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), 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
```

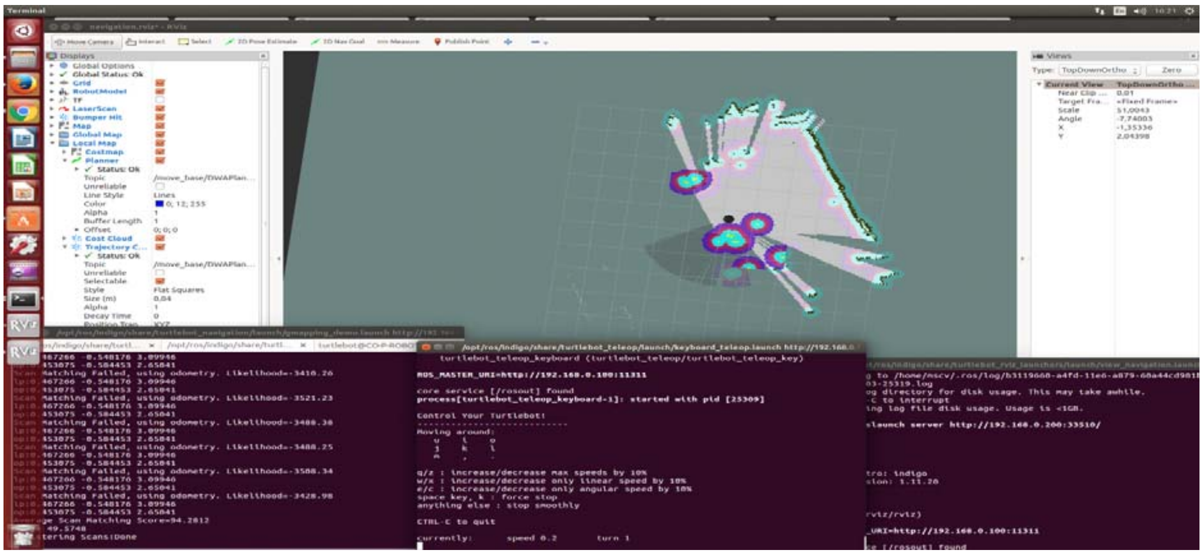
We need to open a monitor, so we chose Rviz just to see what is the robot's vision area.

```
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.



```
467266 -0.548176 3.09946
453075 -0.584453 2.65041
Matching Failed, using odometry. Likelihood=-3410.26
467266 -0.548176 3.09946
453075 -0.584453 2.65041
Matching Failed, using odometry. Likelihood=-3521.23
467266 -0.548176 3.09946
453075 -0.584453 2.65041
Matching Failed, using odometry. Likelihood=-3488.38
467266 -0.548176 3.09946
453075 -0.584453 2.65041
Matching Failed, using odometry. Likelihood=-3488.25
467266 -0.548176 3.09946
453075 -0.584453 2.65041
Matching Failed, using odometry. Likelihood=-3588.34
467266 -0.548176 3.09946
453075 -0.584453 2.65041
Matching Failed, using odometry. Likelihood=-3428.98
467266 -0.548176 3.09946
453075 -0.584453 2.65041
ge Scan Matching Score=94.2812
 49.5748
tering Scans:Done
```

```
/opt/ros/indigo/share/turtlebot_teleop/launch/keyboard_teleop.launch http://192.168.0
    turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)

ROS_MASTER_URI=http://192.168.0.100:11311

core service [/rosout] found
process[turtlebot_teleop_keyboard-1]: started with pid [25309]

Control Your Turtlebot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2       turn 1
```

```
g to /home/mscv/.ros/log/b3119668-a4fd-11e6-a879-68a44cd981
03-25319.log
og directory for disk usage. This may take awhile.
-C to interrupt
ing log file disk usage. Usage is <1GB.

slaunch server http://192.168.0.200:33510/


tro: indigo
sion: 1.11.20


rviz/rviz)

_URI=http://192.168.0.100:11311

ce [/rosout] found
```

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
```

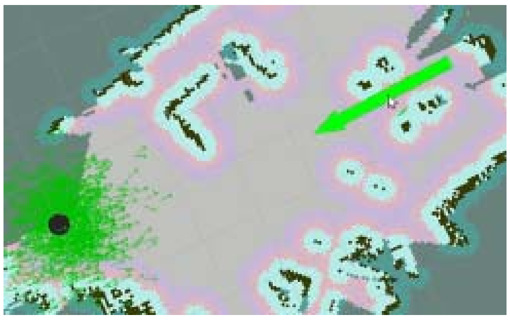
Then, we want to import our map to the system:

```
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

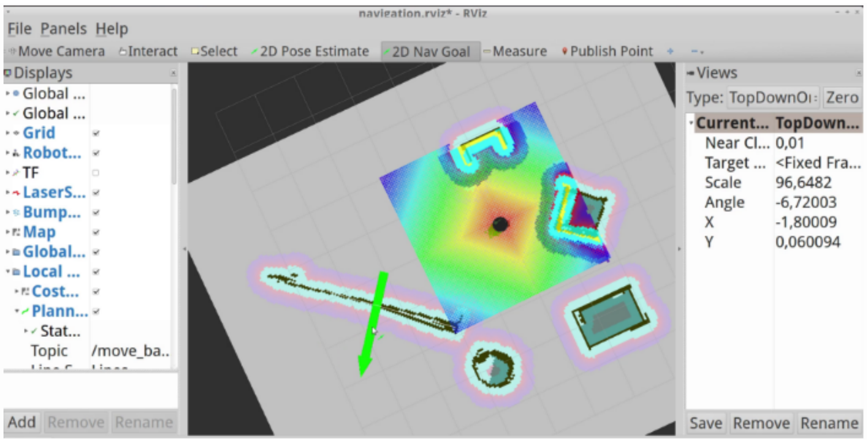
Going Forward and Avoiding Obstacles Using Code (1)

