University of Sheffield

COM2009-3009
Robotics

Lab Assignment #1

[Maxime Fontana]

[Tianyi Zhang]

[Sam Lowth]

Department of Computer Science

April 24, 2020

QUESTION 1
What are the advantages of using a launch file in ROS? What ROS command should be used to load a launch file, and what is the syntax for using this in a Linux terminal?

The advantages of using a launch file in ROS are that it provides a mean to launch multiple ROS nodes at the same time. To load a launch file we must use roslaunch. The relative Linux terminal syntax is roslaunch `[package_name] [launch_file].`

QUESTION 2
What is the difference between a package and a node, and what are the three things that must exist in order for a package to be valid in ROS?

A package is a ROS directory, they wrap up all the source files (nodes, ROS programs). A node is an executable program that uses ROS to communicate with one another (publisher and subscriber) via a topic.
The three things that must exist in order for a package to be valid in ROS are : a package.xml file (Information about the package), a CMakeLists.txt file (rules for compiling the package)and a src directory to store executable files. (python scripts etc)

QUESTION 3
What topic do we need to publish messages to to make the TurtleBot3 Robot move? What type of message does this topic use, and what format does the message take?

We need to publish messages to the `/cmd_vel` topic.
It accepts `geometry_msgs/Twist` which is of the format :
`geometry_msgs/Vector3` linear
float64 x
float64 y
float64 z
`geometry_msgs/Vector3` angular
float64 x
float64 y
float64 z

QUESTION 4
Using the rostopic echo command, we obtain the following data from the /odom topic to inform us of our robot's current odometry:

```
pose:
    pose:
        position:
            x: 0.649356
            y: -0.374484
            z: 0.0
        orientation:
            x: 0.0
            y: 0.0
```

z: -0.469191
w: 0.883097

How far has the robot moved from its origin in the X-Y plane? (Explain briefly how you arrived at this answer)

We can calculate how far the robot has moved from (0,0), its origin, considering that we are working in 2D. So by using the Pythagorean Theorem : $\sqrt{0.649356^2 + 0.374484^2}$ `which is equal to 0.7496 rounded up at` 10^{-4}

QUESTION 5

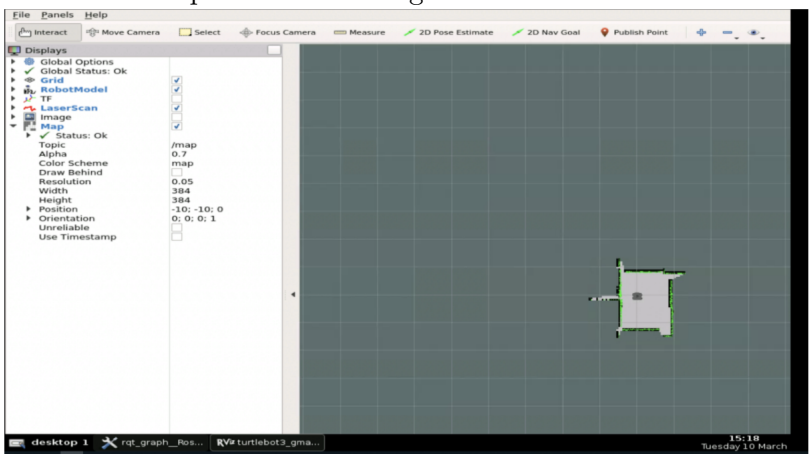
What tool can we use to help us build our own packages in ROS, and where do we need to locate custom packages within the Linux file system of our robot or remote machine?

To create our own packages in ROS, we need to use `catkin_create_pkg`, and we need to be in the `catkin_ws/src` folder before using that command.

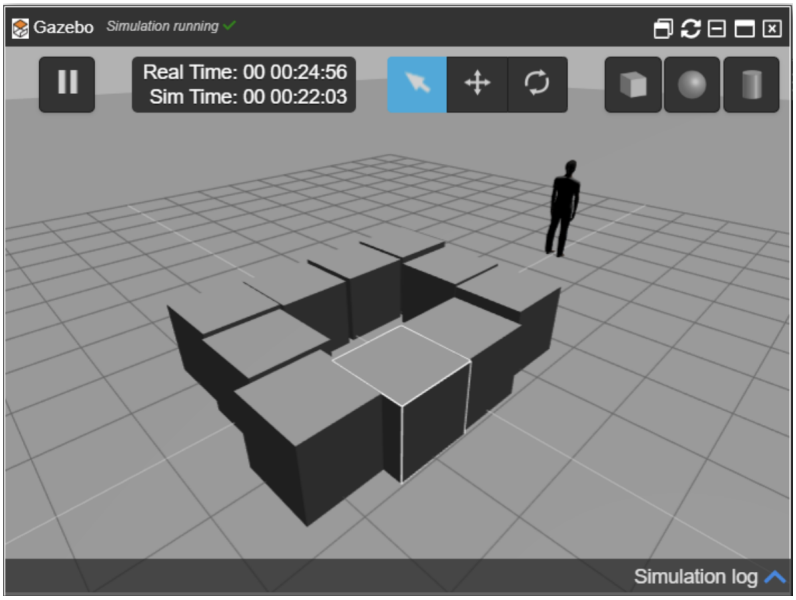
QUESTION 6

Provide two images in your report showing: 1. The map that your robot generated in Exercise 4 2. A photograph of the actual arena that your robot was operating in Compare the two images. How well did your robot map it's environment, were there any areas where the mapping process did not perform well, and why might this be the case?

Here is the map that our robot generated :



Here is the actual environment in which it operated :



Since these cubes were not neatly arranged, the robot had difficulty scanning every corner, therefore the mapping did not perform well for these parts.

QUESTION 7
What information does the SLAM algorithm need to build a map? Where on the TurtleBot3 Robot does this information come from, and which ROS topics could we listen to obtain this information?

- The SLAM algorithm needs the distance measurements and the pose which is the combination of the robot's position(x,y,z) and orientation (x,y,z,w) to build a map.

- The distance measurements come from the LiDAR sensor, and IMU sensor provides the pose of the robot. Then the information will be published to the /scan topic. Hence, we can listen to the /scan topic to obtain data.

QUESTION 8
This week you were provided with a move square template.py file to assist you with developing your move square node. Explain what the callback function is doing here, and explain how you might have accessed and used the data from this in the main loop() to achieve the desired outcome.

Within the callback function, we are able to obtain the data from the input `odom_data` input parameter. For example, `odom_data.pose.pose.position.y is assigned to variable pos_y`, therefore we can get the linear position of our robot in the y axis. In the main loop, when we set a value to linear.x and set angular.x to 0, the robot will go straight, and vice versa, the robot will turn 90 degrees. The robot will not stop until ctrl.c is pressed.

QUESTION 9
How do ROS Services differ from the standard topic-based messaging (publisher / subscriber) approach that you have learnt about in previous weeks? Provide examples of two robot applications where it might be useful to use ROS Services.

