World and Launch File Documentation

## CS 4023/5023 Introduction to Intelligent Robotics

### Alexander Kloppenburg

### Eric Gonzalez

### William Cobb

**World File:** The world file is an XML file that contains the data that defines the environment that the turtlebot is located in.

The XML defines a light source that is directly above the room..The scene is defined and the appearance of the ambient and the background environments are given color values. The physics of the world (time simulation) are defined next. After that comes the ground plane that will support the turtlebot and everything else in the environment. The mobile base of the turtlebot is then defined. The objects in the environment are then defined. These objects can be static (not interactable) or non static (interactable). Static objects in the environment only have collision detection which allows another object to collide with it but doesn’t move. Non static objects can have many forces acting upon them and they are able to be moved. The XML file also defines the physical characteristics of non static objects within the environment. This includes the forces that act upon them such as gravity, friction, etc.

**Launch File**: Our launch file simply launches the project1 package and project1.py file. This python file handles all of the user input and other functions needed for this project.

**How to run/launch:**

1. Be sure to install ROS Kinetic and Gazebo before proceeding.
2. Be sure to refer to <http://wiki.ros.org/catkin/Tutorials/create_a_workspace> to set up catkin\_ws properly, and to source the workspace setup.
3. Download our project1 workspace into your local machine and place it in ~/catkin\_ws/src directory.
4. To run our world file, run the following command:

$ roslaunch turtlebot\_gazebo turtlebot\_world.launch world\_map:=<full path to .../catkin\_ws/src/project1/my\_world.sdf

1. (Optional, if mapping is desired), open a new terminal then run:

$ roslaunch turtlebot\_gazebo gmapping\_demo.launch

1. Now we need to execute the launch file. To do this, run:

$ roslaunch project1 project1.py

1. If you would like to save the map generated by this session, type:

$ rosrun map\_server map\_saver -f <your map name>

Robot Code Documentation

## CS 4023/5023 Introduction to Intelligent Robotics

### Alexander Kloppenburg

### Eric Gonzalez

### William Cobb

**Data Structures & algorithms:**

The code is contained in one python file named project1.py. This file contains all of the code to handle input and also handle collision avoidance and symmetrical object escape and handles any escape behavior respectively.

|  |  |
| --- | --- |
| **Function** | **Usage** |
| **\_\_init\_\_(**self**)** | The starting point of the program. This function is called when the project1.py file is executed. Sets up the boolean variables that will be used to detect conditions that require some sort of action. |
| **shutdown(**self**)** | Shuts down the instance of project1 that is currently running. It prints the message “Stop Turtlebot” to indicate it is stopping. |
| **turn(**self, angle, speed**)** | Turns the turtlebot. Uses the angle and speed parameters provided to accurately turn the turtlebot a given angle and direction. |
| **drive(**self, distance, speed**)** | Drives the turtlebot forward a given distance at a certain speed. This function will not proceed if the bumpers are pressed, a key is pressed, or if the turtlebot detects a symmetric or asymmetric obstacle. |
| **BumperCallback(**self, data**)** | This function detects when the bumper is pressed. |
| **LaserCallback(**self, data**)** | This function is used to detect if there is a symmetric or an asymmetric obstacle in the way of the turtlebot. |
| **on\_press(**self, key) | Detects when a key is pressed. Based on which key is pressed, it will change the velocity of the turtlebot in order to make the turtlebot move. |
| **on\_release(**self, key) | Detects when a key is released. Simply sets *keyPressed* to false. |

**Reactive architecture:**

Our group decided on a schema architecture. We chose this type of architecture because of the simple nature of the tasks we are performing in this iteration of the robot’s code. By separating the perceptual schema (the vision) from the motor schema (the motion), we were able to make the code more efficient and modular. This makes it easy to program the tasks that we want the robot to perform.