a. **A concise description of how the system works. This should be a couple of paragraphs, describing your overall approach at a conceptual level. Some questions to consider: What choices did you make in designing the program, and why?**

Our program consists of two nodes, the velocity translator node, “velocity_translator”, and the navigation controller node, “nav_controller”. We implemented a reactive controller for the nav_controller node to handle the immediate data from the surroundings and compute the robots subsequent actions. The velocity_translator node serves to take in this information of linear and angular velocities from the nav_controller node and interpret it into velocities for the left and right wheels.

The navigation controller node, nav_controller, made up the bulk of the work for this part of the project. Since our navigation controller was not able to subscribe to the “/map” topic we created a reactive controller that would subscribe to the “/scan” topic and use the most recent data in order to determine how to navigate through the environment. It does this by taking in laserscan messages from the “/scan” topic, containing information about distances from the robot to objects in front of it. We then use this information within our handle_scan() function to compute the instantaneous angular and linear velocities that the robot needs in order to avoid the relevant obstacles, publishing this information on the “/cmd_vel” topic. If there are no obstacles detected at a point close enough to the robot and in the same direction as the robot when the data from the “/scan” topic is analyzed then the robot continues to move with the same linear and angular velocities. This makes it such that it only adjusts its course when met with obstacles in real time in accordance with the idea of a reactive controller.

The velocity translator node, velocity_translator, has the single purpose of translating the desired velocities computed by the reactive controller into motor commands that can be sent to the left and right wheels of the robot. It does this by subscribing to the “/cmd_vel” topic where our navigation controller node publishes its computed linear and angular velocities in the form of Twist datatypes. It uses the linear.x and angular.z components of these Twist datatypes in order to compute the correct velocities for each wheel, using the formula learned in class early this semester.

b. **A short answer to these questions: Did the results meet your expectations? Why or why not?**

In our case the results met the expectations we held. Coming into the second part of this project we had a good idea of what the final result should look like due to discussion with professor O'kane in-class where he noted that his own solution took around 40 lines of python code, and that we didn't need to attempt to detect the end corridor after reaching the goal. This meant that we came in expecting a relatively succinct and straightforward implementation and our results met that expectation. Our final python code ended up being around 41 lines of actual code, and it passed all the test cases with flying colors. There are times when the robot sometimes gets very close to obstacles, however never actually runs into any, thus meeting our expectations.