On a standard topic-based messaging approach, Nodes communicate via topics that are being published by any other node in the system. 2 Nodes never directly communicate between one another. However, with Services, 2 nodes communicate together directly, in a request response format with Messages. 2 robot applications where Services might be important are :

- When a specific action needs to take over another, like scan something before operating. The principle of subsumption.
- Turn on/off specific bits of the robot to only care about specific situations to not have a data overload or a battery power being consumed for nothing.

QUESTION 10
Which ROS command would you use to obtain information about a service that is currently active on a ROS robot? Name three important pieces of information that this command would provide us with, and explain the meaning of each of these.

We should use rosservice info, it allows us to find out about a specific service launched by a node such as (the node name, the Type of message the service uses and the Arguments to state the needed input arguments to supply to the service)

QUESTION 11
Which ROS command would you use to obtain information about a particular service message? Explain the format of a service message (there are 4 key points here). If you were building a service client, which part of the message would you be most interested in?

We should use rossrv show [Type of Message] to find out about a particular service message. The format of a service Message is as follows :

- Two parts are separated by "- - -", the part on the top is the Request format, the part at the bottom is the Response format
- When we need to CALL a service, we need to provide the data specified in the Request section.
- The SERVER will send back data in the Response section format. In these 2 formats, variables are stated as Input and Output.

If we were building a service client, we would be interested in the upper part which is the Request section

QUESTION 12

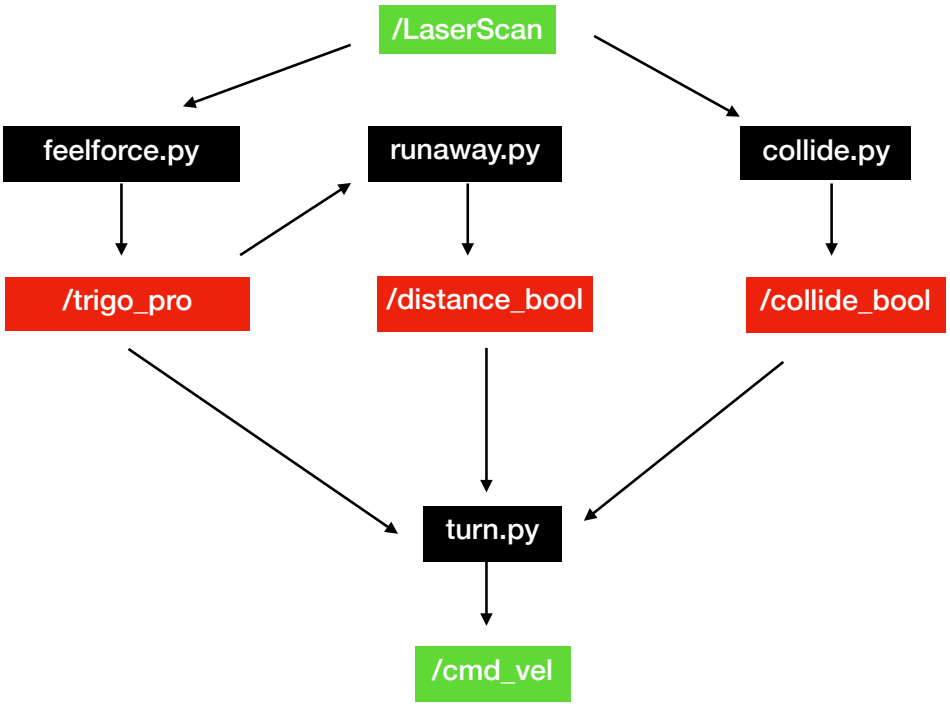
What does the data from the LiDAR sensor tell us about our robot? Which ROS topic is this data published to, and what is the message type? Why might this information be more valuable than odometry data when developing robot navigation algorithms?

The LiDAR sensor tells us about the position of our robot in relation with the surrounding objects, it tells us how far our robot is from our object. It assigns each range with angular values that help the robot build a strong map of its environment. This data is published to the topic /scan and the message type is `sensor_msgs/LaserScan`. This information is more valuable than odometry data because the odometry data only tells us about the position of our robot and also its orientation but not about its environment, however, the LidarSensor gives us a clear picture of the surroundings of our robot which is a lot more valuable.

QUESTION 13

Explain what you would have done to fully implement the Avoid subsumption layer (Layer-0) in terms of the development of your individual ROS nodes and the interconnections between them.

For this assignment, we opted out for a topic-based solution. The graph below describes the interconnections between our ROS nodes, the black boxes are our actual nodes and the red ones are the topics that link them together.



To describe briefly what the nodes are doing, I will provide screenshots of the most important functions we came up with.

- Feelforce.py

This file is the one that is going to process all the trigonometry data we gonna need throughout the entire system. First, feelforce.py is gonna grab data from the Laser-Scan topic. First things first in order to use it, it is going to adapt cos and sin functions (to rotate it in a more conventional way in relation with the Laserscan data).

```
18      def cosinus(self, data):
19          return np.cos(np.radians(data)+np.pi/2)
20
21      def sinuse(self, data):
22          return np.sin(np.radians(data)+np.pi/2)
```

Once provided with this data, we need to find a way to calculate the "path of least resistance", this is done by calculating the mean from the vectors we get with to these 2 functions. Afterwards, we will need to simply send this information as a vector to the topic we created for this specific purpose. This operation is carried out in the callback function :

```
36      def callback(self, data):
37          #rospy.sleep(1)
38          vec_array_x = []
39          vec_array_y = []
40          for index in range(0,360,1):
41              if data.ranges[index] != float("inf"):
42                  vec_array_x.append(self.cosinus(index))
43                  vec_array_y.append(self.sinuse(index))
44          print 'cos :', -np.mean(vec_array_x), ' sin :', -np.mean(vec_array_y)
45          ans.x = -np.mean(vec_array_x)
46          ans.y = -np.mean(vec_array_y)
47          pub.publish(ans)
```

We managed to test this function on simulation and we will show some screenshots of some situations the robot has been put into.

- Runaway.py

This file is meant to set a threshold for which it tells whether the robot should consider moving or not based on the distance to that object. Therefore, this file was meant to use the data we set up previously in feelforce.py. By subscribing to the trigo topic : it is calculating the distance based on a really simple Pythagorean theorem. Based on that threshold that should have been adjusted in real-life situation, it sends a boolean value to the distance bool topic for later use.

```
def calc_range(self,data):
    return (data.x * data.x) + (data.y * data.y)

def callback(self, data):
    if self.calc_range(data) <= 0.9:
        print 'U can go! Too close'
        pub.publish(True)
    else:
        print 'Dont move, The object is quite far from you'
        pub.publish(False)
```

We also managed to test this file on simulation and we will show some screenshots of

some situations the robot has been put into.