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



```
Topic       /move_ba..
Type: TopDownOı  Zero
Current...  TopDown...
Near Cl...  0,01
Target ...  <Fixed Fra...
Scale       96,6482
Angle       -6,72003
X           -1,80009
Y           0,060094
```

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:

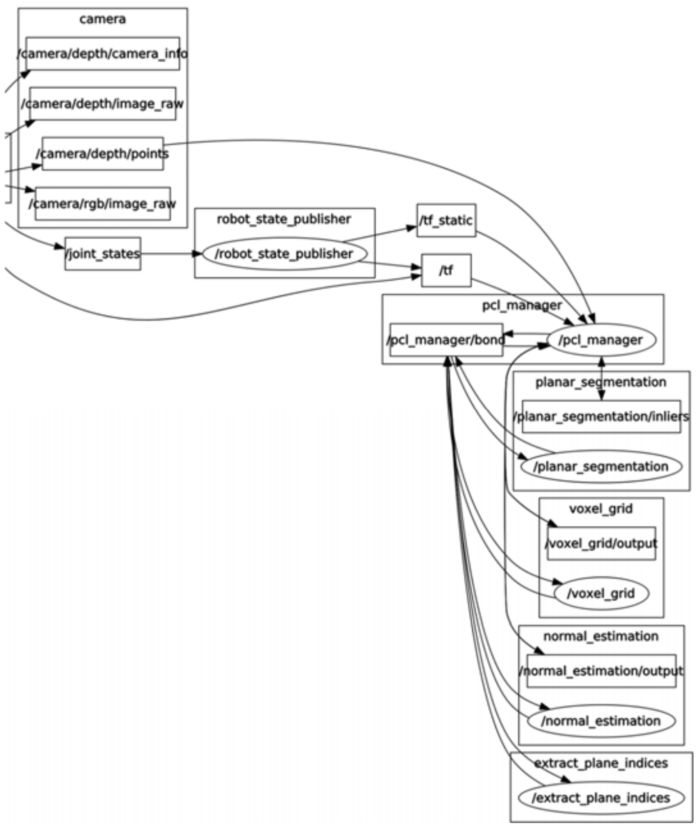
a) Get the image point clound from the Turtlebot Kinect ()
b) Input the raw point clound to a passthrough that will change the type in (Pointcloud2) in order to work with PCL and also reduce points cloud to restricted area.
c) Input this new point cloud (with remap) to get normal and transformed it into a voxel could.
d) This voxel could will be process with the RANSAC in order to make the segmentation, with SAC segmentation from the PCL.

