Author:
Nikolay Prieto Ph.D(c)

Computational Laboratory
Building an Industrial Mobie Manipulator.

Contents

1. Objectives

- To understand how the ROS syntax works with the URDF.
- To build your own mobile and manipulator with ROS.
- To understand how physical variables are set in order to work with simulation environments as Gazebo.

2. Introduction

Mobile manipulators have been in the market for quite a while. Universities and research institutes initially began reusing their mobile robots and robot arms to improve dexterity. When ROS was gaining popularity in early 2007, PR2, a mobile manipulator link from Willow Garage (shown in the following photograph), was the testbed for testing a variety of ROS packages.

PR2 has the mobility to navigate like a human being rationally and the dexterity to manipulate objects in an environment. However, industries didn't prefer PR2 initially as it cost $400k to own one. Soon, mobile robot manufacturers began building robot arms onto their available mobile robot bases and this began gaining popularity due to its lower cost compared to PR2. Some of these well-known manufacturers are Fetch Robotics, Pal Robotics, Kuka, and many more.

Industries used to make use of articulated manipulators that were doing dull and dangerous repetitive tasks. Over time, those robots have grown modern and are capable of working alongside a human operator rather than alone in a work cell. Hence, a combination of such robots with industrial-grade ground vehicles help in certain industrial applications. One of the most common applications is machine tending. It is one of the most trending applications today and a lot of robots are getting deployed in this field. A machine tending robot is one in which certain tasks that help to "tend"to a machine are carried out by the robot. Some tending tasks are the loading and unloading of parts from a machine, assembly operations, meteorological inspections, and more.

Thus, the adoption of ROS as an universal standard framework to use with robotics is completely affirmative. In this tutorial you will learn to set your own differential drive robot and adopt a robot manipulator. In addition, you will be able to implement this model in the simulation environment.

3. Methods

3.1. Initial conditions

By now, you know what mobile manipulators are, what they constitute, and where they are used. Let's get into building one in simulation. As you are very well aware by now, a mobile manipulator would need a good payload mobile robot base and a robot arm, so let's begin building our mobile manipulator in terms of its parts and then combine them. Let's also consider certain parameters and constraints for building and simulating one. To avoid complexities in robot types and to account for a simple and effective simulation, let's consider the following assumptions:

- For a good payload mobile robot base:
    - The robot may move on a flat or inclined flat surface but not an irregular surface.
    - The robot may be a differential drive robot with fixed steering wheels and all wheels driven.
    - The target payload of the mobile robot is 50 kg.
- For a robot arm:
    - 5 DoF
    - The target payload of the robot arm is up to 5 kg.

3.2. Units and coordinate system

Before you begin building the mobile manipulator in Gazebo and ROS, you need to keep the units of measurement and coordinate conventions ROS follows in mind. Information like this is defined in design documents called ROS Enhancement Proposals (REPs). They act as standard references to the community members who use ROS while building their projects. Any new feature that's introduced or is planned to be introduced in ROS would be available as a proposal document for the community. The standard units of measurement and coordinate conventions are defined in REP-0103 link. You can find all the available lists of REPS in the REP index here: link.

As far as we're concerned, the following information is sufficient enough that we can go ahead with building our mobile manipulator.

- For units of measurement:
  - The base units: Length is in meters; mass is in kilograms; time is in seconds
  - The derived units: Angle is in radians; frequency is in hertz; force is in newtons
  - The kinematic-derived units: Linear velocity is in meters per second; angular velocity is in radians per second.
- For coordinate system conventions:
  - The right-hand thumb rule is followed, where the thumb is the z axis, the middle finger is the y axis, and the index finger is the x axis. Also, positive rotation of the z axis is anti-clockwise and the negative rotation of z is clockwise.

3.3. Gazebo and ROS assumptions

As we know, Gazebo is a physics simulation engine with ROS support. It works as a standalone without ROS as well. Most models that are created in Gazebo are in an XML format called Simulation Description Format (SDF). ROS has a different approach to representing robot models. They are defined in an XML format called **Universal Robotic Description Format (URDF)**. So, there is nothing to worry about here because, if the models are created in URDF, with some extra XML tags, they could be easily understood by Gazebo as they're converted automatically into SDF (because of those extra XML tags) under its hood. But if the models are defined in SDF, porting some of the robot's ROS-based features features might be a bit tricky.

There is support for a variety of SDF-based plugins that work or provide message information to ROS, but they're limited to few sensors and controllers. We have Gazebo-11 installed alongside our ROS1 Noetic. Although we have the latest Gazebo and ROS versions (at the time of writing this book), most `ros_controllers` are still not supported in SDF and we need to create custom controllers to make them work. Hence, we shall create robot models in URDF format and spawn it into Gazebo and allow the Gazebo's built-in APIs URDF2SDF do the job of conversion for us.

