Note:

1.This Quiz is from the course, URDF for Robot Modeling on Robot Ignite Academy
: https://www.robotigniteacademy.com/en/course/robot-creation-with-urdf-ros/details/

2.Any content of the quiz belongs to Robot Ignite Academy except for the sample solution written by Samwoo Seong. I.e. I don't own any of quiz contents

3.Any work throughout the quiz is for learning purpose

4.The solution written by Samwoo Seong shouldn't be used to pass the quiz on this course

Samwoo Seong
samwoose@gmail.com

Quiz, URDF FOR ROBOT MODELING
June 23, 2020

1.0 Objectives

-Create the URDF files for a Gurdy Robot from scratch [1]

1.1 Abstract and Motivation

In the field of robotics, running a robot individually could be unaffordable. However, we can still work with many types of robots by simulating on our computer. Therefore, it will be beneficial to know how to design a robot in a simulation. Through this practice, we will have better understanding of robot design and how to make them able to interact with virtual real environments. i.e., We will create a robot in a virtual world, and it can move like a real robot.

1.2 Approach and Procedures

We can break down the whole procedure into a few small procedures.

-Create a package called my_gurdy_description

-Create a URDF file in the package

-Add links and joints of Gurdy robot in the URDF based on Gurdy's structure

-Import 3D CAD models to Gazebo (real-like virtual simulation) in URDF file (i.e., use meshes (e.g. .dae files)

-Add inertias of the robot in URDF file

-Add Gazebo physical properties such as friction coefficients, and stiffness of materials into URDF file

-Add Collision components in URDF file

-Add Visual properties for robot components in URDF file

-Add transmissions of robot in URDF file

-Add sensors onto robot in URDF file

-Write a .yaml file (i.e., control configuration file) so that we can use PID controller for our transmissions

-Write launch files to run a robot and controllers

-Write launch files to spawn Gurdy into Gazebo simulation

Key procedures will be discussed in depth at Discussion section.

1.3 Experimental Results

Video 1. Spawning Gurdy into Gazebo using ROS
https://www.youtube.com/watch?v=mxeKbBixRHU&feature=emb_logo