e) 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
```

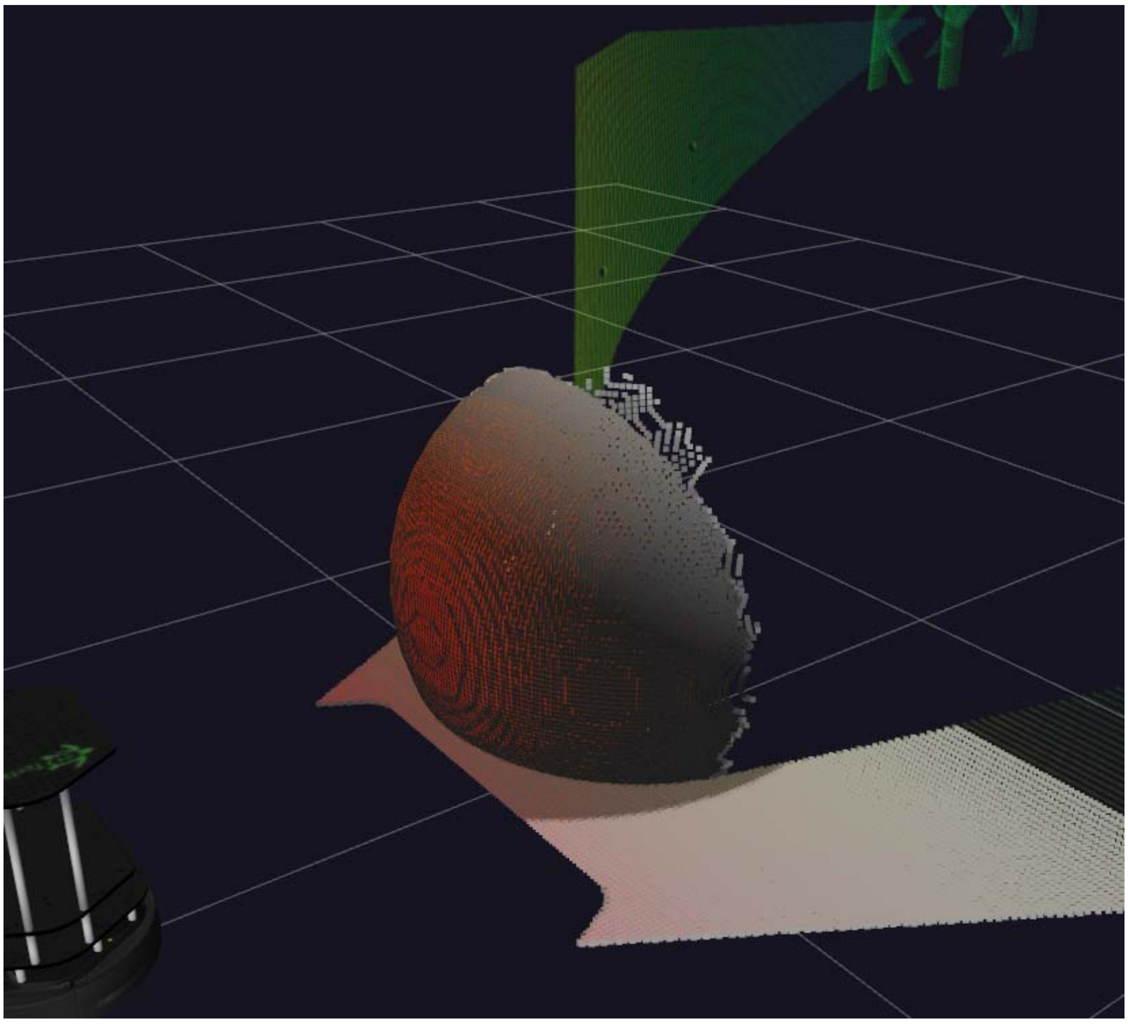
f) Then we will output the object type regarding the SACMODEL that match.

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.

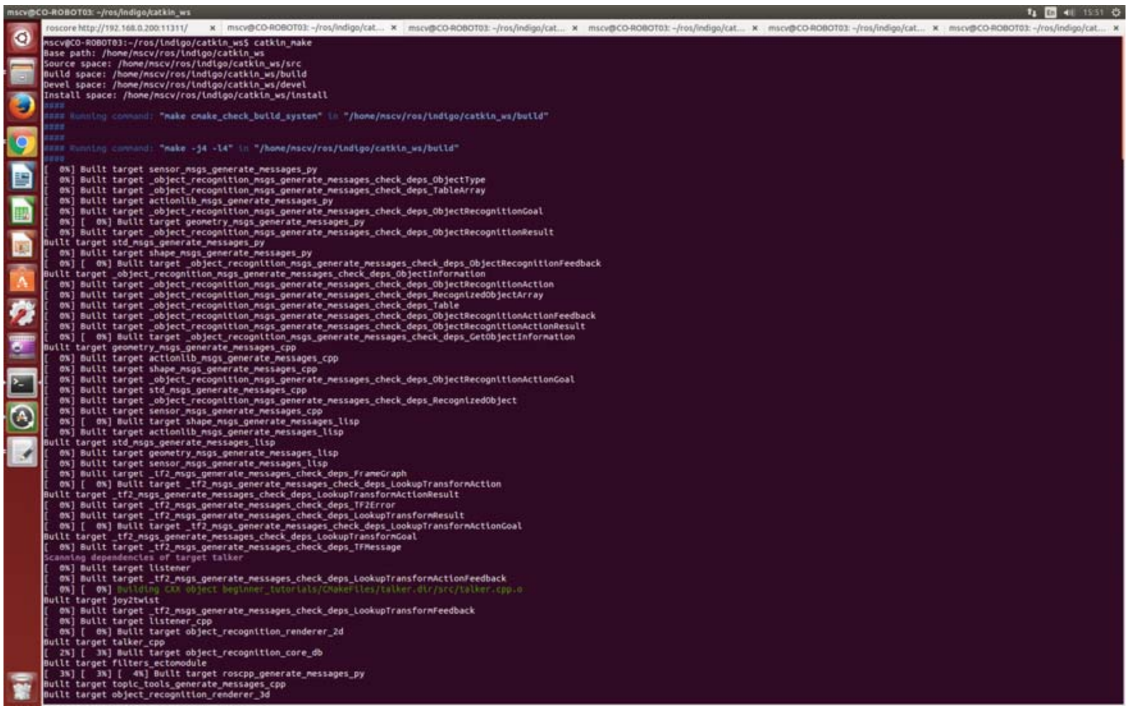
To use this node just build it using **catkin_make** then **rosrun sim_detect sim_detect_node**

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.



