default

February 7, 2024

\#

```
<hr />
<h1 class="text-center">
    <span class="text-primary">Master Class 2023: Checkpoint 1</span>
</h1>
<hr />
```

`- Summary -`

For this project, you will have to build a mobile robot simulation from zero.

In the first part of the project, you will have to create the URDF of your robot, defining all the required links & joints.

Once you have the body of your robot ready, you will add to it the required actuators and sensors. Also, you will spawn the robot inside a Gazebo world in order to generate a robotics simulation.

Finally, you will create a simple ROS1 program to add to your robot some interesting functionality.

`- End of Summary -`

```
<h1 class="text-center">
    <span class="text-primary">Task 1</span>
     
    <span class="">Build RB1 URDF Replica</span>
</h1>
```

In Part1 of this project, you will put into practice what you've learned about Gazebo by creating a simple URDF file of a mobile robot. Find below the tasks that you need to do in order to complete it:

1. Create a new ROS package inside `/catkin_ws` named `my_rb1_description` with `rospy`, `urdf` and `xacro` as dependencies.
2. Inside the package, create a `urdf` folder. Inside this folder, create a file named `my_rb1_robot.urdf`.
3. Following what you learned in the course, fill out the urdf file to create a representation of the RB1 robot, with the following description.
    - The robot is a cilinder 50 cm in diameter and 30 cm in height
    - Has 2 wheels of 5 cm diameter on each side of the base. One at (0.0, -0.2, -0.15) from the `base_link` and another at (0.0, 0.2, -0.15).

- Has two caster wheels: one at the front and another at the back of the robot of 2.5 cm radius. The friction parameters must be ”0”. One at (-0.1, 0.0, -0.15) from the `base_link` and another at (0.1, 0.0, -0.15).

4. Check the following:
   - Make sure that you include visual and collision tags for the `base_link` link as well as for the wheels
   - The robot weighs 25 kg
   - Remember to compute the moments of inertia for all its parts. You can use the following page to remind you the basic formulas of the moment of inertia.
5. Add the following links:
   - `base_footprint` located at the center bottom of the base cylinder.
   - `base_link` located at the center of the rotation axis that connect the two wheels
   - `right_wheel`

   - `left_wheel`
   - `front_caster`
   - `back_caster`
   - `front_laser` link must be located at (0.25, 0.0, 0.075) from the `base_link`. Also the link must be rotated 180º in the `x` axis, this means, the laser is upside-down.
6. You will also need to create the corresponding joints for each one of the links above.

7. To visualize the robot, create a launch file (inside the `launch` folder) named `display.launch`:
   - Provide to the launch file the `my_rb1_robot.urdf` as a parameter
   - Start in the launch file the following nodes:
     - `joint_state_publisher_gui`
     - `robot_state_publisher`
     - `rviz`

You should be able to visualize your robot in rviz after launching.

HINT: To view the frames and links inside the robot, change the Alpha value in the RobotModel dropdown

`- Grading Guide -`

- When you launch the `display.launch` file you can visualize all the robot links correctly, as well as the TFs - **2.25 points**

Execute in Terminal

[ ]:
```
roslaunch my_rb1_description display.launch
```

- When you launch the `display.launch` file you can control the wheel joints through the `joint_state_publisher_gui` app - **0.75 points**

Execute in Terminal

[ ]:
```
roslaunch my_rb1_description display.launch
```

`- End Grading Guide -`

`<h1 class="text-center">`

Task 2

Spawn robot in simulation
</h1>

Now that the URDF is built, it can be used in Gazebo. For Task2, you will spawn the robot inside a Gazebo world in order to generate a simple simulation.

1. Create a new package in `catkin_ws` named `my_rb1_gazebo`. This package will contain only a `launch` folder
2. Inside this `launch` folder, duplicate the `empty_warehouse.launch` file from the following location:

`/home/user/simulation_ws/src/warehouse_robot_lab/rb1_base_sim/rb1_base_gazebo/launch`

Name the new launch file as `my_rb1_robot_warehouse.launch`

That file allows you to launch a simulation of the warehouse without any robot in it. You will have to modify it to include your robot.

