# Memo

**To:** Dr.Berry

**From:** Carson Stone | Peter Garnache

**Date:** 12/15/19

**Re:** Mobile Robotics Lab 1

|  |
| --- |
|  |

## Introduction

The purpose of this lab is to develop the code needed to accurately move the robot.

To do this we were first tasked with developing primitive motion functions that were general in nature. These general motions could then be called to complete more complex movements such as driving in a square, moving in a circle, or moving in a figure 8.

## Method

The first primitive motion function we developed was **forward(int dist, int spd)**. This function has two inputs; an integer for the distance to travel in inches, and an integer for the speed of travel. The function then sets the desired position using stepper.move(dist) on both steppers. It then runs a while loop that waits until both steppers complete their movement that calls left.run() and right.run(). This allows us to improve precision by using the acceleration option in the stepper library to eliminate skipped steps and wheel skid during the start and end of the movement.

The next primitive emotion function we developed was **reverse(int dist, int spd)**. This function has two inputs; an integer for the distance to travel in inches, and an integer for the speed of travel. This function works the exact same as the previous forward function but uses stepper.move(-dist).

Next, we worked on the **stop()** primitive motion function. This function had no inputs. The function stopped all movement of the robot by calling left.stop() and right.stop().

After that, we worked on the **spin(char dir, int theta, int spd)** primitive motion function. This function has three inputs; a char (either ‘L’ or ‘R’) indicating the direction to spin, an int theta indicating the degrees the robot should spin, and an int spd indicating the speed of the spin in inches per sec. Spin used the run() command to drive the robot through a turn using acceleration. This allows us to improve precision by using the acceleration option in the stepper library to eliminate skipped steps and wheel skid during the start and end of movement. We determined the number of steps the wheel must travel using the kinematics attached in the appendix. We will set the distance for each stepper to move to using stepper.move(), then use a loop with stepper.run() on each side to complete the spin. One wheel will be given a positive distance to move, and the other will be given a negative distance to move, and the max speed on both steppers is set the inputted spd value.

The next primitive motion function we worked on was **pivot(char dir, int theta, int spd)**. This function has three inputs; a char (either ‘L’ or ‘R’) indicating the direction to spin, an int theta indicating the degrees the robot should pivot, and an int spd indicating the speed of the spin in inches per sec. Pivot used the run() command to drive the robot through its turn using acceleration. This allows us to improve precision by using the acceleration option in the stepper library to eliminate skipped steps and wheel skid during the start and end of the movement. We will determine the number of steps the wheel must travel using the attached kinematics in the appendix. We will set the distance for the stepper to move to using stepper.move(), then use a loop with stepper.run() to complete the spin. We will choose a stepper to run based on the “dir” variable and have the other stepper remain static.

The last primitive motion function we worked on was **turn(char dir, int theta, int spd, int r)**. This function takes four inputs; direction, theta and speed are the same as described before and the r value indicates the radius of the circle the center of the robot will follow in inches Turn will be using the steppers.runSpeedToPosition function to run both steppers at the same time. We initially tried using the run() command to drive the steppers to their final speed using acceleration, but that ended up not working in the way that we wanted it to. We determined the number of steps each wheel needs to travel using inverse kinematics of the movement attached in the appendix, and created a 2x1 position vector with the target positions of each stepper inside. The runSeedToPosition function of the MultiStepper library calculates the movement speeds for each movement for you, so there was no need to do those calculations on our own. This resulted in both wheels moving at the correct relative speeds to complete their path in the same time.

The first complex movement we worked on was **moveCircle(char dir, int spd, int r)**. moveCircle takes 3 inputs; a character representation of direction, an integer for the speed that the turn should be completed at, and a value for the radius of the circle the center of the robot will follow in inches. To accomplish this, all we had to do was tell the robot to use the turn function shown above and input the same direction, speed, radius, and input a theta of 360 degrees.

The second complex movement we worked on was **moveFigure8(char dir, int spd, int r)**. This function takes 3 inputs; a character representation of direction of the initial turn, an integer for the speed that the turn should be completed at inches per second, and a value for the radius of the circle the center of the robot will follow in inches. moveFigure8 uses the moveCircle function to make a figure 8 by calling it twice using the inputted direction, speed, and radius.

