Create a URDF file by hand

In this lab we are going to familiarise ourselves with the URDF file format. This is important as we are going to be generating URDF files automatically later, so it will be useful to have a grasp of the format.

Objectives

At the end of this worksheet, you should be able to:

- Load in a ready-made URDF file
- Create a new URDF file which specifies a simple robot
- Add additional parts to the robot and specify geometry

Defining terms

Links and joints

Load in some ready-made URDF files

We are going to jump straight in and get some robots onto the screen. Fire up your pybullet environment using iPython and try the following code:

```
import pybullet as p
import pybullet_data as pd
print(pd.getDataPath())
```

In my case, that code prints out the location of a folder:

'/home/myuser/Python/pybullet/lib/python3.9/site-packages/pybullet_data'

If you are in iPython, you can now move into that directory and examine its contents:

```
cd '/home/myuser/Python/pybullet/lib/python3.9/site-packages/pybullet_data'
ls
```

(adjust the path to the one that was printed out)

I see a big list of files and folders including several urdf files. Here is an extract of what I can see:

```
a1/                         __init__.py             soccerball.urdf
args/                       jenga/                  sphere_1cm.urdf
bicycle/                    kiva_shelf/             sphere2red_nocol.urdf
biped/                      kuka_iiwa/              sphere2red.urdf
block.urdf                  laikago/                sphere2.urdf
cartpole.urdf               lego/                   sphere8cube.urdf
checker_blue.png            l_finger_collision.stl  sphere_small.urdf
checker_grid.jpg            l_finger.stl            sphere_smooth.mtl
checker_huge.gif            l_finger_tip.stl        sphere_smooth.obj
```

```
cloth_z_up.mtl              mini_cheetah/           sphere_transparent.urdf
cloth_z_up.obj              mjcf/                   sphere_with_restitution.urd
```

Let's go ahead and load one of those files.

```
p.connect(p.GUI)
rob1 = p.loadURDF(pd.getDataPath() + "/cartpole.urdf")
```

You should see something like this. Note that I removed the display panels on the GUI by pressing 'g' on my keyboard when the pybullet GUI was in focus:

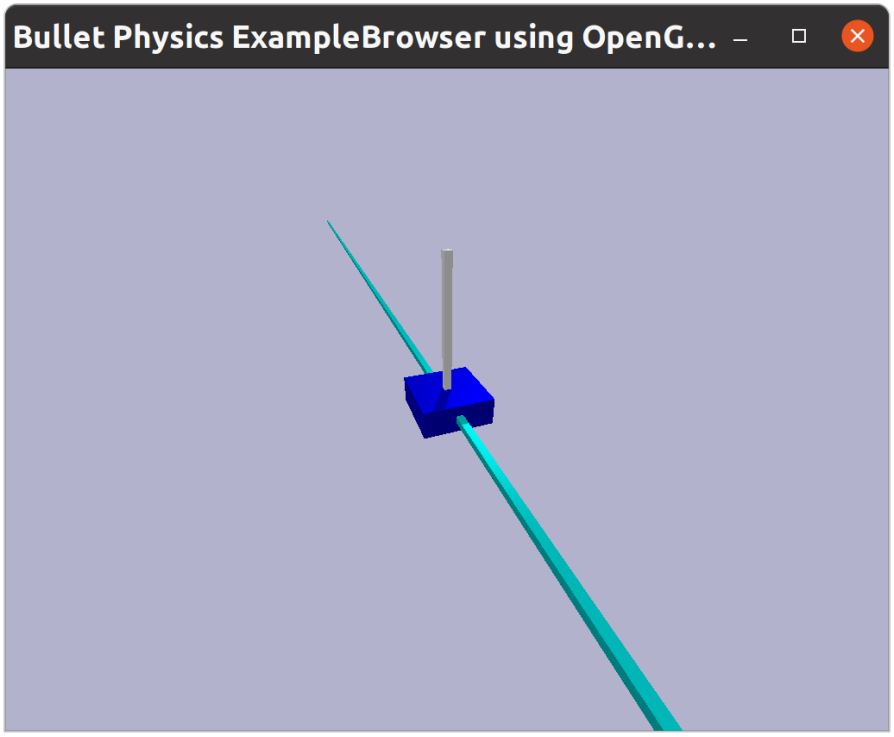

Figure 1: cartpole robot model

Put in a floor, fire up the simulation and try moving the robot around. Here is the complete code to do that:

```
import pybullet as p
import pybullet_data as pd
p.connect(p.GUI)
plane_shape = p.createCollisionShape(p.GEOM_PLANE)
floor = p.createMultiBody(plane_shape, plane_shape)
rob1 = p.loadURDF(pd.getDataPath() + "/cartpole.urdf")
```

```
p.setGravity(0, 0, -10)
p.setRealTimeSimulation(1)

# mac users only to deal with single thread setup
import time
while True:
    p.stepSimulation()
    time.sleep(1.0/240)
```

Try loading in some of the other models in that pd.getDataPath() folder. Can you re-create the scene you saw in the video with r2d2 and the samurai castle?