```
mscv@CO-ROBOT03:~/ros/indigo/catkin_ws$ catkin_make
Base path: /home/mscv/ros/indigo/catkin_ws
Source space: /home/mscv/ros/indigo/catkin_ws/src
Build space: /home/mscv/ros/indigo/catkin_ws/build
Devel space: /home/mscv/ros/indigo/catkin_ws/devel
Install space: /home/mscv/ros/indigo/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/mscv/ros/indigo/catkin_ws/build"
####
####
#### Running command: "make -j4 -l4" in "/home/mscv/ros/indigo/catkin_ws/build"
####
[  0%] Built target sensor_msgs_generate_messages_py
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_ObjectType
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_TableArray
[  0%] Built target actionlib_msgs_generate_messages_py
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_ObjectRecognitionGoal
[  0%] [  0%] Built target geometry_msgs_generate_messages_py
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_ObjectRecognitionResult
Built target std_msgs_generate_messages_py
[  0%] Built target shape_msgs_generate_messages_py
[  0%] [  0%] Built target _object_recognition_msgs_generate_messages_check_deps_ObjectRecognitionFeedback
Built target _object_recognition_msgs_generate_messages_check_deps_ObjectInformation
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_ObjectRecognitionAction
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_RecognizedObjectArray
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_Table
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_ObjectRecognitionActionFeedback
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_ObjectRecognitionActionResult
[  0%] [  0%] Built target _object_recognition_msgs_generate_messages_check_deps_GetObjectInformation
Built target geometry_msgs_generate_messages_cpp
[  0%] Built target actionlib_msgs_generate_messages_cpp
[  0%] Built target shape_msgs_generate_messages_cpp
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_ObjectRecognitionActionGoal
[  0%] Built target std_msgs_generate_messages_cpp
[  0%] Built target _object_recognition_msgs_generate_messages_check_deps_RecognizedObject
[  0%] Built target sensor_msgs_generate_messages_cpp
[  0%] [  0%] Built target shape_msgs_generate_messages_lisp
[  0%] Built target actionlib_msgs_generate_messages_lisp
Built target std_msgs_generate_messages_lisp
[  0%] Built target geometry_msgs_generate_messages_lisp
[  0%] Built target sensor_msgs_generate_messages_lisp
[  0%] Built target _tf2_msgs_generate_messages_check_deps_FrameGraph
[  0%] [  0%] Built target _tf2_msgs_generate_messages_check_deps_LookupTransformAction
Built target _tf2_msgs_generate_messages_check_deps_LookupTransformActionResult
[  0%] Built target _tf2_msgs_generate_messages_check_deps_TF2Error
[  0%] Built target _tf2_msgs_generate_messages_check_deps_LookupTransformResult
[  0%] [  0%] Built target _tf2_msgs_generate_messages_check_deps_LookupTransformActionGoal
