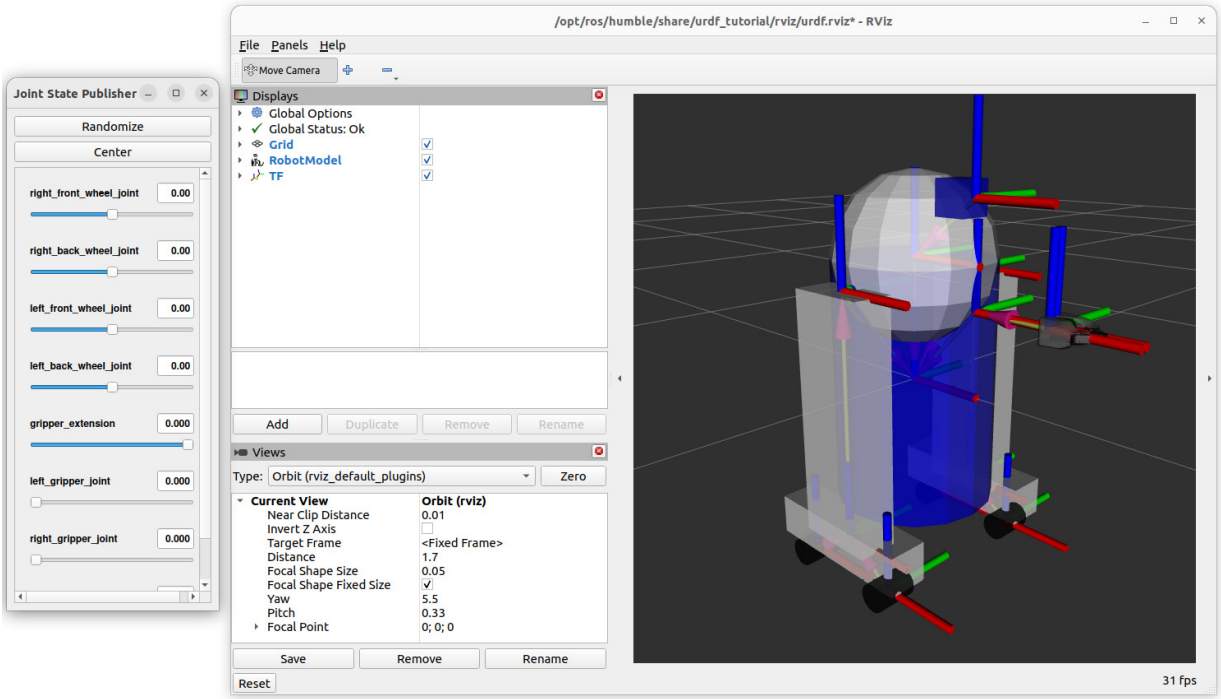

Figure 10.1 – A robot model on RViz

You will get two windows: the main one (**RViz**) with the robot model, and a **Joint State Publisher** window with some cursors. The robot model we have here is a replica of a famous science-fiction movie robot. It has some wheels, a torso, a head, and a gripper.

Let's focus on the main window (**RViz**) for now. Take some time to learn how to navigate in the 3D space and move around the robot. You can use the left click, right click, and mouse wheel. For this, it's best to have a mouse, but you could still manage to navigate with the touchpad of a laptop, although it's less ergonomic.

You can also resize the window and the various sections inside RViz. Pretty much everything you see can be customized. Now that you can load a robot model in RViz, we will start to experiment with TFs.

What are TFs?

There are two main parts in a robot model: links and TFs. In this section, we will visualize them both and understand how they work together.

Let's start with links.

Links

Have a look at the menu on the left side of the **RViz** window. There, you will see, in blue bold letters, **RobotModel** and **TF**. This is what we will focus on in this chapter. As you can see, you can enable or disable both menus.

Disable **TF**, keep **RobotModel**, and expand the menu. There, you can find a submenu called **Links**.

