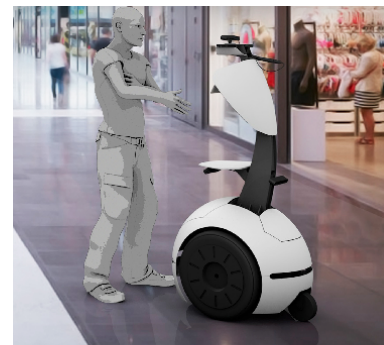




UNIVERSITÀ
di **VERONA**

Dipartimento
di **INFORMATICA**

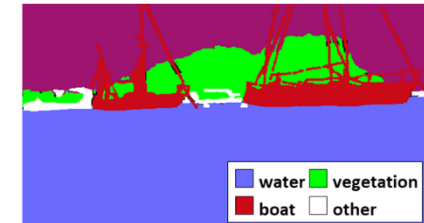
Corso di Laboratorio Ciberfisico
Modulo di Robot Programming with ROS



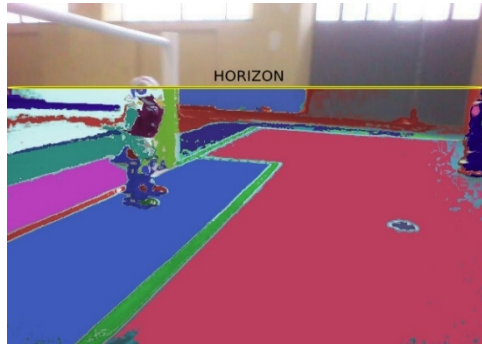
actionlib

ROS

Docente:
Domenico Daniele Bloisi



Giugno 2018



References and Credits

Questo materiale deriva da:

Luca Iocchi – Sapienza Università di Roma
Actions and Plans

<https://www.dis.uniroma1.it/~nardi/Didattica/CAI/matdid/1-ROS-ActionLib.pdf>

ROS actionlib

- Node A sends a request to node B to perform some task
- **Services** are suitable if task is "instantaneous"
- **Actions** are more adequate when task takes time and we want to monitor, have continuous feedback and possibly cancel the request during execution

