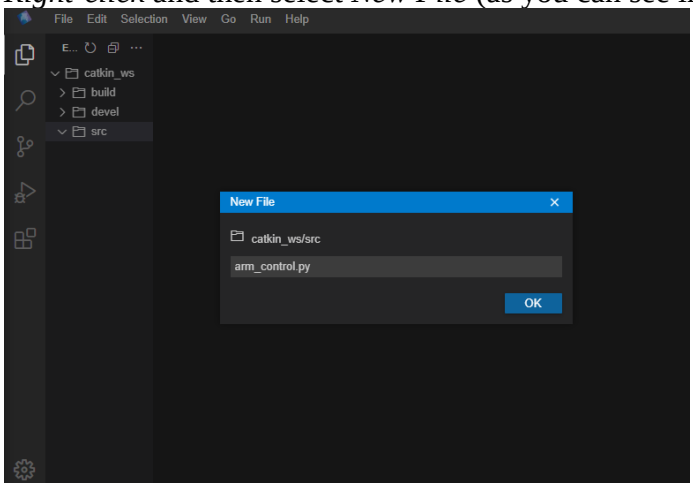
Nama : Nurul Amelia

NIM : 1103194032

[Tugas – 6] - Course Simulation - Python 3 for Robotics - Robotics & ROS Online Courses | The Construct (theconstructsim.com)

[Unit 1]

- ✓ Go to the IDE and select the *src* folder, inside the *catkin_ws*.
- ✓ *Right-click* and then select *New File* (as you can see in the below image)



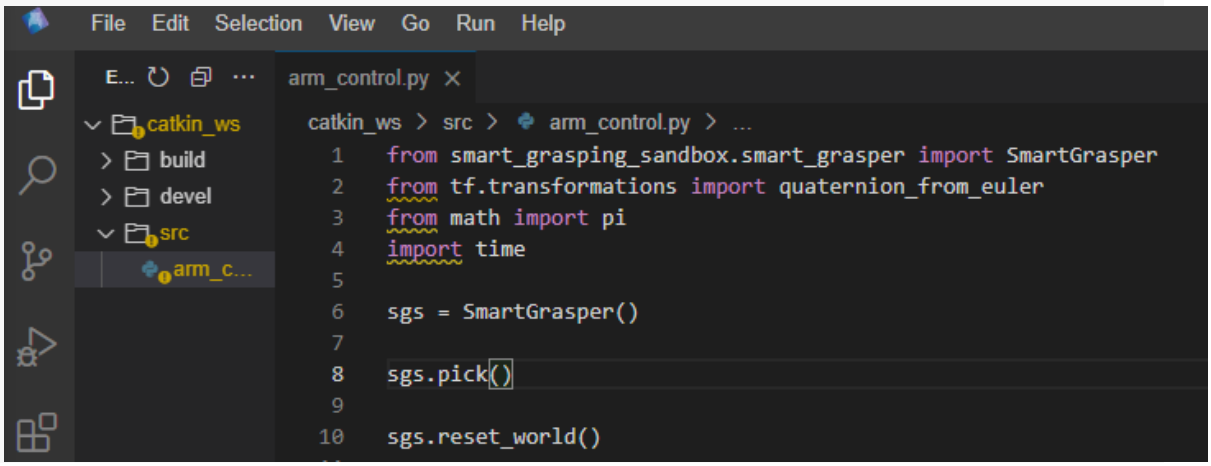
- ✓ Then, this is first Python program in arm_control.py.

```
from smart_grasping_sandbox.smart_grasper import SmartGrasper
from tf.transformations import quaternion_from_euler
from math import pi
import time

sgs = SmartGrasper()

sgs.pick()

sgs.reset_world()
```



- ✓ Execute Program.

```
user:~$ cd /home/user/catkin_ws/src
user:~/catkin_ws/src$ ls
CMakeLists.txt  arm_control.py
user:~/catkin_ws/src$ python arm_control.py
```

Finish

```
#1                    #2                    #3                    #4
t empty. Fix your URDF file by explicitly specifying collision geometry.
[ WARN] [1669640867.483863523]: Link H1_F2_palm_link has visual geometry but no collision geometry. Collision geometry will be lef
t empty. Fix your URDF file by explicitly specifying collision geometry.
[ WARN] [1669640867.529688089]: Link H1_F3_palm_link has visual geometry but no collision geometry. Collision geometry will be lef
t empty. Fix your URDF file by explicitly specifying collision geometry.
[ WARN] [1669640867.625510078, 709.032000000]: Kinematics solver doesn't support #attempts anymore, but only a timeout.
Please remove the parameter '/robot_description_kinematics/arm/kinematics_solver_attempts' from your configuration.
[ INFO] [1669640868.680424575, 710.057000000]: Ready to take commands for planning group arm.
[ INFO] [1669640868.924093095, 710.298000000]: Ready to take commands for planning group hand.
[INFO] [1669640869.245432, 710.611000]: STARTING CONTROLLERS
[INFO] [1669640869.351674, 710.613000]: Moving to Pregrasp
[ INFO] [1669640869.519631258, 710.774000000]: ABORTED: Solution found but controller failed during execution
[INFO] [1669640870.966961, 712.193000]: Grasping
[ INFO] [1669640879.253952483, 720.261000000]: ABORTED: Solution found but controller failed during execution
[INFO] [1669640879.354410, 720.352000]: Lifting
[INFO] [1669640882.946764, 723.675000]: STARTING CONTROLLERS
```

Robot arm in the simulation moves and picks up the red ball, can you see in video demo.