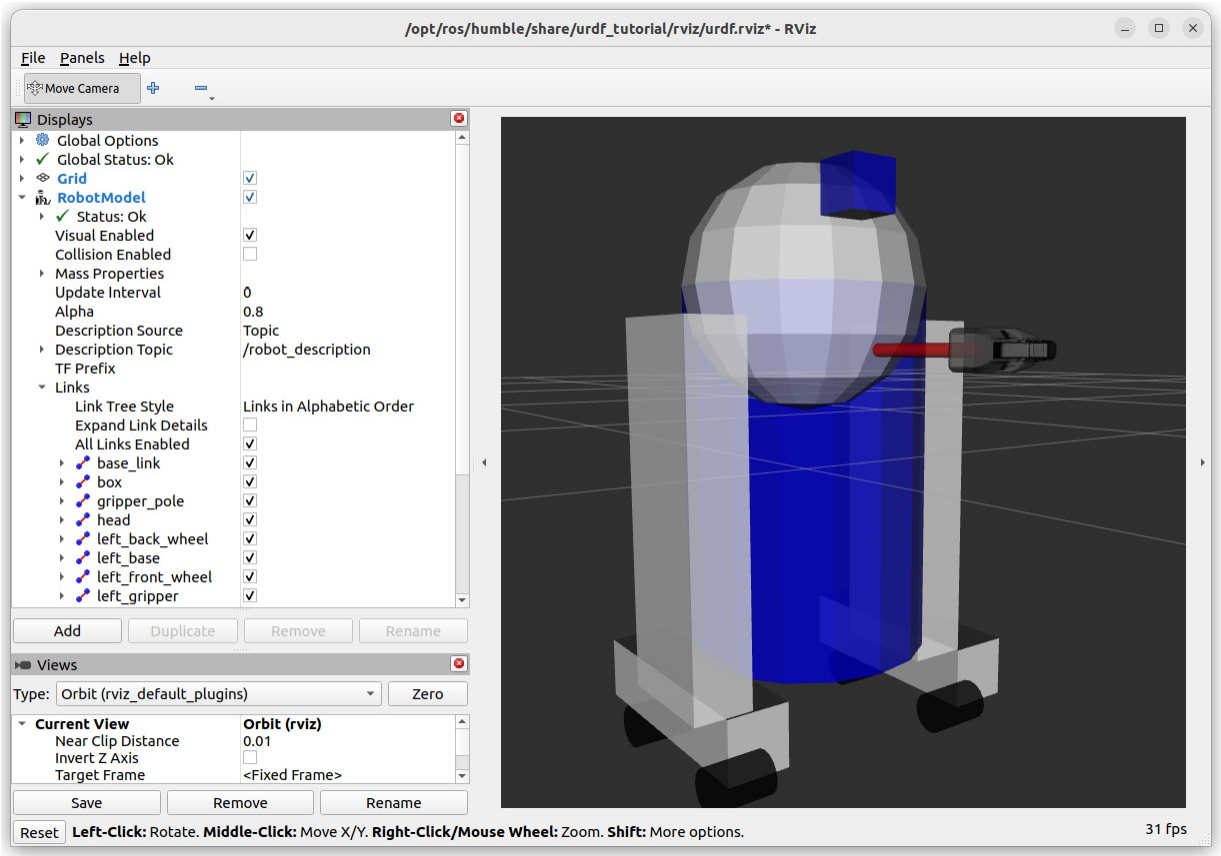

Figure 10.2 – RobotModel with Links menu on RViz

Check and uncheck some boxes. As you can see from this menu, a *link* is one rigid part (meaning one solid part with no articulation) of the robot. Basically, in ROS, a robot model will consist of a collection of rigid parts put together.

In this example, links are represented by basic shapes: boxes, cylinders, and spheres. Those rigid parts do nothing on their own, so how are they connected, and how do they move between each other?

This is where we introduce TFs.