- Collide.py

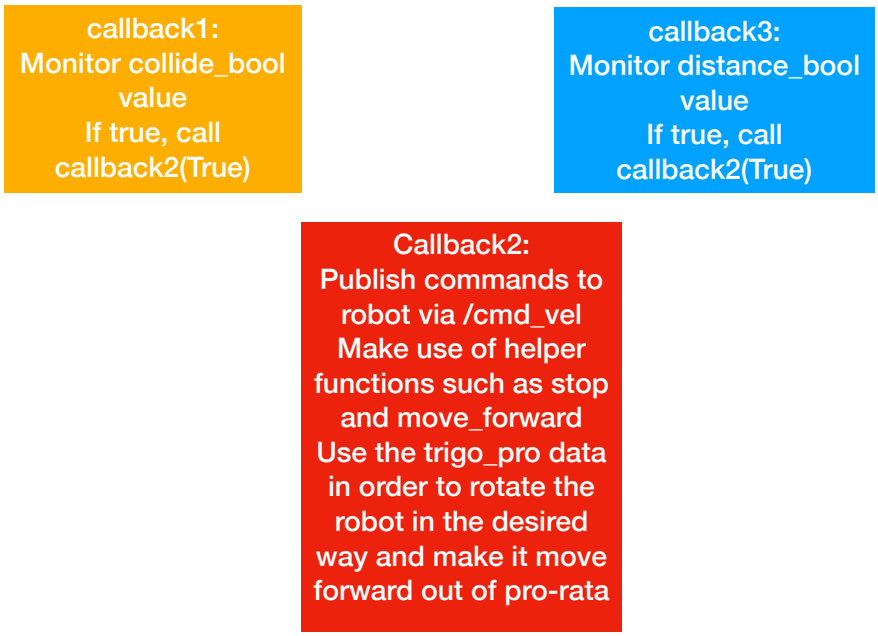
This file is meant to report object detected dead ahead. This is like an additional feature of runaway, but this time, the feature is also to detect where the object is. Therefore, it subscribes to the LaserScan, scan 65 degrees on left ahead, then scan 65 degrees on right ahead. And report to the collide boolean topic :

```
def callback(self, data):
    ans = Bool()
    i = 0
    pube.publish(ans.data)
    ans.data = False
    for index in range(0,65,1):
        if data.ranges[index] != float("inf") and data.ranges[index] < 1 :
            print 'stop, object detected dead ahead left'
            ans.data = True
            pube.publish(ans.data)
            break
        else:
            for indexe in range(294,359,1):
                if data.ranges[indexe] != float("inf") and data.ranges[indexe] < 1:
                    print 'stop, object detected dead ahead right'
                    ans.data = True
                    pube.publish(ans.data)
                    break
                else:
                    continue
```

We managed to test this file on simulation and we will show some screenshots of some situations the robot has been put into.

- Turn.py

This file is handling the command values published to the robot. The system can described with this file below :



First, we will provide the 3 helper functions that are meant to stop the robot from all velocity, calculate the distance (used for the distance for which the robot will move), and the actual move forward function :

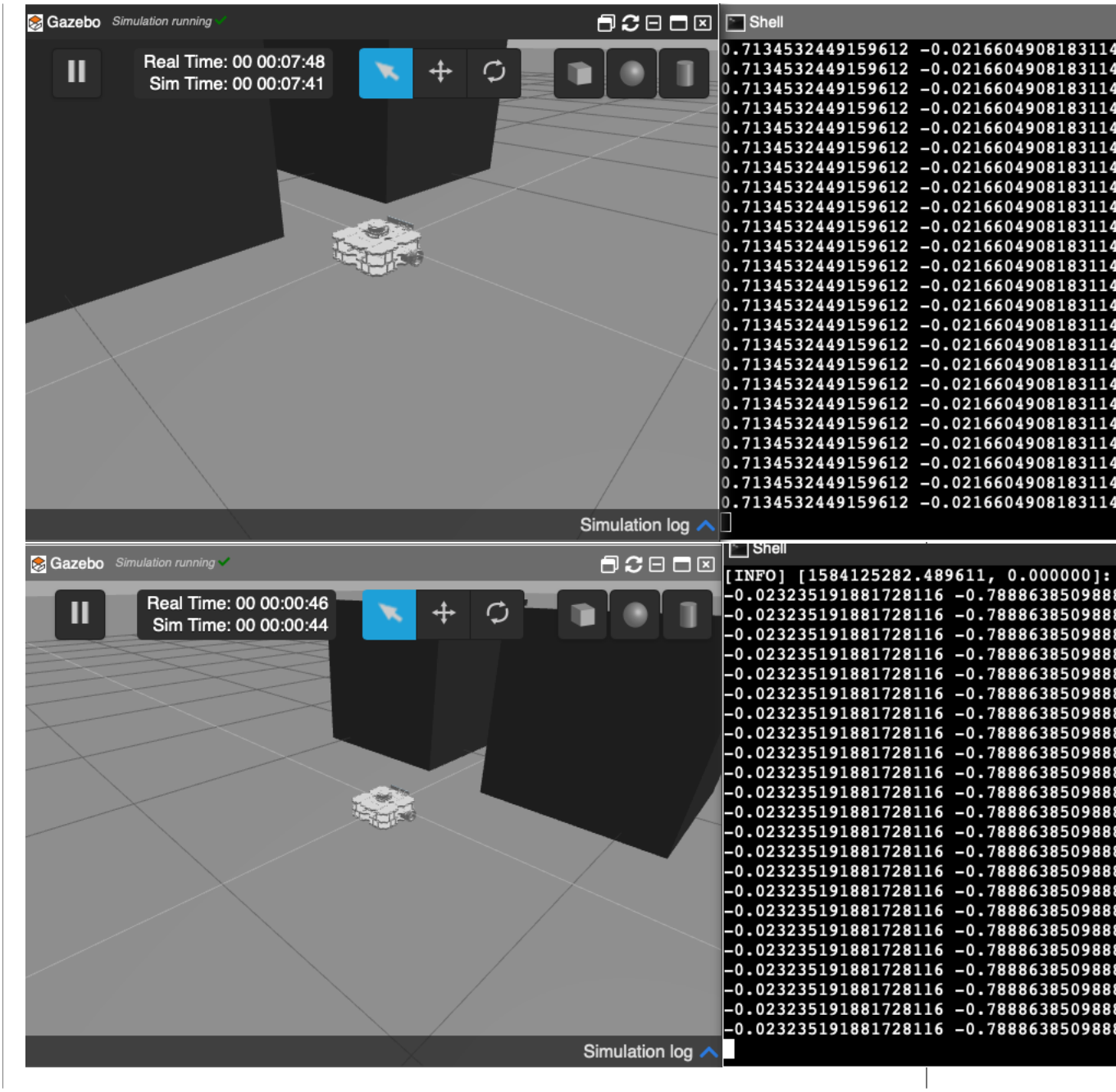
```
class Turn:
    def calc_range(self,x,y):
        return (x * x) + (y * y)

    def stop_motion(self):
        vel = Twist()
        vel.linear.x = 0.0
        vel.angular.y = 0.0
        pub.publish(vel)

    def move_forward(self,distance):
        vel = Twist()
        vel.linear.x = 1
        while not rospy.is_shutdown():
            t0 = rospy.Time.now().to_sec()
            current_distance = 0
            #Loop to move the turtle in an specified distance
            while(current_distance < distance):
                #Publish the velocity
                pub.publish(vel)
                #Takes actual time to velocity calculus
                t1=rospy.Time.now().to_sec()
                #Calculates distancePoseStamped
                current_distance= vel.linear.x*(t1-t0)
            #After the loop, stops the robot
            vel.linear.x = 0
            #Force the robot to stop
            pub.publish(vel)
```

