# 

**Robotic Project**

# Team BCE

# The Turtle Explorer

# Technical Report

Caspani Bastien

Charalampaki Eirini

Katrini Chrysanthi

# Abstract

The Turtlebot, name Tbot arrive on a new planet. Tbot navigates itself and recognizes objects in the new environment. It will be able to recognize 3D objects, like a ball, a cube, a pyramid, etc. The Turtlebot will behave depending on the object in front of it. This behavior can be considering as maintenance task on the planet, but here we will name it the Tbot dance. Then, Tbot needs to find his spaceship and go inside to get back to Le Creusot.

# 1st step

# Creating a Map (1)

Firstly, even if it is not necessary the mapping part, we created a 2D map of our area to give our TurtleBot completely autonomous navigation. On TurtleBot (physical controlling or remote controlling with the command ssh –X [turtlebot@192.168.0.100](mailto:turtlebot@192.168.0.100)), to bring up our robot, we type in a terminal:

roslaunch turtlebot\_bringup minimal.launch

To start making the map, we need:

roslaunch turtlebot\_navigation gmapping\_demo.launch

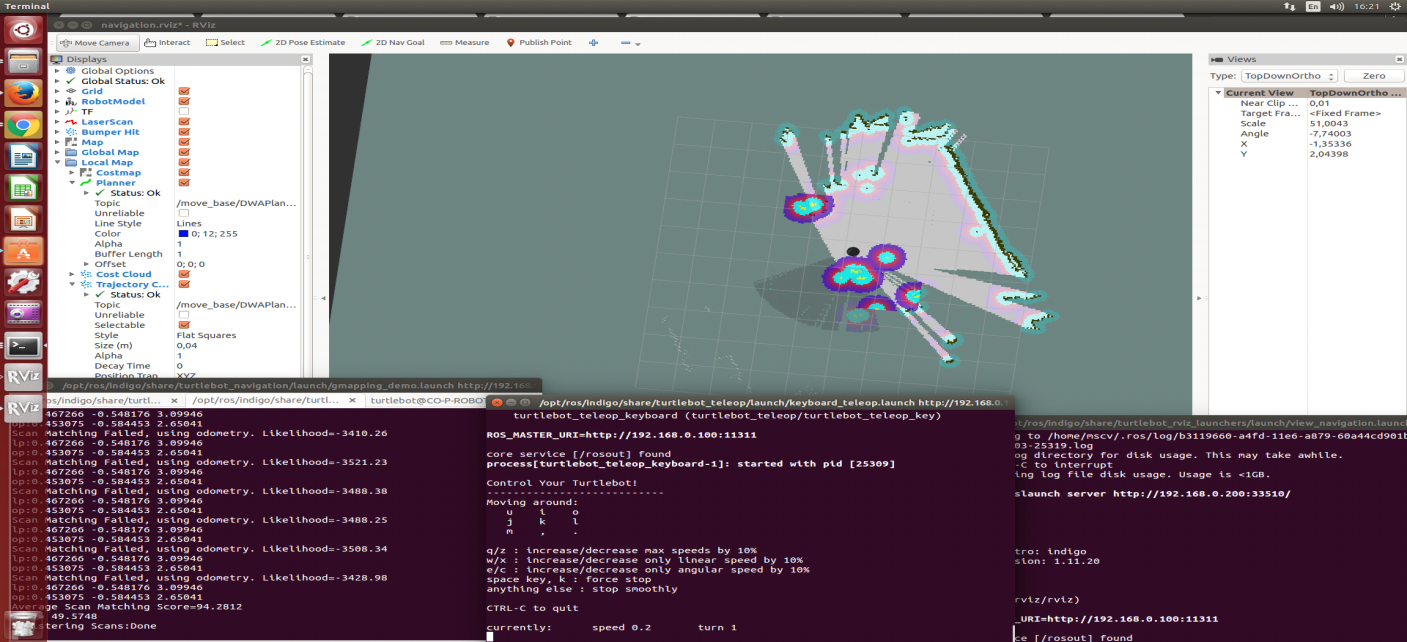
We need to open a monitor, so we chose Rviz just to see what is the robot's vision area.

roslaunch turtlebot\_rviz\_launchers view\_navigation.launch

Also, at this point of our project we prefer to control the robot via keyboard so we have to launch it.

roslaunch turtlebot\_teleop keyboard\_teleop.launch

We navigate our robot around the area so Tbot can take enough pictures and creating the required map.



At the moment we covered the whole area and now we can save the created map typing:

rosrun map\_server map\_saver -f /tmp/my\_map\_BCE

it is creating 2 files for the maps the my\_map\_BCE.yaml configuration file and the map my\_map\_BCE.pgm

# 2nd Step

# Autonomous Driving (1)

In the previous step, we used keyboard to control and navigate the robot, but our main purpose is the autonomous driving.

