World and Launch File Documentation

World File

The world file for this project entails a relatively limited number of items. It includes a plane, upon which four objects are placed upon. These objects include a grey cylinder, a wooden bookcase, a white cube, and a large dumpster. All of these objects are of considerable size, being larger than the robot itself. These objects are located together in a shape of a circle. We found this would give the best map when using gmapping.

In addition, the world file contains a static light source referred to as “sun,” and also places the turtlebot2 slightly outside of the circle of objects. This world file also establishes the physics of the world, including gravity, velocity decay and collision, all of which remain unchanged from the default turtlebot world used in prior tutorials.

Launch File

The launch file for this project, simulation.launch, initiates our python node “major_project.py”. It then launches the turtlebot_world.launch file passing our unique world file, majorworld.world, as a parameter. It then launches the amcl_demo.launch file. While launching, it passes the paramaters for the map, majorworldmap.yaml, as well as the initialpose of the robot. Passing the intialpose is essential for ROS to accurately use the navigation stack to traverse to a certain spot. Finally, we launch the pocketsphinx.launch file so that we may use voice recognition with our project.

The first step in the launch file is to open a node for the package containing our major_project.py file, and identifying this node as required so that when this node is closed all other nodes in the launch file are also exited appropriately. This required tag ensures a safer and more appropriate cleanup before exit. Next, we establish that all nodes in the launch file utilize the same time scaling by setting the param “use_sim_time” to true. This ensures that the multiple nodes running stay in sync with each other in the simulation by providing a consistent time progression between them.

Next in the launch file is to include the majorworld.world file we created, so that the simulation utilizes the appropriate world at all times. This is done by providing an include tag that references the world file with respect to the turtlebot_gazebo file, which is equivalent running a “roscd major_project” after catkin setup is executed, placing the terminal in the root folder of the package.

The third step is to initiate the amcl_demo node which allows the turtlebot sensor data to be utilized to take advantage of ROS’s built in path planning and navigation. It utilizes the navigation stack with a pre-made image of a map. The robot uses the map to generate a costmap, which enables ROS to create a path to a specified point on the plane. It then traverses to the specified point.

Lastly, in order to allow the robot to be able to listen to commands from humans, we launch the pocketsphinx.launch file which should also be in the catkin/src directory. This

launches the voice recognition software as well as the library for recognizing words from the English language.

Launch Instructions

First, you have to install pocketsphinx. In order to do so, you must follow the instructions from the PocketSphinx Setup Guide.md file on the repository. The instructions are as follows:

1.) Open a new terminal and cd into major_project/pocket-sphinx-language-model/
2.) Ensure Dependencies are installed by:
    a. sudo apt-get install -y python python dev python-pip build-essential swig libpulse-dev git
    b. sudo apt-get install python pyaudio
3.) Install Pocketsphinx for python
    a. sudo pip install pocketsphinx
4.) Now use the following commands to create the directories required
    a. sudo mkdir /usr/share/pocketsphinx
    b. sudo mkdir /usr/share/pocketsphinx/model
    c. sudo mkdir /usr/share/pocketsphinx/model/hmm
    d. sudo mkdir /usr/share/pocketsphinx/model/hmm/en_US
5.) Finally copy the language model into the directory
    a. sudo cp -r hub4wsj_sc_8k/ /usr/share/pocketsphinx/model/hmm/en_US

In order to launch the turtlebot simulation with the appropriate behaviors and nodes the user runs the command “roslaunch major_project simulation.launch” after installing the program into the catkin workspace folder, sourcing the catkin setup script and then running a catkin_make from the catkin_ws directory.

In order to install the project, the submitted zip file should be in the “catkin_ws/src” directory. From here, navigate up to the “catkin_ws” directory and execute the command “source ./devel/setup.bash” (or appropriate script for your shell), then run “catkin_make.” Then utilizing the previous roslaunch command (roslaunch major_project simulation.launch) will utilize the launch file detailed previously in order to spin up all required nodes and parameters appropriately.

Shutdown Instuctions

In order to shutdown the ROS nodes in the appropriate method, open a new terminal and enter the command “rosnode kill major_project”. If you have issues launching again after this command is issued and the project is not running, execute “rosnode cleanup” and respond yes to a prompt if given. This is related to a minor issue with Gazebo that occasionally takes significant time to shut itself down. Also, you can press Control C to let ROS control closing down all the processes.

By killing the project using the rosnode kill command, the project is tidily stopped, as well as all related nodes in the project. In this way, we avoid issues of killing individual threads or nodes, but instead stop and exit all nodes and threads safely. As noted, the cleanup is occasionally needed if the project is to be run again quickly after shutdown, as Gazebo takes time to shutdown and a Gazebo instance still running will prevent the creation of a new one as attempted by the launch of the program.