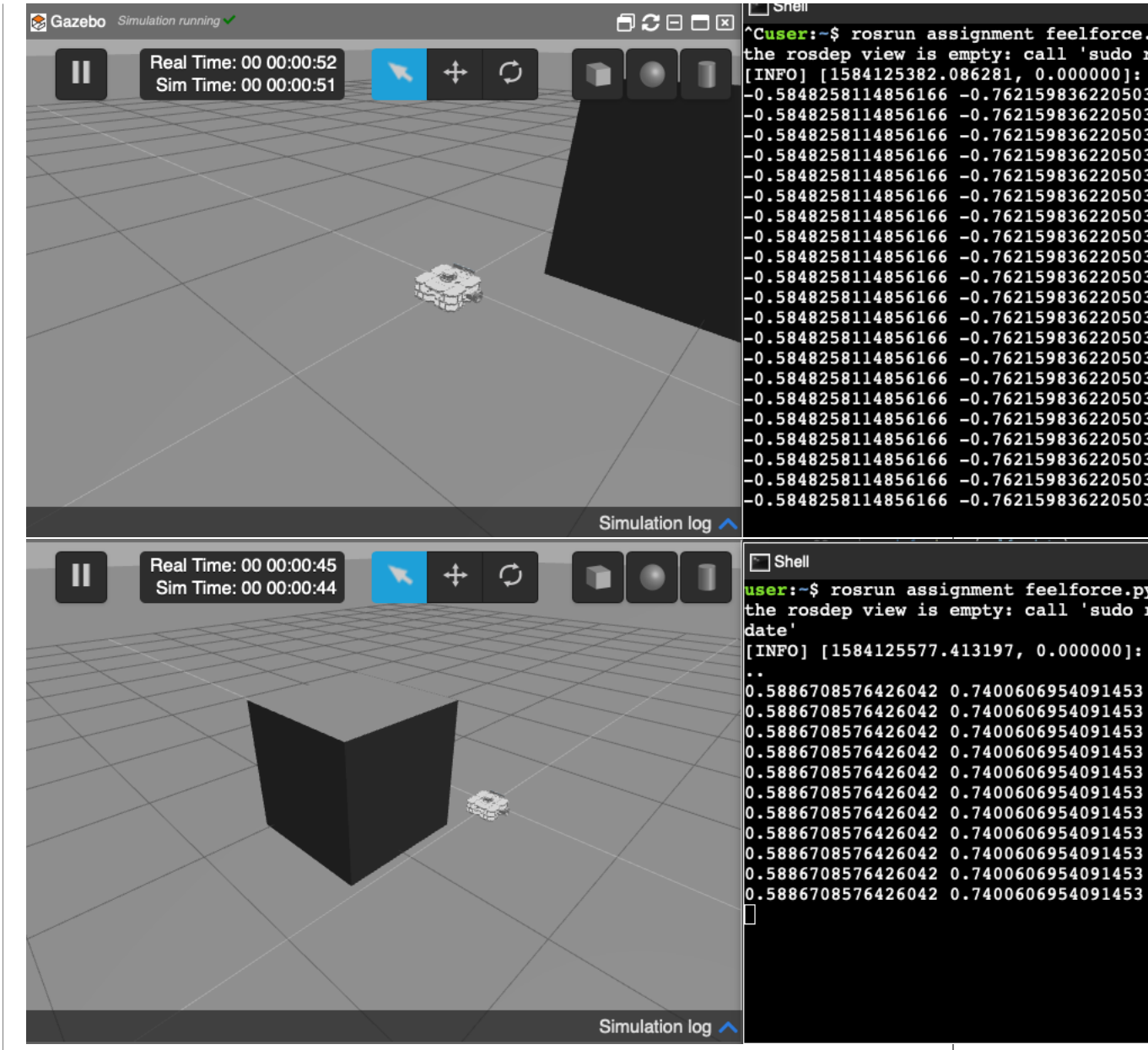
Then, we will provide the code for the 3 callback functions, 2 of them monitor different topics data and giving header to the central one so this one is not constantly running the core functionality :

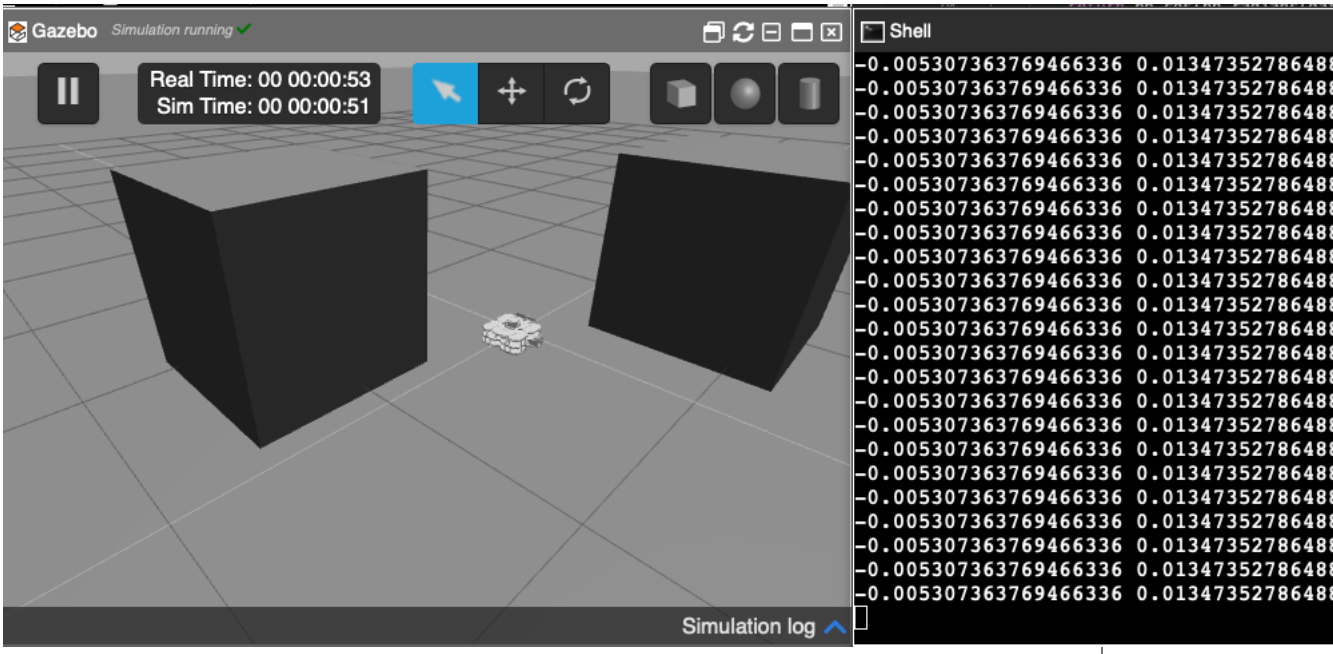
```
def callback1(self, data):
    if data.data == True:
        self.callback2(True)
    else:
        self.callback1()

def callback2(self,data, valid):
    if valid == True
        vel = Twist()
        t0 = rospy.Time.now().to_sec()
        current_angle = 0
        rad = abs(m.atan2(data.y, data.x))
        vel.angular.z = 1.0
        while not rospy.is_shutdown:
            while(current_angle < rad):
                pub.publish(vel)
                t1 = rospy.Time.now().to_sec()
                current_angle = vel.angular.z*(t1-t0)
        vel.angular.z = 0
        pub.publish(vel)
        self.move_forward(self.calc_range(data.x,data.y))
    else
        self.callback2(False)

def callback3(self, data):
    if data.data == True:
        self.stop_motion()
        self.callback2(True)
    else:
        self.callback3()
```

