Unit 3: Gurdy Robot

Warning

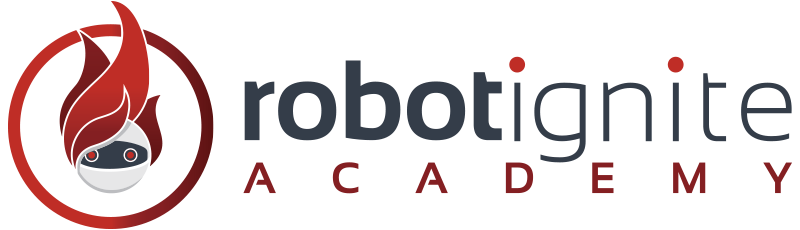
Before you continue, please kill everything that you have launched and remove Mira Robot from the scene by the methods explained.

END Warning

To practice and better learn how URDF are created and used, you are going to create, from scratch and with a little help, the Robot Model of Gurdy Robot.

|  |
| --- |
| </th>  <th>  <figure>  <img id="fig-A.2" src="img/gurdy\_RDS2.png" width="300"/>  </figure>  </th> |

URDF Quiz

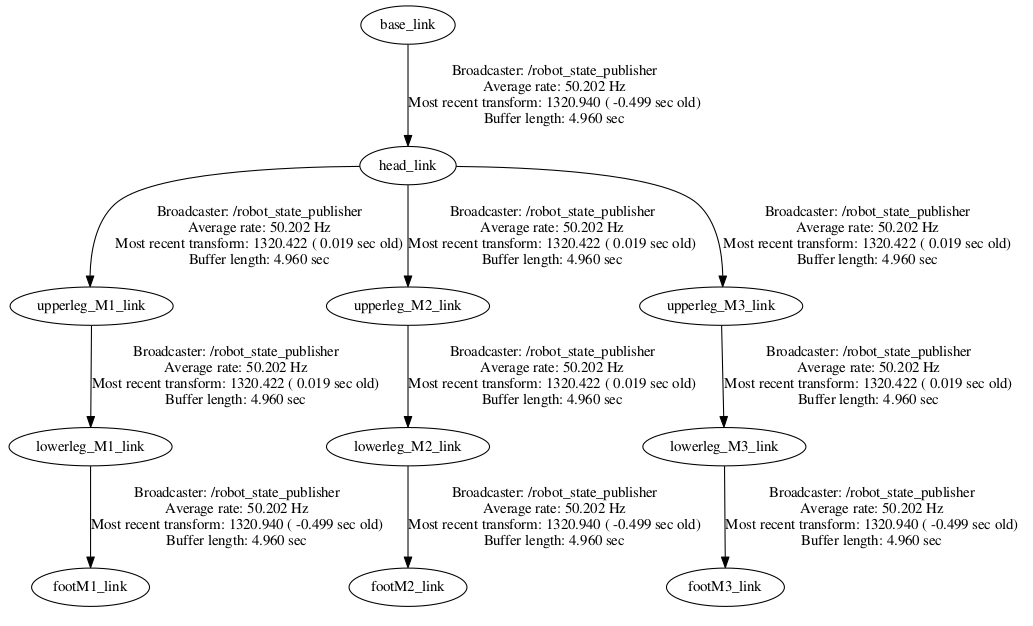


For evaluating this Quiz, we will ask you to perform different tasks. For each task, very **specific instructions** will be provided: name of the package, names of the launch files and Python scripts, topic names to use, etc.