To implement this we had to stop every process on the robot and run:

roslaunch turtlebot\_bringup minimal.launch

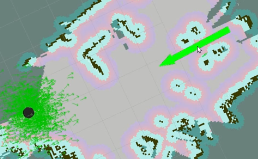
Then, we want to import our map to the system:

roslaunch turtlebot\_navigation amcl\_demo.launch map\_file:=/tmp/my\_map.yaml

If we want to see our map using Rviz monitor, we should type:

roslaunch turtlebot\_rviz\_launchers view\_navigation.launch –screen

In this point, if we click on the TurtleBot in the map, a green arrow will appear that can show us the Tbot’s direction.



So, now the Tbot is going to autonomous driving in the specific point that we gave to it.

# 3rd Step

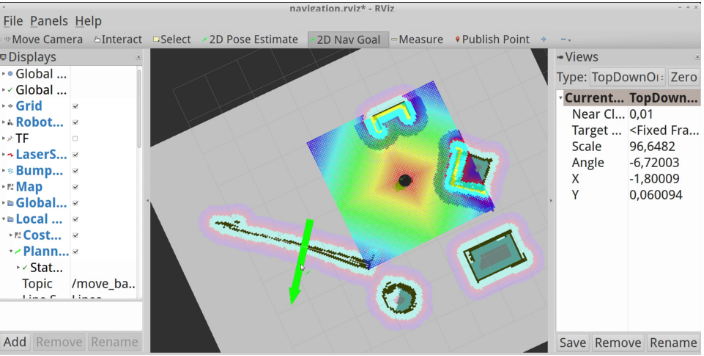
# Going Forward and Avoiding Obstacles Using Code (1)

As we manage to load our map, now we can control the robot and try to make it go forward and avoid obstacles.

To do that we launch Rviz, so we can see the robot and the environment:

roslaunch turtlebot\_rviz\_launches view\_navigation.launch

Then, we change the TurtleBot position by sending a navigation goal



We can also send the navigation goal using a python file and see the result in Rviz:

rosrun bce goforward\_avoid.py

# 4th Step

# Going to a Specific Location on Your Map Using Code (1)

In this point, we are trying to force the robot to go in specific location on our map using python code.

In a new terminal we type:

roslaunch turtlebot\_bringup minimal.launch

roslaunch turtlebot\_navigation amcl\_demo.launch map\_file:=/tmp/my\_mapBCE.yaml

roslaunch turtlebot\_rviz\_launchers view\_navigation.launch –screen

In Rviz, if we move the pointer we can see three different values, which represent the position on the map. We can choose the specific location and write down these three numbers. In a new terminal window, we type:

gedit ~/helloworld/go\_to\_specific\_point\_on\_map.py

In a specific part of the code we can easily change the position of the robot:

# Customize the following values so they are appropriate for your location

position = {'x': 1.22, 'y' : 2.56}

quaternion = {'r1' : 0.000, 'r2' : 0.000, 'r3' : 0.000, 'r4' : 1.000}

# 5th Step

# Object Recognition

In this part of the project we tried several techniques, the **find object 2D**, the **Object Recognition Kitchen (ORK)**, the **Tabletop Object recognition** and the **PCL**.

The ORK was actually working, so we had the server with database of object and clients for the detection. But this approach seems to be very specific and only for small object. And we would need to scan object and work from generated mesh. So, we did not continue in this way.

As suggested we focus on the use of PCL. (Point Cloud Library). The library is multiplatform and not specific for ros so we need to use the ROS integrated part.

First step was to install it and dependency, so we got 3 different packages: perception\_pcl , pcl , pcl\_ros.

The recognition will be done as following pipelining, this in order to be real time:

1. Get the image point clound from the Turtlebot Kinect ()
2. Input the raw point clound to a passthrough that will change the type in (Pointcloud2) in order to work with PCL and also reduce points cloud to restricted area.
3. Input this new point cloud (with remap) to get normal and transformed it into a voxel could.
4. This voxel could will be process with the RANSAC in order to make the segmentation, with SAC segmentation from the PCL.
5. There is SAC model for all object we wanted to detect.