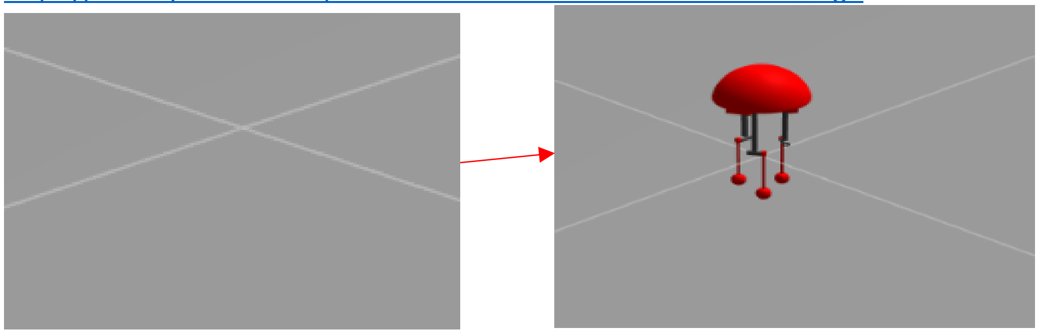


Fig 1. Before and after spawning robot, Gurdy in Gazebo

1.4 Discussion

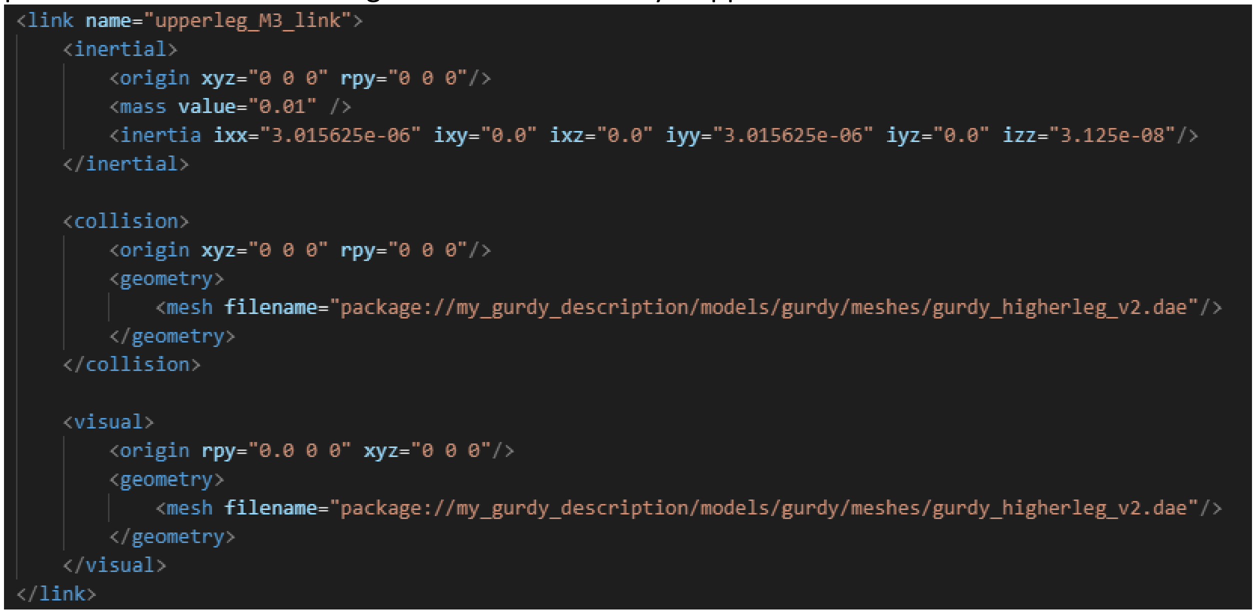
It will be much more intuitive if we go though files that play significant roles to design and spawn a robot, Gurdy. Briefly speaking, there are three types of files are being used.

1. “.urdf”
2. “.yaml”
3. “.launch”

Each file has their own role. First of all, URDF includes the following main components

-links: Gurdy has multiple components such as head, 3 upper legs, 3 lower legs, and 3 feet. This can be created with link tag as shown below. We need to define inertial, collision and visual tag of each component.

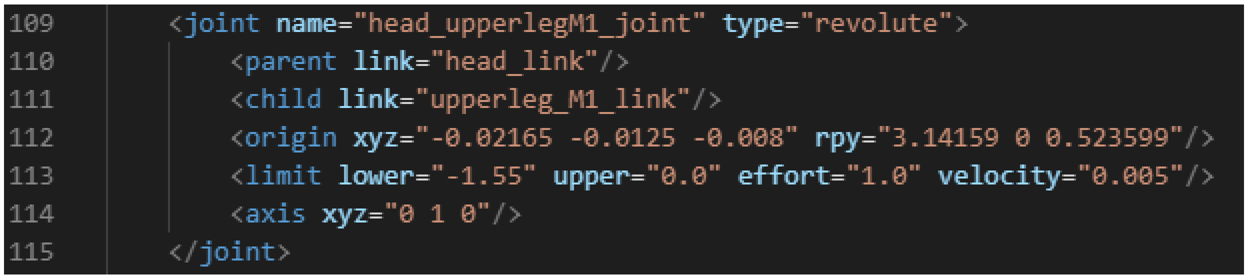
e.g. a link for upper leg in Gurdy’s body. We can see a link contains inertial, collision, and visual parts. Note that we are using 3D CAD file for Gurdy’s appearance.



```
<link name="upperleg_M3_link">
    <inertial>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <mass value="0.01" />
        <inertia ixx="3.015625e-06" ixy="0.0" ixz="0.0" iyy="3.015625e-06" iyz="0.0" izz="3.125e-08"/>
    </inertial>

    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package://my_gurdy_description/models/gurdy/meshes/gurdy_higherleg_v2.dae"/>
        </geometry>
    </collision>

    <visual>
        <origin rpy="0.0 0 0" xyz="0 0 0"/>
        <geometry>
            <mesh filename="package://my_gurdy_description/models/gurdy/meshes/gurdy_higherleg_v2.dae"/>
        </geometry>
    </visual>
</link>
```

-joints of links: Joint tag lets you connect one link to another so that their movement can be associated with each other. In joint tag, we need to specify its type, parent link, child link, origin, limit, and axis depending on its joint type.