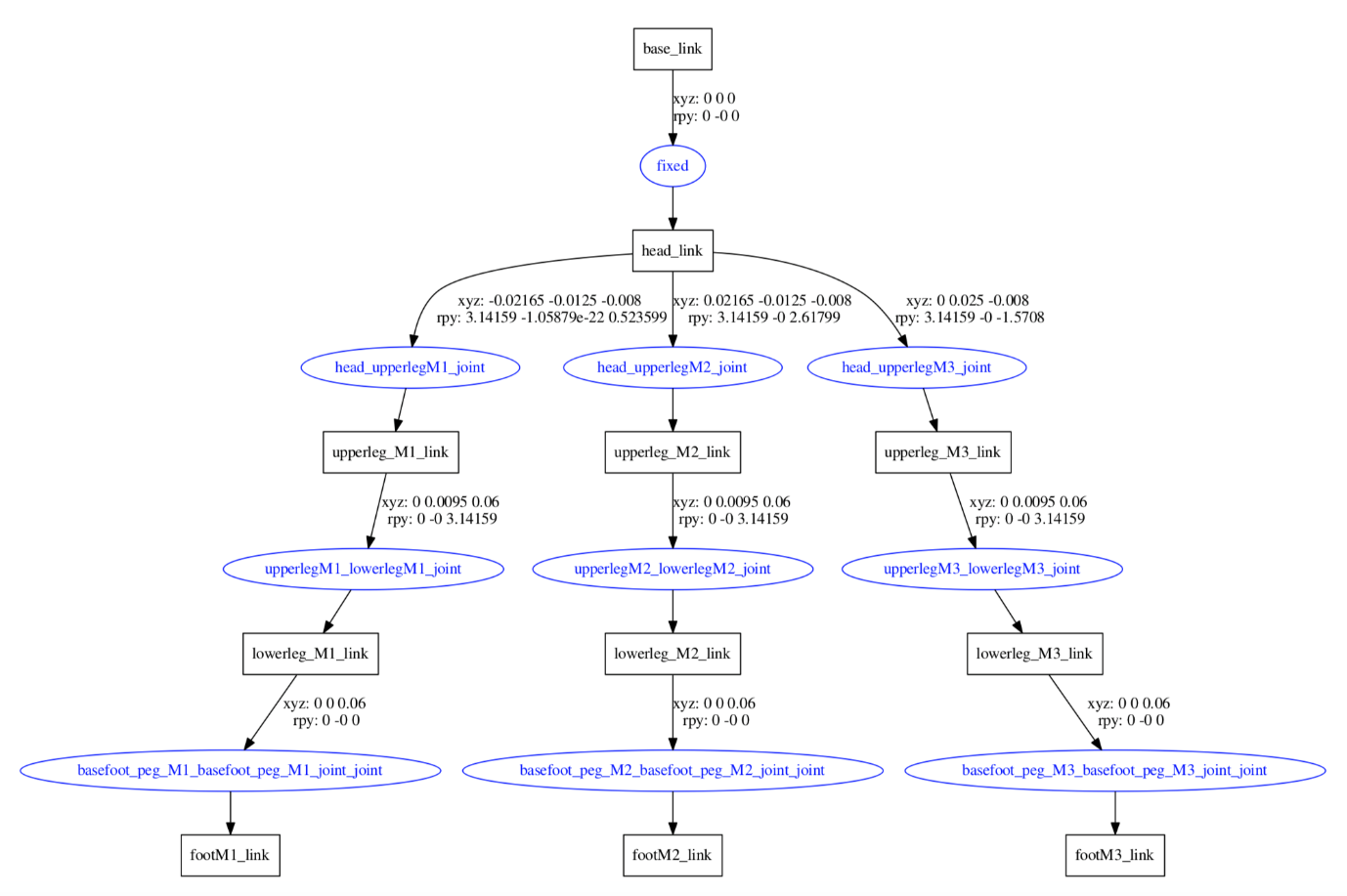
It is **VERY IMPORTANT** that you strictly follow these instructions, since they will allow our automated correction system to properly test your Quiz, and assign a score to it. If the names you use are different from the ones specified in the exam instructions, your exercise will be marked as **FAILED**, even though it works correctly.

Follow the same procedure that we followed to create the Mira URDF. These are some points to guide you through the creation process:

* Discover how many links and joints will be needed to create this robot model. Decide which joints will be fixed, revolve, or be continuous.  
  This Link Tree will give you a good idea of the morphology of Gurdy's link structure.



And you have an even better view with the image below, which shows the links and joints:



* Start by creating a basic geometric URDF model, with no dae files. Use the tools for URDF development.

