Stephen Gregory
CS-460-001 - Intro to Robotics
Dr. Chris Crawford
Homework 5

Homework 5 - Reactive Control

I began the iteration of my turtlebot control program with simple reactive control involving following walls in the robot's environment. First, my program simply monitored the distance between the robot and obstacles in front of the robot at angles of -29 degrees, 0 degrees, and 29 degrees in relation to the robot's front-facing orientation, and turned to the opposite side of the nearest object (i.e. if the robot encountered an obstacle which was closer to its left than its right, it would turn right). This led to a surprising amount of seemingly "intelligent" movement, and the robot was capable of following walls. However, the robot was rendered useless anytime corners were encountered or walls were approached head-on. I first attempted to solve this problem by adding functionality to the robot wherein the bot would decrease its velocity upon getting closer to a wall, eventually moving with a negative velocity, until it was safely far away from a wall and would continue to move forward with a positive velocity. However, this strategy was flawed, and caused my robot to collide with walls when reversing.

I eventually settled on a reactive design that incorporated (1) wall-following using the first aforementioned control mechanism, (2) collision avoidance, which causes the robot to fully stop and turn upon encountering a wall (showing better results than reversing), and (3) adaptive velocity, where the robot's linear velocity is directly proportional to the distance from the nearest detected obstacle in its field of view. This, in addition to a small degree of stochasticity when choosing direction, allows my robot to fully explore most areas of the map without getting permanently stuck in small corridors or corners. The robot explores the yellow portion of the map immediately, as that portion is the origin of the robot's exploration. The speed of the robot's exploration into the green and red zones depends upon the direction taken by the robot at the first wall. If the robot turns left, in the direction of the green zone, it takes between 3-4 seconds and 2-3 minutes for the bot to reach the green zone, and it takes many minutes for the bot to reach the red zone. Conversely, if the robot turns right, in the direction of the red zone, it takes between about 10-11 seconds and 2-3 minutes for the bot to reach the red zone, and about a minute for the robot to reach the green zone. One key change that was added to the control of the robot is the addition of a module I would call "turn commitment". This module ensures that upon encountering a corner or getting too close to a wall, the bot will stop and turn stochastically just as before, but as a novel constraint, the bot will "choose a direction". In other words, if the bot encounters a corner and decides to turn left, this module will now force the bot to keep turning left until it finds a clear path and can stop turning, moving forward. Without this module, the bot would stochastically sample a direction with which to turn from a uniform distribution and turn in that direction, continuing this process as frequently as the robot's callback function is called. This module allows the robot to more completely explore its space, but incurs a time cost as the robot takes a longer time to explore portions of the map. Finally, I added one control to bias the robot towards turning left, and this is done because of the specification that the robot only needs to fully explore the green area of the map. This makes the robot less general, which is undesirable, but because generality is not one of the desired characteristics of the bot, this is acceptable. Additionally, this means that the robot reaches the green area in 3-5 seconds on average, but it will take a very long time to reach the red zone (if it reaches that zone at all). Again, because this is not listed as a requirement for the bot, this should not be an issue, even if it makes the motion of the bot much more coupled to this particular task. In the future, this complicated reactive control should be split into separate publishers and

subscribers to take advantage of compartmentalization, single-responsibility, parallelism, and other important principles of good quality software and hardware engineering.