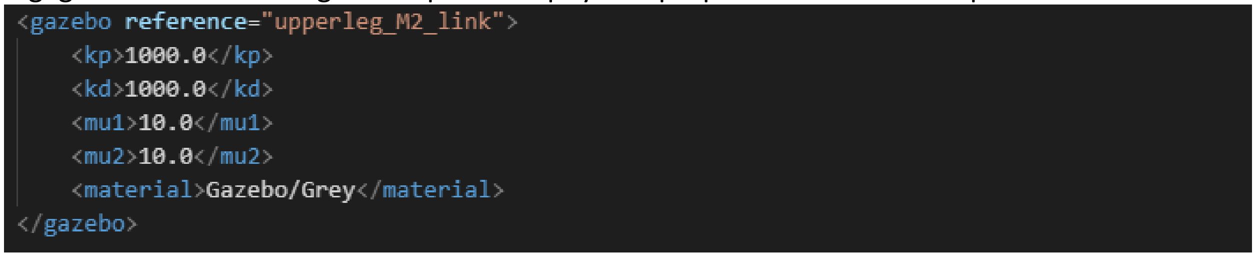
e.g. This is showing an example of joint tag that connect parent link, head_link to child link, upperleg_M1_link.



```xml
    <joint name="head_upperlegM1_joint" type="revolute">
        <parent link="head_link"/>
        <child link="upperleg_M1_link"/>
        <origin xyz="-0.02165 -0.0125 -0.008" rpy="3.14159 0 0.523599"/>
        <limit lower="-1.55" upper="0.0" effort="1.0" velocity="0.005"/>
        <axis xyz="0 1 0"/>
    </joint>
```

-Gazebo physical properties: we can specify physical properties of robot components in Gazebo. For instance, we are defining static contact stiffness (kp), dynamic contact stiffness (kd), static friction coefficient (mu1), dynamic friction coefficient (mu2), and color for Gurdy's 2nd upper leg.

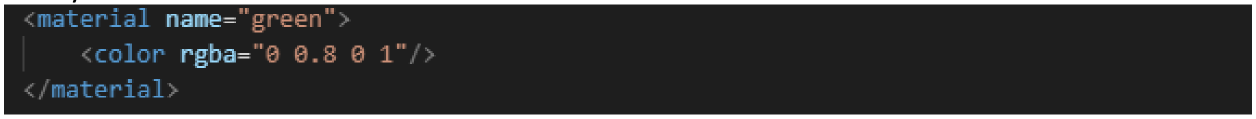
e.g. gazebo reference tag that represents physical properties of robot component in Gozebo