Create a one link robot

NOTES: Add the starter scripts to set things up. Try with no collision, falls through floor. Add collision. Sits on floor.

Now we have seen how we can use ready-made robots in pybullet, it is time to find out how to design our own robot with URDF. We will start by creating a single part robot.

Set up a starter script

First off, let's create a starter script that will set up pybullet with a floor, gravity etc. Put the following code into a file called starter.py

```
import pybullet as p
p.connect(p.GUI)
# configuration of the engine:
p.setPhysicsEngineParameter(enableFileCaching=0)
p.configureDebugVisualizer(p.COV_ENABLE_GUI, 0)
plane_shape = p.createCollisionShape(p.GEOM_PLANE)
floor = p.createMultiBody(plane_shape, plane_shape)
p.setGravity(0, 0, -10)
```

Note that we tell the engine not to cache files. That is important when working iteratively on URDF. We also switch off the GUi panels by default.

Now when you are working in iPython, you can run the command below to set up your physics engine.

```
run starter.py
# sorry mac users - you also need to run
import time
while True:
    p.stepSimulation()
    time.sleep(1.0/240)
```

One link: visual geometry

Create a new file called 101.urdf in your working code folder. Put this code into the file:

```
<?xml version="1.0"?>
<robot name="myfirst">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.25"/>
      </geometry>
    </visual>
  </link>
</robot>
```

Load the file into the simulation, after running the starter.py script. In iPython:

```
run starter.py
rob1 = p.loadURDF('101.urdf')
# mac users only to deal with single thread setup
import time
while True:
    p.stepSimulation()
    time.sleep(1.0/240)
```

Note that 101.urdf and starter.py need to be in the directory you are currently in. Not sure which directory you are in? In iPython:

```
pwd
```

You should see something like this:

If you set the simulation into realtime mode, what happens? Any idea why?

```
p.setRealTimeSimulation(1)
# mac users only
import time
while True:
    p.stepSimulation()
    time.sleep(1.0/240)
```

Collision geometry

The robot fell through the floor because we specified visual geometry but not collision geometry. So there was nothing there to collide with the floor. Update the 101.urdf file to this:

```
<?xml version="1.0"?>
<robot name="myfirst">
  <link name="base_link">
```

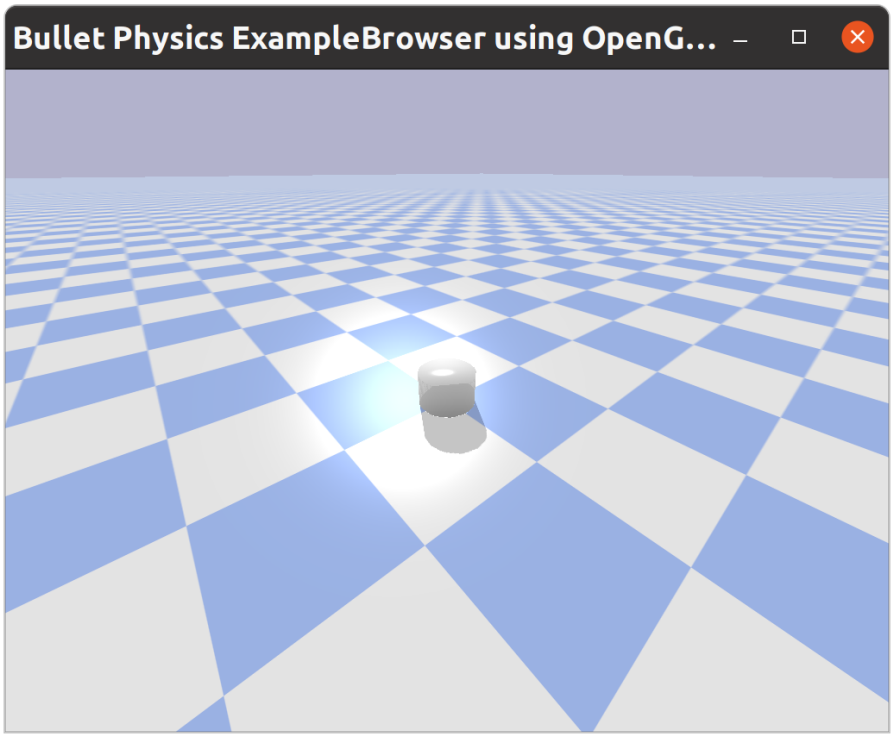

Figure 2: One part robot model

```
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.25"/>
      </geometry>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.6" radius="0.25"/>
      </geometry>
    </collision>
  </link>
</robot>
```

What difference do you notice?

