Lab 02

Getting to Know Your Robot: Locomotion and Odometry

Read this entire lab procedure before starting the lab.

****

Purpose: The purpose of this lab is to get the robot moving and to examine problems with raw odometry.

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

- Describe the primary components of the Arduino robot including the actuators, effectors, and sensors.
- Program the robot to move to a given angle
- Program the robot to move to a given goal position
- Program the Arduino by creating a sketch 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: Arduino Robot

Masking Tape

Ruler

References:

Arduino Robot Getting Started:	http://arduino.cc/en/Guide/Robot
Arduino Robot Boards:	https://www.arduino.cc/en/Main/Robot
Arduino Language Reference:	http://arduino.cc/en/Reference/HomePage
Arduino Robot Library:	http://arduino.cc/en/Reference/RobotLibrary

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.

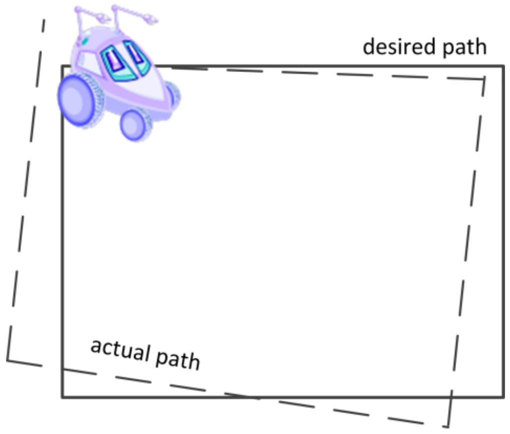


Figure 1: Robot Odometry Error

For accuracy, an encoder or some other feedback sensor should be used to periodically null out or reset the accumulation error. However, when you have a robot with no sensor feedback (encoders) this is the only method for estimating the robot's position. Without the encoders there is no way to correct the position. If the robot was stepper versus DC motors then you could also use steps to estimate position along with time. 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. A DC motor with an encoder for feedback is referred to as a servo motor. This allows the robot to estimate its position relative to a start point. The Arduino robot has a motor but no encoder and it is only possible to control the motor speed and direction.

In this lab, you will use the UMBARK method to find adjustment factors to correct for repeatable odometry error on your robot. You will also use odometry concepts to implement the go-to-angle and go-to-goal behaviors on the robot. Remember to use the adjustment factors that you determined in this lab for the remainder of the robot motion commands in this course for the left and right wheel to correct for systematic errors in odometry. 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.

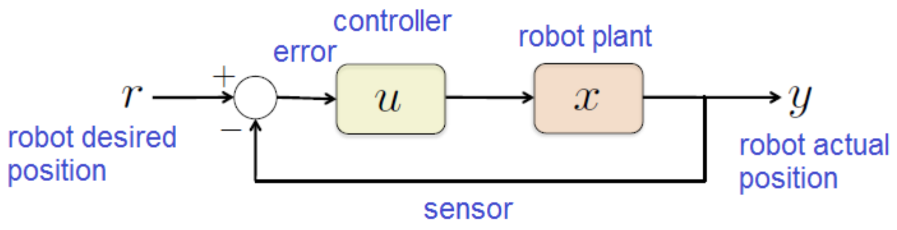
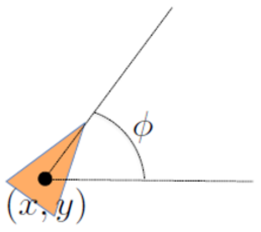


Figure 2: Robot Motion Feedback Control System

Since there are not any encoders on the Arduino robot motors, we will estimate the measured angle or goal by using velocity and time to estimate position using forward kinematics. Figure 2 shows the dynamic model for implementing the Go-To-Goal behavior.

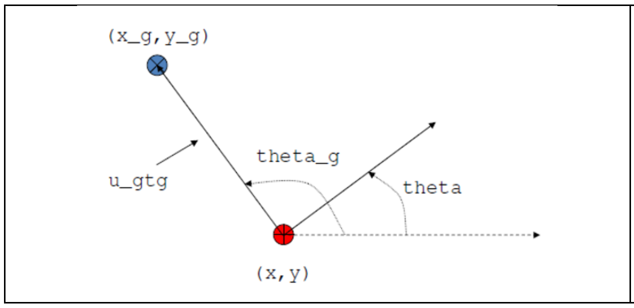


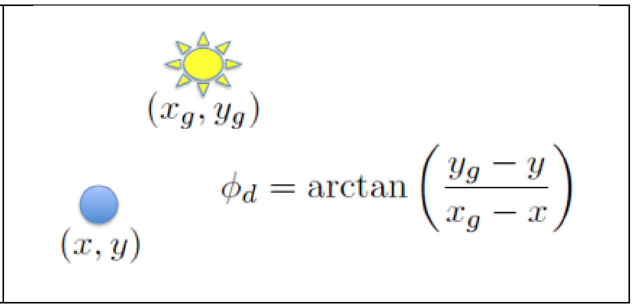
- Inputs:
  v
  ω
- Dynamics:

$$\begin{cases} \dot{x} = v\cos\phi \\ \dot{y} = v\sin\phi \\ \dot{\phi} = \omega \end{cases}$$