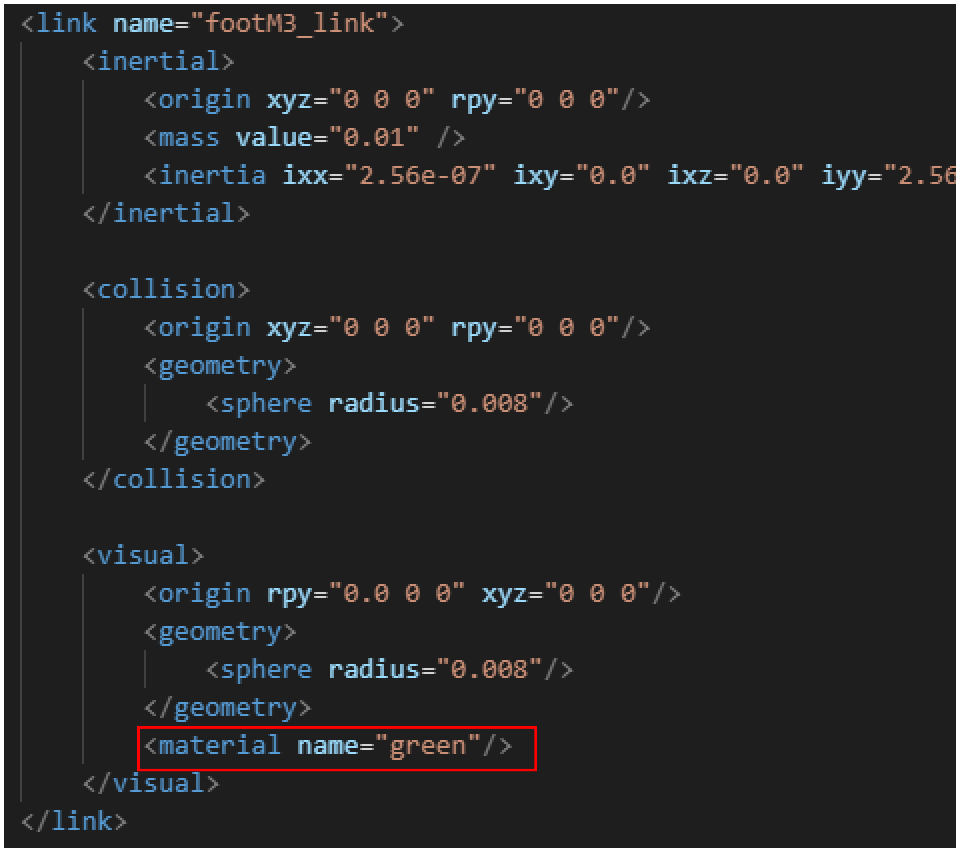
```xml
<gazebo reference="upperleg_M2_link">
    <kp>1000.0</kp>
    <kd>1000.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Grey</material>
</gazebo>
```

-Visual properties for Rviz: Rviz means ROS visualization and it helps developers track robot's transform frames of robot components. There is a way to add color in Rviz. Note that Rviz and Gazebo are used for different purposes. Therefore, some tags exist for different tools.

e.g. you can define a color with material name tag and use it later. Here, we are use this for Gurdy's foot



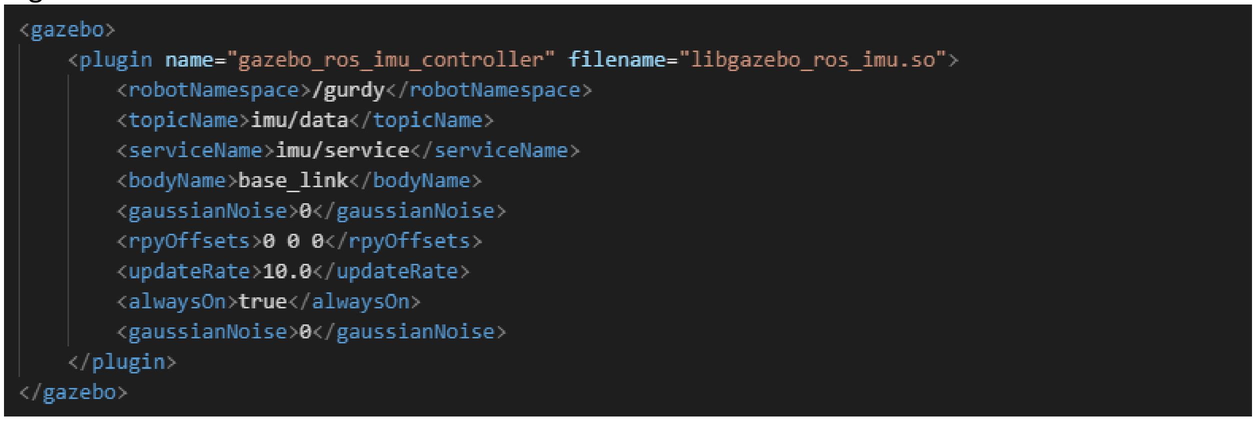
```xml
<material name="green">
    <color rgba="0 0.8 0 1"/>
</material>
```



```
<link name="footM3_link">
    <inertial>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <mass value="0.01" />
        <inertia ixx="2.56e-07" ixy="0.0" ixz="0.0" iyy="2.56
    </inertial>

    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <sphere radius="0.008"/>
        </geometry>
    </collision>

    <visual>
        <origin rpy="0.0 0 0" xyz="0 0 0"/>
        <geometry>
            <sphere radius="0.008"/>
        </geometry>
        <material name="green"/>
    </visual>
</link>
```

-Gazebo Sensors: Gazebo plug in tag make adding sensors process much easier. Here, we are adding a IMU sensor to Gurdy robot.