# model\_type:

# 0: SACMODEL\_PLANE

# 1: SACMODEL\_LINE

# 2: SACMODEL\_CIRCLE2D

# 3: SACMODEL\_CIRCLE3D

# 4: SACMODEL\_SPHERE

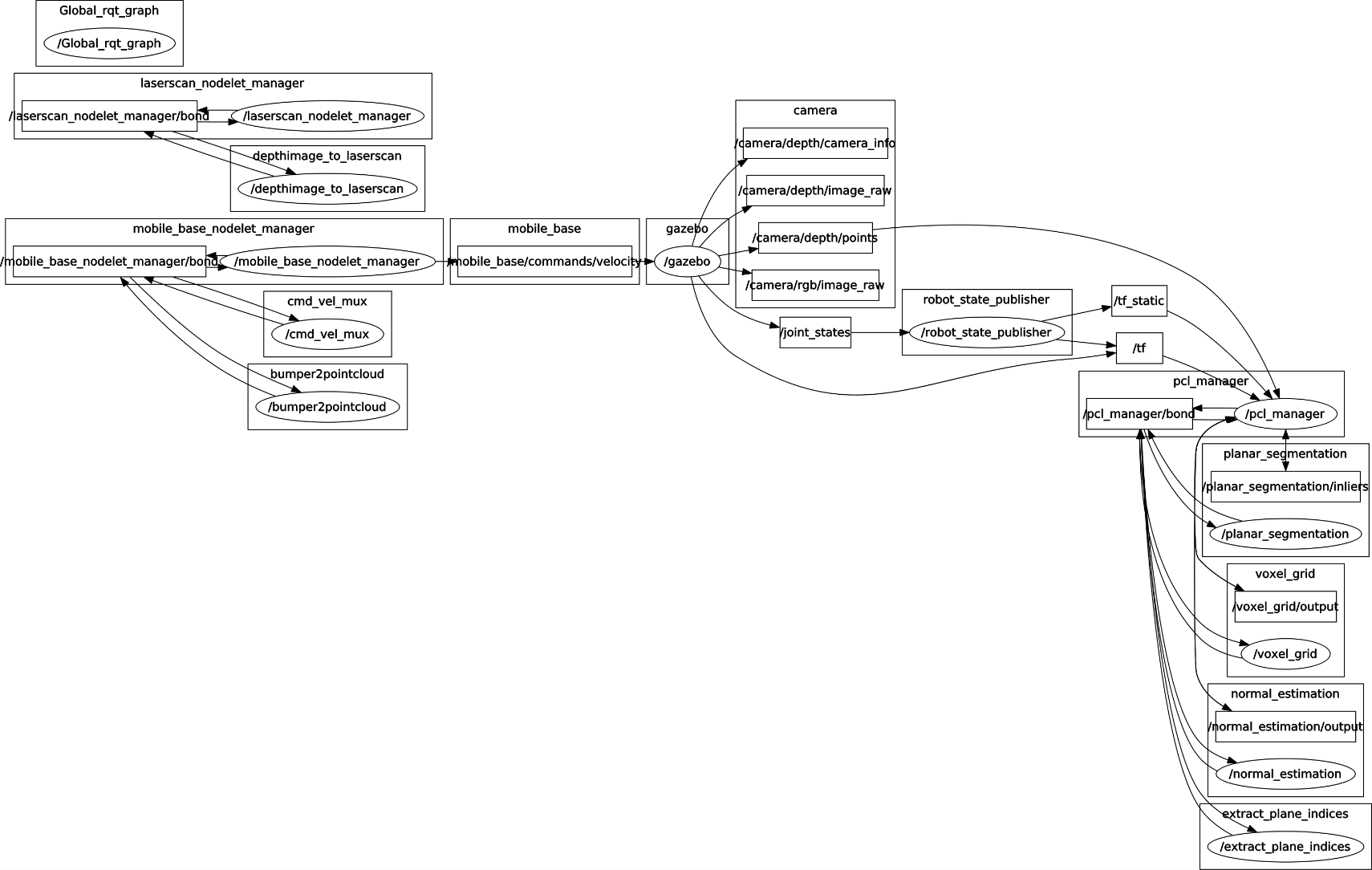
# 5: SACMODEL\_CYLINDER

# 6: SACMODEL\_CONE

# 7: SACMODEL\_TORUS

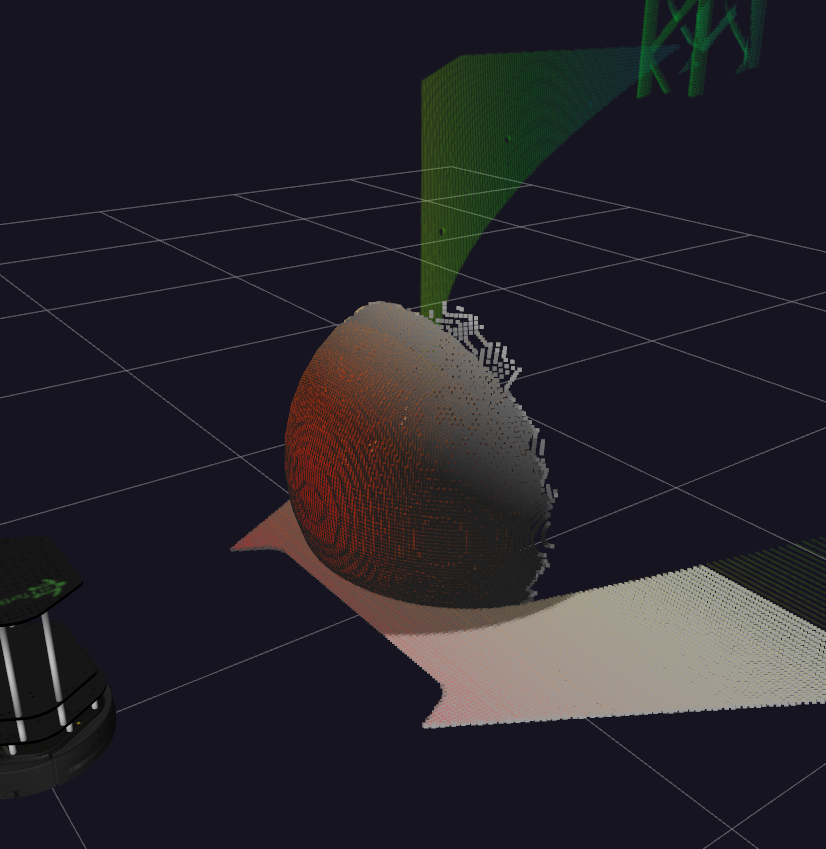
1. Then we will output the object type regarding the SACMODEL that match.