Next, we worked on the **moveSquare(char dir, int spd, int l )** complex movement. moveSquare takes 3 inputs; a character representation of the direction of the initial turn, an integer for the speed that the turn and movements should be completed at, and a value for the length for the sides of the square in inches. To accomplish this both the forward function is called with the inputed length and speed then the spin function is called to turn 90 degrees using the inputted speed and direction in a loop that iterates 4 times.

After that we, worked on the **goToAngle(int spd, int theta)** function. This function takes 2 inputs; an integer for the speed that the turn should be completed at, and a value for the angle the robot will spin to goToAngle will call the spin command and makes the robot end at a specific angle relative to the original direction the robot was facing before the function was called

The last complex movement we made was **goToGoal(int spd, int x, int y)**. goToGoal takes 3 inputs; an integer for the speed that the movement should be completed at, and a x and y position for the robot to go to. This function uses atan2(y, x) to find the angle required to position the robot pointed towards the goal. The robot then uses the spin command to rotate to that angle. After that, the robot uses the Pythagorean theorem to calculate the distance it should travel to end at the goal and then moves to that position using the forward function described above.

## Results

Our robot name is Arbib. Michael A Arbib is a professor at the University of Southern California. He is a theoretical neuroscientist that has done extensive work in computational neuroscience. He has written several books and papers on robotics as well.

### Laboratory Results and Questions:

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.***

Increasing the stepTime variable increased torque and accuracy while decreasing the speed.

Now comment out the first function, *move1()*, in the loop() function and uncomment the next second function, *move2()*, in the main loop. Compile and upload the program to see the difference in the code and robot behavior. ***Comment on this observation in your lab memo.***

Goes faster without skipping as many steps.

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.***

move() sets the target position of a stepper to a relative position. moveTo() sets the target position of a stepper as an absolute position. If you wanted the stepper to move to the position 800 from the origin, you would use moveTo(800), but if you wanted it to move one rotation forward, you would use move(800). These two movements would end at different places.

run() does a single step with a speed determined from acceleration while runSpeed() does a single step at a constant speed. If you call these functions in a loop, then the robot will continue moving seamlessly

runToPosition() moves the stepper from its current position to the target position with an acceleration based movement pattern. runSpeedToPosition() moves the stepper from its current position to the target position using a constant speed. runToNewPosition() sets the target position and then runs to it. These functions use blocking, so they can only be run one at a time. That means if you try to drive the robot forward using these functions, one wheel runs forward, then when it stops, the other one move. This results in the wrong movement behavior for the robot.

Now comment out the move2() function and uncomment the *move3()* function. Repeat steps 11 and 12. ***Comment on your observations in your lab memo.***

Uses steppers.run(). behaved exactly the same but could call both steppers at once. This meant that we could use functions like runToPosition() or runSpeedToPosition() to move the robot because they did not use blocking.

Next repeat steps 11 and 12 for the *move4()* and *move5()* functions.

* + Blocking
  + Bounce
  + ConstantSpeed
  + MultipleSteppers
  + MultiStepper
  + Overshoot
  + ProportionalControl
  + Quickstop
  + Random

***Comment in your lab memo on what you think each of the functions do.***

runToNewPosition runs a stepper continuously with acceleration until it reaches a given position basically the move to command and the runToPosition command

(multisteppers)steppers.runSpeedToPosition run all steppers continuously until reaches given position without accelerating all steppers reach the position at same time

stepper.stop() stops the motor at the current step but saves the current speed the motor was at for the next step

The move makes the robot’s turn its wheel by a given delta value while Moveto the sets an absolute desired position where zero is the position where the robot is turned on

Set max speed sets the maximum speed any run command can reach

Set acceleration sets the acceleration a motors run command speeds up in steps-per-second-squared

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.***

1832 steps

**SPIN**  how many steps does it take to spin the robot 90 degrees? ***State this value in your lab memo.***

496 steps

**PIVOT** how many steps does it take to pivot the robot 90 degrees? ***State this value in your lab memo.***

991 steps

**CIRCLE** 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.***

In order insure that our circle function ended in the same spot it started, we applied a fudge factor multiple to each of the robot wheels. We then adjusted the fudge factor to apply more or less steps to a wheel for a given circle radius based off of trial results. This resulted in a program that met the 2-inch accuracy requirement for all sizes of circles that we tried

**GOTOANGLE** Take measurements to estimate the accuracy of the Go-To-Angle behavior. ***Comment on the results in your lab memo.***

After calibrating a fudge factor for the wheels from the circle function our goToAngle function needed no additional calibration to be accurate.

