Task 2 - URDF

Due: Monday, 2 October 2023, 10:00 AM

Completion requirements

> **Disclaimer:**
>
> - **The tasks usually require a substantial amount of work, please start as early as possible**
> - **If you need help, you can google, or ask on the class forum, you can also attend the labs in person and ask a question directly to the teaching assistants**
> - **Before submitting, make sure your work compiles. Make sure you are not using external packages that we do not ask you to use, that you do not have hardcoded/absolute paths and that your build doesn't rely on previous builds, you can use "rm -r build install log" in your ros2 workspace for that. If your code does not compile, it's not your grader's responsibility to make it work.**
> - **Submission is at 10 am. The intention behind it is that you submit on the previous evening, but we still give you the possibility to work longer if you need to. No extension is allowed unless individually granted by Teaching Assistants.**
> - **Please don't forget to give us feedback on this task.**
> - **We always give you feedback on your previous task before you submit the next. Use this opportunity to correct your mistakes before submitting the next task.**
> - **Read until the end before starting.**

Overview of the task

In order to work on the robot software, we will use a robot in a virtual environment. This robot will be used through the entire course. To create the robot, we will use the URDF format (see below). This task will first require you to complete available ROS tutorials about URDF description, and then apply your knowledge to build your own robot.

Defining a robot: URDF

URDF stands for Unified Robot Description Format. In this exercise, you will learn to make a visual robot in URDF and add actuation, physical, and collision properties to it.
The URDF file you are about to create will define the robot's properties:

- structural: which components it consists of and how they are linked together
- visual: how each component looks like (geometry, position, color)
- physical: mass and inertia of each component, as well as their boundaries for collision detection

The robot you are going to create will accompany you through various lab tasks; future tasks will build upon previous ones, and this current task forms the base for all of them. The URDF file which you are going to create now will contain definitions that are not immediately used in task 2. For example, the mass/inertia, collision, and movable joint properties as well as the transmission tags are currently not used by the software (RViz) to visualize your robot. However, you should try to implement all the mentioned properties correctly already (although you are not yet able to test them all), because you will need them in the future. Implementing as much as possible right now will save you time when accomplishing future tasks which may be more complicated than the current one.
Before continuing, you can read the following conventions to avoid making mistakes in frames and units:

- REP-103 (Standard Units of Measure and Coordinate Conventions)
- REP-105 (Coordinate Frame Conventions).

Here is the list of ROS tutorials to complete

1. Building a Visual Robot Model with URDF from Scratch: Learn how to build a visual model of a robot that you can view in RViz.
   Please read also hints in the end of these descriptions.
   If you do not have urdf-tutorials installed install the package with:

   ```
   sudo apt install ros-humble-urdf-tutorial
   ```

2. Building a Movable Robot Model with URDF: Learn how to define movable joints in URDF.

3. Adding Physical and Collision Properties to a URDF Model: Learn how to add collision and inertial properties to links, and how to add joint dynamics to joints.
4. Using Xacro to Clean Up a URDF File: Learn some tricks to reduce the amount of code in a URDF file using Xacro.
   If you do not have xacro installed install the package with:

   ```
   sudo apt install ros-humble-xacro
   ```

Create your own robot

1. Now prepare for the creation of your own robot in URDF by following the instructions here:
   - Create a package (hereafter referred to as "your package") named **ias0220_<uni_number>** in your workspace (remember from your last assignment), where "uni_number" is your student id truncated to keep only the numbers (example: if my student ID is 189900IASB, my package name will be **ias0220_189900**).
   - Make a directory "urdf" in your ias0220_<uni_number> package and create a file inside it named differential_robot.urdf
   - Make a directory "launch" in your ias0220_<uni_number> package and put the task2.launch.py file in it (you will learn how to make such a file by yourself in the following exercise). In the launch file, replace everywhere you find package_name = "ias0220_<uni_number>" with your number so that the ros2 launch tool will look for your package's path.
   - Make a directory "config" in your ias0220_<uni_number> package and leave it empty for now.
     If you want to understand more about launch files here is the documentation about it: Ros2 launch
2. Based on the knowledge you got, create a differential-drive robot (see pictures below for a possible result) with the specifications laid out in the following:

   Mandatory specifications:
   - Create a dummy link named "**base_footprint**" which should be **below the centre of the base link and lying on the ground.** This link is an EXCEPTION from all others links since it should not have any visual, collision, or inertial property.
   - The base is named "**base_link**".
   - The **track** (distance between the centre of the wheels) is **350mm**.
   - The wheels are named "**left_wheel**" and "**right_wheel**" and their diameter is **72mm**.
   - The joints between the wheels and the base are named "**base_to_left_wheel**" and

"**base_to_right_wheel**", they are **rotational around the symmetry axis of the wheel** and both **wheels are coaxial with the Y axis of the robot's main body**. The z axis of each wheel points outward the robot (i.e., the z axis of both wheels are coaxial with the y axis of the base_link: the z axis of the left wheel goes in the sme direction as the y axis of the base, and the right wheel's z axis goes in the opposite direction) See picture below.
- All the other joints are fixed.
- You need to add a head on the base (blue here), to mark the front of the robot. The shape is free-choice, but keep the dimensions similar (between 20mm and 40mm for each length).
- You must add some passive support (white in the figure) so that the robot can stay standing up without control. Without this part, the robot will look normal in RViz but will fall over once we export it to a simulator (Gazebo). This support length has to be so that the robot is leveled on the two wheels and this support i.e. the robot base is parallel to the ground. The support should be made of two parts: a passive_wheel_holder and a passive_wheel. The joint defining the position of the passive_wheel should do it relative to the passive_wheel_holder, not base_link.
- Add collision and inertial properties to your link, AND orient them in the same way as your visual properties. A useful resource to compute inertia can be found here. They will be needed in future assignments. Don't forget: the inertia components are defined according to the centre of mass of your part, and are defined in the reference frame of the part. Hence, if the part is rotated, or if you put an offset in the origin of your part, you will need to take this rotation and offset in the inertia definition. More information here.
- IMPORTANT: Do not define the offsets of the different parts relative to each other using the <origin> tag of the links. Do it using the <origin> tag in the joints. Even though the former may lead to a working solution (if you do this offset in visual, collision and inertial as well, and take into account the different orientation of the parts) for parts that are fixed relative to each other, this will lead to wrongly defined joints, which will not work as expected. In summary: when you visualize your frames (TF) in Rviz, they should be located in the same places as on the figure below.