Here a picture of our RQT\_graph for the PCL, as we can see it require a PCL\_manager Nodelet and all other are also Nodelet.



The issue was at the end the Sac segmentation did not worked as it returns not enough inliers, we tried increasing Distance Threshold regarding the Kinect resolution and also other method without success. It was really close.

Here a screen shot from the Rviz with PCL nodelets results :



We can see in light gray the voxelgrid and in color the original point cloud from the Kinect.

# 6th Step

# Simulated Recognition

So as we wanted to show some results after so much work we create a new node that will simulate the detection. This one has been done in cpp the new package name is **sim\_detect.** On this node we created a new message objectdetect.msg that will give the object name and the distance to it. For this we use the message\_generation add the build and run depend on the package.xml and change the CMakelist to compile our code and the message. The point is as we know where are placed the object on the gazebo, then we use the robot position to make the detection from the robot.

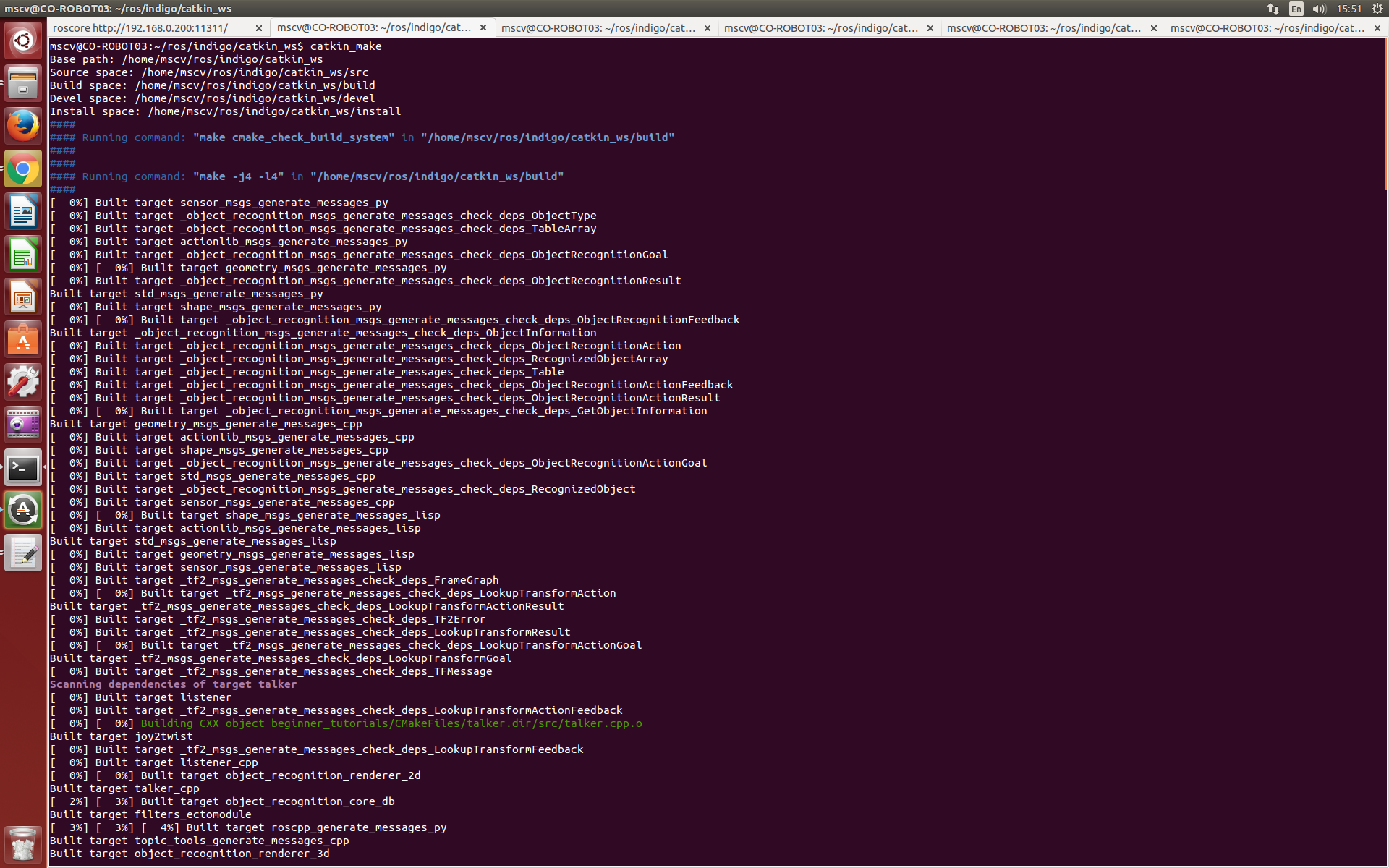
To use this node just build it using **catkin\_make** then **rosrun sim\_detect sim\_detect\_node**

