[Lab 6]

From Unity to ROS

Resources

[1] GitHub of the lab: https://github.com/emarche/Mobile-robotics-4S009023-UniVR

Requirements

Since TurtleBot3's packages are built with python2, we have to downgrade our model trained with python3 and Tensorflow 2.3 to test it with ROS.

Hence, the following libraries are required:

- pip2
- Tensorflow 1.14 installed with pip2

Unity -> ROS

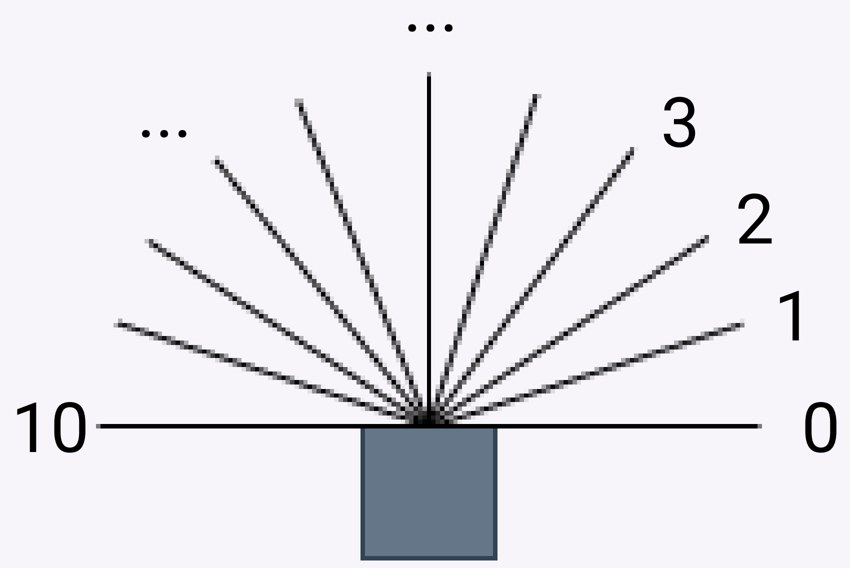
We have to read the data for the input of the network directly from the ROS's topics and encode them to reflect the same information of Unity:

- To read the data from the topics, we use the same functions coded in the bug algorithm.
- Since the input space of a neural network is typically normalized in the range [-1, 1] or [0,1], we also have to normalize the topic's data.

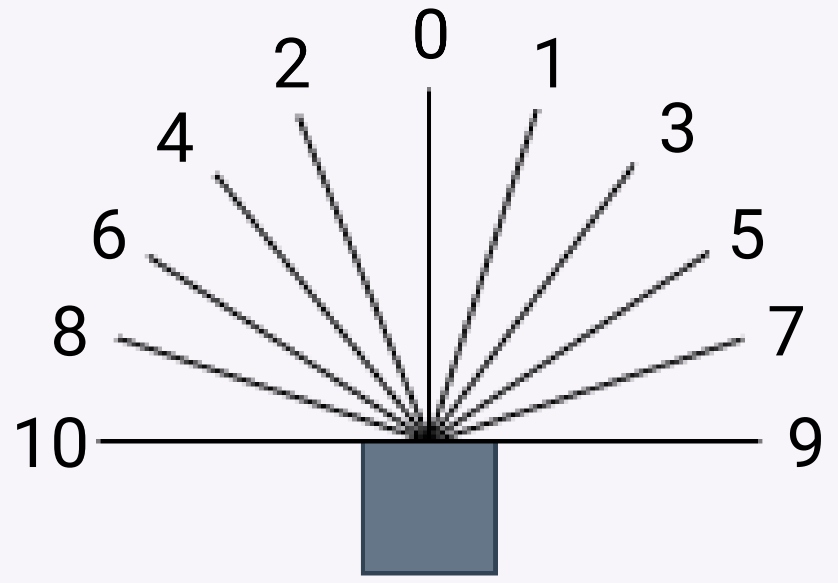
We provide a python script in the GitHub that implements these mappings: "robot_interface.py".

In detail, we provide a set of scripts, with the same structure as the training scripts of the last lesson, that is used to test a model trained in Unity, in a ROS environment (i.e., Gazebo), hence on the real robot.