This was the design we came up with for the implementation of the Layer-0. Below, you will be able to find documentation of the testing in relation with the 3 scripts :

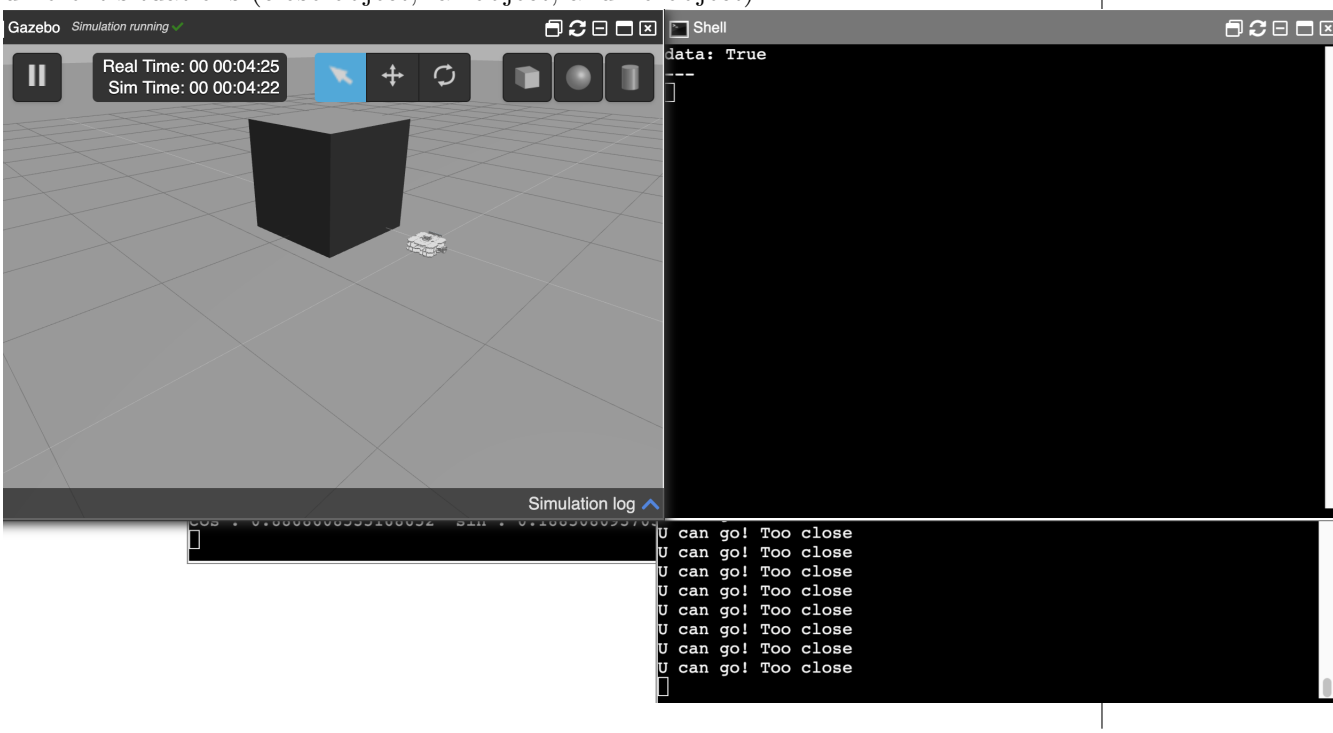
- Feelforce.py : The screenshots demonstrate the calculation of the path of least resistance in different situations