The video of the result can be see here:

<https://www.youtube.com/watch?v=Wah2mYxG0Ig&t=267s>

After many unsuccessful tries, we decided to make a fake recognition by creating a publisher and a listener. So, when the robot detects e.g a cube, we have a publisher, that publish a message “This is a Cube”. We also created a listener, that as soon as it gets the published message, it will print another message eg. “I heard a Cube”, the desired result for our fake recognition.

We make the 2 C++ files (publisher and listener.cpp) executables, and after that we did a catkin make in the catkin\_ws. After we type rospack profile.



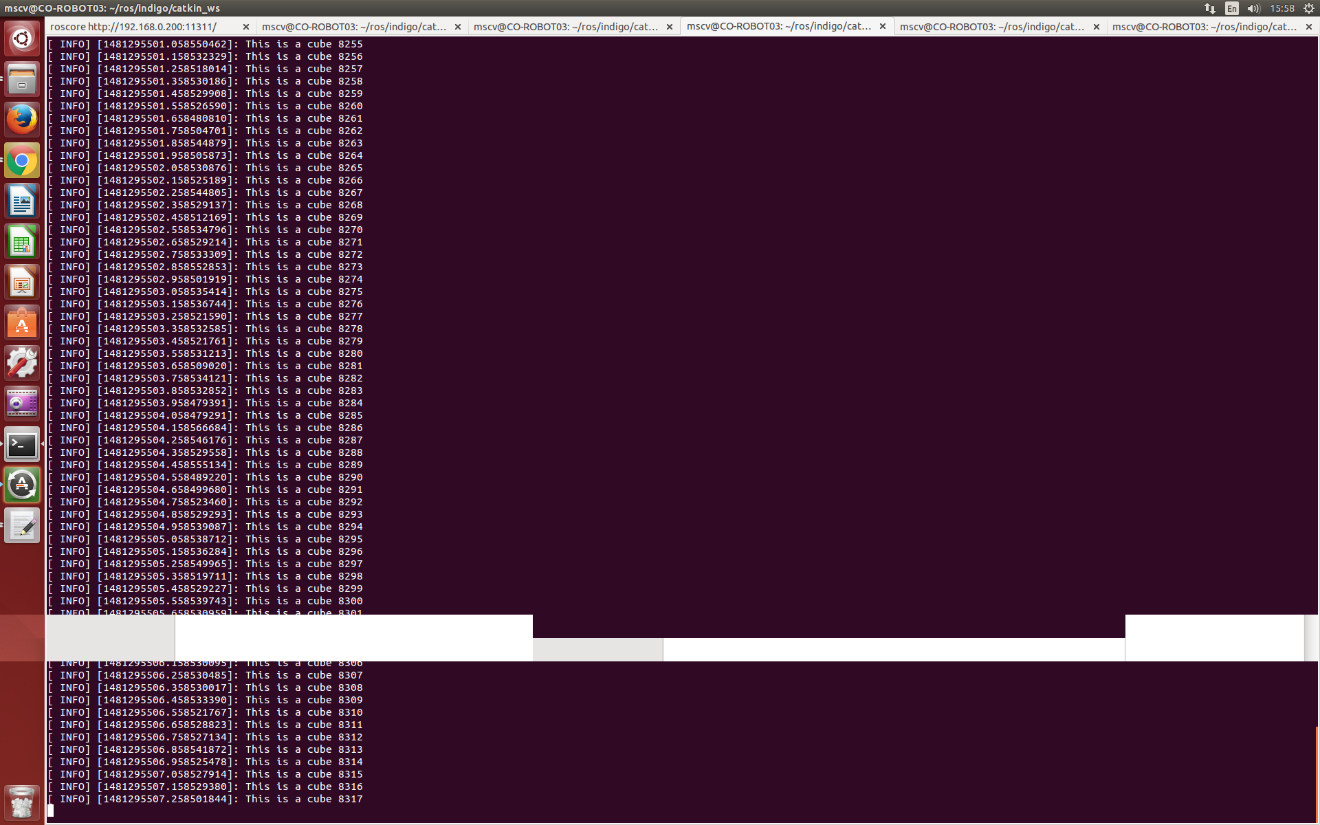
In a different terminal, we run roscore. We open a new terminal, and type in our catkin\_ws