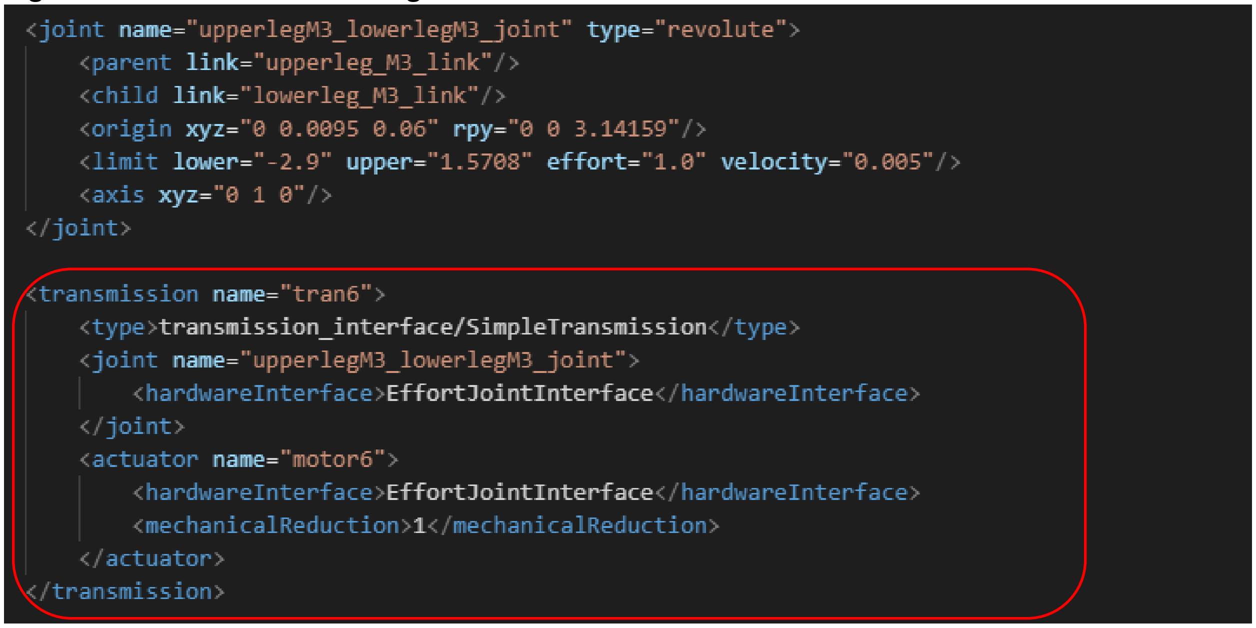
e.g.



```
<gazebo>
    <plugin name="gazebo_ros_imu_controller" filename="libgazebo_ros_imu.so">
        <robotNamespace>/gurdy</robotNamespace>
        <topicName>imu/data</topicName>
        <serviceName>imu/service</serviceName>
        <bodyName>base_link</bodyName>
        <gaussianNoise>0</gaussianNoise>
        <rpyOffsets>0 0 0</rpyOffsets>
        <updateRate>10.0</updateRate>
        <alwaysOn>true</alwaysOn>
        <gaussianNoise>0</gaussianNoise>
    </plugin>
</gazebo>
```

-Transmission: Normally, you can consider a transmission as a motor and its controller. Therefore, you need to add transmission tag whenever a joint is needed to move. Here, we are specifying a transmission tag for joint of 3rd upper leg and 3rd lower leg. Usually, this tag comes after joint tag.

e.g. we name a transmission tag “tran6” and its actuator “motor6”

