Robot modelling & Gazebo simulation activities

Introduction

Robot modelling in ROS is an important skill to create your own model, as well as understand how other third-party models work.

Also, learning how to model your own robot and how to specify parameters for simulation would help you to create a more realistic simulation.

The activities will help the learner to get familiarised with URDF/Xacro modelling and how to load a model in simulation using Gazebo and RViz.

Description of the activities

The learner will find an incomplete model for a differential drive robot. The activities are planned to consolidate the knowledge of URDF/Xacro modelling and give the learner the opportunity to practice the tools and finish a working model for RViz and Gazebo simulation.

Inside the activity workspace folder, you will find the following folders:

- puzzlebot_control – Contains ROS control files
- puzzlebot_gazebo – Contains Robot model files
- puzzlebot_world – Contains worlds for gazebo

These three folders are needed to describe the working robot model and the world for gazebo simulation. For the robot modelling part, the only package used is puzzlebot_gazebo. Here is the list of files inside this package.

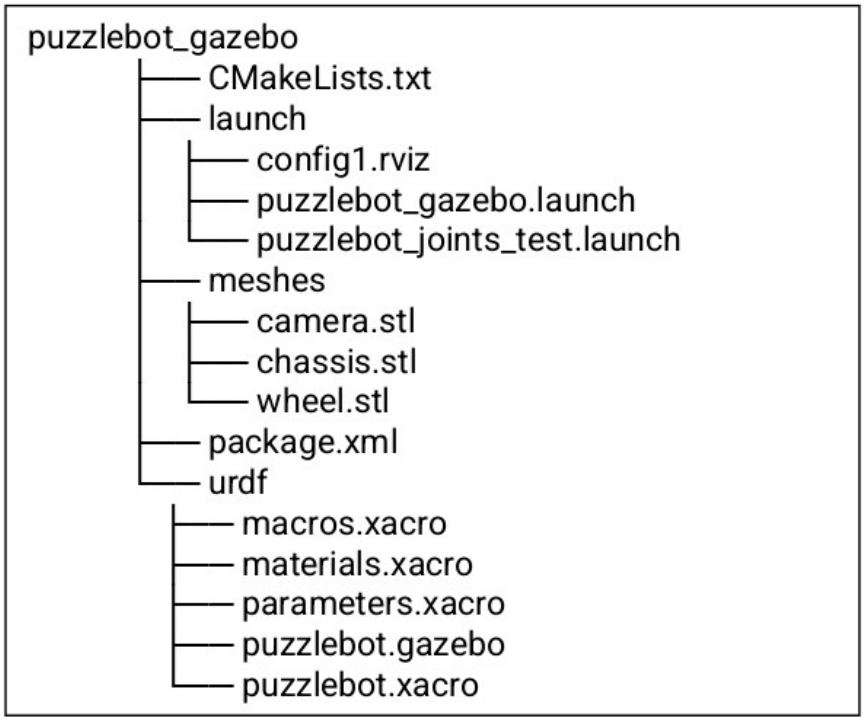

Diagram 1: List of files inside gazebo model

The URDF folder is comprised by several files that together describe the model. The file puzzlebot.xacro will be the main one. At the top of the mainfile, a few lines of code are used to include the rest of the files in the model.

A description of the files inside the robot model is shown in Diagram 2.

```
macros.xacro
        ├── materials.xacro  –This file includes materials, mainly colours.
        ├── parameters.xacro – This file includes any other constant to be used
for xacro
        ├── puzzlebot.gazebo – This file includes most of the Gazebo tags
        └── puzzlebot.xacro   - This is the main file with most of the URDF code
```

Diagram 2. List of files inside robot model and a brief description

Note: It is a standard and recommended as good practise to keep these files split into folders.

Warning: Do not modify any other file that is not specified in the activity, unless you are told to do so.

Activity #1: URDF modelling and Xacro simplification - (time 35 min):

Inside the puzzlebot_gazebo folder, run the launch file `puzzlebot_joints_test.launch` with the following command in the terminal.

```
roslaunch puzzlebot_gazebo puzzlebot_joints_test.launch
```

During the activity, run this command again to track your progress.

Before any changes have been made to the model, the simulation will display what is shown in Figure 1.

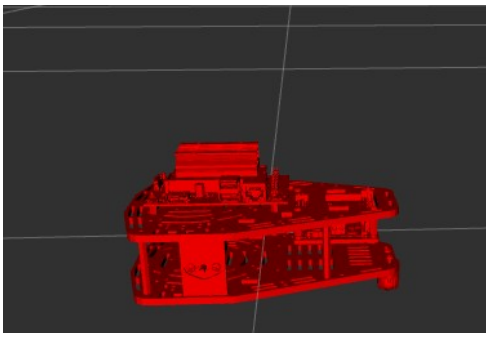

Figure 1. Incomplete model spawn in RViz.

Also, the command will pop up a window called Joint state publisher GUI, which will display a slider when a joint in fully created to test the operation. For the moment will look like Figure 2.

Figure 2: Joint state publisher GUI

This activity is divided into 4 mini sub-tasks:

1. Task 1a: Create one wheel for the robot. The dimensions are already defined in the parameters.xacro file. Use the defined variables and add the joint (the code for one joint is commented out).

2. Task 1b: Spawn both wheels using macros. Based on the code fragment written in task 1a, use xacro to create a macro for both wheels (see Note 2). This macro will add, using symmetry, a wheel on each side of the chassis. Then, add the remaining joint.

   Note: Joints can also be created using the macro for symmetry and constants, but for the purpose of this activity, this is hard-coded. Feel free to experiment and add it in your free time if interested.

   Note 2: Move the macro to the macro.xacro is the best practice.

3. Task 1c: Add meshes to both wheels.

   Hint: Identify where the meshes are (see diagram 1) and check how was added to the base_link link.

4. Task 1d: Check the wheels are movable with joint_state publisher_gui. Run the launch file mentioned above, load file config.rviz (see in diagram 1 where to find it). Once the robot is complete and the joints created, a pop-up window with a slider for each joint will appear. If everything is done correctly, the wheels will rotate when moving the slider.

Activity #2: Gazebo tags, and ROS Control and Gazebo simulation - (time 35 min) :

In this section, some Gazebo tags are added to make the model created in Activity 1 to make it compatible with Gazebo visualization and simulation.

1. Task 2a. Open gazebo interface with an empty world. Use the following command in a terminal window:

```
gazebo
```

This will open the gazebo software with an empty world. Then, add walls using the building editor to create a 4x4 metre room around the origin of the world. Then, add a bookshelf from the model library, and save the world as room.world in puzzlebot_world/world folder.

2. Task 2b. Add Gazebo tags, including the ros_control plugin. The code is mainly done for you. The full idea of this task is that you uncomment the necessary lines (in the main file, look for the include line of code for Gazebo tags and uncomment it) and get familiarized with the folders.

3. Task 2c. Add a transmission for each wheel. Transmissions are the link between the URDF/Xacro model and ROS control. Please, mind the name of the joints and the type of behaviour the transmission will find.

   Note: Add this in the main file for the URDF.

   Note 2: The provided Cheat Sheet has an example, so you can add one to each link.

4. Task 2d. Add ROS controller plugin. As mentioned during the presentation, ROS controllers include some basic already-made pieces of code, but also you can create one if required. For this task, the part of the code that will be used is one already done. So, it can load it and test it. For the sake of time, if you want to see a customised controller, please go to the full model of Puzzlebot Jetson Edition.

5. To test fully run the next command in terminal:

   ```
   roslaunch puzzlebot_gazebo spawn_puzzlebot_gazebo.launch
   ```