actionlib package

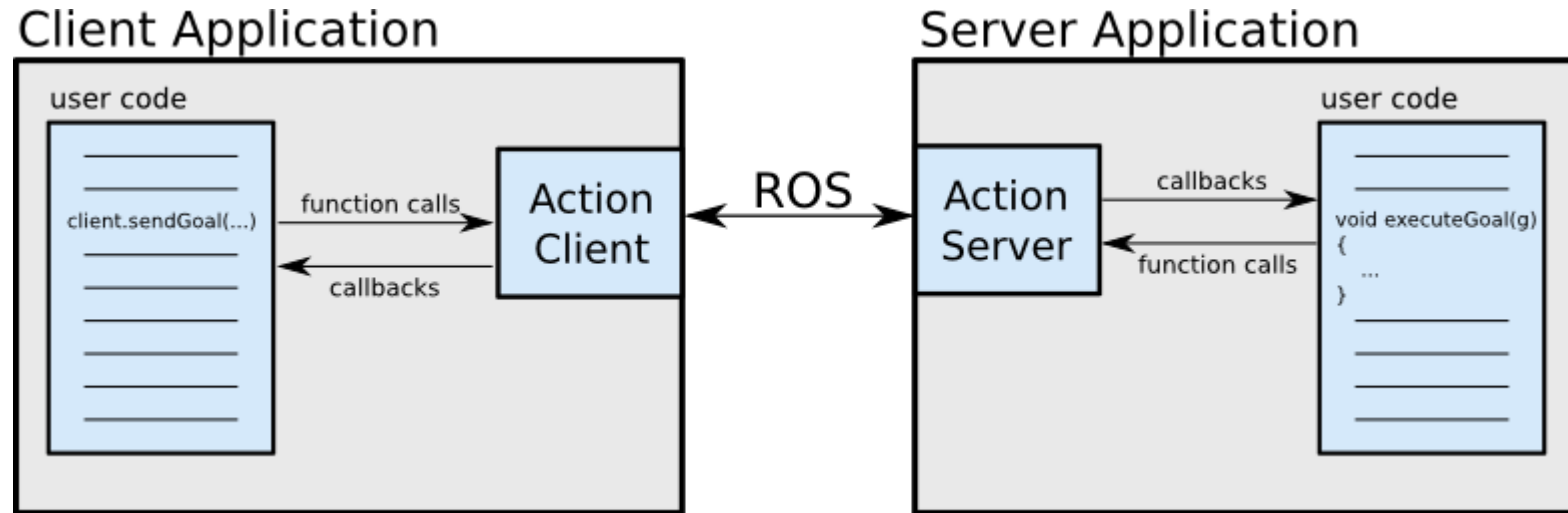
actionlib package provides tools to

- create **servers** that execute long-running tasks (that can be preempted)
- create **clients** that interact with servers

References

- <http://wiki.ros.org/actionlib>
- <http://wiki.ros.org/actionlib/DetailedDescription>
- <http://wiki.ros.org/actionlib/Tutorials>

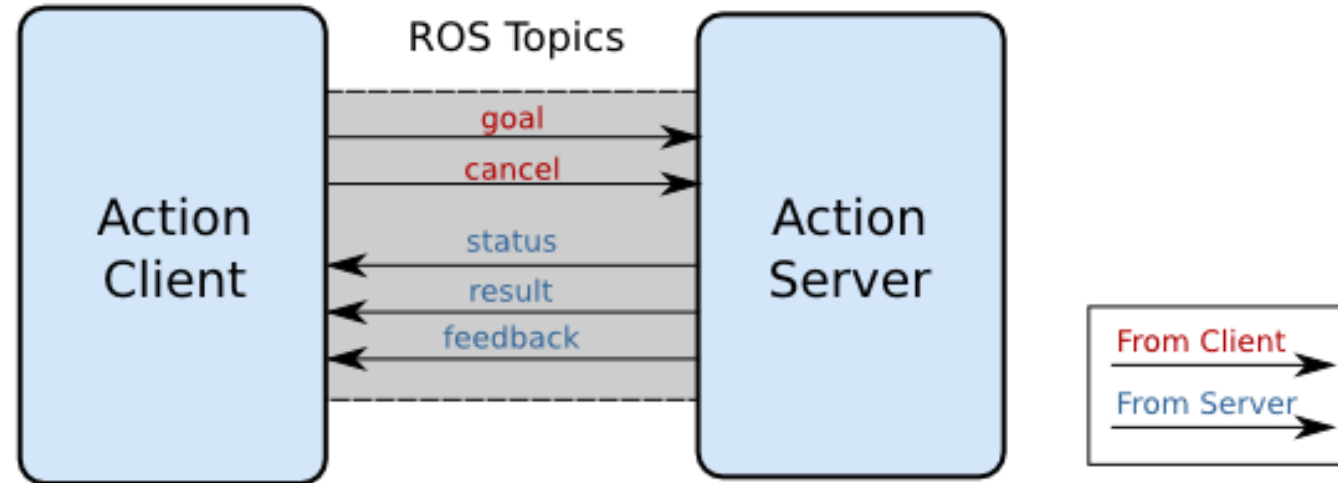
Client-Server interaction



- The ActionClient and ActionServer communicate via a "ROS Action Protocol", which is built on top of ROS messages
- The client and server then provide a simple API for users to request goals (on the client side) or to execute goals (on the server side) via function calls and callbacks