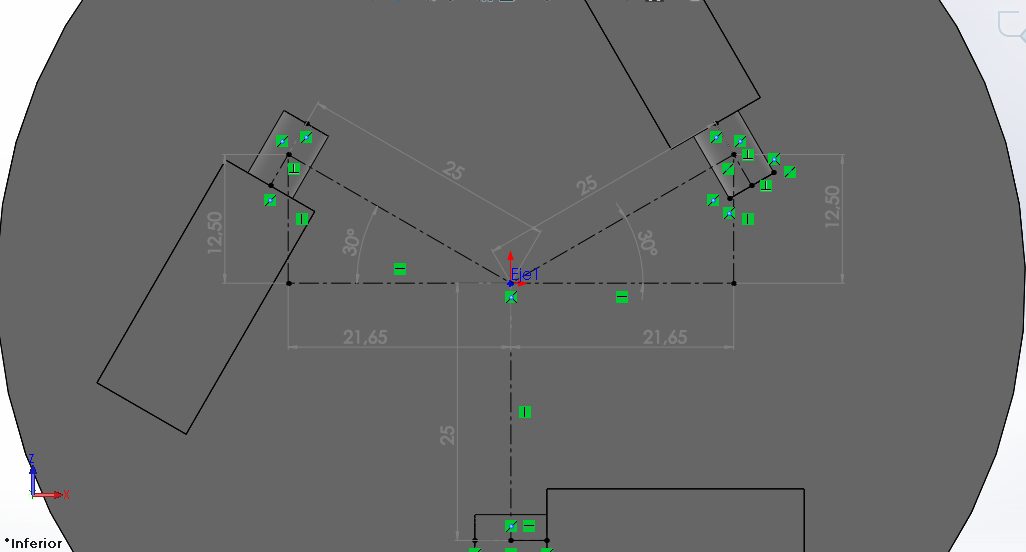
In [ ]:



roslaunch my\_mira\_description urdf\_visualize.launch model:**=**'$(find my\_gurdy\_description)/urdf/gurdy.urdf'

urdf\_to\_graphiz gurdy.urdf

Here you have a blueprint of the distances to place the most difficult part, which is the joints between the head and the legs. The units are millimeters:



You will also need the dimensions of the the basic geometry to have some idea of the way that everything is connected, and the collision elements geometry:

* **base\_link** : box size="0.01 0.01 0.01"
* **head\_link** : cylinder radius="0.05" length="0.04"
* **upper\_leg\_links** : cylinder length="0.06" radius="0.0025"
* **lower\_leg\_links** : cylinder length="0.06" radius="0.0015"
* **foot\_links** : sphere radius="0.008"

When you have the simple main structure, add the dae models. For this, you will have to copy them from the **gurdy\_description** package. Use the following commands to do so:

In [ ]:



roscd; cd ..;

In [ ]:



source devel**/**setup.bash

In [ ]:



rospack profile

In [ ]:



roscd my\_gurdy\_description

In [ ]:



mkdir **-**p models**/**gurdy**/**meshes

In [ ]:



roscd gurdy\_description

In [ ]:



cp .**/**models**/**gurdy**/**meshes**/**gurdy\_head\_v2.dae **/**home**/**user**/**catkin\_ws**/**src**/**my\_gurdy\_description**/**models**/**gurdy**/**meshes

In [ ]:



cp .**/**models**/**gurdy**/**meshes**/**gurdy\_higherleg\_v2.dae **/**home**/**user**/**catkin\_ws**/**src**/**my\_gurdy\_description**/**models**/**gurdy**/**meshes

In [ ]:



cp .**/**models**/**gurdy**/**meshes**/**gurdy\_lowerleg\_v2.dae **/**home**/**user**/**catkin\_ws**/**src**/**my\_gurdy\_description**/**models**/**gurdy**/**meshes

* Once you have the Visual URDF model, it's time to add the collisions, the actuators, and the sensors.  
  The collisions will be geometric figures, as in the Mira Example.  
  The actuators are now six instead of three, so make the appropriate changes.  
  The limits in effort and speed should be around: effort="1.0" and velocity="0.005," but it will depend on the joint.  
  The weights of the links are:
* **base\_link** : mass=None, there is no inertia needed because its a non-functional element.
* **head\_link** : mass value="0.01"
* **upper\_leg\_links** : mass value="0.01"
* **lower\_leg\_links** : mass value="0.01"
* **foot\_links** : mass value="0.01"
* Friction is vital in the foot links. Try different values and see the effect ont locomotion. These are some orientative values:
* **kp** = 1000.0
* **kd** = 1000.0
* **mu1** = 10.0
* **mu2** = 10.0
* PID are tricky. Use the rqt\_reconfigure to get the correct values. These are some orientative values:  
    
