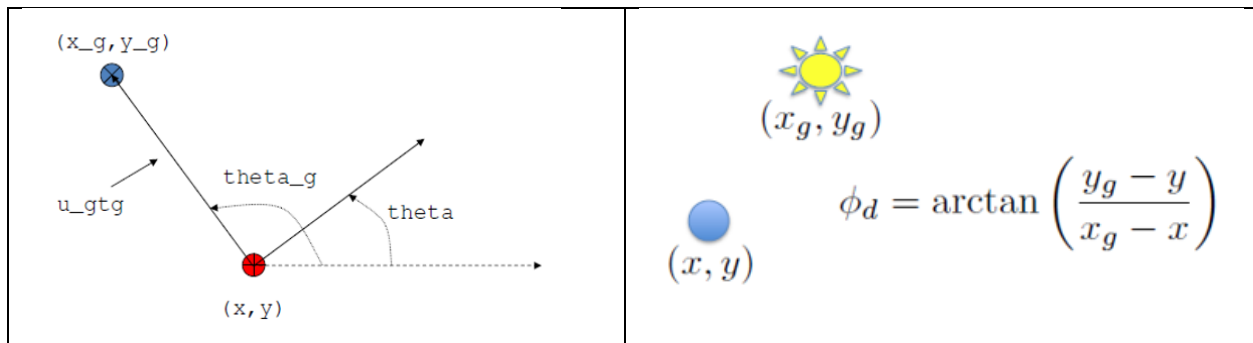


Figure 4: Go-To-Goal Notation

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, $\omega = K(\phi_d - \phi)$.

Bluetooth module for wireless communication

To establish wireless communication between the robot and your laptop. Wireless communication will be used to send and receive data between the robot and laptop and can also be used to program the robot. It will also be used to command the robot to drive remotely (teleoperation), send path plans to the robot for path following (autonomously), send robot locations to the laptop (localization), send robot worlds to and from the laptop (mapping), send robot updates to the laptop (SLAM).

There are several types of wireless communication including RF transceiver, Wi-Fi, Xbee, as well as Bluetooth. In this lab we will use Bluetooth communication with the DSD TECH HC-05



Bluetooth Serial Pass-through Module Wireless Serial Communication with Button for Arduino. It works within 100 meters of the robot at a 2.4 GHz frequency and a maximum data rate of 1Mb/s.

The Bluetooth module operates at 3.3V and has a 3.3V internal regulator so it can be connected to the 5V power line. Figure 5 shows the pinout for the HC-05 Bluetooth Module.

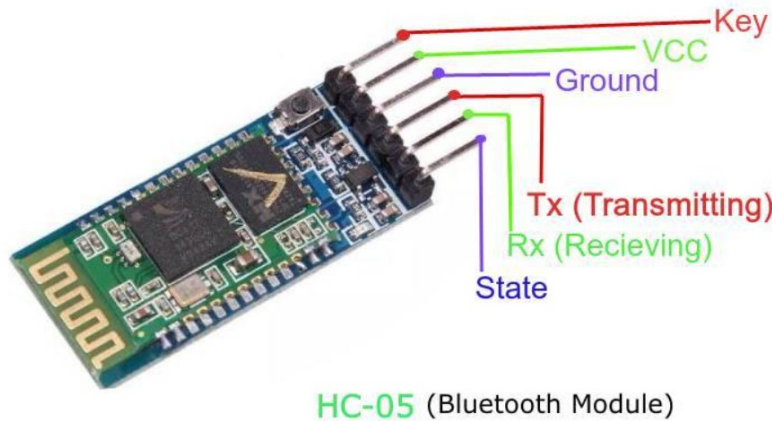


Figure 5: HC-05 Bluetooth Module pinout

The Enable pin sets the module to Data mode if LOW (default) or AT command mode if HIGH. VCC is connected to the +5V power supply. Ground is connected to the ground on the robot. TX (transmitter) is the pin that transmits the received data serially. The RX (receiver) sends data serially over Bluetooth. The state pin is used to check if the Bluetooth module is working properly. The red LED indicates the connection status and slows down to 2 seconds when connected. To send and receive data to the Bluetooth module, connect your computer to a Bluetooth device, find the HC-05 module and use the pin number is '1234'.