Action Interface & Transport Layer

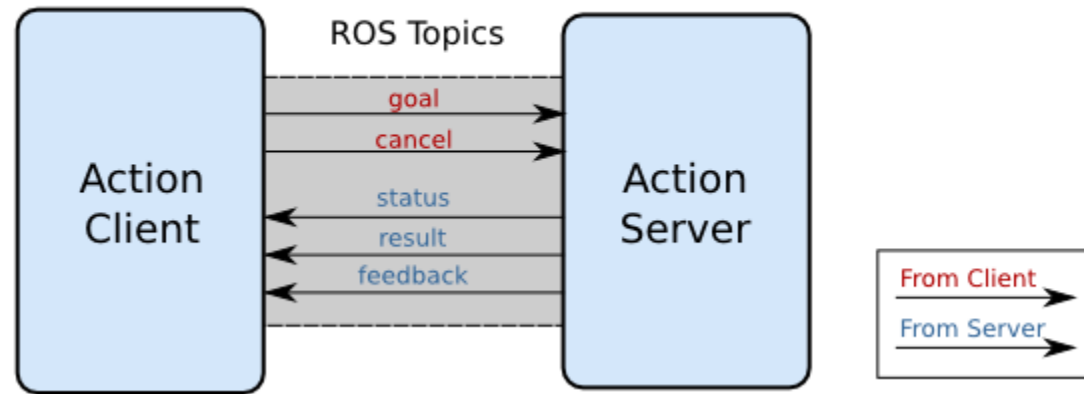
Action Interface



- The action client and server communicate with each other using a predefined action protocol
- This action protocol relies on ROS topics in a specified ROS namespace in order to transport messages

ROS Messages

Action Interface



- **goal** - Used to send new goals to servers
- **cancel** - Used to send cancel requests to servers
- **status** - Used to notify clients on the current state of every goal in the system
- **feedback** - Used to send clients periodic auxiliary information for a goal
- **result** - Used to send clients one-time auxiliary information upon completion of a goal

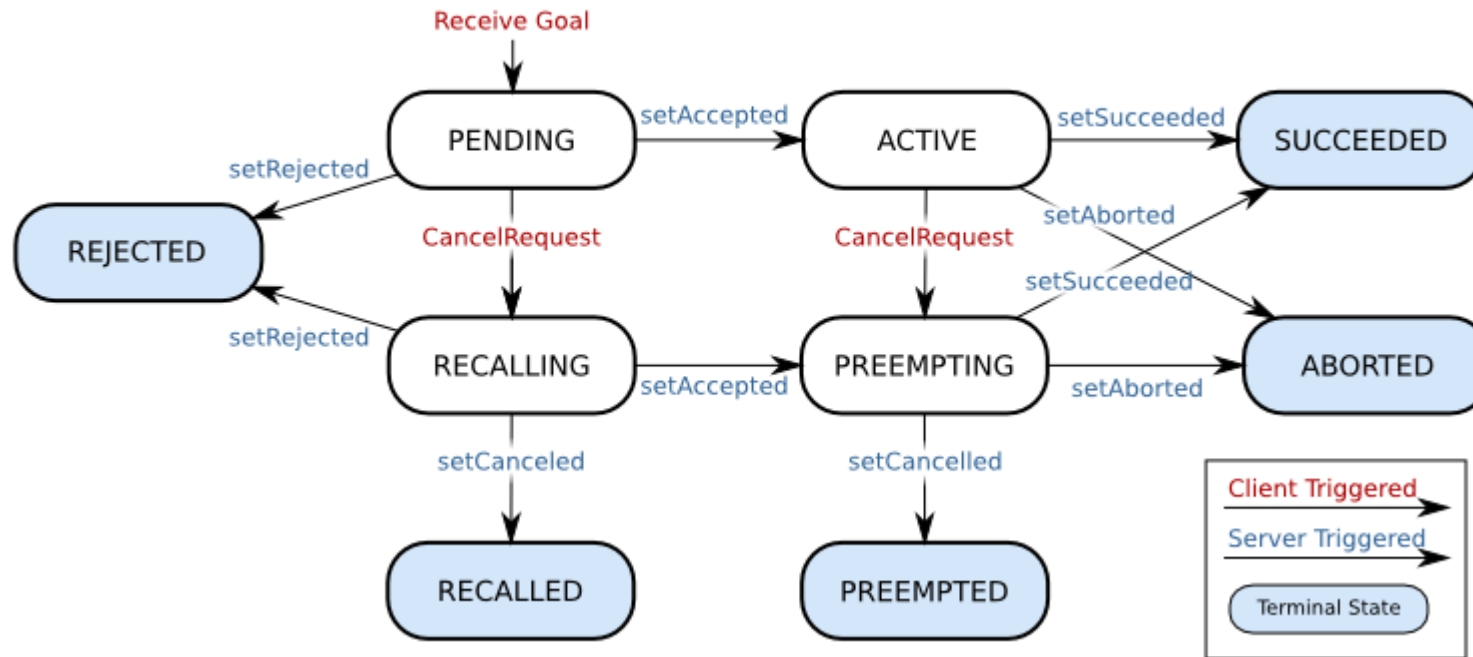
Actions and Goal ID

- Action templates are defined by a name and some additional properties through an `.action` structure defined in ROS
- Each instance of an action has a unique `Goal ID`
- `Goal ID` provides the action server and the action client with a robust way to monitor the execution of a particular instance of an action