TFs

Let's now check the **TF** box. You can keep **RobotModel** checked or unchecked. Inside the **TF** menu, you have a submenu called **Frames**, and you can also enable or disable each frame for the robot.

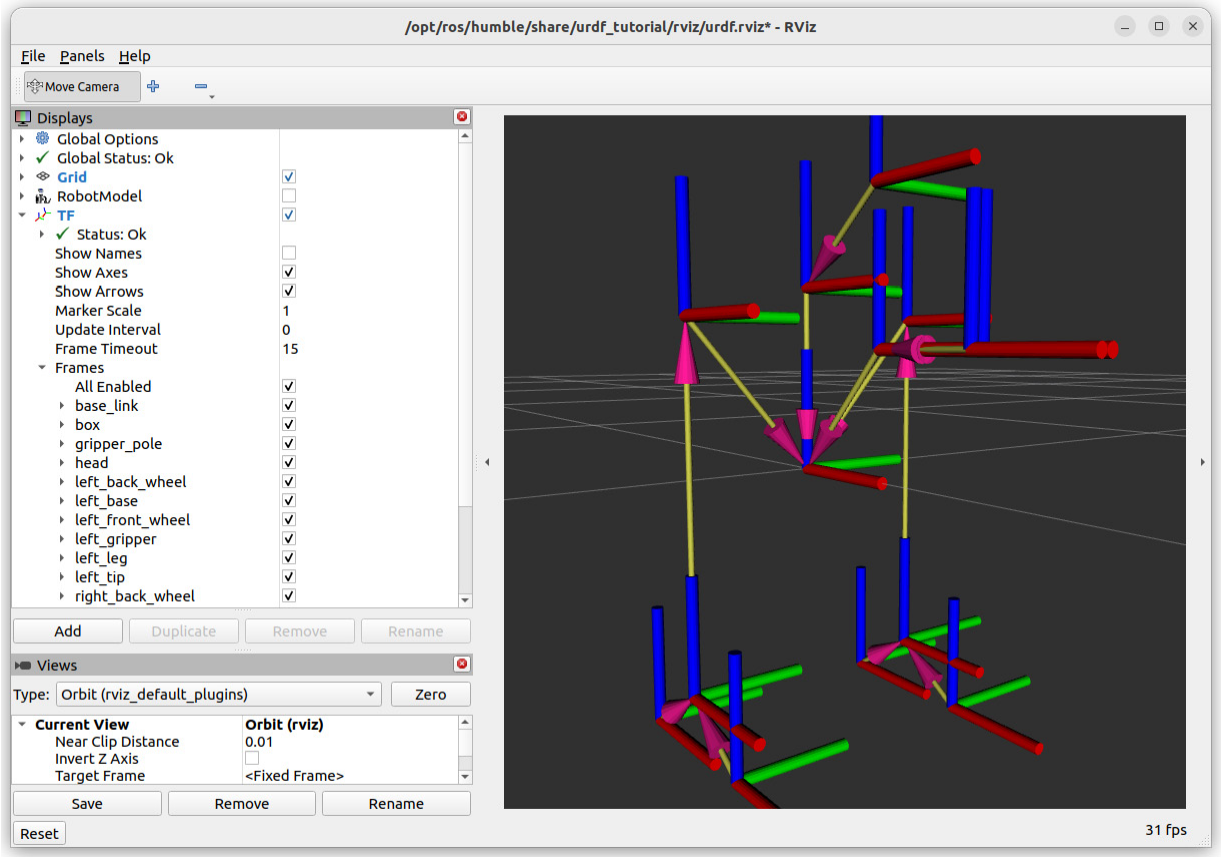

Figure 10.3 – Frames and TFs on RViz

The axes you see here (red, green, and blue coordinate systems) represent the frames, or basically the origin of each link of the robot.