[Unit 2]

✓ Create directory robot_control and a new pyhton script named robot_control_class.py.

```
cd ~/catkin_ws/src/
mkdir robot_control
cd robot_control
touch pyscript1.py
touch robot_control_class.py
```

✓ Python program in robot_control_class.py

```python
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import time


class RobotControl():

    def __init__(self, robot_name="turtlebot"):
        rospy.init_node('robot_control_node', anonymous=True)

        if robot_name == "summit":
            rospy.loginfo("Robot Summit...")
            cmd_vel_topic = "/summit_xl_control/cmd_vel"
            # We check sensors working
            self._check_summit_laser_ready()
        else:
            rospy.loginfo("Robot Turtlebot...")
            cmd_vel_topic='/cmd_vel'
```

```
            self._check_laser_ready()

        # We start the publisher
        self.vel_publisher = rospy.Publisher(cmd_vel_topic, Twist, queue_size=1)
        self.cmd = Twist()

        self.laser_subscriber = rospy.Subscriber(
            '/kobuki/laser/scan', LaserScan, self.laser_callback)
        self.summit_laser_subscriber = rospy.Subscriber(
            '/hokuyo_base/scan', LaserScan, self.summit_laser_callback)

        self.ctrl_c = False
        self.rate = rospy.Rate(1)
        rospy.on_shutdown(self.shutdownhook)


    def _check_summit_laser_ready(self):
        self.summit_laser_msg = None
        rospy.loginfo("Checking Summit Laser...")
        while self.summit_laser_msg is None and not rospy.is_shutdown():
            try:
                self.summit_laser_msg = rospy.wait_for_message("/hokuyo_base/scan",
LaserScan, timeout=1.0)
                rospy.logdebug("Current /hokuyo_base/scan READY=>" +
str(self.summit_laser_msg))

            except:
                rospy.logerr("Current /hokuyo_base/scan not ready yet, retrying for
getting scan")
        rospy.loginfo("Checking Summit Laser...DONE")
        return self.summit_laser_msg

    def _check_laser_ready(self):
        self.laser_msg = None
        rospy.loginfo("Checking Laser...")
        while self.laser_msg is None and not rospy.is_shutdown():
            try:
                self.laser_msg = rospy.wait_for_message("/kobuki/laser/scan",
LaserScan, timeout=1.0)
                rospy.logdebug("Current /kobuki/laser/scan READY=>" +
str(self.laser_msg))

            except:
                rospy.logerr("Current /kobuki/laser/scan not ready yet, retrying for getting
scan")
        rospy.loginfo("Checking Laser...DONE")
        return self.laser_msg

    def publish_once_in_cmd_vel(self):
        """
```

```
        This is because publishing in topics sometimes fails the first time you publish.
        In continuous publishing systems, this is no big deal, but in systems that
publish only
        once, it IS very important.
        """
        while not self.ctrl_c:
            connections = self.vel_publisher.get_num_connections()
            if connections > 0:
                self.vel_publisher.publish(self.cmd)
                #rospy.loginfo("Cmd Published")
                break
            else:
                self.rate.sleep()

    def shutdownhook(self):
        # works better than the rospy.is_shutdown()
        self.ctrl_c = True

    def laser_callback(self, msg):
        self.laser_msg = msg

    def summit_laser_callback(self, msg):
        self.summit_laser_msg = msg

    def get_laser(self, pos):
        time.sleep(1)
        return self.laser_msg.ranges[pos]

    def get_laser_summit(self, pos):
        time.sleep(1)
        return self.summit_laser_msg.ranges[pos]

    def get_front_laser(self):
        time.sleep(1)
        return self.laser_msg.ranges[360]

    def get_laser_full(self):
        time.sleep(1)
        return self.laser_msg.ranges

    def stop_robot(self):
        #rospy.loginfo("shutdown time! Stop the robot")
        self.cmd.linear.x = 0.0
        self.cmd.angular.z = 0.0
        self.publish_once_in_cmd_vel()

    def move_straight(self):

        # Initilize velocities
        self.cmd.linear.x = 0.5
```

```
        self.cmd.linear.y = 0
        self.cmd.linear.z = 0
        self.cmd.angular.x = 0
        self.cmd.angular.y = 0
        self.cmd.angular.z = 0

        # Publish the velocity
        self.publish_once_in_cmd_vel()

    def move_straight_time(self, motion, speed, time):

        # Initilize velocities
        self.cmd.linear.y = 0
        self.cmd.linear.z = 0
        self.cmd.angular.x = 0
        self.cmd.angular.y = 0
        self.cmd.angular.z = 0

        if motion == "forward":
            self.cmd.linear.x = speed
        elif motion == "backward":
            self.cmd.linear.x = - speed

        i = 0
        # loop to publish the velocity estimate, current_distance = velocity * (t1 - t0)
        while (i <= time):

            # Publish the velocity
            self.vel_publisher.publish(self.cmd)
            i += 1
            self.rate.sleep()

        # set velocity to zero to stop the robot
        self.stop_robot()

        s = "Moved robot " + motion + " for " + str(time) + " seconds"
        return s


    def turn(self, clockwise, speed, time):

        # Initilize velocities
        self.cmd.linear.x = 0
        self.cmd.linear.y = 0
        self.cmd.linear.z = 0
        self.cmd.angular.x = 0
        self.cmd.angular.y = 0

        if clockwise == "clockwise":
            self.cmd.angular.z = -speed
```

```
        else:
            self.cmd.angular.z = speed

        i = 0
        # loop to publish the velocity estimate, current_distance = velocity * (t1 - t0)
        while (i <= time):

            # Publish the velocity
            self.vel_publisher.publish(self.cmd)
            i += 1
            self.rate.sleep()

        # set velocity to zero to stop the robot
        self.stop_robot()

        s = "Turned robot " + clockwise + " for " + str(time) + " seconds"
        return s


if __name__ == '__main__':

    robotcontrol_object = RobotControl()
    try:
        robotcontrol_object.move_straight()

    except rospy.ROSInterruptException:
        pass
```

