### **Robot Code Documentation**

Project 2 consisted in the real-life implementation of the code produced in project 1 for an autonomous robot. This code carries out a set of behaviors for a turtlebot in room 300 at Felgar Hall. In this paper we will describe the reactive architecture that was chosen, why it was chosen, the data structures used, the algorithms implemented, and how our algorithms represent the chosen architectural style.

Rodney A. Brooks describes an architectural style in his paper “A Robust Layered Control System for a Mobile Robot” with reactive components which were used for the turtlebot’s reactive architecture. The control system for the turtlebot needs a set of requirements. For that reason we need an architecture with a control system with the following requirements: multiple goals, multiple sensors, robustness, and additivity [1]. Based on the instructions, it is known that the robot will have to perform multiple goals and determine which one is of highest priority. Therefore, to successfully accomplish the goals the robot must have a sensor suit that can gather necessary data.The control system should also be robust enough so that it respond to sensor errors. Lastly, the control system should also be designed to handle all the sensed data, otherwise, it will not be able to perform the behavior effectively; the implemented architecture is reactive and uses inhibitor flags to handle the hierarchy between the robot’s actions.

Based on the chosen architecture, the turtlebot control system uses bumper sensors, a laser sensor, and an odometer. The bumper sensors is used to detect if the robot collided with an object. The laser sensor is used to determine the distance between the robot and the objects around it. Lastly, the odometer is used to determine the distance traveled by the robot. All of these sensors are necessary since the tour robot must be able to perform the following actions: halting if a collision is detected by the bumpers, escaping from the symmetric obstacles within 1 ft in front of the robot, turning randomly after every 1 tf of forward movement, and driving forward.

The project’s code was written in Python. There was no need for any complex data structures due to the nature of the chosen reactive architecture. All the relevant data to perform the robot’s actions was stored using primitive data types such as boolean, double, integers, and strings - the last one being used for testing purposes. Lastly, the robot’s actions were represented and handled by using classes. That way we can create an object for each movement and then update it when necessary. These objects and variables are used with the different robot’s algorithms which will be explained in detailed throughout this paper.

The algorithm consists of having a main method that organizes the different turtlebot actions from highest priority to lowest priority. This way, the turtlebot will be able to prioritize one behavior over another. This goes in accordance with the reactive architecture since the robot’s actions are being handled based on sensed information from the environment.

The program starts by subscribing to the bump, laser scan, and odometry handlers to receive data from the robot’s. Consequently, an object is created for each of the following actions or action handlers: keyboard, turn, obstacle, and random turn. Then, the robot’s logic is set to be executed by calling the method with all the robot’s behaviors in the specified priority order. The robot can now be controlled using the keyboard but none of the inhibitors are activated when the program first starts. Therefore, the robot will just continue to move forward until it bumps into and object, get close enough to avoid an object, or the bot has moved 1 ft and needs to perform a random turn. Additionally, the sensor’s data will be constantly updated through the use of asynchronous calls.

The robot can be moved by pressing the “a” to move left, “d” to move right, “w” to move forward, “x” to move backwards, and “s” to stop movement.

The first behavior involves handling collisions detected by the bumper sensor. The turn handler object uses the linear velocity, angular velocity, current angle, and current direction of the turtlebot. All of these variables will then be used to calculate when to start the turn, stop the turn and continue the turn for the bot. These are accomplished by comparing the current angles and the turn’s new angles. Notice that once the robot has bumped into an obstacle it will ask the user to press the “x” key to backtrack and get away from the object.

The second behavior involves handling the avoidance of obstacles. The obstacle handler then uses the inhibitor variables - which will be set to true or false previously - to determine whether to escape or avoid an obstacle. The robot should avoid if an obstacle is located anywhere near the robot when the distance is 1.75 meters. On the other hand the bot is set to escape if the object can be located to be left and right at the same time.

Lastly, the third robot behavior should be able to perform random turns. The random turn handler uses a random maximum and minimum number of degrees to calculate how much and where to turn. These values were arbitrarily set to -15 and 15 to prevent the robot from moving to abruptly. The new turn degrees and turn radians were randomly generated using the minimum and maximum degrees as boundaries.

**References**

[1] R.A. Brooks. "A Robust Layered Control System for a Mobile Robot." MIT AI Lab

Memo No. 864, September 1985.