Built target _tf2_msgs_generate_messages_check_deps_LookupTransformGoal
[  0%] Built target _tf2_msgs_generate_messages_check_deps_TFMessage
Scanning dependencies of target talker
[  0%] Built target listener
[  0%] Built target _tf2_msgs_generate_messages_check_deps_LookupTransformActionFeedback
[  0%] [  0%] Building CXX object beginner_tutorials/CMakeFiles/talker.dir/src/talker.cpp.o
Built target joy2twist
[  0%] Built target _tf2_msgs_generate_messages_check_deps_LookupTransformFeedback
[  0%] Built target listener_cpp
[  0%] [  0%] Built target object_recognition_renderer_2d
Built target talker_cpp
[  2%] [  3%] Built target object_recognition_core_db
Built target filters_ectomodule
[  3%] [  3%] [  4%] Built target roscpp_generate_messages_py
Built target topic_tools_generate_messages_cpp
Built target object_recognition_renderer_3d
```

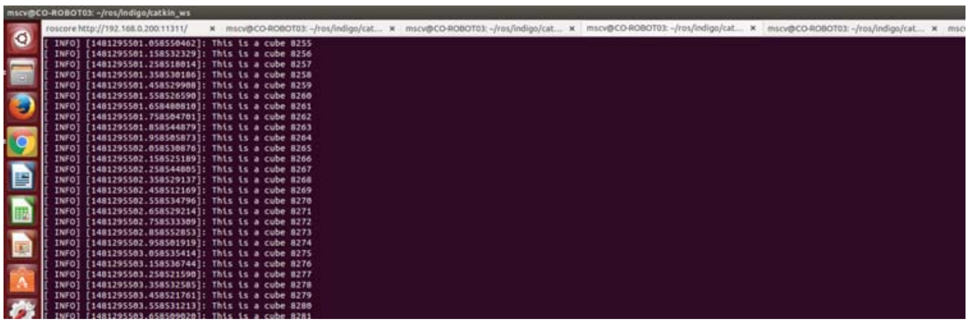
In a different terminal, we run roscore. We open a new terminal, and type in our catkin_ws

source ./devel/setup.bash

and then type rosrun beginner_tutorials talker.

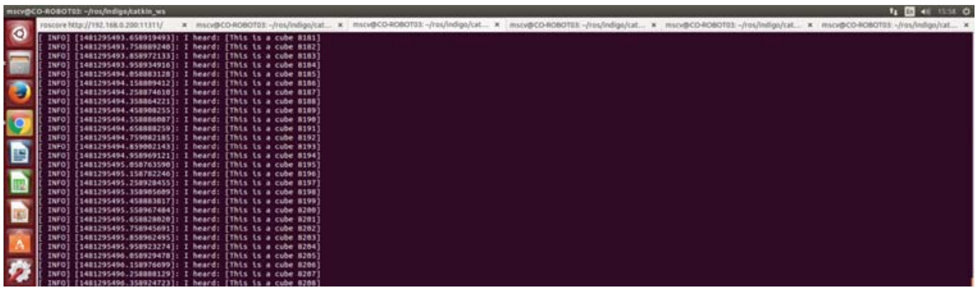
In a different terminal, type rosrun beginner_tutorials listener. Below is the screenshot for the result.

the publisher



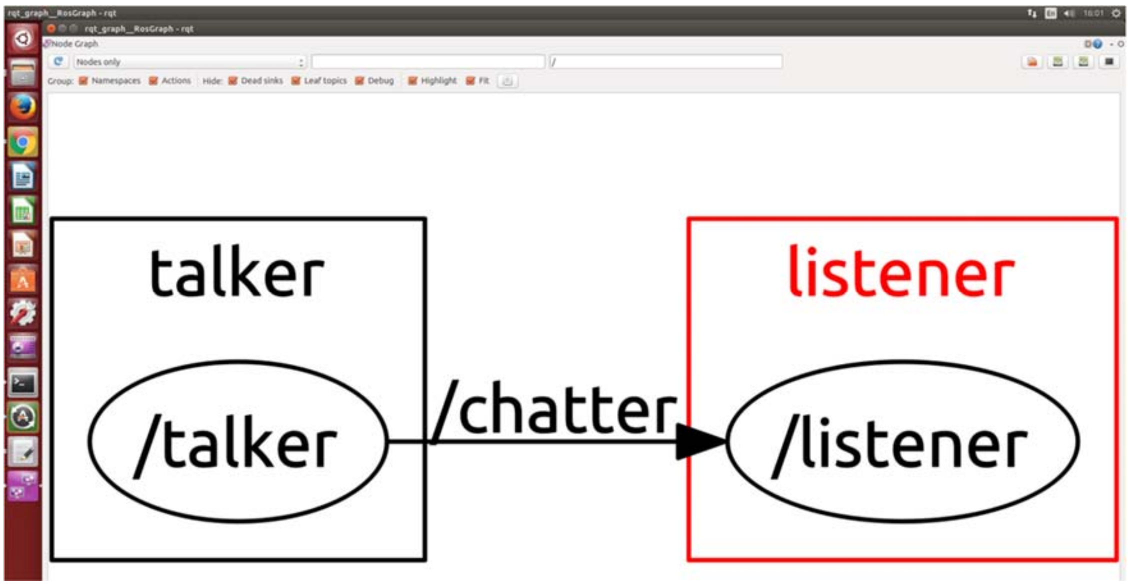
```
mscv@CO-ROBOT03: ~/ros/indigo/catkin_ws
roscore http://192.168.0.200:11311/   mscv@CO-ROBOT03: ~/ros/indigo/cat...   mscv@CO-ROBOT03: ~/ros/indigo/cat...
[ INFO] [1481295501.058550462]: This is a cube 8255
[ INFO] [1481295501.158532329]: This is a cube 8256
[ INFO] [1481295501.258518014]: This is a cube 8257
[ INFO] [1481295501.358530186]: This is a cube 8258
[ INFO] [1481295501.458529908]: This is a cube 8259
[ INFO] [1481295501.558526590]: This is a cube 8260
[ INFO] [1481295501.658480810]: This is a cube 8261
[ INFO] [1481295501.758504701]: This is a cube 8262
[ INFO] [1481295501.858544879]: This is a cube 8263
[ INFO] [1481295501.958505873]: This is a cube 8264
[ INFO] [1481295502.058530876]: This is a cube 8265
[ INFO] [1481295502.158525189]: This is a cube 8266
[ INFO] [1481295502.258544805]: This is a cube 8267
[ INFO] [1481295502.358529137]: This is a cube 8268
[ INFO] [1481295502.458512169]: This is a cube 8269
[ INFO] [1481295502.558534796]: This is a cube 8270
[ INFO] [1481295502.658529214]: This is a cube 8271
[ INFO] [1481295502.758533309]: This is a cube 8272
[ INFO] [1481295502.858552853]: This is a cube 8273
[ INFO] [1481295502.958501919]: This is a cube 8274
[ INFO] [1481295503.058535414]: This is a cube 8275
[ INFO] [1481295503.158536744]: This is a cube 8276
[ INFO] [1481295503.258521590]: This is a cube 8277
[ INFO] [1481295503.358532585]: This is a cube 8278
[ INFO] [1481295503.458521761]: This is a cube 8279
[ INFO] [1481295503.558531213]: This is a cube 8280
[ INFO] [1481295503.658509020]: This is a cube 8281
```

the subscriber