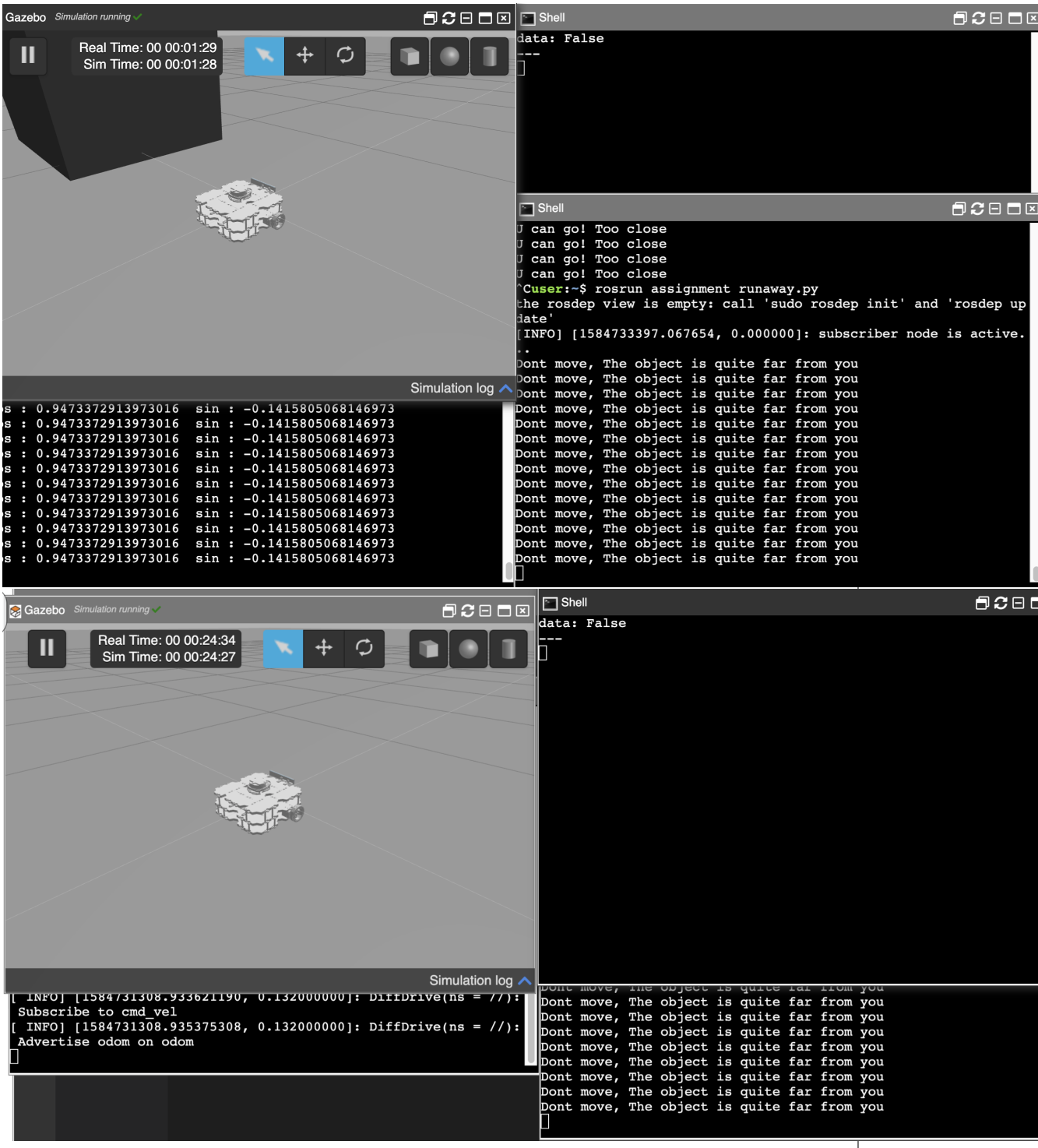

Gazebo
Simulation running
Real Time: 00 00:07:48
Sim Time: 00 00:07:41
Simulation log
Shell
0.7134532449159612 -0.021660490818311
Gazebo
Simulation running
Real Time: 00 00:00:46
Sim Time: 00 00:00:44
Simulation log
Shell
[INFO] [1584125282.489611, 0.000000]:
-0.023235191881728116 -0.788863850988


Gazebo
Simulation running
Real Time: 00 00:00:52
Sim Time: 00 00:00:51
Simulation log
Shell
^Cuser:~$ rosrun assignment feelforce.
the rosdep view is empty: call 'sudo
[INFO] [1584125382.086281, 0.000000]:
-0.5848258114856166 -0.762159836220503
Real Time: 00 00:00:45
Sim Time: 00 00:00:44
Simulation log
Shell
user:~$ rosrun assignment feelforce.py
the rosdep view is empty: call 'sudo
date'
[INFO] [1584125577.413197, 0.000000]:
..
0.5886708576426042 0.7400606954091453