Server State Machine

- Goals are initiated by an ActionClient
- Once a goal is received by an ActionServer, the ActionServer creates a state machine to track the status of the goal

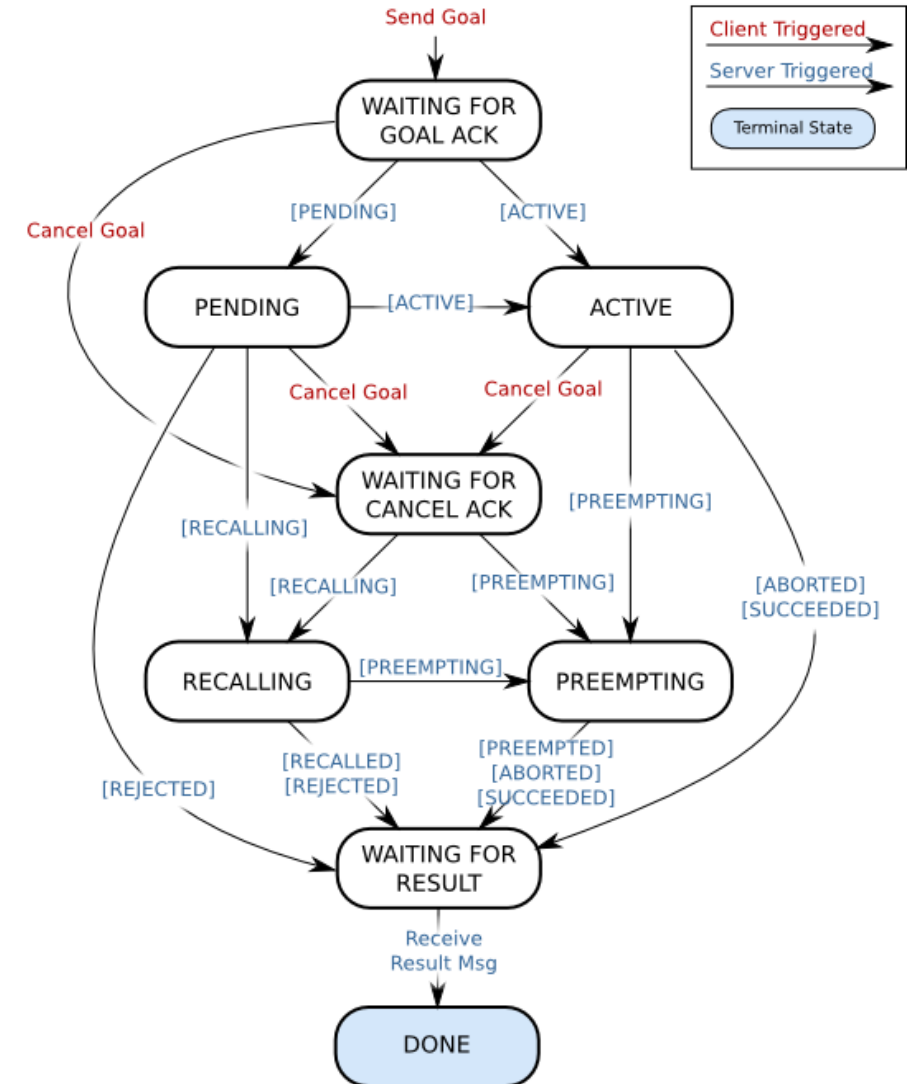
Server State Transitions



Client State Machine

- in actionlib, the server state machine is the primary machine
- the client state machine is the secondary/coupled state machine that tries to track the server's state