```
mscv@CO-ROBOT03: ~/ros/indigo/catkin_ws
roscore http://192.168.0.200:11311/   mscv@CO-ROBOT03: ~/ros/indigo/cat...   mscv@CO-ROBOT03: ~/ros/indigo/cat...
[ INFO] [1481295493.658919493]: I heard: [This is a cube 8181]
[ INFO] [1481295493.758889240]: I heard: [This is a cube 8182]
[ INFO] [1481295493.858972133]: I heard: [This is a cube 8183]
[ INFO] [1481295493.958934916]: I heard: [This is a cube 8184]
[ INFO] [1481295494.058883128]: I heard: [This is a cube 8185]
[ INFO] [1481295494.158809412]: I heard: [This is a cube 8186]
[ INFO] [1481295494.258874610]: I heard: [This is a cube 8187]
[ INFO] [1481295494.358864221]: I heard: [This is a cube 8188]
[ INFO] [1481295494.458908255]: I heard: [This is a cube 8189]
[ INFO] [1481295494.558886087]: I heard: [This is a cube 8190]
[ INFO] [1481295494.658888259]: I heard: [This is a cube 8191]
[ INFO] [1481295494.759082185]: I heard: [This is a cube 8192]
[ INFO] [1481295494.859002143]: I heard: [This is a cube 8193]
[ INFO] [1481295494.958969121]: I heard: [This is a cube 8194]
[ INFO] [1481295495.058763590]: I heard: [This is a cube 8195]
[ INFO] [1481295495.158782246]: I heard: [This is a cube 8196]
[ INFO] [1481295495.258920455]: I heard: [This is a cube 8197]
[ INFO] [1481295495.358985609]: I heard: [This is a cube 8198]
[ INFO] [1481295495.458883817]: I heard: [This is a cube 8199]
[ INFO] [1481295495.558967484]: I heard: [This is a cube 8200]
[ INFO] [1481295495.658828020]: I heard: [This is a cube 8201]
[ INFO] [1481295495.758945691]: I heard: [This is a cube 8202]
[ INFO] [1481295495.858962495]: I heard: [This is a cube 8203]
[ INFO] [1481295495.958923274]: I heard: [This is a cube 8204]
[ INFO] [1481295496.058929478]: I heard: [This is a cube 8205]
[ INFO] [1481295496.158976099]: I heard: [This is a cube 8206]
[ INFO] [1481295496.258888129]: I heard: [This is a cube 8207]
[ INFO] [1481295496.358924723]: I heard: [This is a cube 8208]
```

Also, if we want to see the topics, type rqt_graph.


rqt_graph__RosGraph - rqt
Node Graph
Nodes only
Group: Namespaces Actions Hide: Dead sinks Leaf topics Debug Highlight Fit
talker
/talker
/chatter
listener
/listener

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