✓ In the pyscript1.py :

```
from robot_control_class import RobotControl

rc = RobotControl()

a = rc.get_laser(360)

print ("The distance measured is: ", a)
```

Result :

```
user:~/catkin_ws/src/robot_control$ python pyscript1.py
[INFO] [1669686693.617519, 0.000000]: Robot Turtlebot...
[INFO] [1669686693.618582, 0.000000]: Checking Laser...
[INFO] [1669686693.654377, 891.341000]: Checking Laser...DONE
The distance measured is:  inf
user:~/catkin_ws/src/robot_control$ python pyscript1.py
[INFO] [1669687009.472974, 0.000000]: Robot Turtlebot...
[INFO] [1669687009.474042, 0.000000]: Checking Laser...
[INFO] [1669687009.508860, 1206.382000]: Checking Laser...DONE
The distance measured is:  inf
```

✓ Variable

catkin_ws > src > robot_control > variables.py > ...

```python
from robot_control_class import RobotControl

robotcontrol = RobotControl()

laser1 = robotcontrol.get_laser(0)
print("The laser value received is: ", laser1)

laser2 = robotcontrol.get_laser(360)
print("The laser value received is: ", laser2)

laser2 = robotcontrol.get_laser(719)
print("The laser value received is: ", laser2)
```

Result :

```
user:~/catkin_ws/src/robot_control$ python variables.py
[INFO] [1669687154.906198, 0.000000]: Robot Turtlebot...
[INFO] [1669687154.907524, 0.000000]: Checking Laser...
[INFO] [1669687154.933414, 1351.440000]: Checking Laser...DONE
The laser value received is:  2.8578460216522217
The laser value received is:  inf
The laser value received is:  inf
```

✓ Lists

catkin_ws > src > robot_control > lists.py > ...

```python
from robot_control_class import RobotControl

rc = RobotControl()

l = rc.get_laser_full()

print ("Position 0: ", l[0])
print ("Position 360: ", l[360])
print ("Position 719: ", l[719])
```

\

Result :

```
user:~/catkin_ws/src/robot_control$ python lists.py
[INFO] [1669687195.697972, 0.000000]: Robot Turtlebot...
[INFO] [1669687195.699241, 0.000000]: Checking Laser...
[INFO] [1669687195.723752, 1392.129000]: Checking Laser...DONE
Position 0:  inf
Position 360:  inf
Position 719:  inf
```

✓ Dictionaries

catkin_ws > src > robot_control > dictionaries.py > ...

```python
from robot_control_class import RobotControl

rc = RobotControl()

l = rc.get_laser_full()

dict = {"P0": l[0], "P100": l[100], "P200": l[200], "P300": l[300],

print (dict)
```

Result :

```
user:~/catkin_ws/src/robot_control$ python dictionaries.py
[INFO] [1669687243.519144, 0.000000]: Robot Turtlebot...
[INFO] [1669687243.521450, 0.000000]: Checking Laser...
[INFO] [1669687243.557899, 1439.831000]: Checking Laser...DONE
{'P0': inf, 'P100': inf, 'P200': inf, 'P300': inf, 'P400': inf, 'P500': inf, 'P600': inf, 'P719': inf}
```