Mapping Lidar



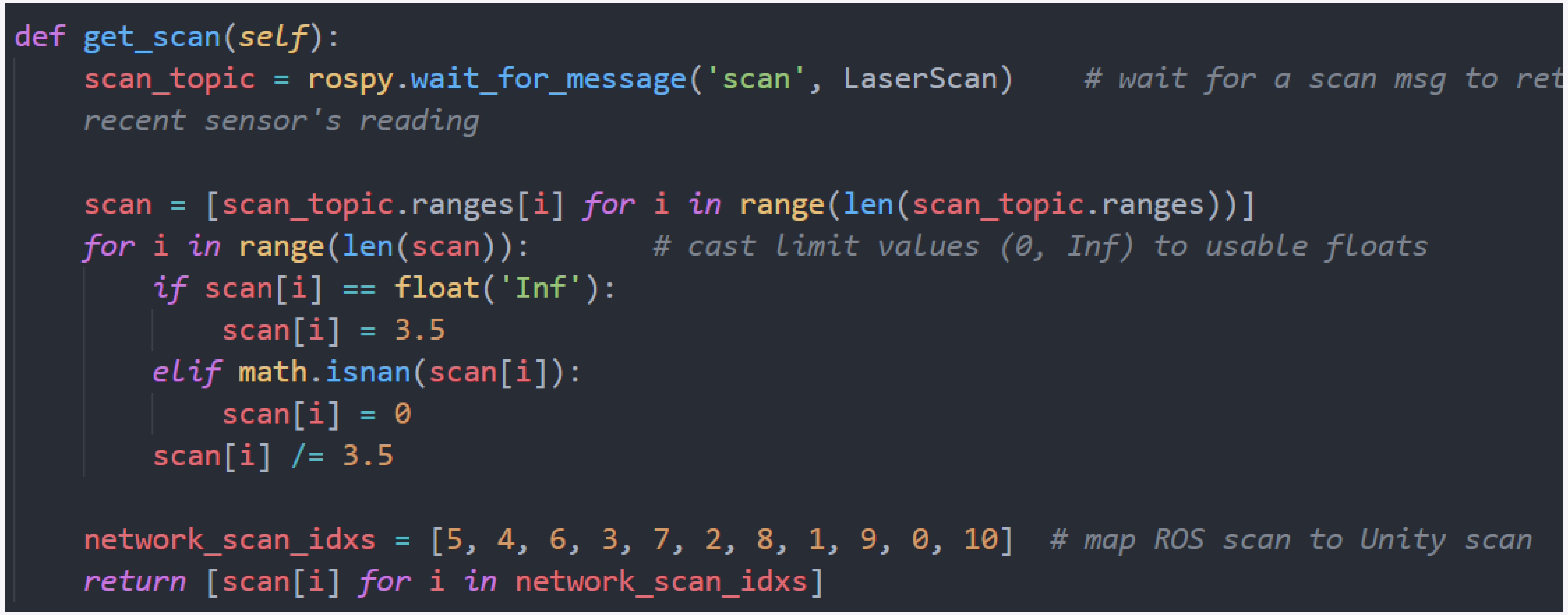
Indexes of ROS scans



Indexes of Unity scans

Mapping Lidar

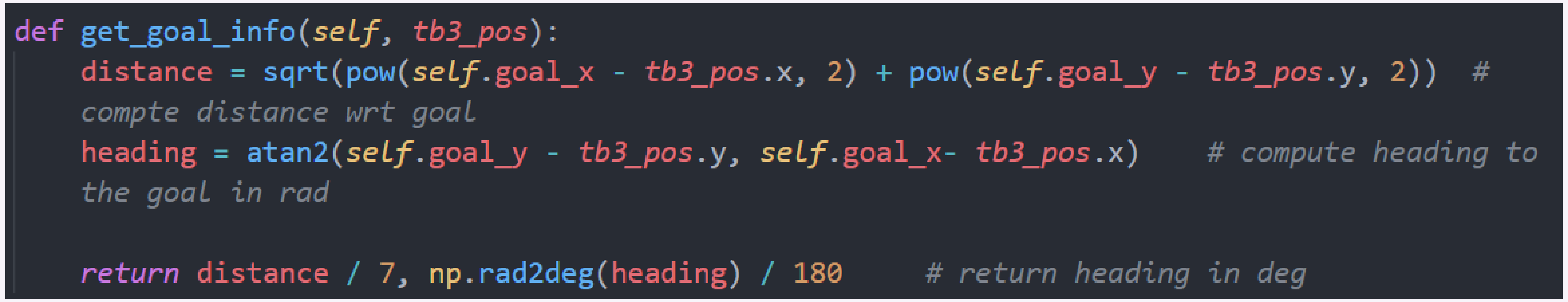
Note that we normalize the scans in range [0.1] by dividing the topic's values by 3.5 (i.e., the limits of the LDS-01)