To program the Arduino over a Bluetooth serial connection, to change the default serial baud rate on the HC-05 Bluetooth module to 115200 bp/s. This requires putting the module in AT data command mode. In this mode the onboard LED will change to a 2s blink.

To program the Arduino over a Bluetooth serial connection, it is necessary to trigger a reboot on the microcontroller. The Arduino uses the DTR (Data Terminal Relay) signal to reset it by pulling the reset line to ground temporarily. To make a temporary connection, it is necessary to insert a capacitor in the circuit. This will block a continuous signal and allow for a quick transition. This can be done with the state pin on the HC-05. Build the circuit as shown in Figure 6 to program your robot using the Bluetooth module.

Run the Arduino IDE or Visual Studio in administrator mode. There are two COM ports connected to the Bluetooth module. The first is for receiving data and the second is for sending data. Select the second COM port to program the Arduino. Verify and upload the code to the COM port as you typically do. See more information about this process at the following link.

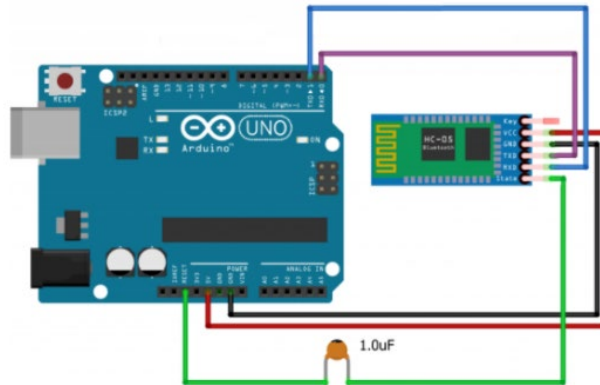


Figure 6: HC-5 Bluetooth Module for programming robot

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 | • Tape Measure |
| • Wall Adapter Power Supply | • 2 Photoresistors and resistors |
| • USB Cable | • 6 IR and sonar sensors (some combination) |
| • Energizer Battery Charger | • HC-05 Bluetooth |
| • 2 Rechargeable Batteries | • Left and right encoders on motor |
| • Pocket Screwdriver | • MPU6050 Module 3 Axis Analog Gyro Sensors+ 3 Axis Accelerometer Module |
| • Game controller (Wiimote or PlayStation) | |
| • Arduino Mega 2560 or Arduino Uno Microcontroller | |
| • Flashlight | |

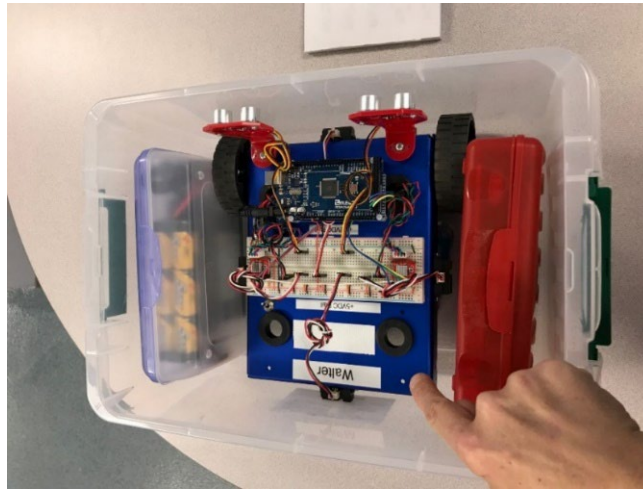


Figure 5: Robot Kit

The robot is shown in Figure 6 highlights some of the key elements on the robot including the stepper motor driver, Arduino MEGA 2560 microcontroller board, IR sensors, motor power, and sensor power busses on the breadboard. You will add more peripherals as the quarter progresses.