✓ Input 1

catkin_ws > src > robot_control > input.py > ...

```python
name = input("What's your name? ")

print("Nice to meet you, " + name)
```

Result :

```
user:~/catkin_ws/src/robot_control$ python input.py
What's your name? Nurul Amelia
Nice to meet you, Nurul Amelia
```

✓ Input 2

catkin_ws > src > robot_control > input2.py > ...

```python
age = int(input("What's your age? "))

age2 = age + 1

print("So next year you will be %d years old!" % age2)
```

Result :

```
user:~/catkin_ws/src/robot_control$ python input2.py
What's your age? 20
So next year you will be 21 years old!
```

✓ Test Input

catkin_ws > src > robot_control > test_input.py > ...

```python
from robot_control_class import RobotControl

num = int(input("Select a number between 0 and 719: "))

rc = RobotControl()
a = rc.get_laser(num)

print ("The laser value received is: ", a)
```

Result :

```
user:~/catkin_ws/src/robot_control$ python test_input.py
Select a number between 0 and 719: 2
[INFO] [1669687441.548615, 0.000000]: Robot Turtlebot...
[INFO] [1669687441.549829, 0.000000]: Checking Laser...
[INFO] [1669687441.595315, 1637.362000]: Checking Laser...DONE
The laser value received is:  inf
```

[Unit 3]

✓ **Condition**

catkin_ws > src > robot_control > fav_movie.py > ...

```python
movie = input("What's your favorite movie? ")

if movie == "Avengers Endgame" or movie == "Titanic":
    print("Good choice!")
elif movie == "Star Wars":
    print("Also a good choice!")
else:
    print("You really are an interesting specimen")
```

Result :

```
user:~/catkin_ws/src/robot_control$ python fav_movie.py
What's your favorite movie? Star Wars
Also a good choice!
```

✓ **Exercise**

test_if.py

catkin_ws > src > robot_control > test_if.py > ...

```python
from robot_control_class import RobotControl

robotcontrol = RobotControl()

a = robotcontrol.get_laser(360)

if a < 1:
    robotcontrol.stop_robot()

else:
    robotcontrol.move_straight()
```

```
user:~$ cd ~/catkin_ws/src/
user:~/catkin_ws/src$ cd robot_control
user:~/catkin_ws/src/robot_control$ python test_if.py
[INFO] [1669883439.823959, 0.000000]: Robot Turtlebot...
[INFO] [1669883439.825176, 0.000000]: Checking Laser...
[INFO] [1669883439.951828, 59.108000]: Checking Laser...DONE
```

test_while.py

catkin_ws > src > robot_control > test_while.py > ...

```python
from robot_control_class import RobotControl

robotcontrol = RobotControl()

a = robotcontrol.get_laser(360)

while a > 1:
    robotcontrol.move_straight()
    a = robotcontrol.get_laser(360)
    print("Current distance to wall: %f" % a)

robotcontrol.stop_robot()

print("Wall is at %f meters! Stop the robot!" % a)
```

```
user:~/catkin_ws/src/robot_control$ python test_while.py
[INFO] [1669883508.200848, 0.000000]: Robot Turtlebot...
[INFO] [1669883508.201865, 0.000000]: Checking Laser...
[INFO] [1669883508.328538, 14.217000]: Checking Laser...DONE
Current distance to wall: 2.058911
Current distance to wall: 1.563021
Current distance to wall: 1.066604
Current distance to wall: 0.570901
```

test_for.py

```
catkin_ws > src > robot_control > test_for.py > ...
1   from robot_control_class import RobotControl
2
3   robotcontrol = RobotControl()
4
5   l = robotcontrol.get_laser_full()
6
7   maxim = 0
8
9   for value in l:
10      if value > maxim:
11          maxim = value
12
13  print("The higher value in the list is: ", maxim)
```

```
user:~/catkin_ws/src/robot_control$ python test_for.py
[INFO] [1669883612.733254, 0.000000]: Robot Turtlebot...
[INFO] [1669883612.734414, 0.000000]: Checking Laser...
[INFO] [1669883612.881407, 30.391000]: Checking Laser...DONE
The higher value in the list is:  inf
```