Coordinate systems follow the right-hand rule in ROS. Following *Figure 10.4*, you have the following:

- X axis (red) pointing forward
- Y axis (green) pointing 90 degrees to the left
- Z axis (blue) pointing up

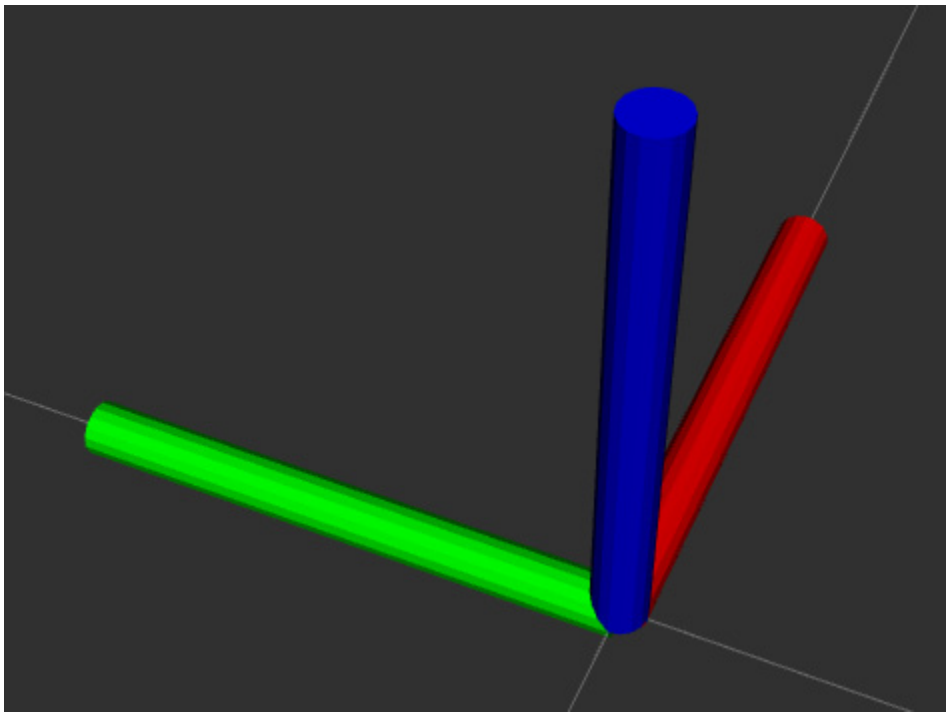

Figure 10.4 – Convention for coordinate systems in ROS

The arrows that you see between each frame in *Figure 10.3* are the relationship between each rigid part (link) of the robot. A TF is represented by an arrow.

> ***NOTE***
>
> *There can be some confusion between the names **links**, **frames**, and **TFs**. Let's make things clear:*
> *- Link: one rigid part of a robot*
> *- Frame: the origin of a link (axis in RViz)*
> *- TF: the relationship between two frames (arrow in RViz)*

So, each rigid part will be linked to another rigid part, thanks to a TF. This transformation defines how those two parts are placed relative to each other. In addition to that, the TF also defines whether the two parts are moving, and if so, how—translation, rotation, and so on.

To make some parts of the robot move, you can move some of the cursors in the **Joint State Publisher** window. You will see the frames and TFs moving in RViz. If you also check the **RobotModel** box again, you will see the rigid parts moving as well.

To better understand, here is an analogy with the human arm: we can define the parts of the arm as *arm* (shoulder to elbow) and *forearm* (after the elbow). Those two are rigid parts (here, links) and do not move on their own. Each link has an origin frame, and there is a TF that defines where the arm

and forearm are connected (imagine an axis in the elbow), and how they move (in this case, it's a rotation with a minimum and maximum angle).

As we will see later in this chapter, TFs are tremendously important. If the TFs for a robot are not correctly defined, then nothing will work.

Now you know what TFs are, but how are they all related to each other? As you can see on RViz, it seems that TFs are organized in a certain way. Let's go one step further and understand what the relationship between TFs is.

