ECEN 3033 - Lab #3 Report

Miles Sanders, Alberto Espinosa, Mark Abbott

1. What happens when you set the "gains" (source-of-error coefficients) of your controller too low? Describe your observations both for distance and bearing.

When I set the gains of the controller too low, the bearing, rho or heading error are not prioritized enough. Therefore, the robot does not take the most optimal route to getting to the desired waypoint and sometimes even goes completely off course. Therefore it is important that the difference in value between the gains are large enough to prioritize what the robot is trying to accomplish, whether it is trying to correct the bearing, rho or heading error.

2. What happens when you set the distance threshold for your stopping criteria (selecting the next waypoint from your list) too low? What happens when you set it too high?

When you set the distance threshold too low, the robot spends too much time trying to correct its exact position to match the waypoint rather than moving on to the next waypoint once you have gotten close enough. If you set the distance threshold too high, the robot does not get close enough to the desired waypoint location before moving on to the next. This causes the robot to crash into objects and not stay on the correct course of achieving the desired final pose.

3. Describe how you implemented your feedback controller in Part 2.

The feedback controller was implemented following the lab instructions… Where we started by computing the Position, Bearing, and Heading Error. We then added a set of conditional statements to determine the controller gains which determined our feedback control law, first prioritizing the bearing error, then the position error, and finally the heading error. Given the correct gains, we then calculated the right and left wheel velocities using inverse kinematics together with our errors and our feedback control law. We made sure to clamp both the wheel velocities and the error values. Finally we calculated the current robot's pose using the odometry equations given on piazza.

4. Does your implementation for Part 2 work? If not, describe the problems it has.

Yes, the implementation for part 2 works smoothly.

5. What are the equations that you use to compute the desired wheel speeds from your implementation? What are your equations for $\theta R'$ and xR' in your feedback controller?

The equations we used to calculate $\theta R'$ and xR' in our feedback controller are as follows:

- xR' = p1 * rho : where p1 stands for the gains made by our position error and rho stands for our position error
- $\theta R'$ = p2 * alpha + p3 * nu : Where p2, and p2 stand for gains made by alpha, our bearing error, and nu, our heading error respectively.

6. What happens if you do not limit the wheel speeds to positive or negative MAX_SPEED? Think about what the robot will do in this case and what your odometry will predict.

If we do not limit the wheel speeds to MAX_SPEED, the lack of limitations on the speed of the left and right wheels results in commands that exceed the absolute value tolerated by Webots. This is due to the fact that the coefficients, specifically p1, are gain values that need to be scaled to the environment in which they operate. The max speed cannot be exceeded, the magnitude of speed applied from the range that is available is what is controlled. If this variable isn't scaled properly, information that hinders the robots performance is introduced. One example of this is when a turn command is introduced, if both speeds exceed the max velocity of the wheel, both wheels will run at maximum velocity, and the robot will move forward instead of at an angle.

7. What would happen if an obstacle was between the robot and its goal?

With the current code, if an obstacle was between the robot and its goal, the robot would just crash into the object and not be able to move any further.

8. (Briefly) How would you implement simple obstacle avoidance using distance sensors if they were placed around the Turtlebot?

I would set a threshold. If the object that was in front of the robot exceeded this threshold, the robot would stop and turn to avoid crashing into it. Now that the robot has detected an object, it will turn until it does not detect the object in front of it anymore. Then, the robot would keep moving forward in this new direction, repeatedly checking if the object was still there. The reason it is still checking if the object is still there is because the desired waypoint is assumed to be behind the object. Therefore, once the object is no longer in the robot's area, the robot will then turn back into the direction of the desired waypoint until it reaches its destination.

9. Roughly how much time did you spend programming this lab?

The lab was completed in a window of about two days.