  {p: 3.0, i: 1.0, d: 0.0}
* As for sensors, you will have to add an IMU to the "base\_link." This is the code for adding an IMU:

In [ ]:



**<**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**>**

Specifications

* The name of the package where you'll place all the code related to the Quiz will be **my\_gurdy\_description**.
* The name of the launch file that will spawn the Gurdy robot with its controllers will be **spawn\_gurdy.launch**. So, this launch file has to:
  + First, spawn the gurdy robot
  + Second, load the controllers
* The names of the links and joints of the robot have to be the same as in the following tree: [gurdy\_link\_and\_joints](https://i-0500135e5b3ce7d04.robotigniteacademy.com/d7e6f789-1484-4bd3-aac3-7d6fe70eb589/jupyter/notebooks/course_urdf_ROS/Course_urdfROS_Unit_3.ipynb" \l "gurdy_links_joints)
* The names of the resulting topics (**joint\_states** and **command** topics) when loading the controllers of the robot have to be the following:
  + **/gurdy/joint\_states**
  + **/gurdy/head\_upperlegM1\_joint\_position\_controller/command**
  + **/gurdy/head\_upperlegM2\_joint\_position\_controller/command**
  + **/gurdy/head\_upperlegM3\_joint\_position\_controller/command**
  + **/gurdy/upperlegM1\_lowerlegM1\_joint\_position\_controller/command**
  + **/gurdy/upperlegM2\_lowerlegM2\_joint\_position\_controller/command**
  + **/gurdy/upperlegM3\_lowerlegM3\_joint\_position\_controller/command**

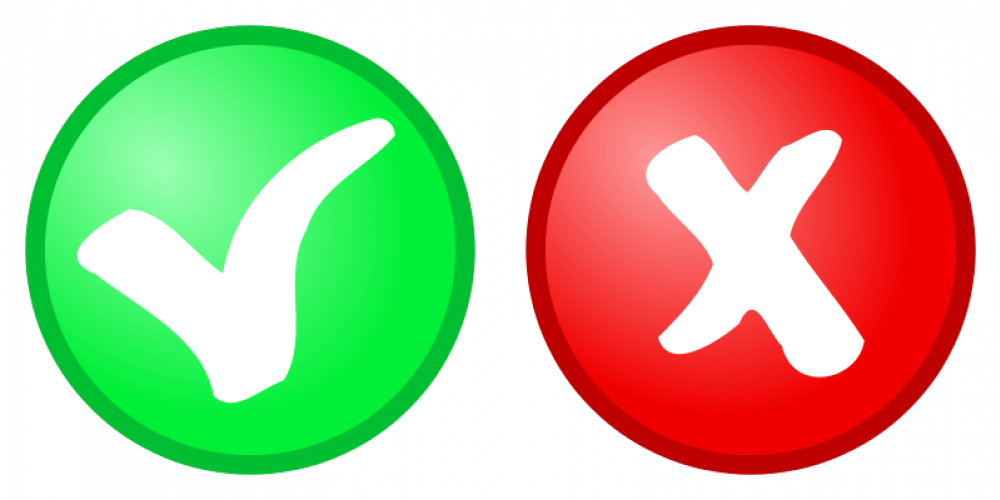
Quiz Correction

When you have finished the Quiz, you can correct it in order to get a Mark. For that, just click on the following button at the top of this Notebook.

**IMPORTANT: Make sure that, before correcting your Quiz, you don't have any robot spawned in the simulations and that you don't have any program running on the Web Shells.**



Final Mark

In case you fail the Quiz, or you don't get the desired mark, do not get frustrated! You will have the chance to resend the Quiz in order to improve your score.