Relationship between TFs

In RViz, we have seen the links (rigid parts) and TFs (connections between the links). Links are mostly used for visual aspects in simulation and will be useful to define inertial and collision properties (when we work with Gazebo). TFs define how links are connected, and how they move between each other.

In addition to that, all the TFs for a robot are organized in a particular way, inside a tree. Let's explore the relationship between TFs and visualize the TF tree for the robot we started on RViz.

Parent and child

Each TF will be connected to another TF, with a parent/child relationship. To see one, you can, for example, disable all TFs on RViz, and only check the `base_link` and `gripper_pole` frames.

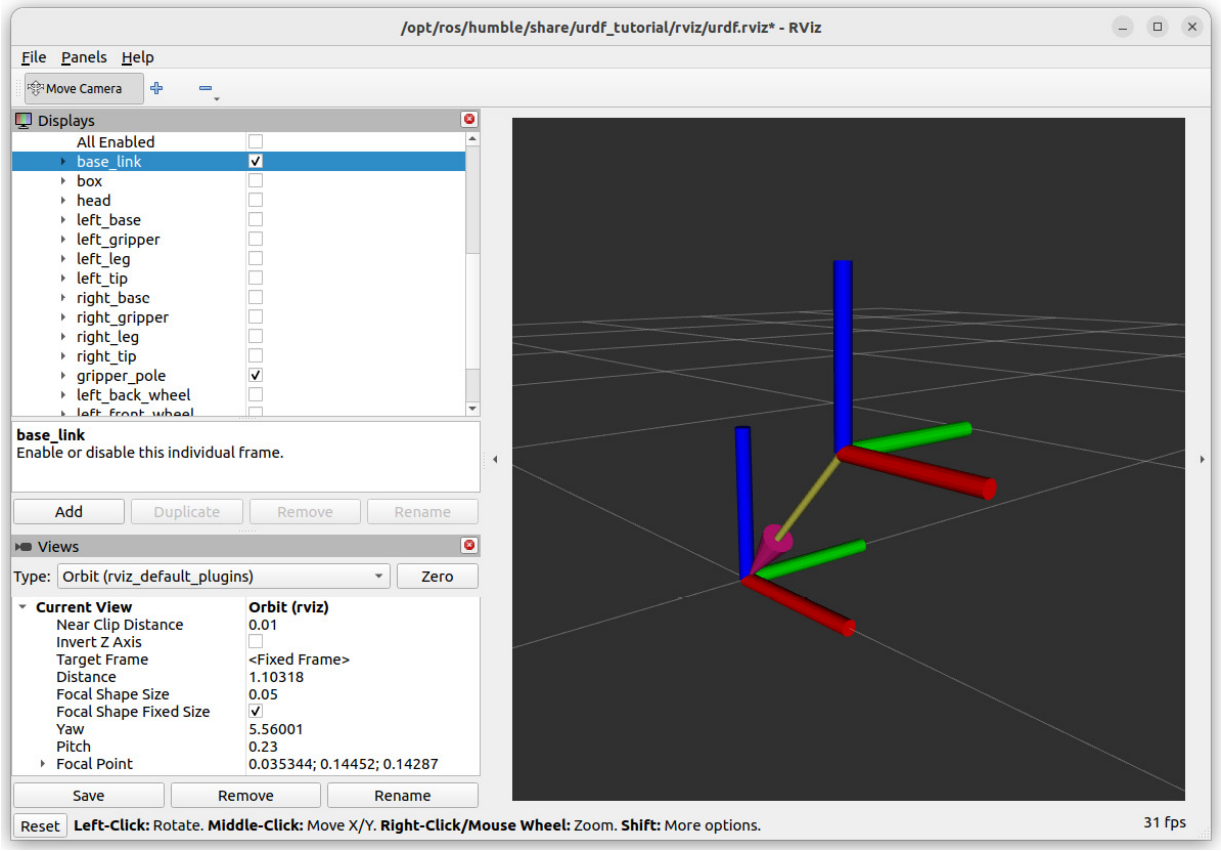

Figure 10.5 – The relationship between two frames

As you can see in this example, an arrow is going from the `gripper_pole` frame to the `base_link` frame. This means that `gripper_pole` is the child of `base_link` (or, `base_link` is the parent of `gripper_pole`).