```
roslaunch sound_play soundplay_node.launch
```

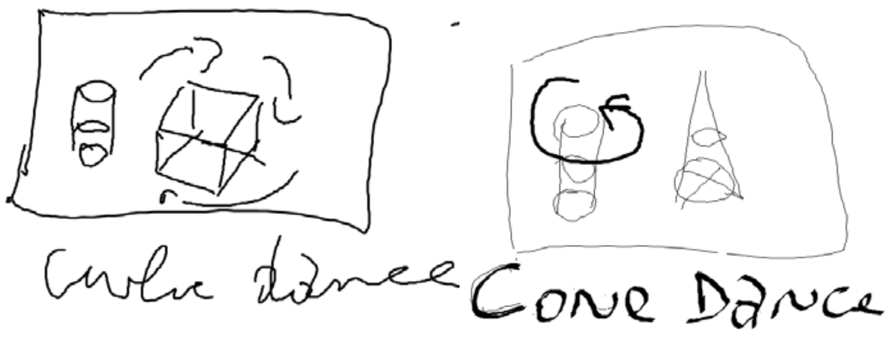
Then again in a new terminal, type:

```
rosrun sound_play say.py "How do I sound now?"
```

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

Movements

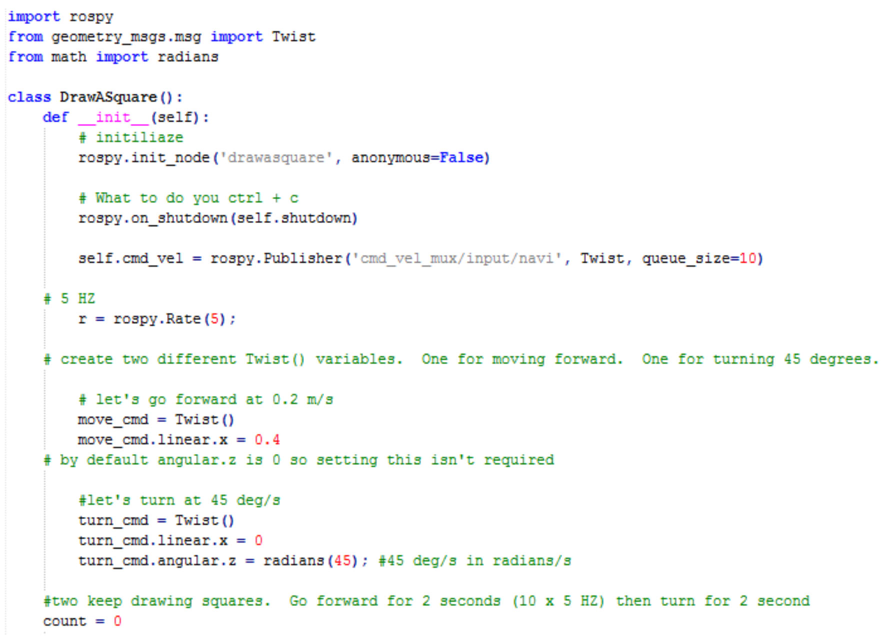
As we had detected and recognized the object, we want the robot to do specific movement based on what it has recognized. If the robot recognizes a cube we want it to do a square around it and if it recognizes a cone we want it to do a cycle around it.



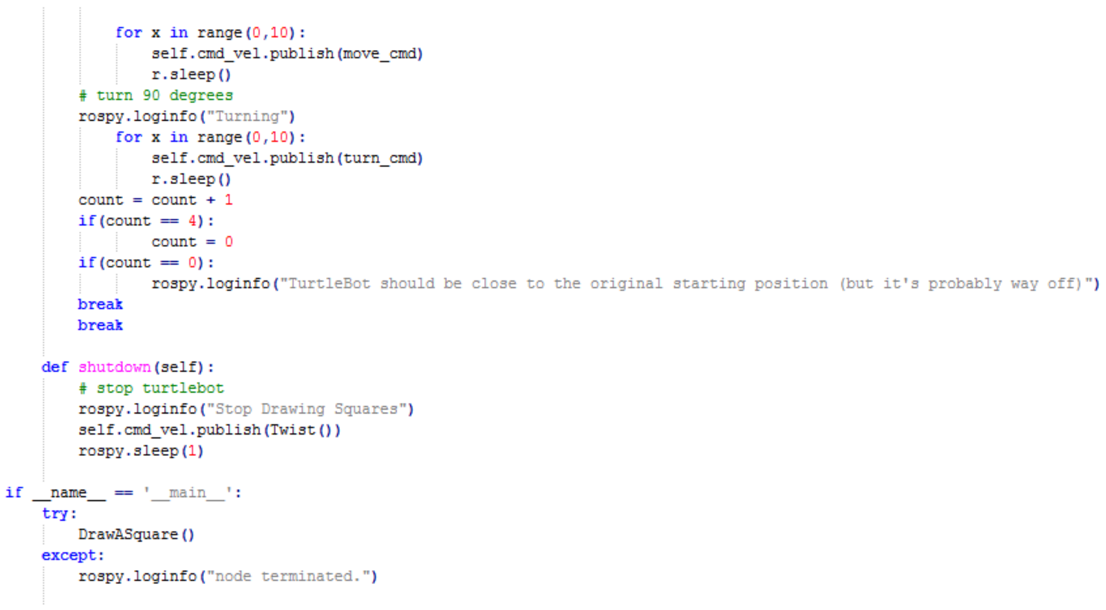
We wrote several python and cpp codes to achieve that, but most of them worked on gazebo but when we tried them to the Tbot they did not worked, and this is something that happened very often since every parameter change (environment, lights, etc).

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

draw_square.py



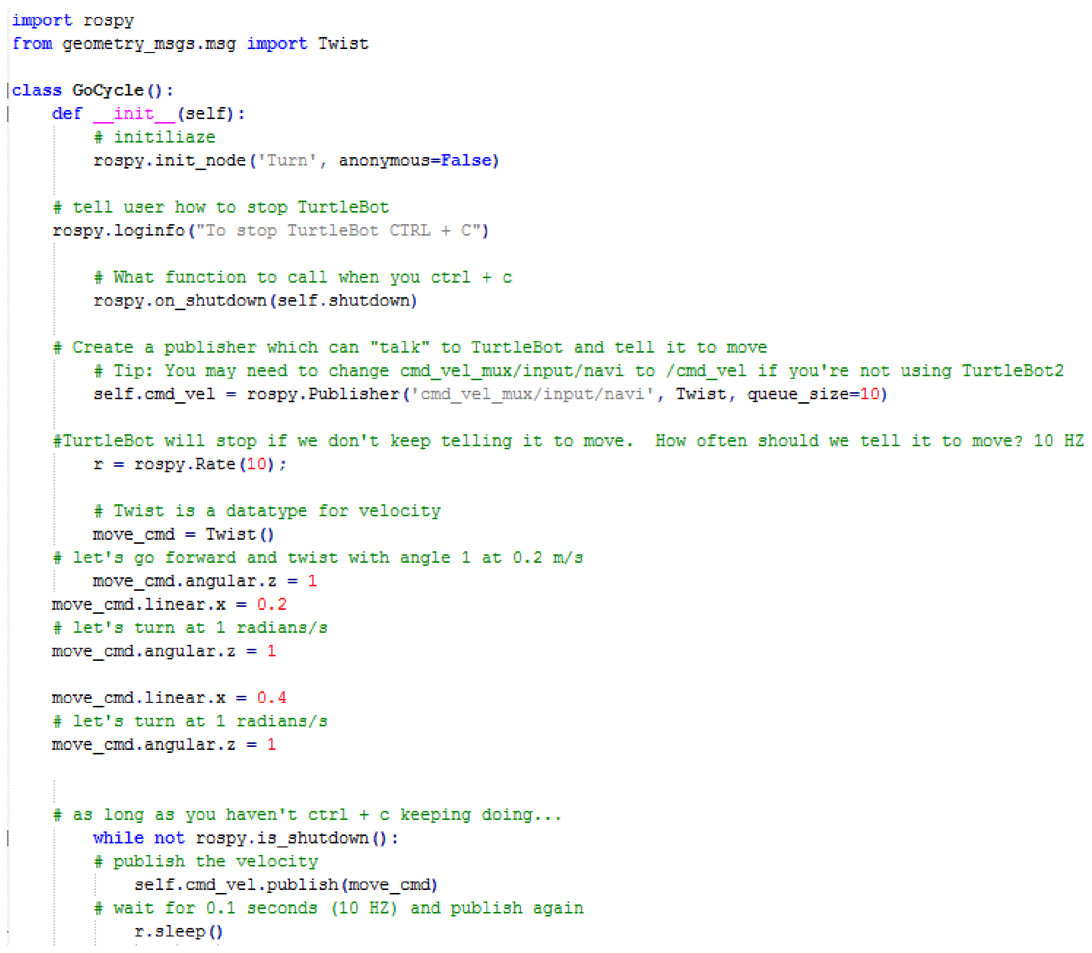
```
import rospy
from geometry_msgs.msg import Twist
from math import radians

class DrawASquare():
    def __init__(self):
        # initiliaze
        rospy.init_node('drawasquare', anonymous=False)