To achieve ROS integration with Gazebo, we need certain dependencies that would establish a connection between both and convert the ROS messages into Gazebo understandable information. We also need a framework that implements real-time-like `robot controllers` that help the robot move kinematically. The former constitutes the `gazebo_ros_pkgs` package, which is a bunch of ROS wrappers written to help Gazebo understand the ROS messages and services, and while latter constitutes the `ros_control` and `ros_controller` packages, which provide robot joint and actuator space conversions and ready-made controllers that control position, velocity, or effort (force). You can install them through these commands:

```
sudo apt install ros-noetic-ros-control
sudo apt install ros-noetic-ros-controllers
sudo apt install ros-noetic-Gazebo-ros-control
```

Note: Please replace the ROS distro if you have a different one as noetic.

We will be using the `hardware_interface::RobotHW` class from `ros_control` as it already has defined abstraction layers and `joint_trajectory_controller` and `diff_drive_controller` from `ros_controllers` for our robot arm and mobile base, respectively. More information about `ros_control` and `ros_controllers` can be found in this article.

4. Building the robot base

Let's begin by modeling our robot base. As we mentioned previously, ROS understands a robot in terms of URDF. URDF is a list of XML tags that contains all of the necessary information of the robot. Once the URDF for the robot base is created, we shall bring in the necessary connectors and wrappers around the code so that we can interact and communicate with a standalone physics simulator such as Gazebo. Let's see how the robot base is built step by step.

4.1. Robot base prerequisites

To build a robot base is needed the following: i) A good solid chassis with a good set of wheels with friction properties; ii) Powerful drives that can help carry the required payload and; iii) Drive controls.

In case you plan to build a real robot base, there are additional considerations you might need to look into, for instance, power management systems to run the robot efficiently for as long as you wish—the necessary electrical and embedded characteristics, and mechanical power transmission systems. What can help you get there is building a robot in ROS. Why, exactly? You would be able to emulate (actually, simulate, but if you tweak some parameters and apply real-time constraints, you could definitely emulate) a real working robot, as in the following examples: i) Your chassis and wheels would be defined with physical properties in URDF; ii) Your drives could be defined using Gazebo-ros plugins; iii) Your drive controls could be defined using `ros-controllers`.

4.2. Robot base specifications

Our robot base might need to carry a robot arm and some additional payload along with it. Also, our robot base should ensure it is electromechanically stable so that it has enough torque to pull its own load, along with the rated payload, and move smoothly with fewer jerks and with marginal pose error.

Let's consider the following specifications for our robot base:

- **Size:** Somewhere within 600 x 450 x 200 (L x B x H, all in mm)
- **Type:** Four-wheel differential drive robot Speed: Up to 1 m/s
- **Payload**: 50 kg (excluding the robot arm)

4.3. Robot base kinematics

Our robot base has only 2 degrees of freedom: a translation along the x axis and rotation along the z axis. Our robot cannot move instantaneously in the y axis due to the fixed steering wheel assumption. Since our robot moves only on the ground, it cannot translate in the z axis as well. I guess it is understood that a rotation along the x or y axes would mean that the robot either summersaults or topples; hence, it is not possible.

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos\omega\delta t & -\sin\omega\delta t & 0 & x - ICC_x & ICC_x \\ \sin\omega\delta t & \cos\omega\delta t & 0 & y - ICC_x & +ICC_y \\ 0 & 0 & 1 & \theta & \omega\delta t \end{bmatrix} \tag{1}$$

The unknown variables are as follows:

$$R = \frac{l}{2}\frac{n_l + n_r}{n_r - n_l} \tag{2}$$

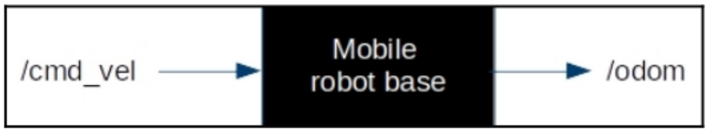

Figura 1: Inputs and outputs from the software point of view

Here, nl and nr are the encoder counts for the left and right wheels, and l is the length of the wheel axis:

$$ICC = [x - R\sin\theta, y + R\cos\theta] \tag{3}$$

and,

$$\omega\delta t = (n_r - n_l) * \text{step}/l \tag{4}$$

5. Software Parameters

Now that we have the robot specifications, let's learn about the ROS-related information we need to know of while building a robot arm. Let's consider the mobile robot base as a black box: if you give it specific velocity, the robot base should move and, in turn, give the position it has moved to. In ROS terms, the mobile robot takes in information through a topic called `/cmd_vel` (command velocity) and gives out `/odom` (odometery). A simple representation is shown as follows:

5.1. ROS messages format

`/cmd-vel` is of the `geometry_msgs/Twist` message format. The message structure can be found in the following link `/odom` is of the `nav_msgs/Odometry` message format. The message structure can be found in the following link Not all the fields are necessary in the case of our robot base since our robot is a 2 degrees of freedom robot.

5.2. ROS controllers

We would define the robot base's differential kinematics model using the `diff_drive_controller` plugin. This plugin defines the robot equation we saw earlier. It helps our robot to move in space. More information about this controller is available at the website.