source ./devel/setup.bash

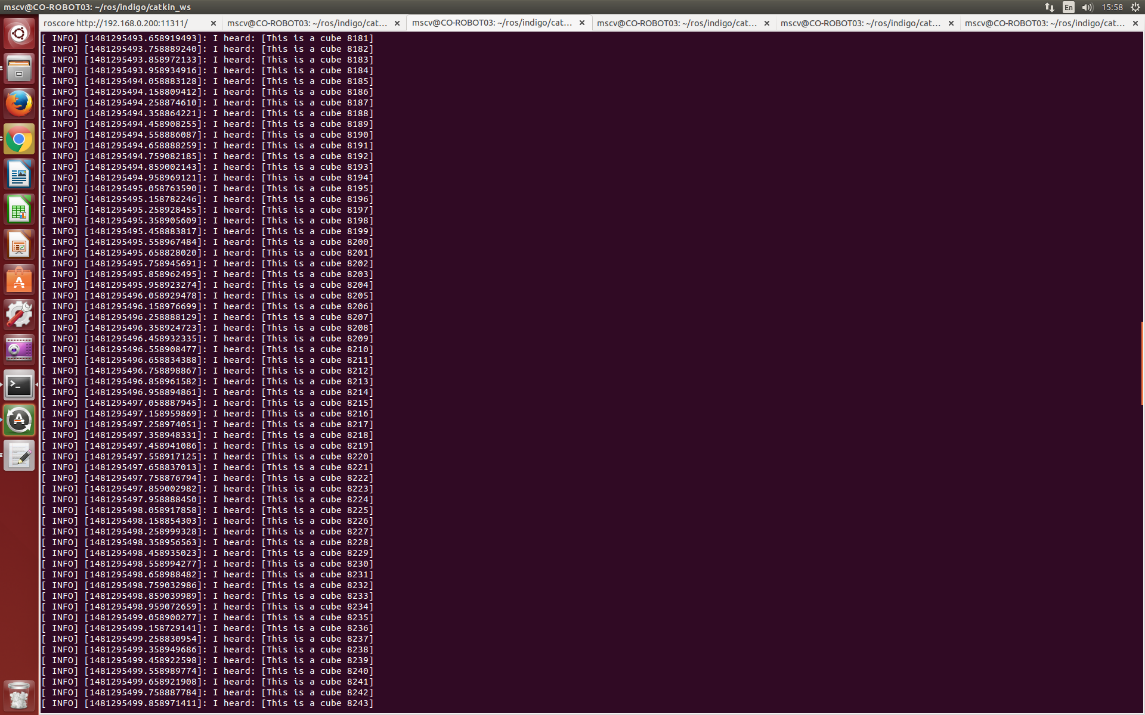
and then type rosrun beginner\_tutorials talker.

In a different terminal, type rosrun beginner\_tutorials listener. Below is the screenshot for the result.