[Unit 4] – Methods

Exercise

test_methods.py

```
catkin_ws > src > robot_control > test_methods.py > ...
1   from robot_control_class import RobotControl
2   import time
3
4   robotcontrol = RobotControl(robot_name="summit")
5
6
7   def move_x_seconds(secs):
8       robotcontrol.move_straight()
9       time.sleep(secs)
10      robotcontrol.stop_robot()
11
12
13  move_x_seconds(5)
```

```
user:~$ cd ~/catkin_ws/src/
user:~/catkin_ws/src$ cd robot_control
user:~/catkin_ws/src/robot_control$ python test_methods.py
[INFO] [1669884256.524403, 0.000000]: Robot Summit...
[INFO] [1669884256.526536, 0.000000]: Checking Summit Laser...
[INFO] [1669884256.695181, 279.097000]: Checking Summit Laser...DONE
```

test_methods2.py

```
catkin_ws > src > robot_control > test_methods2.py > ...
  1  from robot_control_class import RobotControl
  2
  3  robotcontrol = RobotControl(robot_name="summit")
  4
  5
  6  def get_laser_values(a, b, c):
  7      r1 = robotcontrol.get_laser_summit(a)
  8      r2 = robotcontrol.get_laser_summit(b)
  9      r3 = robotcontrol.get_laser_summit(c)
 10
 11      return [r1, r2, r3]
 12
 13
 14  l = get_laser_values(0, 500, 1000)
 15
 16  print("Reading 1: ", l[0])
 17  print("Reading 2: ", l[1])
 18  print("Reading 3: ", l[2])
```

```
user:~/catkin_ws/src/robot_control$ python test_methods2.py
[INFO] [1669884327.525776, 0.000000]: Robot Summit...
[INFO] [1669884327.526951, 0.000000]: Checking Summit Laser...
[INFO] [1669884327.686909, 39.593000]: Checking Summit Laser...DONE
Reading 1:  1.3467020988464355
Reading 2:  7.2234578132629395
Reading 3:  0.94508296251297
```

test_methods3.py

```
catkin_ws > src > robot_control > test_methods3.py > ...
  1  from robot_control_class import RobotControl
  2
  3  robotcontrol = RobotControl(robot_name="summit")
  4
  5  robotcontrol.move_straight_time("forward", 0.3, 5)
  6  robotcontrol.turn("clockwise", 0.3, 7)
```

```
user:~/catkin_ws/src/robot_control$ python test_methods3.py
[INFO] [1669884389.740795, 0.000000]: Robot Summit...
[INFO] [1669884389.742021, 0.000000]: Checking Summit Laser...
[INFO] [1669884389.950615, 33.078000]: Checking Summit Laser...DONE
```

test_methods4.py

```
catkin_ws > src > robot_control > test_methods4.py > ...
1  from robot_control_class import RobotControl
2
3  robotcontrol = RobotControl(robot_name="summit")
4
5  robotcontrol.turn("counter-clockwise", 0.3, 4)
6  robotcontrol.move_straight_time("forward", 0.3, 6)
7  robotcontrol.turn("counter-clockwise", 0.3, 4)
8  robotcontrol.move_straight_time("forward", 0.3, 7)
```

```
user:~/catkin_ws/src/robot_control$ python test_methods4.py
[INFO] [1669884464.893201, 0.000000]: Robot Summit...
[INFO] [1669884464.895336, 0.000000]: Checking Summit Laser...
[INFO] [1669884465.117503, 33.840000]: Checking Summit Laser...DONE
```

[Unit 5] Classes and Object Oriented Programming

jedi_class.py

```
catkin_ws > src > robot_control > jedi_class.py > ...
1  class Jedi:
2      def __init__(self, name):
3          self.jedi_name = name
4
5      def say_hi(self):
6          print('Hello, my name is ', self.jedi_name)
7
8
9  if __name__ == '__main__':
10
11     j1 = Jedi('ObiWan')
12     j1.say_hi()
13
14     j2 = Jedi('Anakin')
15     j2.say_hi()
```

```
user:~$ cd ~/catkin_ws/src/
user:~/catkin_ws/src$ cd robot_control
user:~/catkin_ws/src/robot_control$ python jedi_class.py
Hello, my name is  ObiWan
Hello, my name is  Anakin
```

test_class.py

```
from robot_control_class import RobotControl

class MoveRobot:
    def __init__(self, motion, clockwise, speed, time):
        self.robotcontrol = RobotControl(robot_name="summit")
        self.motion = motion
        self.clockwise = clockwise
        self.speed = speed
        self.time = time
        self.time_turn = 7.0 # This is an estimate time in which the robot will rotate 90 degrees

    def do_square(self):

        i = 0

        while (i < 4):
            self.move_straight()
            self.turn()
            i+=1

    def move_straight(self):
        self.robotcontrol.move_straight_time(self.motion, self.speed, self.time)

    def turn(self):
        self.robotcontrol.turn(self.clockwise, self.speed, self.time_turn)


mr1 = MoveRobot('forward', 'clockwise', 0.3, 4)
mr1.do_square()
mr2 = MoveRobot('forward', 'clockwise', 0.3, 8)
mr2.do_square()
```

```
user:~/catkin_ws/src/robot_control$ python test_class.py
[INFO] [1669885121.930381, 0.000000]: Robot Summit...
[INFO] [1669885121.931678, 0.000000]: Checking Summit Laser...
[INFO] [1669885122.099461, 190.145000]: Checking Summit Laser...DONE
[INFO] [1669885174.964056, 242.152000]: Robot Summit...
[INFO] [1669885174.966017, 242.154000]: Checking Summit Laser...
[INFO] [1669885174.999662, 242.183000]: Checking Summit Laser...DONE
user:~/catkin_ws/src/robot_control$ python test_class.py
[INFO] [1669885289.140489, 0.000000]: Robot Summit...
[INFO] [1669885289.141785, 0.000000]: Checking Summit Laser...
[INFO] [1669885289.180574, 354.443000]: Checking Summit Laser...DONE
[INFO] [1669885342.183331, 406.451000]: Robot Summit...
[INFO] [1669885342.185580, 406.453000]: Checking Summit Laser...
[INFO] [1669885342.200332, 406.466000]: Checking Summit Laser...DONE
```

