Design Principles and Methods - Navigation Lab Report

Harley Wiltzer (260690006)
Juliette Regimbal (260657238)

October 13, 2016

1 Objective

To design a software system that allows a (main) thread of execution to command it to drive the robot to an absolute position on the field while avoiding obstacles, by use of the odometer and an ultrasonic sensor.

2 Method

- Using the nagivation tutorial provided, create a class that extends `Thread` (or, if you want to be very resourceful, a class that implements `TimeListener`) to control the robot's motors to drive to a specified point on the field, given in Cartesian coordinates. Your class should at least implement the following public methods:
  - `void travelTo(double x, double y)`
    This method causes the robot to travel to the absolute field location (x, y). This method should continuously call `turnTo(double theta)` and then set the motor speed to forward(straight). This will make sure that your heading is updated until you reach your exact goal. (This method will poll the odometer for information.)
  - `void turnTo(double theta)`
    This method causes the robot to turn (on point) to the absolute heading `theta`. This method should turn a MINIMAL angle to its target.
  - `boolean isNavigating()`
    This method returns true if another thread has called `travelTo()` or `turnTo()` and the method has yet to return; false otherwise.
- Adjust the parameters of your controller, if any, to minimize the distance between the desired destination and the final position of the robot, while simultaneously minimizing the number of oscillations made by the robot around its destination before stopping.
- Write a program to travel to the waypoints (60, 30), (30, 30), (30, 60), and (60, 0) in that order. Test this program ten (10) times, and record the position the robot a) as reported by the odometer, and b) as measured on the field.
- Modify your code to use the ultrasonic sensor to detect an obstacle (a cinder block) and avoid collision with it. You may mount the ultrasonic sensor as you wish, and may use your wall follower code if desired. Additionally, you may create another class to perform this function.
- Write a program to, with obstacle avoidance, travel to the waypoints (0, 60) and (60, 0) in that order. To demonstrate the effectiveness of your obstacle avoidance, the TA will first trun your robot's program to travel that path without obstacles, then place a single cinder block somewhere along the path (though not at a waypoint), and run your robot's program again.

3 Data

Figure 1: Final Position of Robot (cm) - Part 1

Trial No.	x-odometer (cm)	y-odometer (cm)	x-actual (cm)	y-actual (cm)	x-error (cm)	y-error (cm)
1	60.92	-1.80	61.8	0.4	-0.88	-2.20
2	60.84	-1.52	63.0	-0.2	-2.16	-1.32
3	60.61	-0.87	61.9	0.5	-1.29	-1.31
4	60.88	-1.51	62.2	-0.3	-1.32	-1.21
5	60.94	-1.66	61.5	-0.7	-0.56	-0.96
6	60.86	-1.58	62.4	-0.4	-1.54	-1.18
7	59.92	0.20	61.0	1.6	-1.08	-1.40
8	60.60	-0.65	61.2	0.3	-0.60	-0.95
9	61.04	-1.86	61.5	-1.2	-0.46	-0.66
10	61.08	-2.04	63.0	-0.2	-1.92	-1.84
Mean	60.77	-1.33	61.95	-0.02	-1.18	-1.30
St. Dev.	0.34	0.69	0.70	0.77	0.58	0.44

4 Data Analysis

4.1 Calculations of Mean and Standard Deviation for Error in Odometer Readings

$$\text{Mean} = \frac{1}{n}\sum_{k=1}^{n}\mathcal{E}_k \tag{1}$$

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{k=1}^{n}(\mathcal{E}_k - \mu_{\mathcal{E}})^2}{n-1}} \tag{2}$$

In formulae (1) and (2), n represents the number of trials for which data was acquired, $\mu_{\mathcal{E}}$ represents the mean of the error, and $\mathcal{E}_k$ represents the error in measurement in some coordinate axis. Below, the calculations for these values are explicitly shown.

4.1.1 Calculations of the means

$$\overline{x} = \frac{-0.88 - 2.16 - 1.29 - 1.32 - 0.56 - 1.54 - 1.08 - 0.60 - 0.46 - 1.92}{10} = -1.18\text{cm}$$

$$\overline{y} = \frac{-2.20 - 1.32 - 1.31 - 1.21 - 0.96 - 1.18 - 1.40 - 0.95 - 0.66 - 1.84}{10} = -1.30\text{cm}$$

4.1.2 Calculations of the standard deviations

$$s_x = \sqrt{\frac{(-0.88-(-1.18))^2+(-2.16-(-1.18))^2+(-1.29-(-1.18))^2+(-1.32-(-1.18))^2+(-0.56-(-1.18))^2+(-1.54-(-1.18))^2+(-1.08-(-1.18))^2+(-0.60-(-1.18))^2+(-0.46-(-1.18))^2+(-1.92-(-1.18))^2}{9}}$$
$$= 0.58\text{cm}$$

$$s_y = \sqrt{\frac{(-2.20-(-1.30))^2+(-1.32-(-1.30))^2+(-1.31-(-1.30))^2+(-1.21-(-1.30))^2+(-0.96-(-1.30))^2+(-1.18-(-1.30))^2+(-1.40-(-1.30))^2+(-0.95-(-1.30))^2+(-0.66-(-1.30))^2+(-1.84-(-1.30))^2}{9}}$$
$$= 1.44\text{cm}$$