### Accuracy Results:

In order to determine the accuracy of our robot’s movements, we tested some of our movement functions for their ability to return back to the starting position. We had the robot execute a 2 foot circle, a 1 foot diameter figure 8, and a 1.5 foot square 5 times each to get a measurement for the robot’s accuracy. We measured the distance that the robot was from the starting point and recorded them in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Trial** | **Circle (2ft)** | **Figure 8 (1ft)** | **Square (1.5ft)** |
| 1 | 0.10" | 0.50" | 0.50" |
| 2 | 0.25" | 0.50" | 1.00" |
| 3 | 0.25" | 0.25" | 1.50" |
| 4 | 0.25" | 1.00" | 1.25" |
| 5 | 0.50" | 0.75" | 1.50" |
| Average | 0.27" | 0.60" | 1.15" |

All of these movements are fairly accurate and within the expected accuracy. Because figure 8 calls circle twice, we can expect the accuracy to be less than the circle function because the error is compounding. The square function makes 8 function calls throughout its operation, so a higher inaccuracy is expected for that. All of our movement accuracies are within our 2” desired accuracy.

### Additional Questions:

1. What is the diameter and circumference of the robot wheels?

The left wheel has a diameter of 3.338 in. The right wheel has a diameter of 3.337 in. This gives us a circumference of 10.487in on the left, and 10.483 on the right.

1. How many inches or feet can it move in a quarter, half, full, and two rotations?

The robot can move 2 5/8 inches in a quarter rotation, 5.25 inches in a half rotation, 10.5 in a full turn, and 21 inches in two rotations.

1. How many encoder ticks do you get in a quarter, half, full and two rotations?

We get 5 encoder ticks in a quarter rotation, 10 in a half rotation, 20 in a full rotation, and 40 in two rotations.

1. Compare the accuracy between encoder ticks and steps for a given distance (i.e. 2 feet).

Over the course of 2 feet, we would travel 1831 steps with an error of 0.45 steps from rounding. We would also travel 46 ticks on the encoder with an error of 0.22 ticks from rounding. The encoder error would convert into 8.8 steps. The encoder would be less accurate over 2 feet than the encoder because of the difference in resolution between the sensors.

1. Describe how you can use a proportional controller to move the robot a given distance and correct for odometry error.

We could use a proportional controller to move the robot by calculating the error between the desired position and the current position based on the feedback from the encoders, then change the speed of the robot linearly based on that error. The robot would slow down as it approaches its target position, and hopefully land perfectly on the desired position.

1. How did you calculate the turn angle for the robot? Explain and show formula in memo.

We calculated the distance each wheel needed to travel using forward kinematics. With variables w = width of the robot, r = radius of the circle, d = diameter of wheels, and spr = steps per revolution on the wheels. the inner wheel distance and outer wheel distances can be calculated using

and .

These equations calculate the circumference of the portion of the circle that the robot should traverse, then convert that into the number of steps the robot should take. We decided to use the pivot command for our goToAngle command so the center of our robot did not move during the turn which made the goToGoal function easier.

1. What type of accuracy/error did you have in the go-to-angle behavior?

We had very good accuracy in the goToAngle function. We moved to positions with greater than 3 degree accuracy.

1. How did you calculate the move distance given the x and y position? Explain and show formula in memo.

We calculated the move distance given an x and y position using the quadratic formula. Now that we are pointed at the goal, we can model it as a right triangle with the move distance being the hypotenuse. Knowing both side lengths we can calculate the hypotenuse using

We then converted this value to the number of steps for our robot to traverse using the number of steps per revolution divided by circumference of the wheels.

1. What type of accuracy/error did you have in the go-to-goal behavior?

We had very good accuracy in the goToAngle function. We could go to positions with error less than 1 inch in any direction.

1. What could you do to improve the accuracy of the behaviors?

One thing to improve the accuracy of the behaviors would be to add a gyroscopic sensor to improve the accuracy of our turns using feedback control. The wheels have a large width so they do not rotate about the same point each time which means that we cannot use the encoders to improve accuracy in turn measurements.