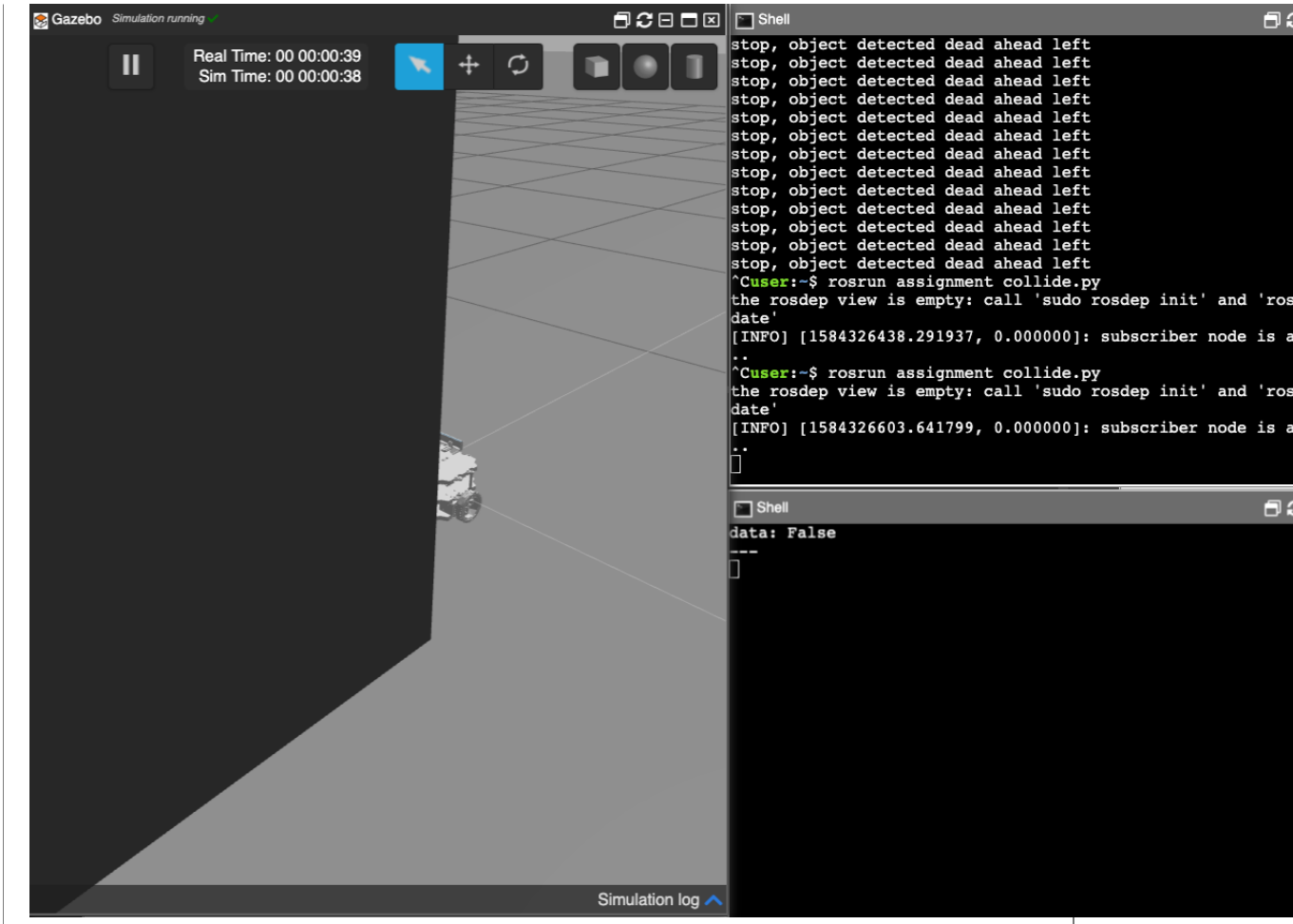


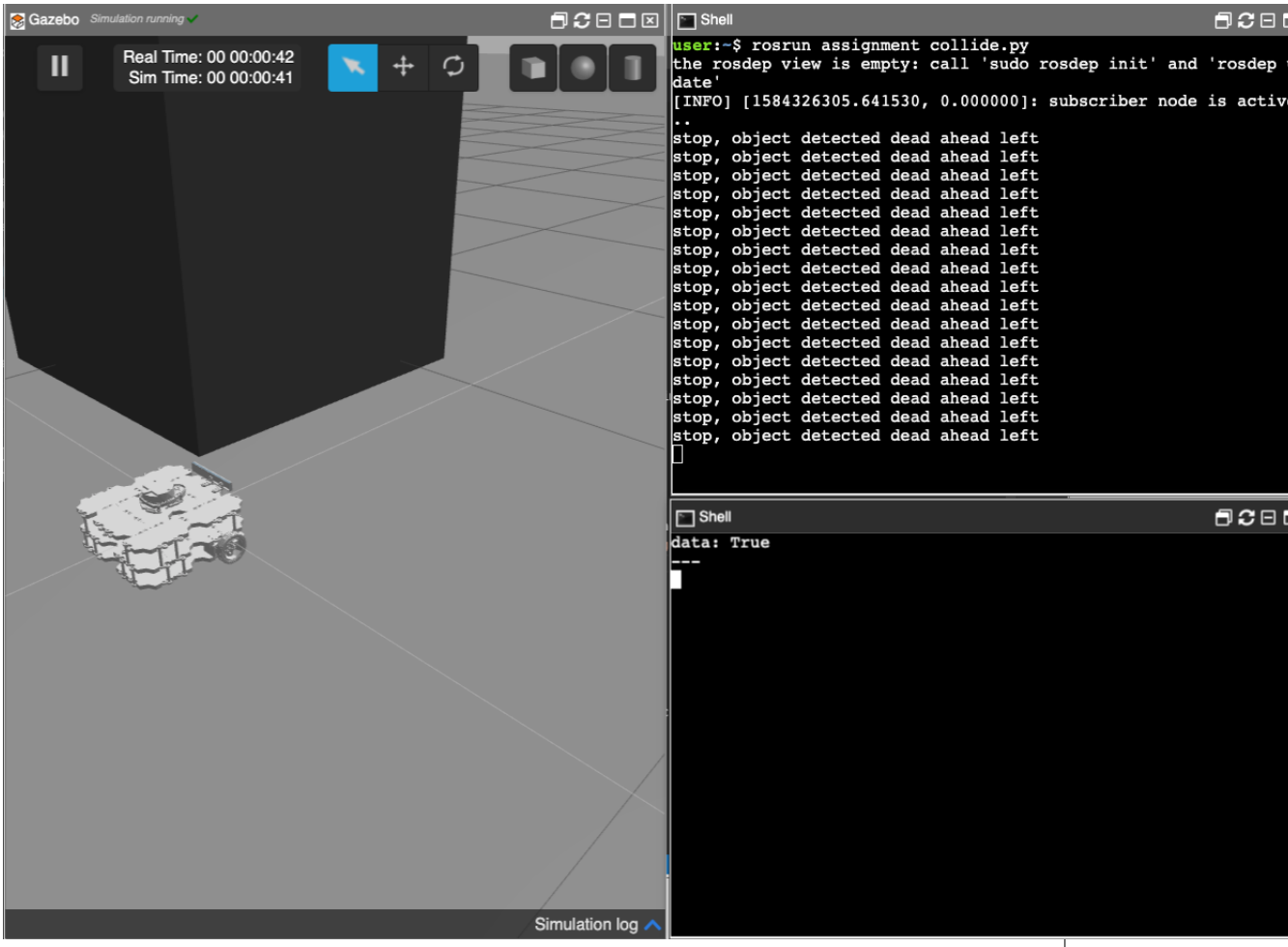
- Runaway.py : The screenshots demonstrate the runaway boolean value based on 3 different situations (close object, far object, and no object)