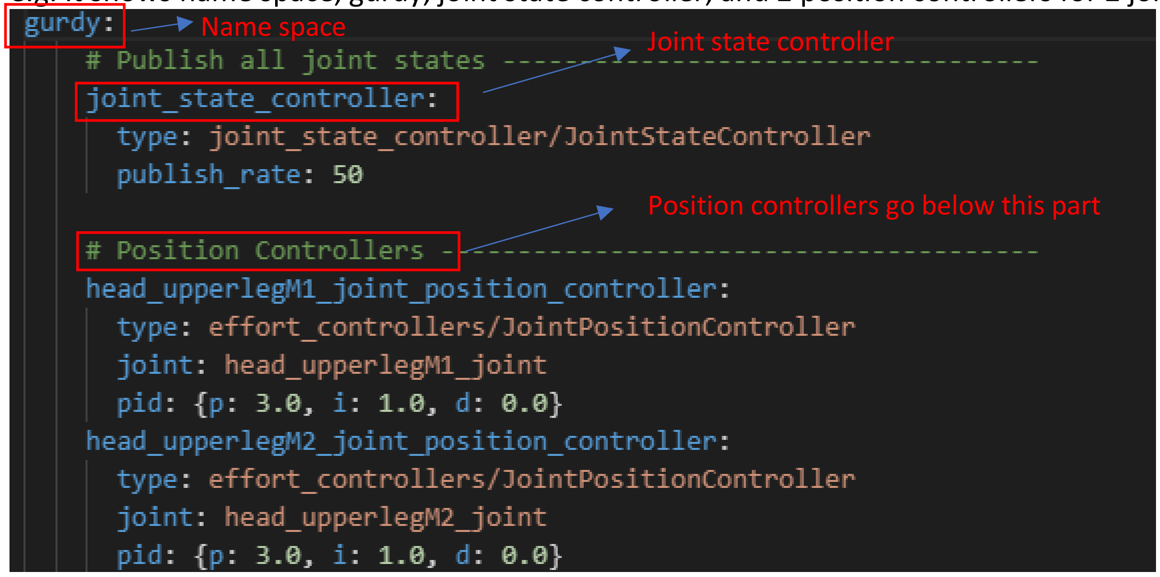


```
<joint name="upperlegM3_lowerlegM3_joint" type="revolute">
    <parent link="upperleg_M3_link"/>
    <child link="lowerleg_M3_link"/>
    <origin xyz="0 0.0095 0.06" rpy="0 0 3.14159"/>
    <limit lower="-2.9" upper="1.5708" effort="1.0" velocity="0.005"/>
    <axis xyz="0 1 0"/>
</joint>

<transmission name="tran6">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="upperlegM3_lowerlegM3_joint">
        <hardwareInterface>EffortJointInterface</hardwareInterface>
    </joint>
    <actuator name="motor6">
        <hardwareInterface>EffortJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
</transmission>
```

We need to iteratively specify all tags for all robot components.

Next, we need to write up “.yaml” file and it is located in config folder in my_gurdy_description package. In this file, we are defining its name space which is gurdy, joint state controller, and position controllers. Here, we are using PID controller to stabilize behaviors of motors. The more motors you use, the more position controllers you need to add in this file.

e.g. It shows name space, gurdy, joint state controller, and 2 position controllers for 2 joints.



```
gurdy:
    # Publish all joint states -----------------------------------
    joint_state_controller:
      type: joint_state_controller/JointStateController
      publish_rate: 50

    # Position Controllers ---------------------------------------
    head_upperlegM1_joint_position_controller:
      type: effort_controllers/JointPositionController
      joint: head_upperlegM1_joint
      pid: {p: 3.0, i: 1.0, d: 0.0}
    head_upperlegM2_joint_position_controller:
      type: effort_controllers/JointPositionController
      joint: head_upperlegM2_joint
      pid: {p: 3.0, i: 1.0, d: 0.0}
```

Lastly, we need a few ".launch" files which are located in launch folder in my_gurdy_descrition package.