If you look back at *Figure 10.3*, you can see all the frames for the robot, with all the relationships between them (TFs).

The order of those relationships is very important. If you move `gripper_pole` relative to `base_link` (the `gripper_extension` cursor in the **Joint State Publisher** window), then anything that's attached to `gripper_pole` (meaning children of `gripper_pole`) will also move with it.

That makes sense: when you rotate your elbow, your forearm is moving, but also your wrist, hand, and fingers. They don't move relative to the forearm, but as they are attached to it, they move relative to the arm.

Now, you can visualize all the links and TFs on RViz, see the relationship between TFs, and also how they are related to each other. Let's go further with the `/tf` topic.

The /tf topic

At this point, you might think that what we did in *Part 2* of this book has nothing to do with what we are doing now. Well, everything we have seen here is still based on nodes, topics, and so on.

Let's list all nodes:

```
$ ros2 node list
/joint_state_publisher
/robot_state_publisher
/rviz
/transform_listener_impl_5a530d0a8740
```

You can see that RViz is actually started as a node (`rviz`). We also have the `joint_state_publisher` and `robot_state_publisher` nodes, which we will come back to in the following chapters of this book. Now, let's list all topics:

```
$ ros2 topic list
/joint_states
/parameter_events
/robot_description
/rosout
/tf
/tf_static
```

You can see that the nodes that were started are using topics to communicate with each other. In this topic list, we find the `/tf` topic. This topic will be there for any robot you create. The TFs you saw on RViz are actually the 3D visualization of this topic—meaning the `rviz` node is a subscriber to the `/tf` topic.

You can subscribe to the topic from the terminal with the following:

```
$ ros2 topic echo /tf
```

If you do so, you will receive lots of messages. Here is an extract:

```
transforms:
- header:
        stamp:
        sec: 1719581158
        nanosec: 318170246
        frame_id: base_link
  child_frame_id: gripper_pole
  transform:
        translation:
        x: 0.19
        y: 0.0
        z: 0.2
        rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
```

This extract matches what we previously saw on RViz. It represents the transformation between `base_link` and `gripper_pole`. Here are the important pieces of information we can get from this

message:

- Timestamp for the TF
- Parent and child frame IDs
- The actual transformation, with a translation and a rotation

> ***NOTE***
>
> *The rotation is not represented by Euler angles (**x**, **y**, **z**), but by a quaternion (**x**, **y**, **z**, **w**). Quaternions are usually better suited for computers, but it's difficult for humans to visualize them. Do not worry about this—we won't really have to deal with quaternions. If you ever have to do so in the future, you will have access to libraries that can convert angles into something you can understand.*

One important thing we can get here is that the transformation is for a specific time. It means that with the topic data, you can follow the evolution of all TFs over time. You can know where `gripper_pole` is relative to `base_link`, now or in the past.

This `/tf` topic contains all the information we need, but it's not really human-readable. That's why we started with RViz, so we could see a 3D view with all the TFs.

Let's now finish this section by printing the TF tree, so we can see all the relationships in one single image.

Visualizing the TF tree

For each robot, you can visualize the complete TF tree in a simplified way, so that you can see all the relationships between all the TFs.

To do that, you will need to use the `tf2_tools` package. Make sure it is installed:

```
$ sudo apt install ros-<distro>-tf2-tools
```

Don't forget to source the environment after installing the package. Now, keep the robot running on RViz, and execute this command in a second terminal:

```
$ ros2 run tf2_tools view_frames
```

As you will see with the logs, it will listen to the `/tf` topic for five seconds. After this, the command exits with a big log that you can ignore.