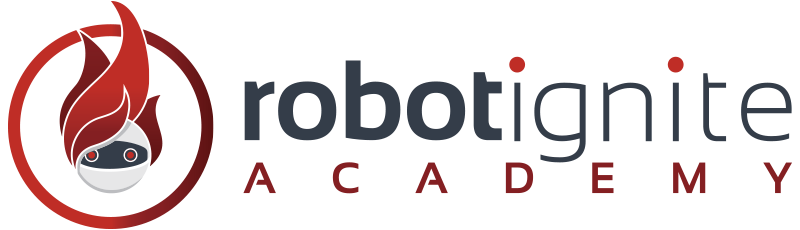
Exercise U3-1

* Create a python script that moves the joints as you wish to test that it moves correctly, in an easy and comfortable way.

END Exercise U3-1

Solution Exercise U3.1

Please try to do it by yourself, unless you get stuck or need some inspiration. You will learn much more if you fight for each exercise.



An example of a movement script for Gurdy:

In [ ]:



*#!/usr/bin/env python*

​

**import** rospy

**import** time

**from** math **import** pi, sin, cos, acos

**import** random

**from** std\_msgs.msg **import** Float64

**from** sensor\_msgs.msg **import** JointState

"""

Topics To Write on:

type: std\_msgs/Float64

/gurdy/head\_upperlegM1\_joint\_position\_controller/command

/gurdy/head\_upperlegM2\_joint\_position\_controller/command

/gurdy/head\_upperlegM3\_joint\_position\_controller/command

/gurdy/upperlegM1\_lowerlegM1\_joint\_position\_controller/command

/gurdy/upperlegM2\_lowerlegM2\_joint\_position\_controller/command

/gurdy/upperlegM3\_lowerlegM3\_joint\_position\_controller/command

"""

​

**class** gurdyJointMover(object):

​

**def** \_\_init\_\_(self):

rospy.init\_node('jointmover\_demo', anonymous**=**True)

rospy.loginfo("Gurdy JointMover Initialising...")

​

self.pub\_upperlegM1\_joint\_position **=** rospy.Publisher(

'/gurdy/head\_upperlegM1\_joint\_position\_controller/command',

Float64,

queue\_size**=**1)

self.pub\_upperlegM2\_joint\_position **=** rospy.Publisher(

'/gurdy/head\_upperlegM2\_joint\_position\_controller/command',

Float64,

queue\_size**=**1)

self.pub\_upperlegM3\_joint\_position **=** rospy.Publisher(

'/gurdy/head\_upperlegM3\_joint\_position\_controller/command',

Float64,

queue\_size**=**1)

self.pub\_lowerlegM1\_joint\_position **=** rospy.Publisher(

'/gurdy/upperlegM1\_lowerlegM1\_joint\_position\_controller/command',

Float64,

queue\_size**=**1)

self.pub\_lowerlegM2\_joint\_position **=** rospy.Publisher(

'/gurdy/upperlegM2\_lowerlegM2\_joint\_position\_controller/command',

Float64,

queue\_size**=**1)

self.pub\_lowerlegM3\_joint\_position **=** rospy.Publisher(

'/gurdy/upperlegM3\_lowerlegM3\_joint\_position\_controller/command',

Float64,

queue\_size**=**1)

joint\_states\_topic\_name **=** "/gurdy/joint\_states"

rospy.Subscriber(joint\_states\_topic\_name, JointState, self.gurdy\_joints\_callback)

gurdy\_joints\_data **=** None

rate **=** rospy.Rate(2)

**while** gurdy\_joints\_data **is** None:

**try**:

gurdy\_joints\_data **=** rospy.wait\_for\_message(joint\_states\_topic\_name, JointState, timeout**=**5)

**except**:

rospy.logwarn("Time out " **+** str(joint\_states\_topic\_name))

**pass**

rate.sleep()

​

self.gurdy\_joint\_dictionary **=** dict(zip(gurdy\_joints\_data.name, gurdy\_joints\_data.position))

