URDF Introduction - Part 1

Welcome to this first video about URDF, or the Unified Robot Description Format.

If you've heard about ROS before, you may also have heard about URDF.

You may have heard that URDF is the format ROS uses to store robot models in or that URDF is the simulation format of ROS.

You may have even found posts on the internet claiming that if you're using ROS and you don't have a URDF you're doing it wrong somehow.

Even though that last expression is a bit strong, it's certainly true that URDF is a central piece of technology in ROS and is used in many of its packages.

And while it is definitely possible to use ROS without a URDF, it will make many parts of it not function optimally or at all.

So what is URDF really?

Technically, URDF is a domain specific modeling language based on XML, that allows us to encode the kinematics, some parts of the dynamics and various other pieces of metadata related to a robot in a format that is readable and writable to a certain extent by both humans and machines.

That's rather formal, so to put this in other words: URDF is the file format ROS uses to describe the layout of the body of a robot with all its links, joints, shapes and colors.

It also allows us to store pieces of extra information, such as the range of motion of all joints in the robot, and how fast it can perform those motions.

It is a domain specific language because it uses names and terminology specific to the robotics and robot modeling domains. This allows us to quickly create robot models using the actual names of the parts of a robot instead of having to resort to generic descriptions.

On the implementation level a URDF file is just a text file containing XML tags: specific keywords that ROS recognizes as being part of URDF.

Some of those XML tags refer to other files or 3D models of parts of robots, which allows us to make use of external files to quickly incorporate a detailed shape complete with colors from a 3D mesh file, such as a upper or lower arm.

So now that we know what URDF is and what URDF files are, we can take a look at what we actually model with such a file.

The bulk of a URDF file is made up of elements called links and joints.

These directly correspond to the links and joints of a robot:

links give robots their shape;

joints connect links and determine how they may move with respect to each other.

So a robot in URDF is a set of links connected by joints in a certain order.

If the order changes, the motions a robot can make change as well, so not only the body layout changes by changing the way joints connect links, but also the behavior of the robot.

Let's look at a tiny example of a URDF:

It models this very small robot called "tiny_robot".

Something like this.

This is of course not really a robot, it's just a single link, but it *is* a valid URDF at just 3 lines long.

There are no joints here, so this cannot move and also no references to 3D meshes, so it will not have an appearance in RViz.

Let's add a second link and a joint, to make it a little bit more interesting.

'link_1' is the parent of 'link_2', and they are connected by 'joint_1'.

Schematically, that looks like this.

Still no references to any 3D shapes here, so this robot will not look like much in RViz.

Two dots is actually not a valid 'joint type', and we must specify one or ROS will not know what to do with our URDF.

So while there is only one type of link, there are as many as 6 types of joints supported by URDF.

The first are fixed joints: these rigidly connect the parent and child, making motion impossible.

The second are revolute joints: these allow rotation of the child with respect to the parent, but only in one dimension. They are commonly used to model industrial robots.

The third are continuous joints: these are the same as revolute but without any limits. They can rotate an infinite amount of times.

The fourth type is prismatic: only translation of the child with respect to the parent in one dimension is allowed.

Fifth are planar joints: these are 2 dimensional variants of prismatic joints.

And the sixth and last type are floating joints:

these allow motion and rotation in all directions without any limits whatsoever. Connecting a drone to the rest of a URDF world could be done with this type of joint.

We've mostly talked about what a URDF file is and the types of elements that go in it.

However, another important aspect of URDF is that it enforces a number of conventions that help align the workflow and expectations of the global but distributed group of ROS users and developers.

The two conventions I mention here come from a ROS standards document called REP 103, or "Standard Units of Measure and Coordinate Conventions".

There are many more and if you're interested check some of the links we've added on the course page for this video.

The first convention is that ROS uses a right-handed coordinate system, everywhere. In ROS, positive X points forwards and positive Y points left, so that positive Z must point upward.

The second is that ROS uses SI units and derived SI units for everything that represents a physical quantity. So lengths or distances are expressed in meters, and angles or rotations are expressed in radians.

As radians are not the most intuitive to most of us, conversion functions are often available, so we don't have to take out our calculators whenever we need to specify an angle.

That's it for the first part of this presentation.

Summarizing:

URDF is a text format based on XML that allows us to model robots using links and joints;

various joint types are supported;

links can be given an appearance using 3D mesh files;

ROS uses meters and radians for specifying lengths and angles.

Now as you've already heard in the introduction video, we're going to make use of external resources for some of the exercises and tutorials in this week.

With URDF demystified a little bit, head over to the course page for this video and follow the two tutorials we've selected for you and try creating a robot with URDF yourself.

Be sure to post on the forum if something is unclear or if you're stuck with something.

Once you're done, we'll continue with part 2.