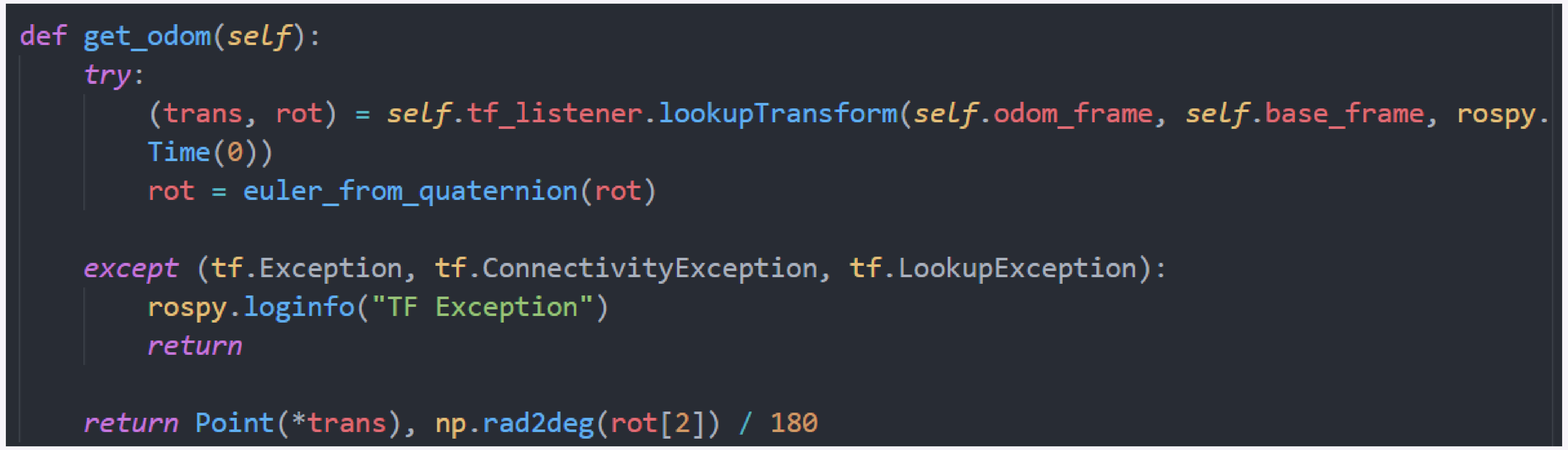
```
def get_scan(self):
    scan_topic = rospy.wait_for_message('scan', LaserScan)    # wait for a scan msg to ret
    recent sensor's reading

    scan = [scan_topic.ranges[i] for i in range(len(scan_topic.ranges))]
    for i in range(len(scan)):       # cast limit values (0, Inf) to usable floats
        if scan[i] == float('Inf'):
            scan[i] = 3.5
        elif math.isnan(scan[i]):
            scan[i] = 0
        scan[i] /= 3.5

    network_scan_idxs = [5, 4, 6, 3, 7, 2, 8, 1, 9, 0, 10]  # map ROS scan to Unity scan
    return [scan[i] for i in network_scan_idxs]
```

Goal Information

We also normalize distance and heading towards the goal as well as the current heading of the TurtleBot3, under the assumption that the max distance between two points in the environment is approximately 7 meters (you have to change this value in different scenarios)



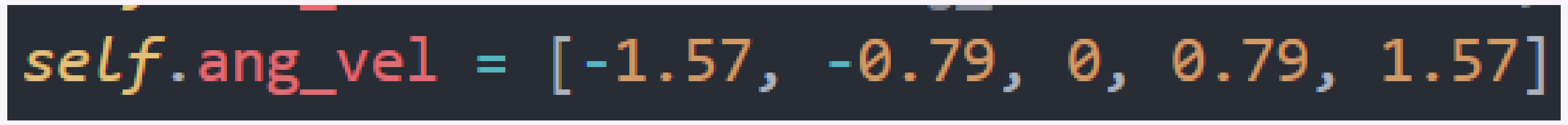
```python
def get_goal_info(self, tb3_pos):
    distance = sqrt(pow(self.goal_x - tb3_pos.x, 2) + pow(self.goal_y - tb3_pos.y, 2))  #
    compte distance wrt goal
    heading = atan2(self.goal_y - tb3_pos.y, self.goal_x- tb3_pos.x)    # compute heading to
    the goal in rad

    return distance / 7, np.rad2deg(heading) / 180     # return heading in deg
```