​

**def** move\_gurdy\_all\_joints(self, upperlegM1\_angle, upperlegM2\_angle, upperlegM3\_angle, lowerlegM1\_value, lowerlegM2\_value ,lowerlegM3\_value):

upperlegM1 **=** Float64()

upperlegM1.data **=** upperlegM1\_angle

upperlegM2 **=** Float64()

upperlegM2.data **=** upperlegM2\_angle

upperlegM3 **=** Float64()

upperlegM3.data **=** upperlegM3\_angle

​

lowerlegM1 **=** Float64()

lowerlegM1.data **=** lowerlegM1\_value

lowerlegM2 **=** Float64()

lowerlegM2.data **=** lowerlegM2\_value

lowerlegM3 **=** Float64()

lowerlegM3.data **=** lowerlegM3\_value

​

self.pub\_upperlegM1\_joint\_position.publish(upperlegM1)

self.pub\_upperlegM2\_joint\_position.publish(upperlegM2)

self.pub\_upperlegM3\_joint\_position.publish(upperlegM3)

​

self.pub\_lowerlegM1\_joint\_position.publish(lowerlegM1)

self.pub\_lowerlegM2\_joint\_position.publish(lowerlegM2)

self.pub\_lowerlegM3\_joint\_position.publish(lowerlegM3)

​

​

**def** gurdy\_joints\_callback(self, msg):

"""

sensor\_msgs/JointState

std\_msgs/Header header

uint32 seq

time stamp

string frame\_id

string[] name

float64[] position

float64[] velocity

float64[] effort

​

:param msg:

:return:

"""

self.gurdy\_joint\_dictionary **=** dict(zip(msg.name, msg.position))

​

​

**def** convert\_angle\_to\_unitary(self, angle):

"""

Removes complete revolutions from angle and converts to positive equivalent

if the angle is negative

:param angle: Has to be in radians

:return:

"""

*# Convert to angle between [0,360)*

complete\_rev **=** 2 **\*** pi

mod\_angle **=** int(angle **/** complete\_rev)

clean\_angle **=** angle **-** mod\_angle **\*** complete\_rev

*# Convert Negative angles to their corresponding positive values*

**if** clean\_angle **<** 0:

clean\_angle **+=** 2 **\*** pi

​

**return** clean\_angle

​

**def** assertAlmostEqualAngles(self, x, y,):

c2 **=** (sin(x) **-** sin(y)) **\*\*** 2 **+** (cos(x) **-** cos(y)) **\*\*** 2

angle\_diff **=** acos((2.0 **-** c2) **/** 2.0)

**return** angle\_diff

​

**def** gurdy\_check\_continuous\_joint\_value(self, joint\_name, value, error**=**0.1):

"""

Check the joint by name 'base\_waist\_joint', 'body\_head\_joint', 'waist\_body\_joint is near the value given

We have to convert the joint values removing whole revolutions and converting negative versions

of the same angle

:param joint\_name:

:param value:

:param error: In radians

:return:

"""

joint\_reading **=** self.gurdy\_joint\_dictionary.get(joint\_name)

**if** **not** joint\_reading:

**print** "self.gurdy\_joint\_dictionary="**+**str(self.gurdy\_joint\_dictionary)

**print** "joint\_name===>"**+**str(joint\_name)

**assert** "There is no data about that joint"

clean\_joint\_reading **=** self.convert\_angle\_to\_unitary(angle**=**joint\_reading)

clean\_value **=** self.convert\_angle\_to\_unitary(angle**=**value)

​

dif\_angles **=** self.assertAlmostEqualAngles(clean\_joint\_reading, clean\_value)

similar **=** dif\_angles **<=** error

​

**return** similar

​

**def** gurdy\_movement\_look(self, upperlegM1\_angle, upperlegM2\_angle, upperlegM3\_angle, lowerlegM1\_value, lowerlegM2\_value ,lowerlegM3\_value):

"""

Move:

'head\_upperlegM1\_joint',

'head\_upperlegM2\_joint',

'head\_upperlegM3\_joint',

'upperlegM1\_lowerlegM1\_joint',

'upperlegM2\_lowerlegM2\_joint',

'upperlegM3\_lowerlegM3\_joint'

:return:

"""

check\_rate **=** 5.0

position\_upperlegM1 **=** upperlegM1\_angle

position\_upperlegM2 **=** upperlegM2\_angle

position\_upperlegM3 **=** upperlegM3\_angle

​

position\_lowerlegM1 **=** lowerlegM1\_value

position\_lowerlegM2 **=** lowerlegM2\_value

position\_lowerlegM3 **=** lowerlegM3\_value

​

similar\_upperlegM1 **=** False

similar\_upperlegM2 **=** False

similar\_upperlegM3 **=** False

​

similar\_lowerlegM1 **=** False

similar\_lowerlegM2 **=** False

similar\_lowerlegM3 **=** False

​

rate **=** rospy.Rate(check\_rate)

**while** **not** (similar\_upperlegM1 **and** similar\_upperlegM2 **and** similar\_upperlegM3 **and** similar\_lowerlegM1 **and** similar\_lowerlegM2 **and** similar\_lowerlegM3):

self.move\_gurdy\_all\_joints(position\_upperlegM1,

position\_upperlegM2,

position\_upperlegM3,

position\_lowerlegM1,

position\_lowerlegM2,

position\_lowerlegM3)

similar\_upperlegM1 **=** self.gurdy\_check\_continuous\_joint\_value(joint\_name**=**"head\_upperlegM1\_joint",

value**=**position\_upperlegM1)

similar\_upperlegM2 **=** self.gurdy\_check\_continuous\_joint\_value(joint\_name**=**"head\_upperlegM2\_joint",

value**=**position\_upperlegM2)

similar\_upperlegM3 **=** self.gurdy\_check\_continuous\_joint\_value(joint\_name**=**"head\_upperlegM3\_joint",

value**=**position\_upperlegM3)

similar\_lowerlegM1 **=** self.gurdy\_check\_continuous\_joint\_value(joint\_name**=**"upperlegM1\_lowerlegM1\_joint",

value**=**position\_lowerlegM1)

similar\_lowerlegM2 **=** self.gurdy\_check\_continuous\_joint\_value(joint\_name**=**"upperlegM2\_lowerlegM2\_joint",

value**=**position\_lowerlegM2)

similar\_lowerlegM3 **=** self.gurdy\_check\_continuous\_joint\_value(joint\_name**=**"upperlegM3\_lowerlegM3\_joint",

value**=**position\_lowerlegM3)

​

rate.sleep()

​

**def** gurdy\_init\_pos\_sequence(self):

"""

UPPER limits lower="-1.55" upper="0.0"

LOWER limits lower="-2.9" upper="1.5708"

:return:

"""

upperlegM1\_angle **=** **-**1.55

upperlegM2\_angle **=** **-**1.55

upperlegM3\_angle **=** **-**1.55

lowerlegM1\_angle **=** 0.0

lowerlegM2\_angle **=** 0.0

lowerlegM3\_angle **=** 0.0

self.gurdy\_movement\_look(upperlegM1\_angle,

upperlegM2\_angle,

upperlegM3\_angle,

lowerlegM1\_angle,

lowerlegM2\_angle,

lowerlegM3\_angle)

​

lowerlegM1\_angle **=** **-**1.55

lowerlegM2\_angle **=** **-**1.55

lowerlegM3\_angle **=** **-**1.55

self.gurdy\_movement\_look(upperlegM1\_angle,

upperlegM2\_angle,

upperlegM3\_angle,

lowerlegM1\_angle,

lowerlegM2\_angle,

lowerlegM3\_angle)

​

**def** gurdy\_hop(self, num\_hops**=**15):

"""

UPPER limits lower="-1.55" upper="0.0"

LOWER limits lower="-2.9" upper="1.5708"

:return:

"""

​

upper\_delta **=** 1

basic\_angle **=** **-**1.55

angle\_change **=** random.uniform(0.2, 0.7)

upperlegM\_angle **=** basic\_angle

lowerlegM\_angle **=** basic\_angle **-** upper\_delta **\*** angle\_change **\*** 2.0

​

*#self.gurdy\_init\_pos\_sequence()*

**for** repetitions **in** range(num\_hops):

self.gurdy\_movement\_look(upperlegM\_angle,

upperlegM\_angle,

upperlegM\_angle,

lowerlegM\_angle,

lowerlegM\_angle,

lowerlegM\_angle)