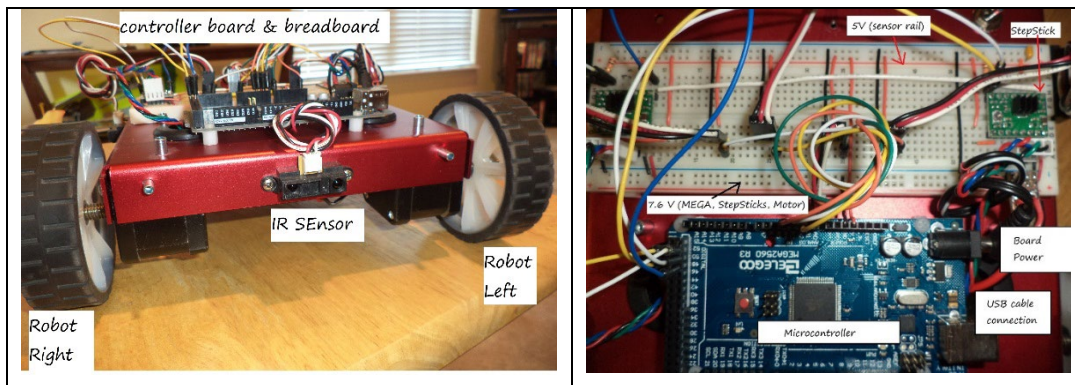


Figure 6: Robot Hardware

Part 1 – Software Installation

In this section you will install the software to move your robot.

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. My recommendation is the PlatformIO IDE available [here](#).
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 following libraries in the Arduino IDE by following the directions [here](#). Install the following libraries in the PlatformIO IDE by following the directions [here](#).
 - Arduino
 - AccelStepper
 - MultiStepper
 - AdaFruit_MPU6050
 - SoftwareSerial
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 use blocking or reset speed to zero. *Look at the Blocking, Bounce, ConstantSpeed, MultipleSteppers, MultiStepper, Overshoot, Random, ProportionalControl, Quickstop, and Random Examples.*
6. Download “**RobotIntro.ino**” and read the comments at the top. Compare the motor wiring to your robot and change it to match if it does not (see Table 1).

Table 1: Motor Wiring

	Left Motor	Right Motor
Stepper enable pin	48	48
Step pin	52	50
Direction pin	53	51

7. This file 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. Finally, rename and save this file as “**YourRobotName-Lab01.ino**”.
9. 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>
10. You can view the documentation about the library classes, functions, methods, and procedures in the folder where the library was installed.
11. In the Arduino IDE click Tools->Board->Arduino/Genuino Mega or Mega 2560
12. In the Arduino IDE, click Tools->Port->COM# (where board is connected, see Device Manager)
13. Review the PlatformIO documentation for the proper way to set Project Settings.



Part 2– Robot Motion

In this section, you will run the program to move your robot.

1. 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 or get it on the test stand. You should also use LEDs to illuminate to indicate when the robot is about to move.
2. 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/>
3. In Arduino IDE, click Sketch->Verify/Compile (CTRL-R) or click check icon
4. In Arduino IDE, click Sketch->Upload (CTRL-U) or click right arrow icon
5. 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.
6. WATCH THE ROBOT! DON'T LET IT DRIVE OFF THE TABLE! USE A TEST STAND! A BOOK OR BOX! IF
7. The robot should continue to move forward and backward one full rotation until you turn the power off.
8. 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.***
9. Now comment out the first function, *move1()*, in the loop() function and uncomment the next second function, *move2()*, in the main loop.
10. Compile and upload the program to see the difference in the code and robot behavior. ***Comment on this observation in your lab memo.***
11. 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()*, *runToPositionI()*, *runToNewPosition()* and *runSpeedToPosition()*. ***Comment on these observations in your lab memo.***
12. Now comment out the *move2()* function and uncomment the *move3()* function. Repeat steps 11 and 12. ***Comment on your observations in your lab memo.***
13. Next repeat steps 11 and 12 for the *move4()* and *move5()* functions.
14. 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 – AccelStepper Library

In this section, you will examine the different functions in the AccelStepper library that you can use to move your robot.