robot_control_class.py

```
#!/usr/bin/env python
```

```
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import time


class RobotControl():

    def __init__(self, robot_name="turtlebot"):
        rospy.init_node('robot_control_node', anonymous=True)

        if robot_name == "summit":
            rospy.loginfo("Robot Summit...")
            cmd_vel_topic = "/summit_xl_control/cmd_vel"
            # We check sensors working
            self._check_summit_laser_ready()
        else:
            rospy.loginfo("Robot Turtlebot...")
            cmd_vel_topic='/cmd_vel'
            self._check_laser_ready()

        # We start the publisher
        self.vel_publisher = rospy.Publisher(cmd_vel_topic, Twist, queue_size=1)
        self.cmd = Twist()

        self.laser_subscriber = rospy.Subscriber(
            '/kobuki/laser/scan', LaserScan, self.laser_callback)
        self.summit_laser_subscriber = rospy.Subscriber(
            '/hokuyo_base/scan', LaserScan, self.summit_laser_callback)

        self.ctrl_c = False
        self.rate = rospy.Rate(1)
        rospy.on_shutdown(self.shutdownhook)


    def _check_summit_laser_ready(self):
        self.summit_laser_msg = None
        rospy.loginfo("Checking Summit Laser...")
        while self.summit_laser_msg is None and not rospy.is_shutdown():
            try:
                self.summit_laser_msg = rospy.wait_for_message("/hokuyo_base/scan",
LaserScan, timeout=1.0)
                rospy.logdebug("Current /hokuyo_base/scan READY=>" +
str(self.summit_laser_msg))

            except:
                rospy.logerr("Current /hokuyo_base/scan not ready yet, retrying for getting scan")
        rospy.loginfo("Checking Summit Laser...DONE")
        return self.summit_laser_msg
```

```
    def _check_laser_ready(self):
        self.laser_msg = None
        rospy.loginfo("Checking Laser...")
        while self.laser_msg is None and not rospy.is_shutdown():
            try:
                self.laser_msg = rospy.wait_for_message("/kobuki/laser/scan", LaserScan,
timeout=1.0)
                rospy.logdebug("Current /kobuki/laser/scan READY=>" + str(self.laser_msg))

            except:
                rospy.logerr("Current /kobuki/laser/scan not ready yet, retrying for getting scan")
        rospy.loginfo("Checking Laser...DONE")
        return self.laser_msg

    def publish_once_in_cmd_vel(self):
        """
        This is because publishing in topics sometimes fails the first time you publish.
        In continuous publishing systems, this is no big deal, but in systems that publish only
        once, it IS very important.
        """
        while not self.ctrl_c:
            connections = self.vel_publisher.get_num_connections()
            if connections > 0:
                self.vel_publisher.publish(self.cmd)
                #rospy.loginfo("Cmd Published")
                break
            else:
                self.rate.sleep()

    def shutdownhook(self):
        # works better than the rospy.is_shutdown()
        self.ctrl_c = True

    def laser_callback(self, msg):
        self.laser_msg = msg

    def summit_laser_callback(self, msg):
        self.summit_laser_msg = msg

    def get_laser(self, pos):
        time.sleep(1)
        return self.laser_msg.ranges[pos]

    def get_laser_summit(self, pos):
        time.sleep(1)
        return self.summit_laser_msg.ranges[pos]

    def get_front_laser(self):
        time.sleep(1)
```

```
    return self.laser_msg.ranges[360]

def get_laser_full(self):
    time.sleep(1)
    return self.laser_msg.ranges

def stop_robot(self):
    #rospy.loginfo("shutdown time! Stop the robot")
    self.cmd.linear.x = 0.0
    self.cmd.angular.z = 0.0
    self.publish_once_in_cmd_vel()

def move_straight(self):

    # Initilize velocities
    self.cmd.linear.x = 0.5
    self.cmd.linear.y = 0
    self.cmd.linear.z = 0
    self.cmd.angular.x = 0
    self.cmd.angular.y = 0
    self.cmd.angular.z = 0

    # Publish the velocity
    self.publish_once_in_cmd_vel()

def move_straight_time(self, motion, speed, time):

    # Initilize velocities
    self.cmd.linear.y = 0
    self.cmd.linear.z = 0
    self.cmd.angular.x = 0
    self.cmd.angular.y = 0
    self.cmd.angular.z = 0

    if motion == "forward":
        self.cmd.linear.x = speed
    elif motion == "backward":
        self.cmd.linear.x = - speed

    i = 0
    # loop to publish the velocity estimate, current_distance = velocity * (t1 - t0)
    while (i <= time):

        # Publish the velocity
        self.vel_publisher.publish(self.cmd)
        i += 1
        self.rate.sleep()

    # set velocity to zero to stop the robot
    self.stop_robot()
```

```
        s = "Moved robot " + motion + " for " + str(time) + " seconds"
        return s


    def turn(self, clockwise, speed, time):

        # Initilize velocities
        self.cmd.linear.x = 0
        self.cmd.linear.y = 0
        self.cmd.linear.z = 0
        self.cmd.angular.x = 0
        self.cmd.angular.y = 0

        if clockwise == "clockwise":
            self.cmd.angular.z = -speed
        else:
            self.cmd.angular.z = speed

        i = 0
        # loop to publish the velocity estimate, current_distance = velocity * (t1 - t0)
        while (i <= time):

            # Publish the velocity
            self.vel_publisher.publish(self.cmd)
            i += 1
            self.rate.sleep()

        # set velocity to zero to stop the robot
        self.stop_robot()

        s = "Turned robot " + clockwise + " for " + str(time) + " seconds"
        return s


if __name__ == '__main__':

    robotcontrol_object = RobotControl()
    try:
        robotcontrol_object.move_straight()

    except rospy.ROSInterruptException:
        pass
```

```
user:~/catkin_ws/src/robot_control$ python robot_control_class.py
[INFO] [1669886255.225669, 0.000000]: Robot Turtlebot...
[INFO] [1669886255.227427, 0.000000]: Checking Laser...
[ERROR] [1669886256.245882, 489.896000]: Current /kobuki/laser/scan not ready yet, retrying for getting scan
[ERROR] [1669886257.253511, 490.580000]: Current /kobuki/laser/scan not ready yet, retrying for getting scan
^C[ERROR] [1669886258.431314, 490.955000]: Current /kobuki/laser/scan not ready yet, retrying for getting scan
[INFO] [1669886258.433554, 490.955000]: Checking Laser...DONE
```