​

upper\_delta **\*=** **-**1

**if** upper\_delta **<** 0:

upperlegM\_angle **=** basic\_angle **+** angle\_change

**else**:

upperlegM\_angle **=** basic\_angle

lowerlegM\_angle **=** basic\_angle **-** upper\_delta **\*** angle\_change **\*** 2.0

​

​

**def** gurdy\_moverandomly(self):

"""

UPPER limits lower="-1.55" upper="0.0"

LOWER limits lower="-2.9" upper="1.5708"

:return:

"""

upperlegM1\_angle **=** random.uniform(**-**1.55, 0.0)

upperlegM2\_angle **=** random.uniform(**-**1.55, 0.0)

upperlegM3\_angle **=** random.uniform(**-**1.55, 0.0)

lowerlegM1\_angle **=** random.uniform(**-**2.9, pi**/**2)

lowerlegM2\_angle **=** random.uniform(**-**2.9, pi**/**2)

lowerlegM3\_angle **=** random.uniform(**-**2.9, pi**/**2)

self.gurdy\_movement\_look(upperlegM1\_angle,

upperlegM2\_angle,

upperlegM3\_angle,

lowerlegM1\_angle,

lowerlegM2\_angle,

lowerlegM3\_angle)

​

**def** movement\_random\_loop(self):

"""

Executed movements in a random way

:return:

"""

rospy.loginfo("Start Moving Gurdy...")

**while** **not** rospy.is\_shutdown():

self.gurdy\_init\_pos\_sequence()

*#self.gurdy\_moverandomly()*

self.gurdy\_hop()

​

**if** \_\_name\_\_ **==** "\_\_main\_\_":

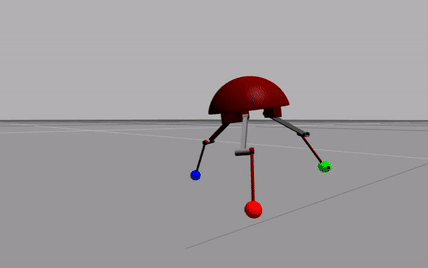
gurdy\_jointmover\_object **=** gurdyJointMover()

gurdy\_jointmover\_object.movement\_random\_loop()

​

​

If everything went well, you should have something similar to this:



Exercise U3-2

Create alternative versions of Gurdy by following these ideas:

1. Add an extra joint in each upper leg that allows it to rotate on the Z axis and change the angle at will.
2. Create an alternative version of Gurdy with an extra arm on the top of the head, and place a camera on the tip of the arm.
3. Change the foot joints from fix to primatic so that it can change the length of its legs.

END Exercise U3-2

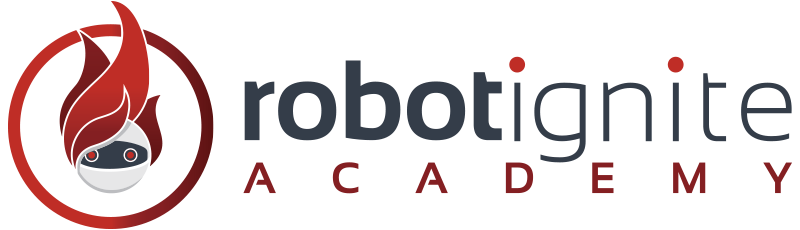
Exercise U3-3

1. Create an algorithm that moves Gurdy in the direction that you desire through a cmd\_vel topic publishing.
2. Create a function that detects data through the IMU when it has fallen and is upside-down, and starts a recovery routine to get on its feet again.
3. Try to make him able to jump high. Bear in mind that you will have to change effort limits, PID values, and so on, without losing control of Gurdy.

END Exercise U3-3

Solution Exercise U3.2 and U3.3

Please try to do it by yourself, unless you get stuck or need some inspiration. You will learn much more if you fight for each exercise.



Follow this link to open the solutions notebook for Adapt URDF for Gazebo Simulator: [solutions\_Course\_urdfROS\_Unit\_3](https://i-0500135e5b3ce7d04.robotigniteacademy.com/d7e6f789-1484-4bd3-aac3-7d6fe70eb589/jupyter/notebooks/course_urdf_ROS/extra_files/solutions_Course_urdfROS_Unit_3.ipynb)