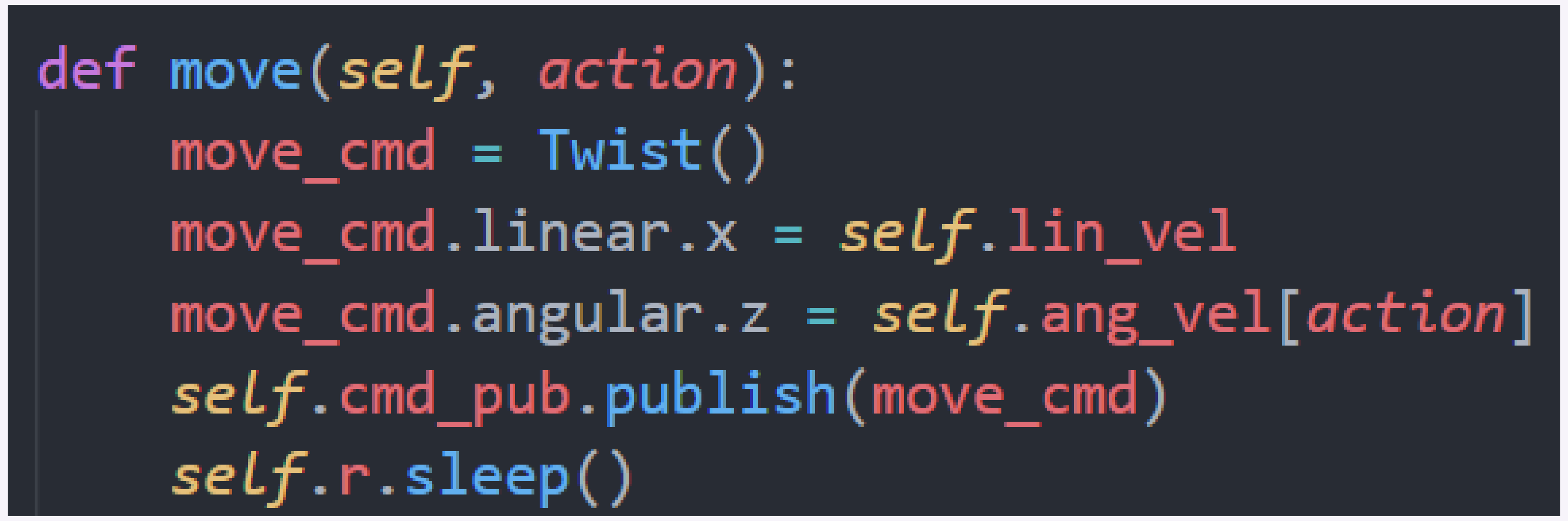
```python
def get_odom(self):
    try:
        (trans, rot) = self.tf_listener.lookupTransform(self.odom_frame, self.base_frame, rospy.
        Time(0))
        rot = euler_from_quaternion(rot)

    except (tf.Exception, tf.ConnectivityException, tf.LookupException):
        rospy.loginfo("TF Exception")
        return

    return Point(*trans), np.rad2deg(rot[2]) / 180
```

Map Actions

Finally we have to map the discrete actions of the double DQN, to actual velocity commands; this is already implemented in the move function that retrieves the angular velocity from a set of 5 predefined angular velocities (that are a direct map of the actions of the Unity environment, but in rad/s)



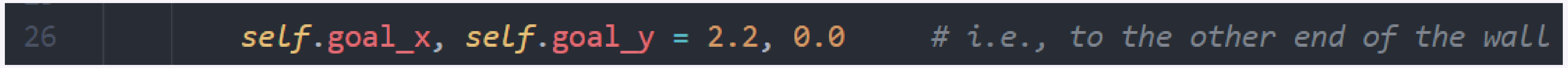
```python
self.ang_vel = [-1.57, -0.79, 0, 0.79, 1.57]
```



```python
def move(self, action):
    move_cmd = Twist()
    move_cmd.linear.x = self.lin_vel
    move_cmd.angular.z = self.ang_vel[action]
    self.cmd_pub.publish(move_cmd)
    self.r.sleep()
```

Robot Interface

As in the Bug script of the previous lesson, you can specify the desired goal position in the "robot_interface.py" script (inside the "utils" folder) at line 26.



```
26          self.goal_x, self.goal_y = 2.2, 0.0     # i.e., to the other end of the wall
```

Furthermore, this ROS interface publishes a velocity message with velocities set to 0 on the shutdown of the node, in order to stop the robot.

Exercise

Test a model trained in Unity on Gazebo, using the provided python scripts:

- Launch the Bug0 environment of [Lab 4] (manually remove the obstacle in the "model.sdf" file if you are testing a model trained without obstacles).
- Launch the provided python script (with python2).

- What happens by setting different velocities or discretizations?
- Can you use the same scripts on the real robot?

Projects

Some project ideas are available on the course website:

- http://profs.sci.univr.it/~farinelli/courses/mr/progetti-mr.html