[Unit 6] MicroProject

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist, Point, Quaternion
from sensor_msgs.msg import LaserScan
from nav_msgs.msg import Odometry
import tf
from tf.transformations import euler_from_quaternion, quaternion_from_euler
import time
from math import radians, copysign, sqrt, pow, pi
import PyKDL


class RobotControl():

    def __init__(self):
        rospy.init_node('robot_control_node', anonymous=True)
        self.vel_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
        self.summit_vel_publisher = rospy.Publisher('/summit_xl_control/cmd_vel', Twist,
queue_size=1)
        self.laser_subscriber = rospy.Subscriber(
            '/kobuki/laser/scan', LaserScan, self.laser_callback)
        self.summit_laser_subscriber = rospy.Subscriber(
            '/hokuyo_base/scan', LaserScan, self.summit_laser_callback)
        self.odom_sub = rospy.Subscriber ('/odom', Odometry, self.odom_callback)
        self.cmd = Twist()
        self.laser_msg = LaserScan()
        self.summit_laser_msg = LaserScan()
        self.roll = 0.0
        self.pitch = 0.0
        self.yaw = 0.0
        self.ctrl_c = False
        self.rate = rospy.Rate(10)
        self.tf_listener = tf.TransformListener()
        self.odom_frame = '/odom'
        self.base_frame = '/base_link'
        self.angular_tolerance = radians(2)
        rospy.on_shutdown(self.shutdownhook)

    def publish_once_in_cmd_vel(self):
        """
        This is because publishing in topics sometimes fails the first time you publish.
        In continuos publishing systems there is no big deal but in systems that publish only
        once it IS very important.
        """
        while not self.ctrl_c:
            connections = self.vel_publisher.get_num_connections()
            summit_connections = self.summit_vel_publisher.get_num_connections()
```

```
            if connections > 0 or summit_connections > 0:
                self.vel_publisher.publish(self.cmd)
                self.summit_vel_publisher.publish(self.cmd)
                #rospy.loginfo("Cmd Published")
                break
            else:
                self.rate.sleep()

    def shutdownhook(self):
        # works better than the rospy.is_shutdown()
        self.ctrl_c = True

    def laser_callback(self, msg):
        self.laser_msg = msg

    def summit_laser_callback(self, msg):
        self.summit_laser_msg = msg

    def odom_callback(self, msg):
        orientation_q = msg.pose.pose.orientation
        orientation_list = [orientation_q.x, orientation_q.y, orientation_q.z, orientation_q.w]
        (self.roll, self.pitch, self.yaw) = euler_from_quaternion (orientation_list)

    def get_laser(self, pos):
        time.sleep(1)
        return self.laser_msg.ranges[pos]

    def get_laser_summit(self, pos):
        time.sleep(1)
        return self.summit_laser_msg.ranges[pos]

    def get_front_laser(self):
        time.sleep(1)
        return self.laser_msg.ranges[360]

    def get_laser_full(self):
        time.sleep(1)
        return self.laser_msg.ranges

    def stop_robot(self):
        #rospy.loginfo("shutdown time! Stop the robot")
        self.cmd.linear.x = 0.0
        self.cmd.angular.z = 0.0
        self.publish_once_in_cmd_vel()

    def move_straight(self):

