To: Dr. Berry

From: Team Moravec, Devon Adair and Hunter LaMantia

Date: 12/9/2018

Re: Robot Odometry Lab 1

The purpose of this lab was to program basic movement functions into our mobile robot. The basic movement functions were then used to create more advanced movement functions.

Our first advanced movement function was going to a goal based on given coordinates. This function called another function that calculated the angle in which the robot had to turn, and then moved in a straight line to the goal.

Our next advanced movement function was moving in a square. This called our function for going to a goal four times, and then turned to face its original direction.

Another advanced movement function was moving in a circle. This called a basic movement function meant for moving in an arc which specified to move a full 360 degrees.

Our last advanced movement option was moving in a figure eight. This just called the function for moving in a circle twice.

Our pseudocode represents our initial plans for how we wanted to make our functions. After experimentation and finding bugs in our original code, our functions are different in a few ways than we planned. The pseudocode is attached.

In all of our functions, we wanted the inputs to have degrees, inches, and inches per second as units. A large portion of our process was experimentally finding conversion factors to convert these units into distances and speeds that are meaningful to the robot.

To find the conversion factors we needed, we measured the wheels of the robot. We found them to have a diameter of 3.34 inches and a circumference of 10.5 inches. This means that a full rotation moves the robot 10.5 inches, a half rotation moves it 5.25 inches, a quarter rotation moves it 2.625 inches, and two rotations moves it 21 inches.

Through use of the encoders we found how many ticks the wheels move when making those turns. A full turn takes 20 ticks, a half rotation takes 10 ticks, a quarter rotation takes 5 ticks, and two rotations takes 40 ticks. Therefore, to move two feet, we would need 1829 steps or 91.4 ticks.

One way to correct for error is to use the encoders as proportional controllers. We could do this by programming the robot to check whether the encoder value is equal to, less than, or greater than the target value, and making the motors move or not move based on that information.

The turning angle was calculated with a simple arctangent, using the given goal’s coordinates. Specifically, our equation was

,

which was followed by a series of if-else statements based on whether x or y were positive or negative to determine which quadrant to put the robot in. The move distance was also simple. We used the Pythagorean Theorem, which is

.

Neither the function for going to a specific angle nor the function for going to a goal had any noticeable error, as far as we could tell. However, we did not use our encoders to check for accuracy, so we could use them to reduce any error we may have. We believe that one reason for our accuracy is that we turned before moving, rather than attempting to do both together.

Our function for moving in a circle was not as accurate. The diameter of the circle that the robot moves in and the ending point is generally within three inches of what we expect, but the error is still noticeable.

Our biggest source of odometry error is our rear wheel slipping or gripping to hard when making a turn. Besides that, we believe that imperfect terrain and hardware likely caused what little other error we had. We would correct for this error by using the proportional controller described above. Implementing this would improve all of our advanced movement function, such as moveSquare(), moveCircle(), and moveFigure8().

We experimented a bit with the basic movement function in order to gain a higher understanding of the given code. We found that hanging the step time from 500 to 4000 slows the robot down a lot. This is because a lot more time now passes between each step.

We found that move1(), which moves the robot forwards and backwards, allows the robot to turn a little bit on startup and on stopping. However, move2() is exact, likely because of the usage of the run function of the library. On the other hand, move3() was very rough, likely because the multistepper does not give us complete control over the individual steppers.

Within move2(), we changed a few functions to see what would happen. We changed move(0) to moveTo(0), which caused the robot to stop moving backwards. This is because moveTo(0) told the robot to return to position zero, the original position, but move(0) told the robot to move a distance of zero, so it did not move. We also changed runSpeedToPosition() to runSpeed(), which greatly lowered the accuracy of the robot’s movements. We believe this to be because runSpeed() does not set a specific position.

We also changed functions within move3() to see what would happen. We changed runSpeedToPosition() to run(), resulting in the robot stuttering in place and not moving. We believe this to be because no position was given.

We also found the number of steps required to move the robot in a few ways. To move it forward two feet, both wheels must move 91.4 steps. To pivot it 90 degrees, it must move 960 steps. To spin it 90 degrees, both wheels must move 480 steps.

In conclusion, our robot worked very well. It was not quite as accurate as it could have been if we had used encoders, but it was always within our tolerance of three inches.

Our robot is named after Hans Moravec, who is famous for his discovery that making robots do things intelligently is not nearly as difficult as making them able to detect their surroundings and move around in them.

Attached is our pseudocode.

Pseudocode:

Go to Angle (int angle) {

Set a constant speed but different direction

Calculate delay time based upon angle in degrees

If angle is positive

Spin counter clockwise at the calculated speed

Else // angle is negative

spin clockwise in the same speed

Delay for delay time

Stop()

}

Go to Goal (x, y) {

Calculate the angle based on given position

Call the angle function given the calculated angle

Calculate distance to move to next

Call forward to the calculated distance

}

Circle (diameter, direction) {

Set both speeds at a constant speed

Calculate the speed at which to move the diameter

Calculate the difference in the speed based upon the circle diameter

Set the speed of the robot

Call the turn function given the direction left or right

If left make right go faster

If right make left go faster

}

Figure Eight (diameter) {

Set initial Direction to left

Call the circle function with the given diameter the initial direction

Call the circle function again with the given diameter and opposite direction

}

Square (side) {

// side is the length of every side of the square

Call go to goal with the x = side and y = 0

Call go to goal with the x = 0 and y = -side

Call go to goal with the x = 0 and y = -side

Call go to goal with the x = 0 and y = -side

Turn -90 degrees to end up in same direction we started

}