        # What to do you ctrl + c
        rospy.on_shutdown(self.shutdown)

        self.cmd_vel = rospy.Publisher('cmd_vel_mux/input/navi', Twist, queue_size=10)

    # 5 HZ
        r = rospy.Rate(5);

    # create two different Twist() variables.  One for moving forward.  One for turning 45 degrees.

        # let's go forward at 0.2 m/s
        move_cmd = Twist()
        move_cmd.linear.x = 0.4
    # by default angular.z is 0 so setting this isn't required

        #let's turn at 45 deg/s
        turn_cmd = Twist()
        turn_cmd.linear.x = 0
        turn_cmd.angular.z = radians(45); #45 deg/s in radians/s

    #two keep drawing squares.  Go forward for 2 seconds (10 x 5 HZ) then turn for 2 second
    count = 0
```



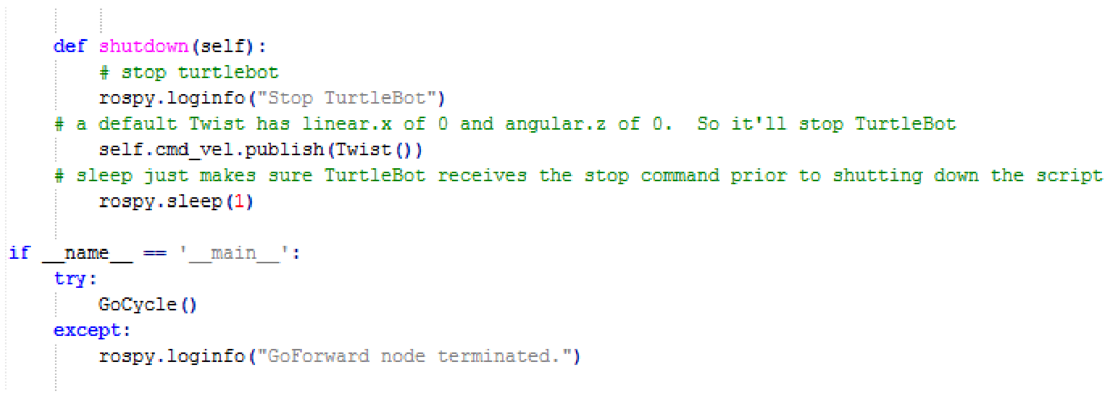
```
            for x in range(0,10):
                self.cmd_vel.publish(move_cmd)
                r.sleep()
        # turn 90 degrees
        rospy.loginfo("Turning")
            for x in range(0,10):
                self.cmd_vel.publish(turn_cmd)
                r.sleep()
        count = count + 1
        if(count == 4):
                count = 0
        if(count == 0):
                rospy.loginfo("TurtleBot should be close to the original starting position (but it's probably way off)")
        break
        break

    def shutdown(self):
        # stop turtlebot
        rospy.loginfo("Stop Drawing Squares")
        self.cmd_vel.publish(Twist())
        rospy.sleep(1)

if __name__ == '__main__':
    try:
        DrawASquare()
    except:
        rospy.loginfo("node terminated.")
```

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

gocycle.py



```
import rospy
from geometry_msgs.msg import Twist

class GoCycle():
    def __init__(self):
        # initiliaze
        rospy.init_node('Turn', anonymous=False)

    # tell user how to stop TurtleBot
    rospy.loginfo("To stop TurtleBot CTRL + C")

        # What function to call when you ctrl + c
        rospy.on_shutdown(self.shutdown)

    # Create a publisher which can "talk" to TurtleBot and tell it to move
        # Tip: You may need to change cmd_vel_mux/input/navi to /cmd_vel if you're not using TurtleBot2
        self.cmd_vel = rospy.Publisher('cmd_vel_mux/input/navi', Twist, queue_size=10)

    #TurtleBot will stop if we don't keep telling it to move.  How often should we tell it to move? 10 HZ
        r = rospy.Rate(10);

        # Twist is a datatype for velocity
        move_cmd = Twist()
    # let's go forward and twist with angle 1 at 0.2 m/s
        move_cmd.angular.z = 1
    move_cmd.linear.x = 0.2
    # let's turn at 1 radians/s
    move_cmd.angular.z = 1

    move_cmd.linear.x = 0.4
    # let's turn at 1 radians/s
    move_cmd.angular.z = 1


    # as long as you haven't ctrl + c keeping doing...
        while not rospy.is_shutdown():
        # publish the velocity
            self.cmd_vel.publish(move_cmd)
        # wait for 0.1 seconds (10 HZ) and publish again
            r.sleep()
```



```
    def shutdown(self):
        # stop turtlebot
        rospy.loginfo("Stop TurtleBot")
    # a default Twist has linear.x of 0 and angular.z of 0.  So it'll stop TurtleBot
        self.cmd_vel.publish(Twist())
    # sleep just makes sure TurtleBot receives the stop command prior to shutting down the script
        rospy.sleep(1)

if __name__ == '__main__':
    try:
        GoCycle()
    except:
        rospy.loginfo("GoForward node terminated.")
```

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

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