You will get two new files, in the same directory as where you ran the command.

Open the PDF file. You will see something like this (I'm just adding the left side of the image, otherwise the text would be too small to read in the book):

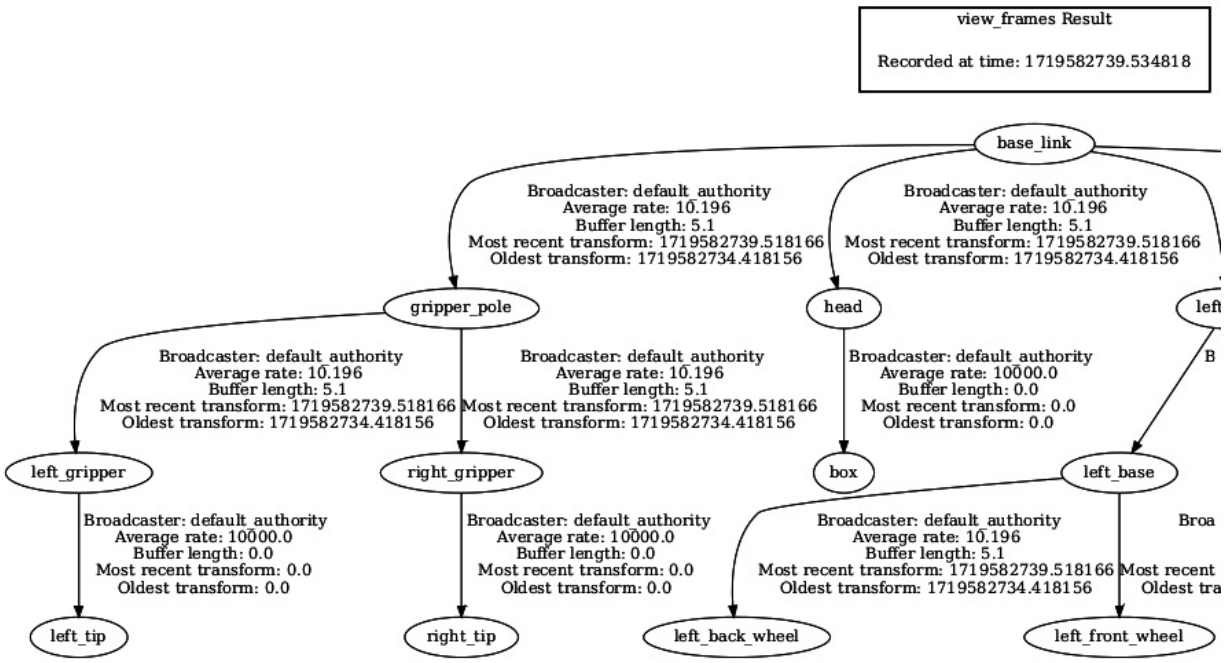

Figure 10.6 – TF tree for a robot

In this file, you get all the links and TFs at once, and it's quite clear to see which link is a child of which other link. Each arrow on the PDF represents a TF between the links.

As you can see, the root link for that robot is named `base_link` (the name `base_link` is used for most robots as the first link). This link has four children: `gripper_pole`, `head`, `left_leg`, and `right_leg`. Then, those links also get more children. Here, we can clearly see all the children for the `gripper_pole` link.

We can now understand that when we previously moved `gripper_pole` relative to `base_link`, then all children of `gripper_pole` were also moved relative to `base_link`.

> ***NOTE***
>
> *In ROS, one link can have several children, but only one parent. We will come back to this in the next chapter when we define the links and TFs ourselves.*

In this example, we have just one robot. If you were to have several robots in your application, then you would have, for example, a `world` frame as the root link. Then, this frame would have several children: `base_link1`, `base_link2`, and so on. Each robot base link would be connected to the `world` frame. Thus, you can get a complete TF tree, not just for one robot but also for a complete robotics application with several robots.