1. 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?

We used the turn then forward approach for the go to goal program because it was much simpler to program and much more accurate. One con of this program is that at the end of the movement, the robot was positioned in a different direction than when it started.

1. What are some sources of the odometry error?

Odoetry error comes from several sources. First, when moving the robot with a high acceleration and high speed, it is possible to skip steps during acceleration and deceleration. Another source of error is the width of the wheels when turning. Because the wheels are wide, we do not know he exact point that the wheels are turning about, and that point changes over time.

1. How could you correct for this error?

We corrected for the acceleration skipping steps by using a small enough acceleration that we stopped seeing the motors skip steps. We also could have implemented PID control with the encoders, but because of the low resolution on the encoders, this option would likely increase our error. We estimated the point that the robot spun about, and used fudge factor variables to calibrate the robot so it actually reached the points that we wanted.

1. How could you improve the three motions (move*Square, moveCircle, moveFigure8*) functions?
2. Describe the method, pseudocode, flow chart, or state diagram.

## Conclusion

This lab, we learned the basics of controlling the mobile robotics platform that this class is based off of. We did this by familiarizing ourselves with the library that we will be using to perform movement functions. We completed every objective of the lab within the accuracy requirements but have also realized that the math isn’t always perfectly precise when it comes to these robots. We had to apply a fudge factor to many of our movements to get the robot to return to its starting location.

We discovered some interesting things through finishing this lab. The first is in the provided function runToStop(). The way that this function appears to work is that it querys the left.run() and right.run() functions in if statements to determine if the wheels have stopped, then exits the loop when both wheels finish their movement. At first glance this appears to only stall the code until the robot is done moving, but in reality the calls to the left.run() function inside the logical statements is still moving the motor. This saves us a lot of space as we could write an entire movement command with one line using something like this:

While(!stepperLeft.run() || !stepperRight.run()){ }

And that function would have the same effect as the entire group of like 15 lines of code that move the stepper to the target position.

We successfully completed the target of the lab, and understand more about the mobile robotics system now than we did before this point.

## Appendix:

# **Common Variable Glossary**

## **Global Constants**

spr Steps per Revolution for the wheel

d Diameter of the wheel

w Distance between the centers of the wheels

dstStep Linear distance of one wheel step

speedD Default speed

accelD Default acceleration

# **forward(int dist, int spd)**

forward has two inputs; an integer for the distance to travel in inches, and an integer for the speed of travel.

dist is a value from -1000 to 1000 in inches

spd is a value from 1 to 100 in inches per sec

The function then sets the desired position using stepper.move(dist) on both inputs. It then runs a while loop that waits until both steppers complete their movement that calls left.run() and right.run() each loop. This allows us to improve precision by using the acceleration option in the stepper library to eliminate skipped steps and wheel skid during the start and end of movement.

# **reverse(int dist, int spd)**

reverse has two inputs; an integer for the distance to travel in inches, and an integer for the speed of travel.

dist is a value from -1000 to 1000 in inches

spd is a value from 1 to 100 in inches per sec

The function then sets the desired position using stepper.move(-dist) on both inputs. It then runs a while loop that waits until both steppers complete their movement that calls left.run() and right.run() each loop. This allows us to improve precision by using the acceleration option in the stepper library to eliminate skipped steps and wheel skid during the start and end of movement.

# **stop()**

stop has no inputs.

Stop calls the function left.stop() and right.stop() in order to stop the robot.

# **spin(char dir, int theta, int spd)**

spin takes three inputs; a character representation of direction, an integer for the angle rotation to be completed, and an integer for the speed that the turn should be completed at.

dir has a value of either ‘L’ or ‘R’

theta is a value from -3600 to 3600 in degrees

spd is a value from 1 to 100 in inches per sec

Spin will be using the run() command to drive the robot through a turn using acceleration. This allows us to improve precision by using the acceleration option in the stepper library to eliminate skipped steps and wheel skid during the start and end of movement. We will determine the number of steps the wheel must travel using kinematics. We will set the distance for each stepper to move to using stepper.move(), then use a loop with stepper.run() on each side to complete the spin. One wheel will be given a positive distance to move, and the other will be given a negative distance to move.