Additional information:

- All the colors are free of choice, and the shapes of the base (red here), the head and the support are free of choice, provided that they don't interfere with the main function of the robot: moving around.
- For reference only, we attached a figure with some dimensions and masses of our robot. You don't have to copy them even though you can. Don't hesitate to personalize yours. The only mandatory values are the ones stated above.
- If you wish, you can add a marker on the wheel to see better when it rotates (white rectangle in the image below), but it is up to you to add one, and if you do add one, you can make anything: different shape, replace the wheel by a STL shape (like the griper in the R2D2 tutorial), or add a texture to the wheel.

Hints

Hint 1:

You can check at any time how your robot looks like by typing ros2 launch ias0220_<uni_id> task2.launch.py (after doing colcon build)

This will launch the file task2.launch.py located in your launch directory within your ias0220_<uni_id> package. The launch file "task2.launch.py" instructs ROS to use the xacro package to read your URDF file, start the nodes joint_state_publisher (publishing the joint states), robot_state_publisher (publishing the transforms tf using the joint states) and RViz (displaying the robot model using tf). As you can see, launch files are a useful tool to start several nodes and read various configuration files without the need of extensive typing of multiple or complicated commands in the terminal. You will learn more about launch files in next week's lab.

If launching doesn't work: Did you run your colcon build on your workspace? Did you source your setup.bash file?

Sourcing command:

source ~/ros2_ws/install/setup.bash

Hint 2:

When RViz opens, you won't see anything because you need to tell RViz what you want to see. To do that, click add at the bottom of the window, and chose RobotModel. Repeat with TF to see the frames. Change fixed frame to an existing frame to see your robot. You can change the frames size under TF Marker Scale. You can save these settings by going into file-> save config as, and save it as

task2_config.rviz in ias0220_<uni_id>/config directory that you made earlier on. The launch file is instructing Rviz to use this file (you can check the syntax in the launch file if you want) as default configuration, so that you don't have to set all these details each time you launch this program.

Hint 3:

If you get a warning in RViz that the "map" frame doesn't exist, it is because this is true. Just set the fixed frame to "base_footprint" manually (click on the drop down menu in RViz).

Hint 4:

To check whether your robot has the correct wheels positioning, you can add this visual rectangle to your base_link in urdf file:

```
<visual>
  <geometry>
    <box size="0.0001 0.350 0.072"/>
  </geometry>
```

```
    </visual>
```

The short edges of the rectangle should end in the middle of the wheels (and be coincident with the wheel frames), and the height of the rectangle should be equal to the diameter of the wheels. You can modify "alpha" parameter in RVIZ to make the robot more or less transparent and see the rectangle better.

Similarly, you can add

```
    <visual>
      <geometry>
        <box size="0.350 0.0001  0.072"/>
      </geometry>
    </visual>
```

To check that your support wheel is at the correct height. It should be tangent to the bottom of the rectangle.

Hint 5:

If you cannot launch your robot visualization after inserting xacro syntax, check the second line of your urdf file and make sure you haven't redefined the robot name. You should have only one line starting with "<robot" here. Also, avoid using the minus sign outside/before the macro statement: don't write "-${...}", but instead write "${-(...)}". Otherwise you will get an error if the macro evaluates to something negative.

Hint 6:

You can check the tree presenting the structure of the frames to check whether the robot is structured as you intended by typing rosrun

rqt_tf_tree rqt_tf_tree. Below is the expected structure.

Hint 7:

During the R2D2 tutorial, you will see many errors similar to this one: "TIFFFieldWithTag: Internal error, unknown tag 0xa210." These are linked to the R2D2 model and can be safely ignored.

Summary:

In this task, you learned how to define a robot in URDF, which defines the robot physical, visual and structural properties. You then made your own robot, who so far has only two actuated joints (all other joints are static). This robot will follow you until the end of this course, congratulation for bringing it to life!
In the next task, you will start to actuate it. You will make it move using your keyboard and visualize it moving in Rviz.

Submission **(Don't forget to make sure your code compiles before submitting! Use catkin_make clean, and make sure you don't have any hard-coded path)**

Compressed (.zip) folder of your package, including:

- **Your URDF file, correctly named and placed in the correct directory.**

- **The screenshot robot_rviz_with_frames_task2.png in the data directory**
- **Your configuration file in /config**
- **All the rest that makes up a package, or that we gave you (setup.py, package.xml, launch file, ...)**

Please don't forget to give us feedback on this task! You get bonus points for it.

Submission should ideally be completed before the next lab. However, you still have time during that lab to ask some minor questions and make changes to your solution before submitting it, but be aware that if many students are in this situation, the teaching assistants may be very busy. Final deadline of submission is fixed, and if nothing is submitted by that time, your grade is 0. Extensions are allowed only after request, and the request should be made before deadline. Teaching assistants will decide whether to grant an extension.