1. WATCH THE ROBOT! DON'T LET IT DRIVE OFF THE TABLE! USE A TEST STAND! A BOOK OR BOX!
2. Run each of the example programs from the AccelStepper library to understand what the various functions do. These examples are in the AccelStepper library zip folder.
3. You can access them from the IDE by click File->Examples->AccelStepper->
 - Blocking
 - Bounce
 - ConstantSpeed
 - MultipleSteppers
 - MultiStepper
 - Overshoot
 - ProportionalControl
 - Quickstop
 - Random
4. Comment in your lab memo on what you think each of the functions do.

Part 4 – Encoders

In this section you will confirm that the encoders attached to the left and right wheel are working correctly. I have provided the interrupt for the encoders in the ***“RobotIntro.ino”*** file.

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. Confirm that left encoder is connected to pin 18 and the right encoder is connected to pin 19. Also confirm that voltage and ground connections.
3. Upload the program to the robot and open the serial monitor to examine the encoder ticks as the robot moves.
4. 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. How consistent are the values between movements? How consistent are the values between the left and right motor? ***Comment on the results in your lab menu.***

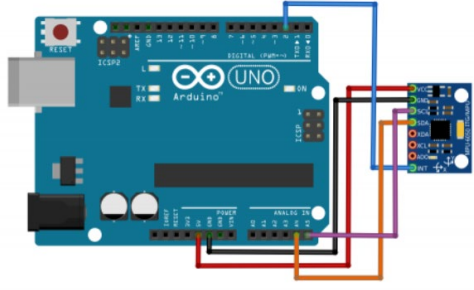
Part 5 – Gyro and Accelerometer

In this section you will confirm that the gyroscope and accelerometer are attached and working properly. I have provided code for reading the values in the ***“RobotIntro.ino”*** file.



1. Use wires to connect the IMU to your robot as shown in Table 2.
2. Install the Arduino MPU 6050 and I2C libraries if they are not already installed. Read more details [here](#).
3. Run the code and open the serial monitor to confirm that the IMU is working correctly. Move the robot around and observe how the data changes. **Comment on the results in your lab memo.**

Table 2: IMU Wiring

IMU pin	Arduino Pin	 <p>Arduino MPU 6050 connections</p>
VCC	5V	
GND	GND	
INT	2(interrupt pin 0)	
SDA	20 (SDA)	
SCL	21 (SCL)	

Part 6 – 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 6, 7, and 8 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 forward 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 reverse 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 stop is obviously when the robot stops moving. Write a *stop()* function.



5. A robot spin 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.

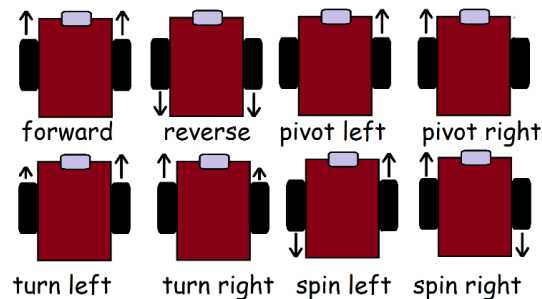


Figure 7: Robot Motion Summary

Hint: Use *runSpeed()* or *runSpeedToPosition()* for the turn function. You will need to call them in a *loop()* to continuously increment the steps. You will also need to *setSpeed()* right before the run function. If *setSpeed()* does not seem to be working then use *setMaxSpeed()*.

Part 7 – 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. **The circle diameter is measured from the center of the robot halfway between the wheels.**
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.**

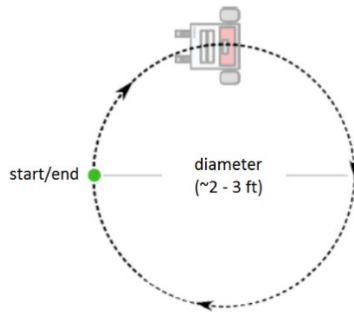


Figure 8: Circle Robot Motion

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 8.