# **pivot(char dir, int theta, int spd)**

pivot takes three inputs; a character representation of direction, an integer for the angle rotation to be completed, and an integer for the speed that the turn should be completed at.

dir has a value of either ‘L’ or ‘R’

theta is a value from -3600 to 3600 in degrees

spd is a value from 1 to 100 in inches per sec

Pivot will be using the run() command to drive the robot through a turn using acceleration. This allows us to improve precision by using the acceleration option in the stepper library to eliminate skipped steps and wheel skid during the start and end of movement. We will determine the number of steps the wheel must travel using kinematics. We will set the distance for the stepper to move to using stepper.move(), then use a loop with stepper.run() to complete the spin. We will choose the stepper to run based on the “dir” variable.

# **turn(char dir, int theta, int spd, int r)**

Turn takes four inputs; a character representation of direction, an integer for the angle rotation to be completed, an integer for the speed that the turn should be completed at, and a value for the radius of the circle the center of the robot will follow in inches.

dir has a value of either ‘L’ or ‘R’

theta is a value from -3600 to 3600 in degrees

spd is a value from 1 to 100 in inches per sec

r is a value from 0 to 100 in inches

Turn will be using the run() command to drive the robot through a turn using acceleration. This allows us to improve precision by using the acceleration option in the stepper library to eliminate skipped steps and wheel skid during the start and end of movement. We will determine the number of steps the wheel must travel using kinematics. We can also calculate the speeds that each wheel must travel at to achieve the desired average speed of the center of the robot, and turn the robot at the desired radius. Because we are using acceleration, and each wheel is spinning at different speeds, we will need to use kinematics equations for constant acceleration systems to solve for the acceleration needed by each wheel so that they accelerate over the same time period, so that the ratio of speeds from the outer wheel to the inner wheel is always the right ratio for the radius of turn that we want.  We will set the distance for each stepper to move to using stepper.move(), then use a loop with stepper.run() on each side to complete the turn.

# **moveCircle(char dir, int spd, int r)**

moveCircle takes 3 inputs; a character representation of direction, an integer for the speed that the turn should be completed at, and a value for the radius of the circle the center of the robot will follow in inches.

dir has a value of either ‘L’ or ‘R’

spd is a value from 1 to 100 in inches per sec

r is a value from 0 to 100 in inches

This function will call the turn() function with the following call:

turn(dir, 360, spd, r);

# **moveFigure8(char dir, int spd, int r)**

moveFigure8 takes 3 inputs; a character representation of direction of the initial turn, an integer for the speed that the turn should be completed at, and a value for the radius of the circle the center of the robot will follow in inches.

dir has a value of either ‘L’ or ‘R’

spd is a value from 1 to 100 in inches per sec

r is a value from 0 to 100 in inches

moveFigure uses the moveCircle command to create a figure 8. It calls it twice with the “dir” input changing based on the input to moveFigure8. We will include this change in the if loops that check the dir variable character.

For a “dir” input of ‘L’, the calls would be as follows:

moveCircle(L, spd, r);

moveCircle(R, spd, r);

# **moveSquare(char dir, int spd, int l )**

moveSquare takes 3 inputs; a character representation of the direction of the initial turn, an integer for the speed that the turn and movements should be completed at, and a value for the length for the sides of the square in inches.

dir has a value of either ‘L’ or ‘R’

spd is a value from 1 to 100 in inches per sec

l is a value from 0 to 100 in inches

In a for loop that iterates 4 times, the forward function is called with the inputted length and speed and then the pivot function is called to turn 90 degrees using the inputted speed and direction

# **goToAngle(int spd, int theta)**

goToAngle takes 2 inputs; an integer for the speed that the turn should be completed at, and a value for the angle the robot will spin to

spd is a value from 1 to 100 in inches per sec

Theta is a value from -360 to 360 in degrees

This function just calls the spin command and makes the robot end at a specific angle relative to the original direction the robot was facing before the function was called

# **goToGoal(int spd, int x, int y)**

goToGoal takes 3 inputs; an integer for the speed that the movement should be completed at, and a x and y position for the robot to go to

spd is a value from 1 to 100 in inches per sec

l is a value from 0 to 100 in inches

This function uses atan2(y, x) to find the angle required to position the robot pointed towards the goal. The robot then uses the spin command to rotate to that angle. The robot uses the Pythagorean theorem to calculate the distance it should travel to end at the goal. It then moves forward to that position.