(1) "spawn_urdf.launch":

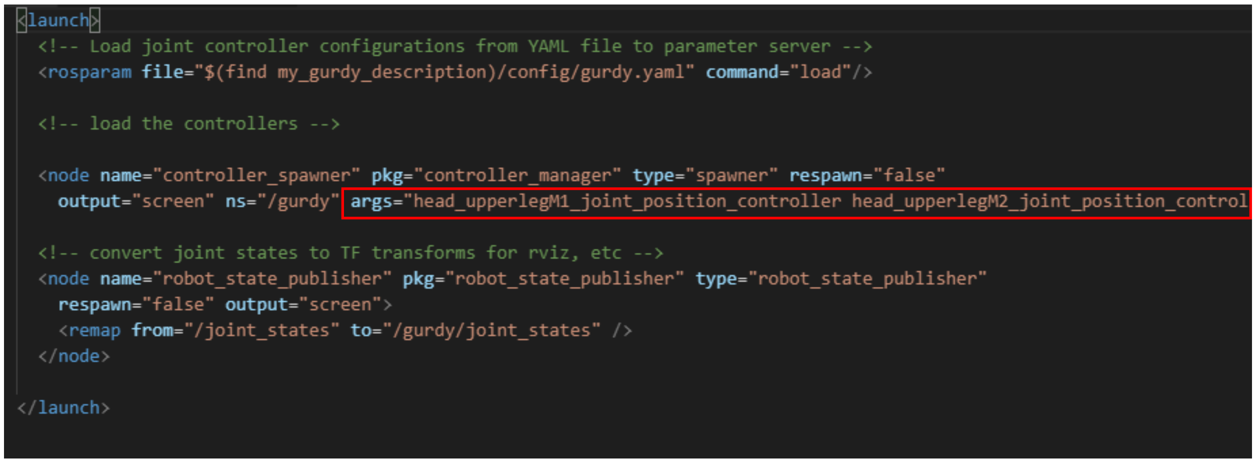
It specifies where to locate gurdy, name of urdf you want to spawn, and robot's name. In our case, all of these will be specified in spawn_gurdy.launch file.

(2) "spawn_gurdy.launch" :

This includes spawn_urdf.launch and then specifies where to spawn gurdy, name of urdf file which is gurdy.urdf. and robot's name, gurdy

(3) "gurdy_control.launch":

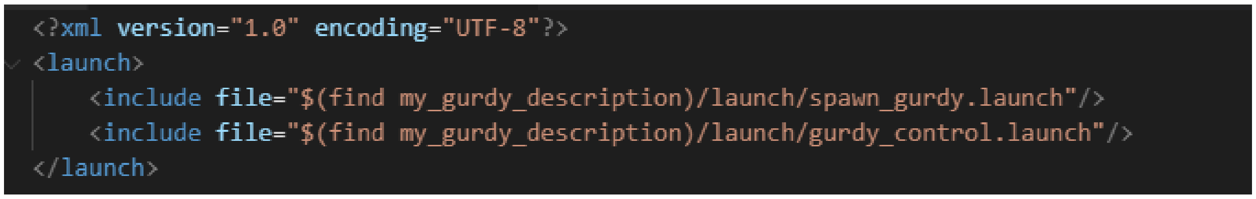
This file loads "gurdy.yaml" file that contains configuration of all controllers we are using in URDF file and then all controllers is spawned and managed by first node tag. Then, all joint states are converted to TF transforms in the second node tag for Rviz. Note that you need to add all position controllers defined in .yaml file into "args" in the first node tag.



```
<launch>
  <!-- Load joint controller configurations from YAML file to parameter server -->
  <rosparam file="$(find my_gurdy_description)/config/gurdy.yaml" command="load"/>

  <!-- load the controllers -->

  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
    output="screen" ns="/gurdy" args="head_upperlegM1_joint_position_controller head_upperlegM2_joint_position_control

  <!-- convert joint states to TF transforms for rviz, etc -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
    respawn="false" output="screen">
    <remap from="/joint_states" to="/gurdy/joint_states" />
  </node>

</launch>
```

(4) "start_gurdy_withcontroller.launch":

Lastly, we write up a launch file that will run all 3 launch files we create previously. It can be done with include tag.



```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
    <include file="$(find my_gurdy_description)/launch/spawn_gurdy.launch"/>
    <include file="$(find my_gurdy_description)/launch/gurdy_control.launch"/>
</launch>
```

References

[1] Quiz Contents Credit: Robot Ignite Academy

https://www.robotigniteacademy.com/en/course/robot-creation-with-urdf-ros/details/