5.3. Modeling the robot base

Now that we have all the necessary information about the robot, let's get straight into modeling the robot. The robot model we are going to build is as follows:

There is something you need to know before you come out with thoughts about modeling robots using URDF. You could make use of the geometric tags that define standard shapes such as cylinder, sphere, and boxes, but you cannot model complicated geometries or style them. These can be done using third-party software, for example, sophisticated Computer Aided Design (CAD) software such as Creo or Solidworks or using open source modelers such as Blender, FreeCAD, or Meshlab. Once they are modeled, they're are imported as meshes. The models in this book are modeled by such open source modelers and imported into URDFs as

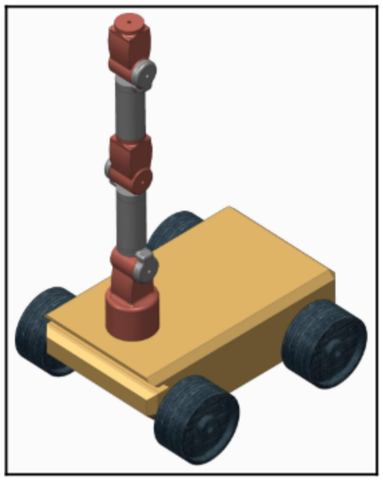

Figura 2: Mobile manipulator to control.

meshes. Also, writing a lot of XML tags sometimes becomes cumbersome and we might get lost while building intricate robots. Hence, we shall make use of macros in URDF called xacro (`http://wiki.ros.org/xacro`), which will help to reduce our lines of code for simplification and to avoid the repetition of tags.

Our robot base model will need the following tags:

- `<xacro>`: To help define macros for reuse.
- `<links>`: To contain the geometric representations of the robot and visual information.
- `<inertial>`: To contain the mass and moment of inertia of the links.
- `<joints>`: To contain connections between the links with constraint definitions ¡Gazebo¿: To contain plugins to establish a connection between Gazebo and ROS, along with simulation properties.

The chassis is named `base_link` and you can see the coordinate system in its center. Wheels (or `wheel_frames`) are placed with respect to the `base_link` frame. You can see that, as per our REP, the model follows the right-hand rule in the coordinate system. You can now make out that the forward direction of the robot will always be toward the x axis and that the rotation of the robot is around the z axis. Also, note that the wheels rotate around the y axis with respect to its frame of reference (you shall see this reference in the code in the next section).

```
source /opt/ros/noetic/setup.bash
$ mkdir -p ~/chapter3_ws/src
catkin_init_workspace
cd ~/chapter3_ws/src
catkin_create_pkg robot_description catkin
cd ~/chapter3_ws/ $ catkin_make
cd ~/chapter3_ws/src/robot_description/
```

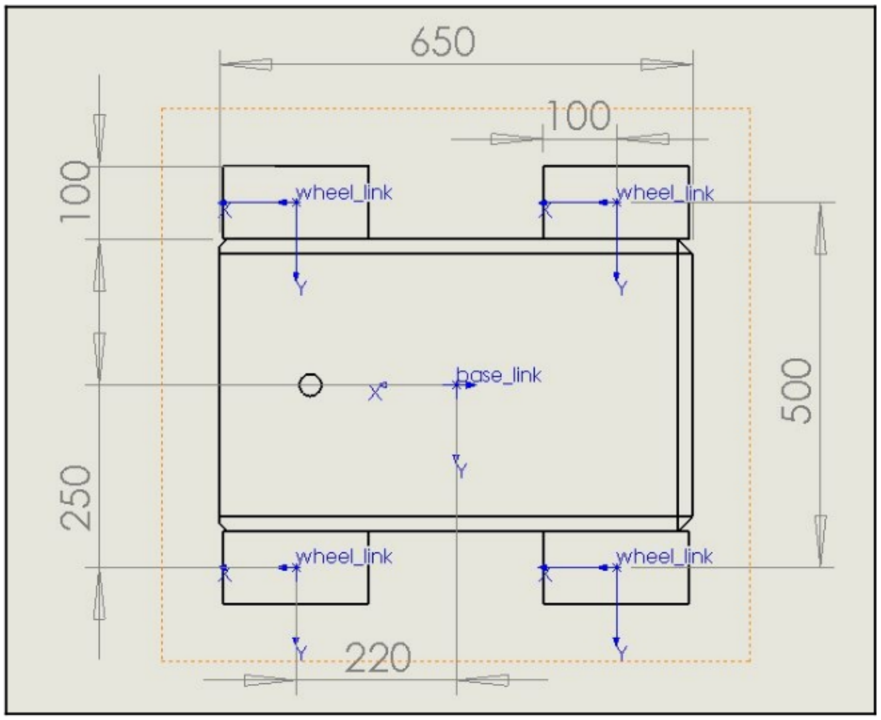

Figura 3: General measurement of the mobile base.