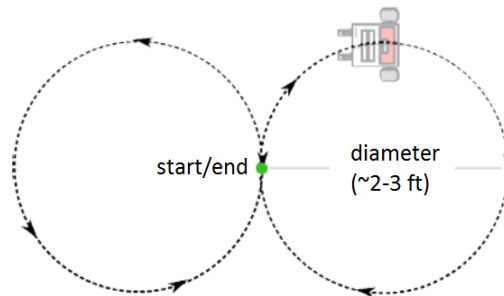


Figure 9: Figure Eight Robot Motion

Part 8 – 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. *Use the $\text{atan2}(y,x)$ function to put your angles in the correct quadrant.*
2. The user will input a given angle in degrees and the robot should spin or pivot to the given angle. The robot should be able to take negative and positive angles to move clockwise or counterclockwise. You may have to add a variable to the function to indicate whether the robot should move left or right. Remember to use the right hand rule, positive angles move 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.



- 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 angles are counterclockwise and negative angles are clockwise. Turn on the GREEN LED when the robot is executing the Go To Angle behavior.

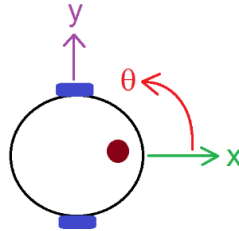


Figure 10: Robot Relative Reference Frame

Part 9 – Go To Goal Behavior

- Create a method to implement the Go-To-Goal behavior.
- The robot should move to the given goal position and stop within a certain error.
- Take measurements to estimate the accuracy of the Go-To-Goal behavior.
- 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 could use the *goToAngle()* and *goToGoal* functions or some of the other primitive move functions.
- Use the reference frame shown in Figure 8 for the *goToGoal()* behavior.
- Turn on the GREEN LED and YELLOW LED when the robot executes the Go To Goal behavior.

Part 10 – Square Path

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

- Write a program to move the robot in a square path with sides specified in the function (see Figure 11).
- 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.

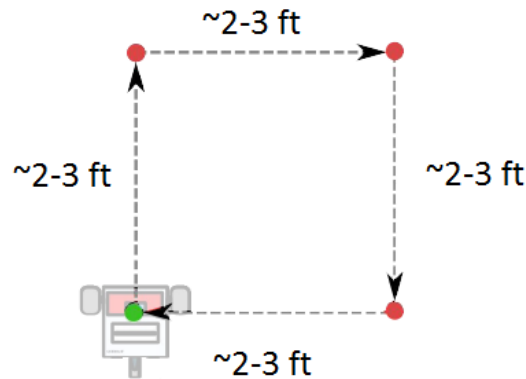


Figure 11: Square Robot Motion

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.

Part 11 – Wireless Communication (OPTIONAL, must be done by Lab 2)

As the quarter progresses, it will be important to have a way to send commands to and receive data from the robot. In this section, you will set up wireless communication for your robot.

1. There are two ways to achieve this goal, the first is Bluetooth communication, which is preferable. We will try this method first and if it does not work you can then use the wireless transceiver with the Arduino Uno attached to your laptop. See Dr. Berry for wireless transceiver instructions after you have exhausted all options to get Bluetooth working.
2. Install “SoftwareSerial.h” library
3. Connect VCC to 5V, GND to ground, TX to pin 10, and RX to pin 11.
4. Use Bluetooth on your computer to connect to the “DSD Tech HC-05” module. The password for the Bluetooth module is ‘1234’ and the baud rate is ‘9600’.
5. When the Bluetooth and laptop are paired, the LED on the module switches from quickly flashing to a double blink every two seconds. Check the device manager to confirm which COM port is the Standard Serial Over Bluetooth Link. If there are two it is most likely the first one.
6. If you open up the COM port where the Arduino Mega is connected, you should see “Goodnight Moon”, if you open up the Bluetooth COM port, you should see, “Hello, World?”.
7. Then if you type data in either the Bluetooth or Robot com port you should see it mirrored on the other one to indicate Bluetooth communication is working.
8. Now that you have bidirectional communication working, download “HC05.ino” and a serial port monitor and try to blink an LED on robot without being tethered to laptop.