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.





$$\phi_d = \arctan\left(\frac{y_g - y}{x_g - x}\right)$$

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

****

Part I – Go To Angle Behavior

1. Create a method to implement the Go-To-Angle behavior.
2. The robot should turn the given angle and move forward for a 5 seconds.
3. Take measurements to estimate the accuracy of the Go-To-Angle behavior.
4. You should create an interface on the LCD where the user inputs the angle by pressing a pushbutton. The robot should then select a different pushbutton in order to make the robot move. For help on how to program the pushbuttons, please view Files→ Examples→ Robot Control→ explore→ R05_inputs.
5. The LCD should display the behavior as well as the requested angled. The robot should then turn the requested angle relative to the initial starting position. Use the LCD and/or buzzers to indicate behaviors as well as robot intentions or current state. For help on how to write to the LCD please review the reference library or example at Files→ Examples→ Robot Control→ explore→ LCDWriteText. For help on how to sound the buzzer please review the reference library or example at Files→ Examples→ Robot Control→ learn→ Beep.
6. Your code should be modular with the goToAngle() written as a function that is called from the void loop() function when the button is pressed.
7. The reference frame for the robot is shown in Figure 5, where the robot's start position is $(x,y,\theta) = (0,0,0)$.

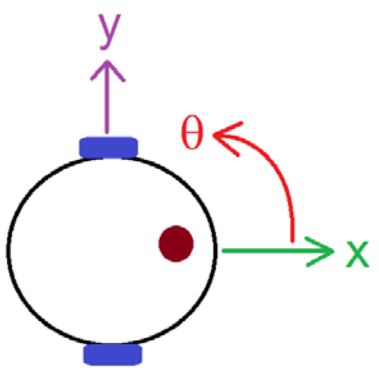


Figure 5: Robot Relative Reference Frame

Part II – 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. You should create an interface on the LCD where the user inputs the x and y values by pressing a pushbutton. The robot should then select a different pushbutton in order to make the robot move.
8. The LCD should display the requested behavior and goal position. The LCD should show the current motion being executed by the robot. Use the LCD and/or buzzers to indicate behaviors as well as robot intentions or current state.
5. Your code should be modular with the goToGoal() written as a function that is called from the void 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.
6. Use the reference frame shown in Figure 5 for the goToGoal() behavior.

Part III – Square Path

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

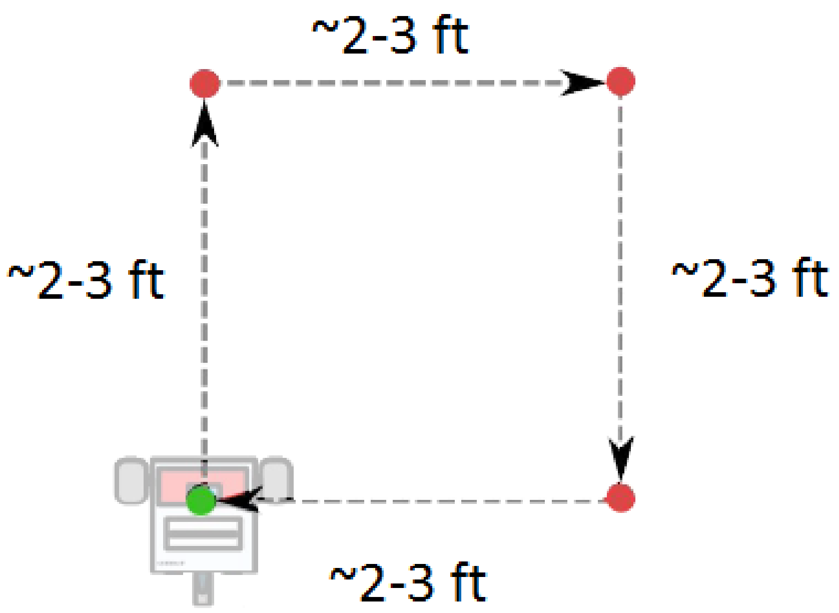


Figure 6: Square Robot Motion

1. Write a program to move the robot in a square path with sides between 2 and 3 feet (see Figure 6).
2. The Square() function should be called from the void loop() function at the push of a button. Since there are only 5 buttons on the Arduino, you may need to create a menu on the LCD with items selected by using the pushbuttons. You should call the goToAngle() and goToGoal functions from the Square() function.
3. Place masking tape on the ground where the robot will start.
4. Run the *"Square"* program 5 times with the robot traversing the path in a clockwise direction.
5. After each run, find the difference in the final x and y position from the original start position.
6. Run the *"Square"* program 5 more times with the robot traversing the path in the counterclockwise direction.
7. After each run, find the difference in the final x and y position from the original start position.
8. 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 7).
9. You should include the table of data for the 10 square traversals (cw, ccw) and the graph similar to Figure 7 for your data in the lab memo submission.