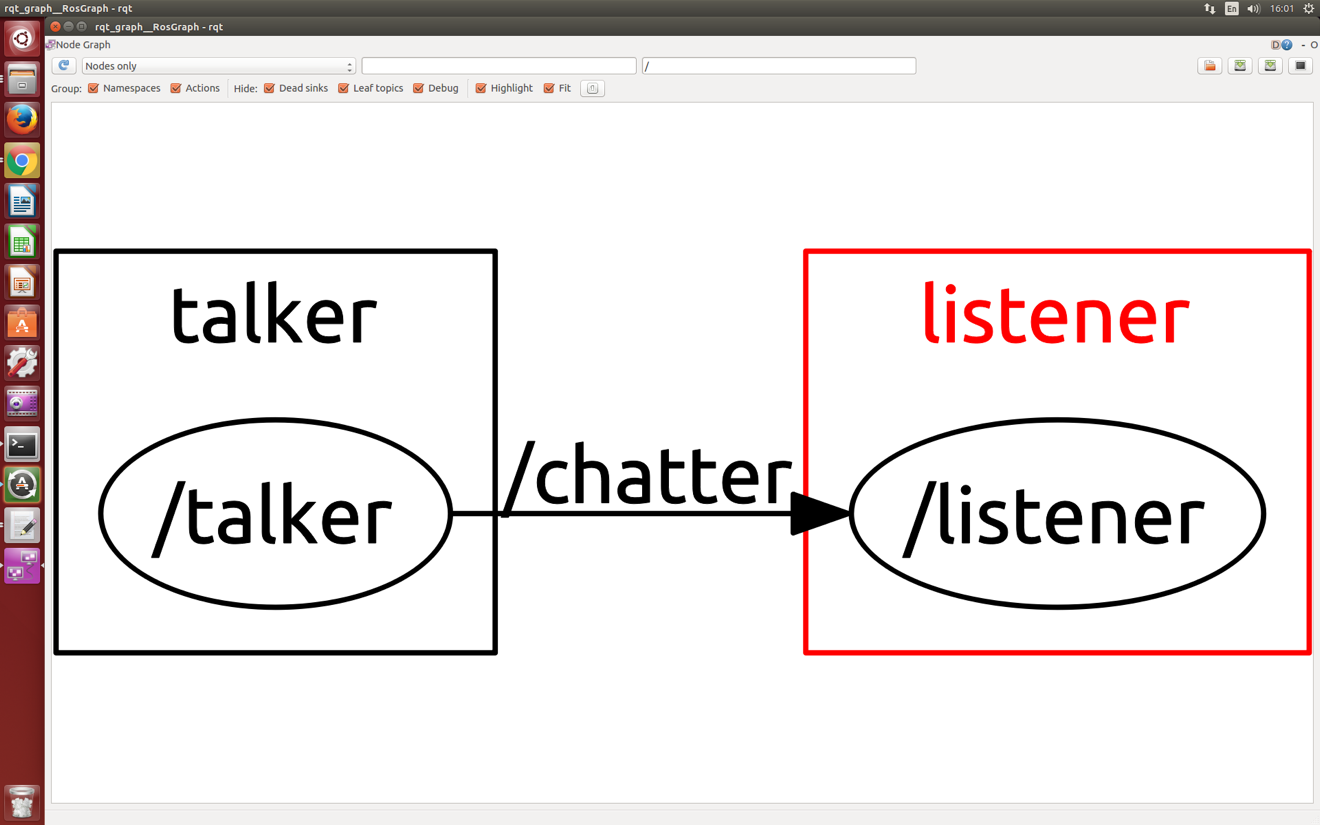
the publisher



the subscriber



Also, if we want to see the topics, type rqt\_graph.



If we want our Robot to speak, we type in a new terminal:

roslaunch sound\_play soundplay\_node.launch

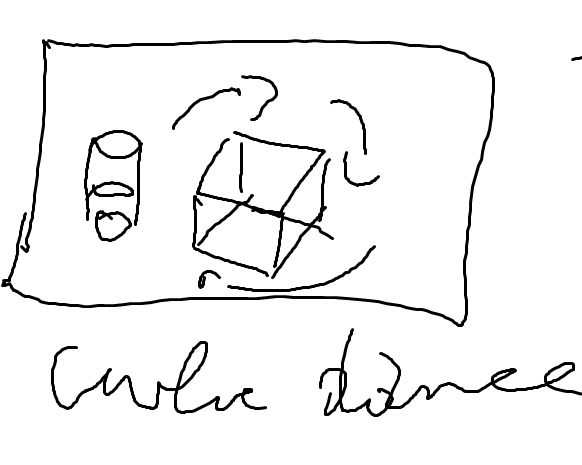
Then again in a new terminal, type:

rosrun sound\_play say.py "How do I sound now?"

(or the text of our choice, e.g Ohh eh cube)

# Movements

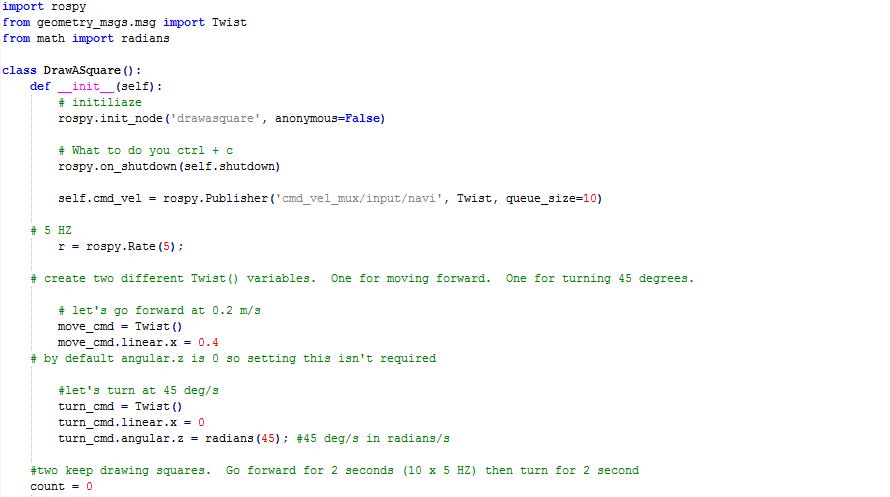
As we had detected and recognized the object, we want the robot to do specific movement based on what it has recognized. If the robot recognizes a cube we want it to do a square around it and if it recognizes a cone we want it to do a cycle around it.

