World File Documentation

The included world file used for this project's demonstration, included in /worlds/project1.world, is an XML-formatted file that contains all aspects of the robot's environment. It begins with the SDF version, 1.4. <light> sets the “mood” of the file, and it defines various graphic variables, such as <pose>, <range>, <cast_shadows>, and other lighting-related elements. <physics> includes the physics type (ode), as well as various other physics-related elements, such as <max_step_size>, the amount of distance covered by a single step from the robot and <real_time_factor>, which sets the speed of the simulation. The world (and therefore the world file) is absolutely overloaded with jersey barrier elements, which were chosen for their strength and modularity, as well as compatibility with the machines in Felgar Hall. Each <model> tag includes a pose (its position), a velocity (our jersey barriers do not attack), an acceleration, and a wrench. Aside from the jersey barriers, the only other elements in the scene are the ground_plane, which is necessary to keep the robot and the jersey barriers from descending into the pits of Hell, and the robot itself. <state world_name='default'> sets other environment variables, such as the simulated time, real time, and the wall time. <model name='mobile_base'> is unique compared to the other models; it includes <inertial> and <inertia>, which deal with its traveling physics.

For each model, including every single jersey barrier, the ground plane, and the mobile base, there are various <collision> tags, which allow for collisions from various angles of the model, as well as how the model will respond to such an unpleasant event, including its friction and maximum contacts.

Launch File Documentation

A launch file is provided, placed in /launch/project1.launch. In ROS, launch files are similar to startup scripts. Much simpler than world files, they serve as a guide to help ROS understand exactly how the world file and user commands around this world file should be handled. The launch file is also XML-formatted, all encapsulated in <launch>. There are various “arguments” in arg_name, which help ROS know what to launch. For this project, I have set the world_file argument to $(arg world_file); the dollar sign allows the user to specify different parameters when running the launch file. In this case, the world_file argument is user-defined, and when calling the launch file, they will need to also add “world_file:=[world file]”. Various other arguments are called for the base, battery, GUI, stacks, and 3d sensor. All of these are

set to the default environment variables in ROS; these arguments are telling ROS how each of these things should act. Following this is the <include> tag, which signals that other files supporting this launch file can be found in the given arguments. For instance, <include file="$(find_turtlebot_gazebo)/launch/includes/$(arg base).launch.xml"> specifies that another file is needed in the "includes" folder in this package.

To create an easy user experience, one need not directly call the launch file. I have created two scripts which can be run from the terminal. Following the instructions in the READ_ME file, one will find that it is only necessary to run "cd ~/Desktop/robotics-master; ./run_me.sh", then open a new terminal window and run "cd ~/Desktop/robotics-master; ./run_me2.sh". These scripts include the commands needed to initialize the project in full. Essentially, run_me creates the project using catkin_make, sources the included setup file, then runs the launch file, which opens Gazebo and displays the robot and surrounding world. run_me2 then runs the project code, allowing the robot to explore to its heart's content.