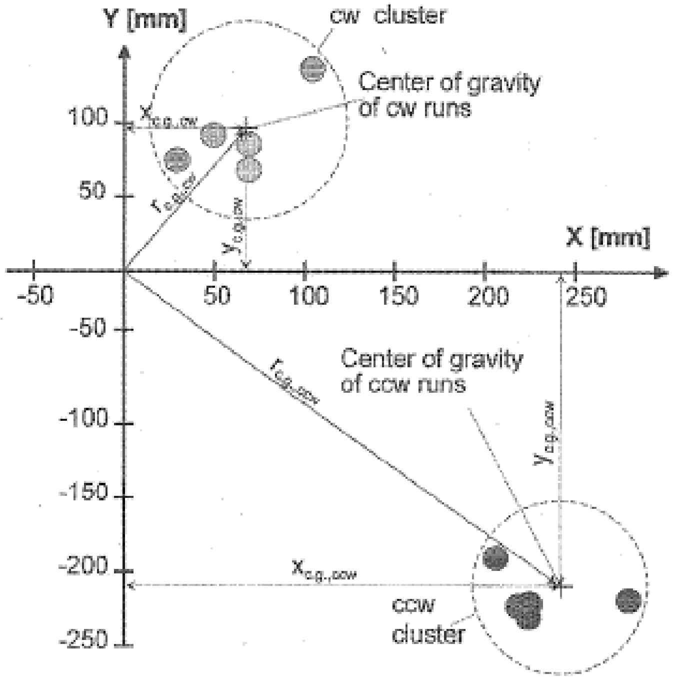


Figure 7: Odometery error on an uncalibrated vehicle

Part IV - 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

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

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

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

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

where

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

and

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

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 V – 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 8). You should call the Circle() function from the void loop() function on a button push.
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.

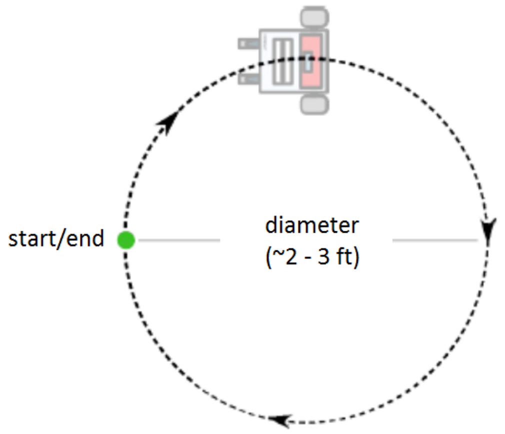


Figure 8: Circle Robot Motion

3. Finally, modify the *"Circle"* program to create a *"FigureEight"* program to move the robot in a figure eight using two circles (see Figure 9).
4. Once again try to adjust the tuning factors so that the robot passes through the same center point of the figure 8 each time. You should call the Circle() function twice to create the FigureEight() function. This means you may need to modify the Circle() function pass it a direction variable.
5. The LCD should show the current motion being executed by the robot.

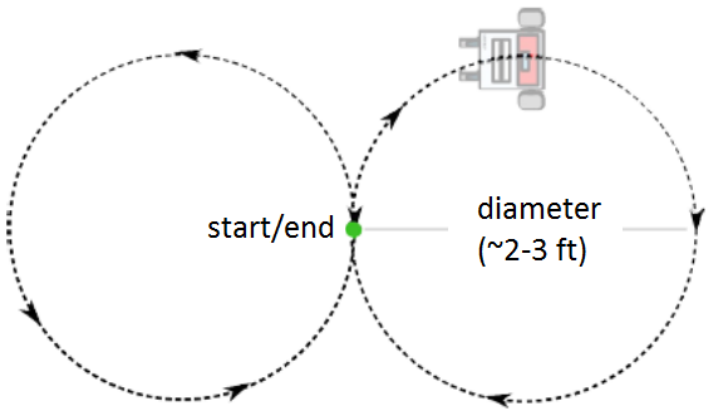


Figure 9: Figure Eight Robot Motion

Demonstration:

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

- Ablity to explain how code works
- Go-To-Angle motion
- Go-To-Goal motion

- Square motion
- Circle motion
- Figure Eight motion

Bring your robot fully charged to class every day!

Questions to answer in the lab memo

1. How did you calculate the turn angle for the robot using velocity and time? Explain and show formula in memo.
2. What type of accuracy/error did you have in the go-to-angle behavior?
3. How did you calculate the move distance given the x and y position? Explain and show formula in memo.
4. What type of accuracy/error did you have in the go-to-goal behavior?
5. What could you do to improve the accuracy of the behaviors?
6. 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?
7. What are some sources of the odometry error?
8. How could you correct for this error?
9. How could you improve the three motions (*Square, Circle, FigureEight*) programs?
10. Describe the method, pseudocode, flow chart, or state diagram.

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 Written as a combination of sentences or paragraphs and only bulleted list, if necessary
  - 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

- Starts with a statement of purpose
- Discusses the strategy or pseudocode for implementing the robot remote control (includes pseudocode, flow chart, state diagram, or control architecture in the appendix)
- Discusses the tests and methods performed
- 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

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

Submission Requirements:

You must submit your properly commented Sketch code & memo to the Moodle DropBox by midnight on Sunday. Check the course calendar for the lab demonstration due date.