**Part 12 – Wireless Programming (OPTIONAL, Your Choice to Implement, Can Be Buggy)**

1. See more resources at [here](#), [here](#), [here](#), [here](#), [here](#), [here](#), and [here](#).
2. Next, figure out a way to wirelessly read encoder data from the robot and print it to the serial monitor.
3. Rewire Bluetooth module by connecting GND to GND, RX to pin 11 and TX to pin 10 (**Do not connect VCC yet!**)
4. Hold down button on Bluetooth module and connect VCC to 5V, then release the button. The blink rate on the LED should slow down to 2s,
5. Change the baud rate on the Bluetooth to 115200 bps.
6. See details on this process [here](#).
7. Rewire Bluetooth module by connecting GND to GND, RX to pin 11 and TX to pin 10 (**Do not connect VCC yet!**)
8. Hold down button on Bluetooth module and connect VCC to 5V, then release the button. The blink rate on the LED should slow down to 2s,
9. Upload the RobotIntro.ino code in the resources folder to the microcontroller.
10. Change serial monitor to both NL & CR to send AT commands to Bluetooth module
11. Type 'AT' and click enter. Confirm you receive 'OK' to make sure the module is communicating.
12. Type 'AT+UART=115200,0,0' and click enter. Confirm you receive 'OK' to confirm the baud rate has changed. Do the same for all the following commands:
 - AT+ROLE=0
 - AT+POLAR=1,0
 - AT+INIT
13. To change name use 'AT+NAME=<PARAM>'.
14. To change password use 'AT+PSWD=<PARAM>'.
15. See all commands at the following [link](#).

Upload a program to the Arduino using the Bluetooth module

1. Remove all connections and wire the Bluetooth module to your robot as shown in Figure 2. (RX to TX pin 1, TX to RX pin 0, VCC to 5V, GND to GND, STATE to 1 μ F capacitor to RESET pin on Arduino)
2. Open the Blink code at File → Examples → Basics → Blink.ino
3. Select the second COM port for the Bluetooth module and upload to the Arduino.
4. After this is working correctly move on.

**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.

Additional questions to answer in the lab memo (all responses should be in the results section, include the question with your answer)

1. What is the diameter and circumference of the robot wheels? How did you use these in a formula to design the circle or turn function?
2. How many inches or feet can it move in a quarter, half, full, and two rotations?
3. Compare the accuracy between encoder ticks and steps for a given distance (i.e. 2 feet). How many ticks/step or steps/tick?
4. Describe how you can use a proportional controller to move the robot a given distance and correct for odometry error. Hint: You will need sensor feedback where the input is steps or distance.
5. How did you calculate the turn angle for the robot? Explain and show formula in memo.
6. What type of accuracy/error did you have in the go-to-angle behavior?
7. How did you calculate the move distance given the x and y position? Explain and show formula in memo.
8. What type of accuracy/error did you have in the go-to-goal behavior?
9. What could you do to improve the accuracy of the behaviors?
10. 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?
11. What are some sources of the odometry error?
12. How could you correct for this error?
13. How could you improve the three motions (*moveSquare*, *moveCircle*, *moveFigure8*) functions?
14. Describe the method, pseudocode, flow chart, or state diagram. Discuss the reality with the theory and software design plan. What was similar? What was different?
15. What is your robot's name? Write 2 or 3 sentences to describe how your robot's name relates to some historical figure, robot, or theory in robotics. Make sure to use proper citations and referencing.



Grading Rubric:

The lab is worth a total of 40 (prelab 5 pts, code, 10 pts, demo 10 pts, worksheet 15 pts). points and is graded by the following rubric. The grading rubrics is shown on the following table.

Points	Demonstration	Code	Memo
Excellent 100%	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
Average 75%	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
Poor 50%	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
Missing	Meets none of the design specifications or not submitted	Not submitted	Not submitted

Upload Details:

You must submit you properly commented code to Moodle DropBox and lab memo (worksheet) to GradeScope by **11:59 pm on Sunday**. Your code should be modular with functions and classes in order to make it more readable. You should use wireless communication to command the robot and indicate the robot state during program execution.