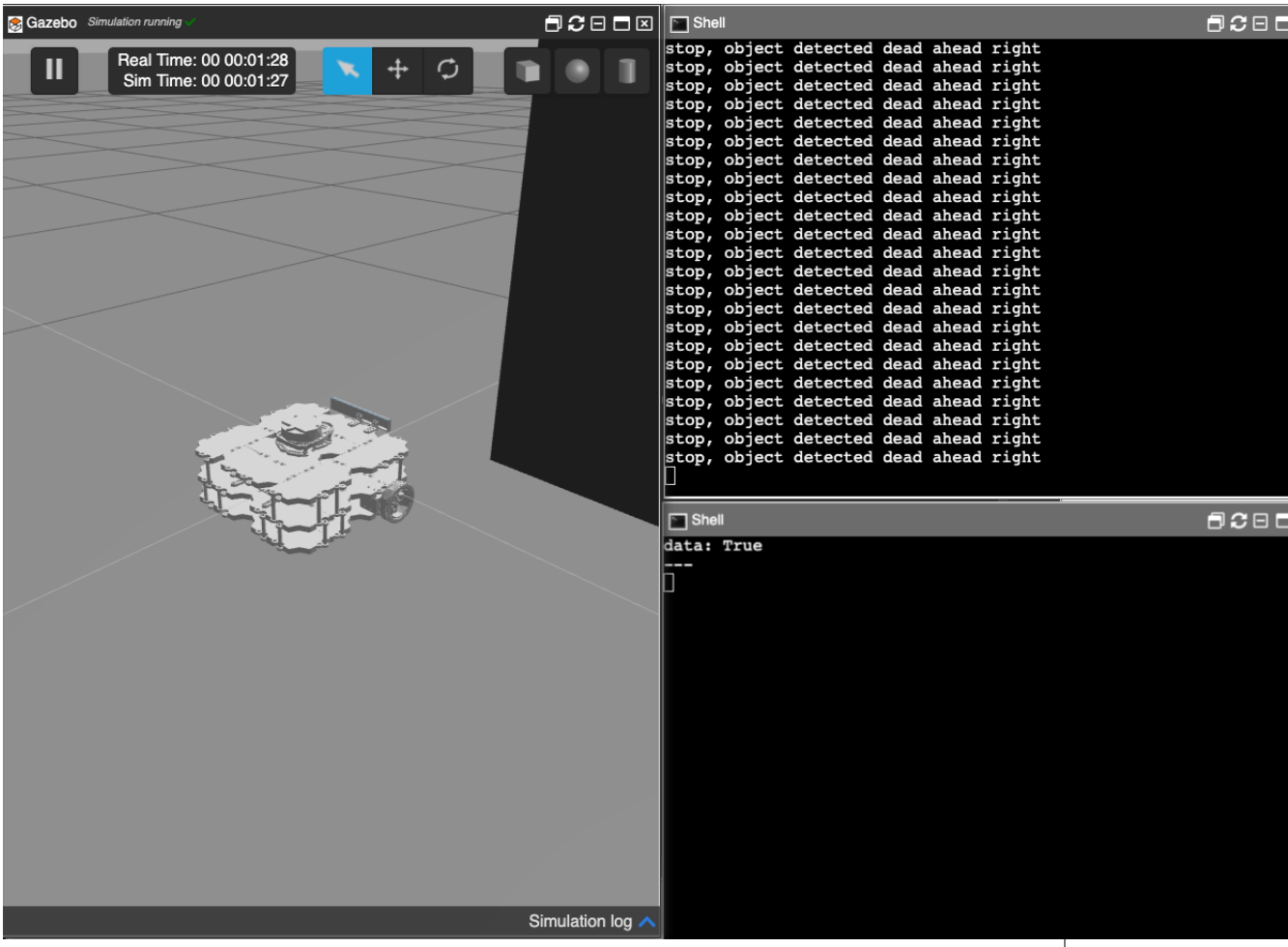


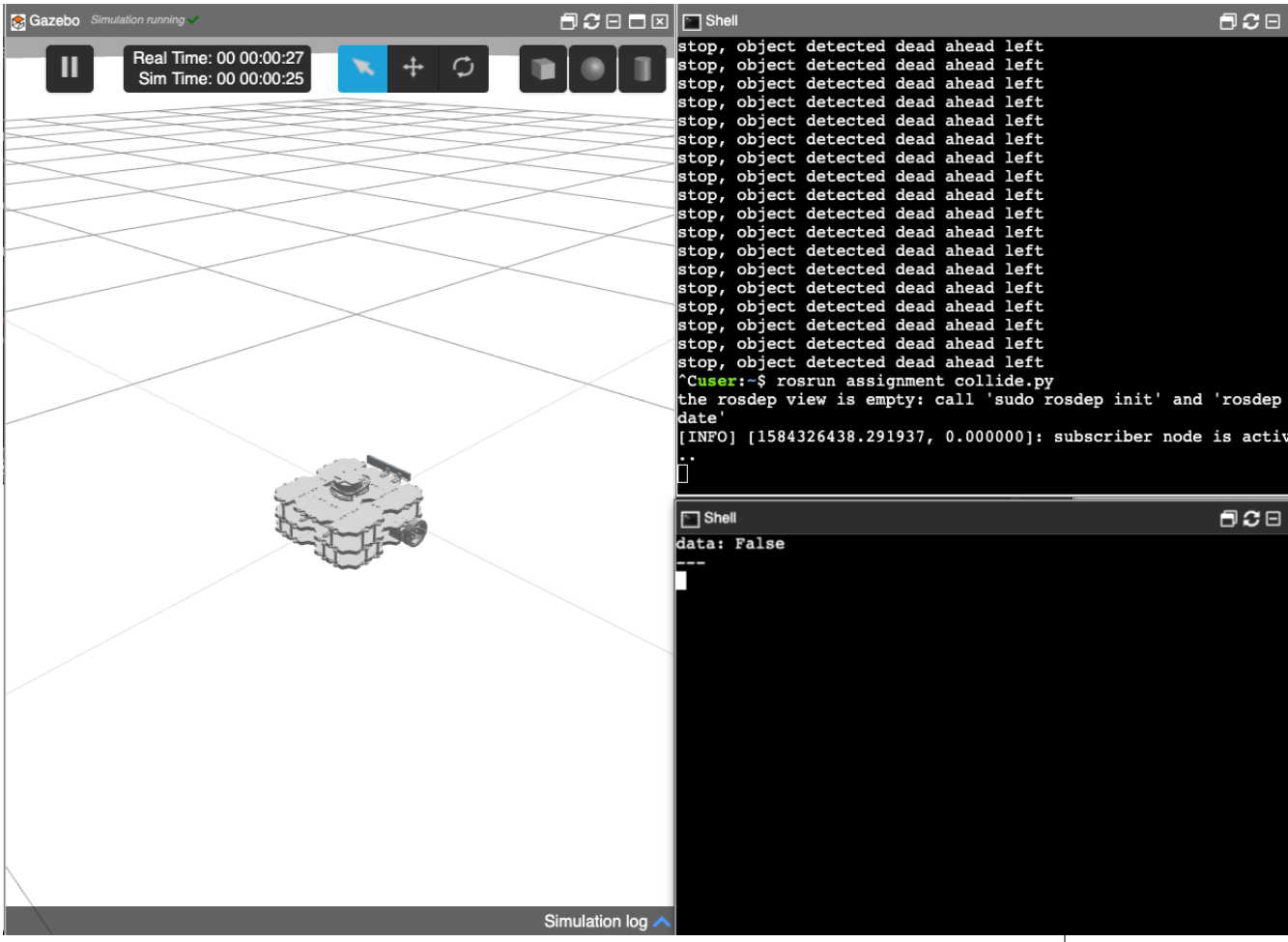


- Collide.py : The screenshots demonstrate the collide boolean value based on 3 different situations (Object behind, left ahead, right ahead, no object)


Gazebo
Simulation running
Real Time: 00 00:00:39
Sim Time: 00 00:00:38
Simulation log
Shell
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
^Cuser:~$ rosrun assignment collide.py
the rosdep view is empty: call 'sudo rosdep init' and 'ros
date'
[INFO] [1584326438.291937, 0.000000]: subscriber node is a
..
^Cuser:~$ rosrun assignment collide.py
the rosdep view is empty: call 'sudo rosdep init' and 'ros
date'
[INFO] [1584326603.641799, 0.000000]: subscriber node is a
..
Shell
data: False


Gazebo
Simulation running
Real Time: 00 00:00:42
Sim Time: 00 00:00:41
Simulation log
Shell
user:~$ rosrun assignment collide.py
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep
date'
[INFO] [1584326305.641530, 0.000000]: subscriber node is activ
..
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
stop, object detected dead ahead left
Shell
data: True


Gazebo
Simulation running
Real Time: 00 00:01:28
Sim Time: 00 00:01:27
Simulation log
Shell
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
stop, object detected dead ahead right
Shell
data: True



QUESTION 14

Explain what you would have done to implement the Wander subsumption layer (Layer- 1) in terms of the development of ROS nodes, the interconnections between them and the interactions with Layer-0.

Due to a technical issue that occurred with the ros website we were using, showing no content (IDE/Notebook) at a the stage of this question I will try to describe the system the best I can.

The aim of this layer is to wander, therefore with the implementation we had, it was mainly about tracking for inactivity in terms of movement of the robot and to simply make it move in any direction considering that we already had the implementation of the Avoid Layer (rotation and stop).
One way to do this is to use Odometry data to get coordinates of the robot, when subscribing to that topic, monitor the x and y coordinates, and using rospy.time API, it is possible to see if the robot has not moved for a given amount of time. Once such threshold reached, simply make call a pre-defined function that would make the robot move to a specific distance not too large so the callback functions cannot take over the situation, and not to small to have our robot moving just a bit between time lapses.

The other way would be to use this Odometry data, but instead of making the robot move based on no input whatsoever. It would be given an input from an input that would work the same way as the Runaway.py file works, same goes for the collide bool topic, it would calculate the ranges around based on on an incremented threshold or just bigger and hopefully return a True value for something around, saying that the force applied is sufficient to the situation.

QUESTION 15
Please provide a table within your report like the one below, to indicate each group member's contribution to this assignment as a percentage (adding up to 100). Include as many rows as appropriate for the size of your group.

Group Member	%
Maxime Fontana	75
Tianyi Zhang	20
Sam Lowth	5