Remove inertia error messages

When I load our 101.urdf file, I see the following error message:

b3Printf: No inertial data for link, using mass=1, localinertiadiagonal = 1,1,1, identity local inertial frame

Let's fix that. Add the following tag to your 101.urdf file, inside the link tag:

```
  <inertial>
    <mass value="0.25"/>
    <inertia ixx="0.0003" iyy="0.0003" izz="0.0003"
             ixy="0" ixz="0" iyz="0"/>
  </inertial>
```

Here is a link to the corresponding ROS manual page about the inertia tag:

http://wiki.ros.org/urdf/XML/link#Elements

Now reload the file - have the error messages gone?

Experiment with shapes

Now let's try some different geometries. Locate the information about the geometry tag here:

http://wiki.ros.org/urdf/XML/link#Elements

Experiment with the box, cylinder and sphere tags.

Create a two link robot

Now we will add another link to the robot. To do that, we need to add another link tag and a joint tag. The joint tag specifies how the two links (robot body parts) should be joined together.

Create a copy of your 101.urdf file called 102.urdf. Add another link tag to the file, making sure it is inside the robot tag (not inside the other link tag). You will also need to give it a different name than the first link tag - edit the name attribute of the link tag:

```
<link name="sub_link">
```

Try to load the new file into the simulation. Does it work?

On my system, the python interpreter actually crashes with the following warning:

b3Printf: URDF file with multiple root links found: base_link sub_link

The concept here is that when we have more than one link, one link becomes the base link and all other links must be connected to that, or some descendant of that. Figure 3 illustrates valid and non-valid structures.

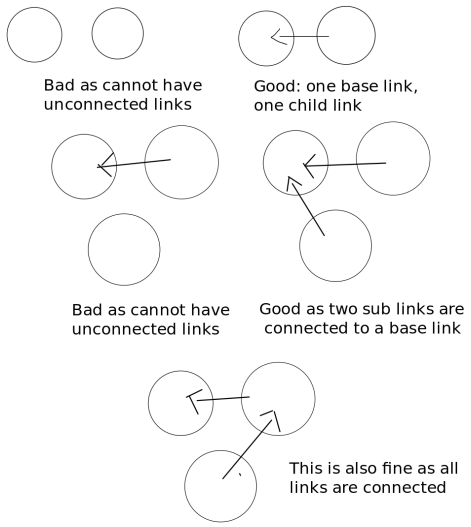

Figure 3: Valid and non-valid link arrangements

Connect the links with a joint

To connect links together, we need a joint tag. Add the following tag inside the robot tag, at the top level of the hierarchy, i.e. not inside the link tag:

```
<joint name="base_to_sub" type="revolute">
  <parent link="base_link"/>
  <child link="sub_link"/>
  <axis xyz="1 0 0"/>
  <limit effort="10" upper="0" lower="10" velocity="1"/>
  <origin rpy="0 0 0" xyz="0 0.5 0"/>
</joint>
```

Load this into your simulation:

```
# note we are calling it rob3 in case
# you already have a rob1 !
rob3 = p.loadURDF('102.urdf)
```

You should see something a bit like this, depending what you ended up with for the geometry in the link tag:

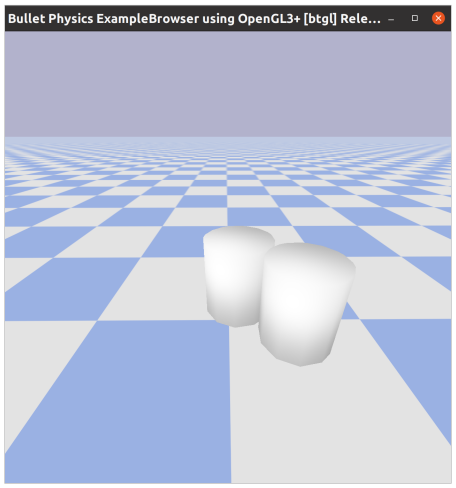

Figure 4: Two link robot

Adjust the joint parameters

Here is a link to the documentation for the joint tag:

http://wiki.ros.org/urdf/XML/joint

Experiment with the origin tag. We will look at the axis tag later. Try moving the sub link to a different position relative to the base link. Try rotating it to a different angle.

Challenge

Time to create some more complex URDF files and robot scenes.

- Can you load in a few different URDF files at the same time?
- Can you create some specific geometries? E.g. a starfish?
- Can you move the robots around programmatically using resetBasePositionAndOrientation?

Check against the objectives

Here are the objectives from the start of the lab. How did you do?

- Load in a ready-made URDF file
- Create a new URDF file which specifies a simple robot
- Add additional parts to the robot and specify geometry