3. Add to the launch file your urdf file as a parameter
4. Add also the `spawn_model` node from `gazebo_ros`
5. Make sure to also launch the `joint_state_publisher` and `robot_state_publisher` nodes
6. Launch the file using the following command. It should spawn your robot inside the warehouse world.

[ ]:
```
roslaunch my_rb1_gazebo my_rb1_robot_warehouse.launch
```

`- NOTES -`

Make sure that the `ROS_PACKAGE_PATH` of the terminal you are launching includes both `catkin_ws` and `simulation_ws`, otherwise the system won't find the needed packages:

[ ]:
```
export ROS_PACKAGE_PATH='/home/user/catkin_ws/src:/opt/ros/noetic/share:/home/user/simulation_ws/src'
```

`- END OF NOTES -`

The result of your launch should be something like this:

7. Now, change the spawning location of the robot to be in the middle of the *starting position*, that is the squared area delimited by the yellow-and-white tape.
8. Also, add the required gazebo color tags in order to make the robot black

`- Grading Guide -`

- When you launch the `my_rb1_robot_warehouse.launch` file the robot spawns in the simulation (inside the *starting position*) - **1.5 points**

Execute in Terminal

[ ]:
```
roslaunch my_rb1_gazebo my_rb1_robot_warehouse.launch
```

- End Grading Guide -

```
<h1 class="text-center">
    <span class="text-primary">Task 3</span>
     
    <span class="">Add sensor plugins</span>
</h1>
```

Now that the model is in Gazebo, you will add the actuators and sensors to the robot in order for it to control the motors of the robot and perceive its surroundings.

1. In `my_rb1_robot.urdf` you must add two plugins:
    - `libgazebo_ros_diff_drive.so` from `differential_drive_controller`
        - Linked to the model's wheel joints
        - Make sure to include the wheel separation and radius tags
    - `libgazebo_ros_laser.so` from `gazebo_ros_head_hokuyo_controller`
        - Attached to `front_laser` link
        - Publishing on topic `/scan`
2. Once the plugins are added, the model should be ready to start working. Check that you can move the robot with `/cmd_vel` and are able to see the `/scan` and `/odom` topics.

Great! You now have your own minimal version of the RB1

- Grading Guide -

- You can move the robot by sending velocities to the `/cmd_vel` topic - **0.75 points**
- You can read the laser data from the `/scan` topic - **0.75 points**

Execute in Terminal

[ ]:
```
roslaunch my_rb1_gazebo my_rb1_robot_warehouse.launch
```

- End Grading Guide -

```
<h1 class="text-center">
    <span class="text-primary">Task 4</span>
     
    <span class="">Control the robot with ROS</span>
</h1>
```

Now that the robot has its sensors and actuators working, it's time to control it with ROS! In Task 4, you will create a simple ROS service for your robot in order to provide a nice functionality to end users.

You will create a ROS Service that allows users to rotate the robot for an specific number of degrees.

1. Inside `catkin_ws`, create a new ROS1 package named `my_rb1_ros`.
2. Inside the new package, create a new Python C++ script named **`rotate_service.cpp`**.
3. The C++ program will contain a ROS1 node that provides a ROS Service named `/rotate_robot`. This service will make the robot rotate for a specific number of degrees (defined by the user).

4. The service will use a custom message named `Rotate.srv`. This message will contain the following:
    - A **request** that contains an integer field, named `degrees`, to specify the number of degrees to rotate
    - A **response** that contains an string, named `result`, that specifies if the rotation has been completed successfully or not.
5. Use the Odometry data from the `/odom` topic in order to compute the rotation of the robot.
6. Create a launch file named `rotate_service.launch` that starts your service server.

`- NOTES -`

The rotation angles are defined as indicated in the below image:

`- END OF NOTES -`

`- Grading Guide -`

- You can call the `/rotate_robot` service specifying the degrees in the request, and the robot rotates for the specified amount of degrees - **2 points**
- The program uses Odometry data (from the `/odom` topic) in order to compute the rotation of the robot - **1 point**
- When the rotation finishes, the Service returns a message indicating the rotation has been successful - **1 point**

Execute in Terminal

[ ]:
```
roslaunch my_rb1_ros rotate_service.launch
```

Execute in Terminal

[ ]:
```
rosservice call /rotate_robot "degrees: -90"
```

`- End Grading Guide -`