        # Initilize velocities
        self.cmd.linear.x = 0.5
        self.cmd.linear.y = 0
```

```
        self.cmd.linear.z = 0
        self.cmd.angular.x = 0
        self.cmd.angular.y = 0
        self.cmd.angular.z = 0

        # Publish the velocity
        self.publish_once_in_cmd_vel()

    def move_straight_time(self, motion, speed, time):

        # Initilize velocities
        self.cmd.linear.y = 0
        self.cmd.linear.z = 0
        self.cmd.angular.x = 0
        self.cmd.angular.y = 0
        self.cmd.angular.z = 0

        if motion == "forward":
            self.cmd.linear.x = speed
        elif motion == "backward":
            self.cmd.linear.x = - speed

        i = 0
        # loop to publish the velocity estimate, current_distance = velocity * (t1 - t0)
        while (i <= time):

            # Publish the velocity
            self.vel_publisher.publish(self.cmd)
            self.summit_vel_publisher.publish(self.cmd)
            i += 0.1
            self.rate.sleep()

        # set velocity to zero to stop the robot
        self.stop_robot()

        s = "Moved robot " + motion + " for " + str(time) + " seconds"
        return s


    def turn(self, clockwise, speed, time):

        # Initilize velocities
        self.cmd.linear.x = 0
        self.cmd.linear.y = 0
        self.cmd.linear.z = 0
        self.cmd.angular.x = 0
        self.cmd.angular.y = 0

        if clockwise == "clockwise":
            self.cmd.angular.z = -speed
```

```
        else:
            self.cmd.angular.z = speed

        i = 0
        # loop to publish the velocity estimate, current_distance = velocity * (t1 - t0)

        while (i <= time):

            # Publish the velocity
            self.vel_publisher.publish(self.cmd)
            self.summit_vel_publisher.publish(self.cmd)
            i += 0.1
            self.rate.sleep()

        # set velocity to zero to stop the robot
        self.stop_robot()

        s = "Turned robot " + clockwise + " for " + str(time) + " seconds"
        return s

    def get_odom(self):

        # Get the current transform between the odom and base frames
        tf_ok = 0
        while tf_ok == 0 and not rospy.is_shutdown():
            try:
                self.tf_listener.waitForTransform('/base_link', '/odom', rospy.Time(),
rospy.Duration(1.0))
                tf_ok = 1
            except (tf.Exception, tf.ConnectivityException, tf.LookupException):
                pass

        try:
            (trans, rot)  = self.tf_listener.lookupTransform('odom', 'base_link', rospy.Time(0))
        except (tf.Exception, tf.ConnectivityException, tf.LookupException):
            rospy.loginfo("TF Exception")
            return

        return (Point(*trans), self.quat_to_angle(Quaternion(*rot)))

    def rotate(self, degrees):

        position = Point()

        # Get the current position
        (position, rotation) = self.get_odom()

        # Set the movement command to a rotation
        if degrees > 0:
            self.cmd.angular.z = 0.3
```

```
        else:
            self.cmd.angular.z = -0.3

        # Track the last angle measured
        last_angle = rotation

        # Track how far we have turned
        turn_angle = 0

        goal_angle = radians(degrees)

        # Begin the rotation
        while abs(turn_angle + self.angular_tolerance) < abs(goal_angle) and not
rospy.is_shutdown():
            # Publish the Twist message and sleep 1 cycle
            self.vel_publisher.publish(self.cmd)
            self.rate.sleep()

            # Get the current rotation
            (position, rotation) = self.get_odom()

            # Compute the amount of rotation since the last lopp
            delta_angle = self.normalize_angle(rotation - last_angle)

            turn_angle += delta_angle
            last_angle = rotation

        self.stop_robot()

    def quat_to_angle(self, quat):
        rot = PyKDL.Rotation.Quaternion(quat.x, quat.y, quat.z, quat.w)
        return rot.GetRPY()[2]

    def normalize_angle(self, angle):
        res = angle
        while res > pi:
            res -= 2.0 * pi
        while res < -pi:
            res += 2.0 * pi
        return res


if __name__ == '__main__':
    #rospy.init_node('robot_control_node', anonymous=True)
    robotcontrol_object = RobotControl()
    try:
        robotcontrol_object.move_straight()

    except rospy.ROSInterruptException:
        pass
```