4.2 Are the errors present as a result of the odometer or the navigator?

The errors are caused by both the odometer and the navigator. As seen in the Odometer experiment, there is significant slippage occuring when the wheels spin. As a consequence of such slippage, there must be error in the measurement of the x and y position of the robot, as well as in the measurement of its heading.
However, due to the design of the navigator, more error emerges. Since the navigator must correct its angle continously, one cannot simply tell the navigator to travel some distance in a given direction. Instead, the navigator

is programmed to move on to the next waypoint when it is within a certain range of the desired target position. Therefore, with this design, error is inherent and even *allowed*, as a tradeoff for better heading correction and a more dynamic controller. The result is that while the robot can very accurately reach the waypoint in the appropriate direction, it only gets to within a certain error margin of the waypoint. These errors, however, are independent from that of the odometer error. Although the robot may not reach the exact waypoint, the odometer *knows* the robot has not reached the exact waypoint - that is to say, although there is error in the robots position due to the design of the navigator, this does not hinder the performance of the odometer.
Furthermore, since the robot is continuously correcting its heading, the wheels undergo a lot of acceleration (in order to change the robot's direction). Although usually the corrections are very small, in some situations the corrections may be large enough to cause significant slipping (because of the acceleration) which can lead to significant error in the odometer readings.
Finally, another source of error lies in the model of the robot. Odometer calculations are done given certain measurements, such as the radii of the wheels of the robot and the distance between them. The controller of the robot assumes that these are perfectly accurate, which is not true in reality, thus any error on those measurements leads to error in the odometer's performance. Also, as the wheels accelerate, they undergo stress which can lead to deformation. The model of the robot in the controller assumes perfectly circular wheels, and such an assumption grows less and less accurate as the wheels wear out, thus introducing more error in the odometer.

5 Observations and Conclusion

5.1 Explain the operation of the controllers for navigation. How accurately does it move the robot to its destination? How quickly does it settle on its destination?

The controller for navigation works by continuously correcting the heading of the robot and having it move forward until it is within a certain euclidean distance from the waypoint. The heading correction works by using the waypoint coordinates and the odometer's readings to calculate by which angle the robot must rotate to be facing its destination, using Java's `atan2` method. To avoid constant, tiny corrections (which would result in very jerky behavior and possibly more slippage), the robot only corrects its heading when it is outside some threshold.
The controller for obstacle avoidance is actually implemented in the same class as that for basic navigation. The difference is that when the ultrasonic sensor detects that it is close to a wall, the normal operation of the path follower is skipped, and the robot turns its ultrasonic sensor some θ degrees, and rotates on point $-\theta$ degrees to keep the sensor in the same relative position. At this point, the robot's heading Θ is stored, and it contours the wall using a Bang-Bang controller until its heading is within a small margin of $\Theta + 180$ degrees. Next, the robot turns its sensor back $-\theta$ degrees, and turns θ degrees while moving one of its wheels faster than the other so as to move away from the wall as it rotates, at which point it can return to its normal navigation method.
After tweaking the threshold value associated with the minimum error for which the robot corrects its heading in `turnTo()`, the robot was found to settle relatively quickly on its target. In the worst case, the robot was seen to make up to three oscillations near a waypoint, however its accuracy and precision were sublime. Furthermore, the threshold corresponding to the maximum euclidean distance from a waypoint for which the robot moves on to the next waypoint was tweaked until the robot reached its destination within an appropriate error margin. As seen in section **3**, the robot was always accurate to within 3cm from its target destination.

5.2 How would increasing the speed of the robot affect the accuracy of the robot? What is the main source of error in navigation (and odometry)?

The main source of error in odometry and odometry-based navigation is wheel slippage. Since the position of the robot is tracked by using wheel rotation with an accurate physical model of the chassis, any deviation from its expected behavior - such as wheel slipping - will introduce error into odometry and thus navigation. Since the robot makes corrections in navigation and changes paths to go to different waypoints, a higher traveling speed would require more acceleration. With the wheels accelerating more, the likelihood that they will slip more often and for longer when making adjustments to wheel direction and speed increases. And as this error increases, the overall accuracy in navigation decreases.

6 Further Improvements

To decrease error, higher friction wheels could be substituted to make it more difficult for the robot to slip during normal use. Predictions could be made by finding if there is a constant amount of error introduced over a certain distance (for example, the robot could be consistently +0.5cm off per 100cm travelled). This correction could be factored into odometry in software, allowing for more accurate positioning in navigation. Furthermore, reducing the speed of the robot would in turn reduce the amount of acceleration of the robot, effectively reducing the amount of slipping of the wheels. Finally a method of landmark based correction could be implemented, similar to how the light sensor detected evenly-spaced lines in Lab 2. Landmark detection would provide an absolute distance from either a point or line on the track, allowing for the robot to reset its odometry readings and essentially zero the error accumulated through travel.