URDF Introduction Part 2

Good to see you back and I hope that you succeeded in completing the first two tutorials.

While right now it may seem like you could model anything in URDF using links and joints and 3D files, in this video we'll take a look at some of the shortcomings of URDF and see that is not necessarily the case.

Even though URDF is a quite OK system to model robots in, it's far from perfect and it would be good to already be aware of some of its limitations.

I'm going to discuss four of these limitations today, namely:

The immutability of robot descriptions.

The fact that closed-loop systems cannot be modelled.

The lack of a proper sensor tag in URDF,

and the low reusability of URDF models and how that influences their composability.

Let's look at the first one, the immutability.

URDFs are essentially immutable, meaning that once they've been loaded on the parameter server, they cannot be changed.

While to be honest this is not directly a problem with URDF itself, but more with the way it is used and stored and read from the ROS parameter server, this is a serious limitation that can be annoying to discover.

Adding a new link or joint to the model of a robot while ROS nodes are running is certainly possible it's a dangerous thing to do, as there is no standardized way of notifying all consumers of the URDF (so your nodes) of those changes and update all of them in a coherent way.

Imagine the possibilities for errors and serious accidents if one part of your robot believes it has an extra leg, while the other part thinks it can fly.

More realistically this means that applications targeting self-assembling robots or robots with changing tools can have difficulty using URDF.

A second limitation of URDF is that it is impossible for joints to have more than a single parent or child.

Technically, URDF only supports 'tree structures' or robot body layouts with a single starting point or root and no cycles.

While that may not seem like a serious limitation, it can be if you want to model a robot that for example has two arms with hands on them and is supposed to grasp boxes.

The robot's torso, together with the two arms, hands and the box form a cycle, with the two hands the leafs, meaning that the joints for the box need two parents. As this is not supported such setups cannot be modelled with URDF.

A third limitation is that URDF cannot be used to describe any sensors your robot may have.

Imagine a robot with a camera or a laser scanner. URDF does not have any keywords that allow us to describe the resolution of the camera, what its rate of capture is or whether the laser scanner is a single or dual-return device or what its Field-of-View is.

This is typically not really a problem though, as ROS has multiple other ways of encoding and storing that sort of information.

However, if we consider a URDF file as a description of our physical robot that any ROS application can use to learn about the structure and hardware components in it, then not being able to add information about sensors can be considered a limitation.

Finally, the fourth limitation is the reusability of a plain URDF.

As we saw earlier, all URDFs contain a 'robot' tag, and there can only be one of those 'robot' tags in a URDF file.

So what if we'd have the desire to create a world with multiple robots in it?

Well, there is no way in URDF to import other URDF files directly, that is only supported for 3D meshes.

So we'd have no choice but to take all links and joints of all our robots and merge them into a single giant robot, consisting of all the components of all robots that we want to put in the scene.

Now that would not be so bad, if that didn't actually mean that we'd have to copy-and-paste the contents of all the URDFs describing all the objects and robots that we'd like to add into a single file, while keeping track of joint and link names and making sure the aggregated whole still makes a valid URDF.

This obviously doesn't scale and seriously limits the reusability of URDF robot models.

Luckily, while this is a limitation of URDF, ROS has a system in place called XACRO that solves most of these issues.

Xacro supports macroing, which in this context can be explained as programmatic URDF generation. It allows you to create stand-alone URDF snippets, called macros, that can be added multiple times to a scene by inserting just the name of the macro. The XACRO system will take this name and replace it -- intelligently -- with the full URDF snippet.

So instead of directly copy-and-pasting URDF snippets ourselves, xacro can do this for us, and much more efficiently as well, similar to using templates.

The nice thing about XACRO macros is that they accept parameters, just like C++ or Python functions.

Let's take a look at a small example of this.

As we can see, this defines a macro called 'arm'. It takes one parameter, called 'parent' that allows us to specify what the parent link is of the link that gets added by the macro, and another, 'arm_name' that we use as a prefix for all joint and link names so that they are unique.

The lines between 'xacro:macro' will be copy-pasted into the resulting URDF file automatically, and the parameters will be replaced by the values we supply in the macro call.

Now with this macro we can easily add two arms to a robot.

Notice how we supply a different value for the 'arm_name' parameter, but both arms have 'torso' as their parent link .

This is a very powerful system, as XACRO macros can call other macros, as well as use mathematical and Python expressions to generate URDF snippets.

So how does XACRO address model composability?

Well, XACRO macros don't have to be defined in the same file as they are used in.

With a special include tag, macros defined in other files can be imported, after which they can be called with any arguments they require.

This makes adding multiple robots to a single scene much easier, as instead of having to copy-and-paste everything into a single file, only an import statement and a macro invocation are needed.

There is one downside to XACRO though: a file containing XACRO macros and statements cannot be directly read by URDF parsers.

We must first convert the XACRO file to a URDF file with the 'xacro' tool.

We can use this command for that.

The xacro tool will not only convert the file to valid URDF, it also checks whether the structure of the XACRO we've created is legal and complain loudly if that is not the case.

If all is ok, it will generate the corresponding URDF for us which we can then use as normal with all ROS tools.

This was quite a bit of theory and this video has gotten quite long.

Let's summarize what we've learned:

URDFs once loaded cannot easily be changed or updated;

URDFs cannot be used to model robots with cycles or closed-loop kinematics;

URDF does not include a tag to describe sensors with;

and URDFs cannot easily be combined with other URDFs;

but we have XACRO for that, which supports macros and other advanced functionality to generate URDF for us.

Now that we've tasted the power of XACRO and have seen a very simple example, it's again time to try this for yourself.

Check the links to the tutorials we've selected for you and we'll continue with the last part after you come back.

Be sure to post on the forum in case you need help, we'll try to assist you if we can.