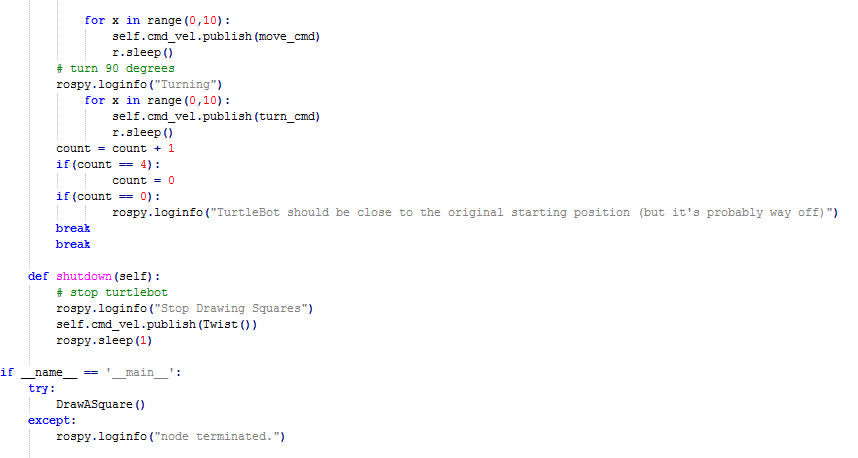


We wrote several python and cpp codes to achieve that, but most of them worked on gazebo but when we tried them to the Tbot they did not worked, and this is something that happened very often since every parameter change (environment, lights, etc).

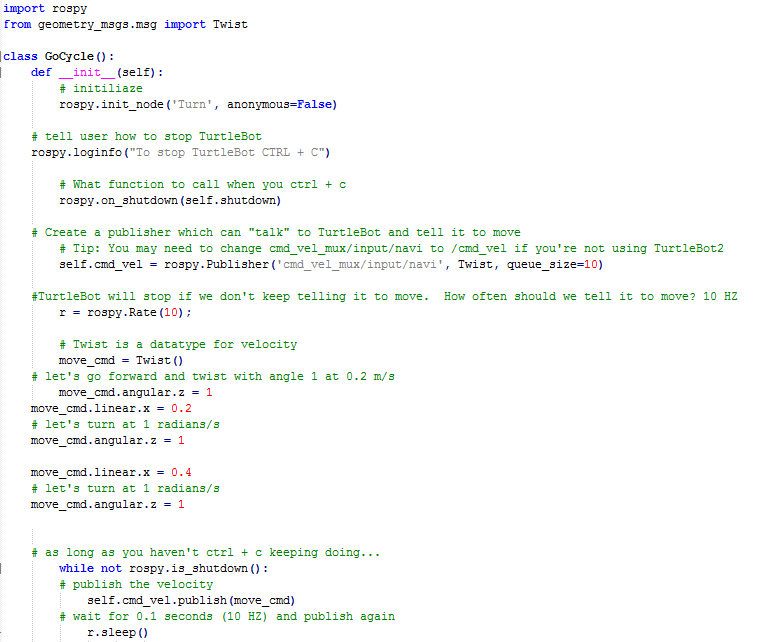
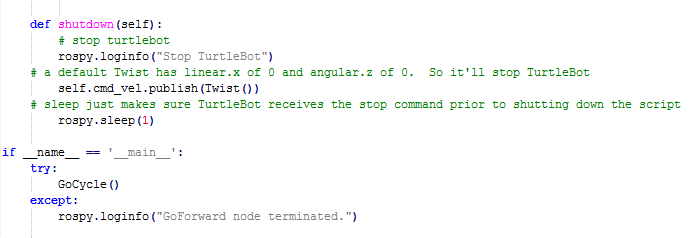
Finally, we use the codes that correspond better to our robot.

**draw\_square.py**

****

****

<https://www.youtube.com/watch?v=WO2-XAnb5Ks>

**gocycle.py  **

<https://www.youtube.com/watch?v=YnviLrt-0LM>

# Conclusion

Unfortunately the project did not end fully complete as expected. But most of the steps has been achieved. We have the SLAM working, so the robot can evaluate with the obstacle avoidance. We have also the PCL working to the limited range and nearly detected object in front of him.

We have the dance behavior and sound after the result of detection.

The point is that we did not succeed to put everything together but we would suggest using actionlib and smach.

# References

1. http://learn.turtlebot.com/. [Online] http://learn.turtlebot.com/.

2. http://wg-perception.github.io. [Online] http://wg-perception.github.io/tabletop/.

3. http://wiki.ros.org. [Online] http://wiki.ros.org/tabletop\_object\_detector.

4. http://wg-perception.github.io/object\_recognition\_core/index.html

5. <http://www.pointclouds.org/>

6.  [http://pcl.ros.org](http://pcl.ros.org/)

7. <https://github.com/ros-perception/perception_pcl>

8. http://wiki.ros.org/ROS/Tutorials