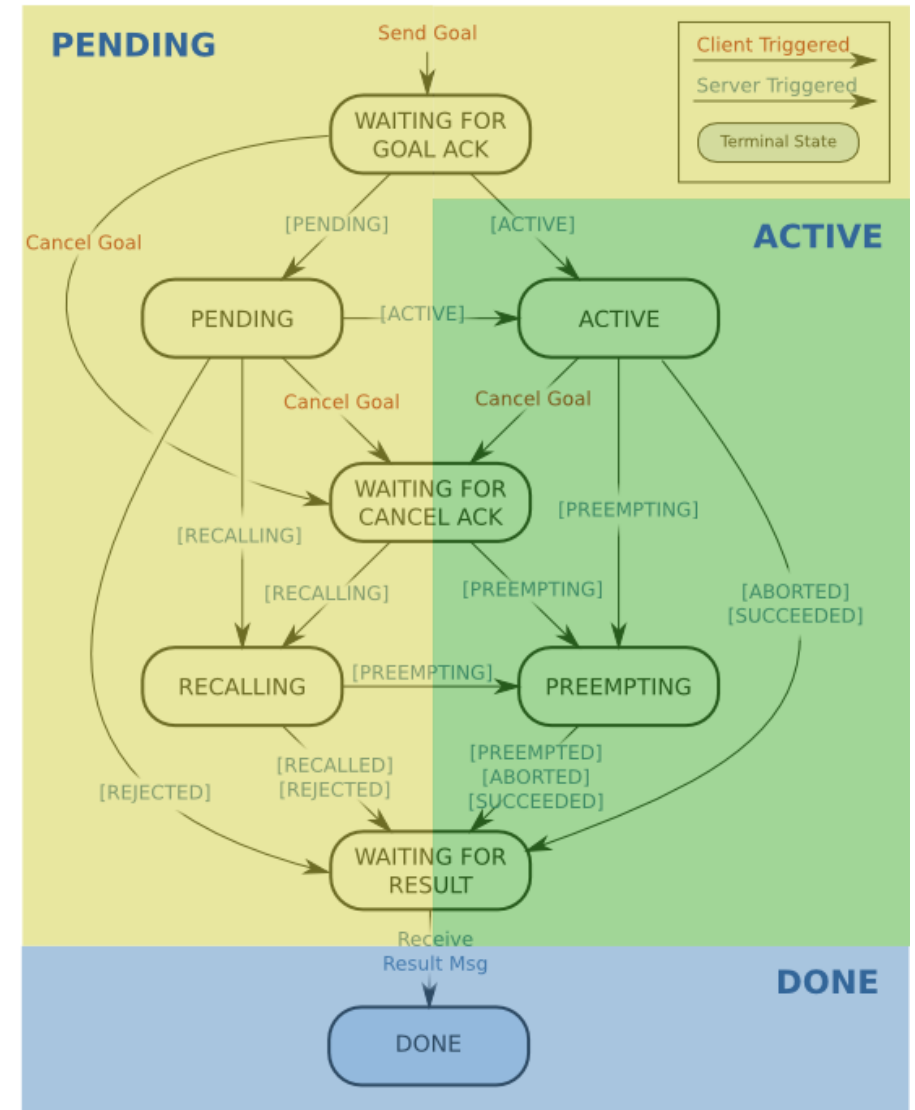
Client State Transitions



SimpleActionServer/Client

- SimpleActionServer: implements a single goal policy
- Only one goal can have an active status at a time
- New goals preempt previous goals based on the stamp in their GoalID field
- SimpleActionClient: implements a simplified ActionClient

Simple Client State Transitions



Example: move_base action server

- **Action Subscribed Topics**

- move_base/goal ([move_base_msgs/MoveBaseActionGoal](#)): A goal for move_base to pursue in the world
- move_base/cancel ([actionlib_msgs/GoalID](#)): A request to cancel a specific goal

- **Action Published Topics**

- move_base/feedback ([move_base_msgs/MoveBaseActionFeedback](#)): Feedback contains the current position of the base in the world
- move_base/status ([actionlib_msgs/GoalStatusArray](#)): Provides status information on the goals that are sent to the move_base action
- move_base/result ([move_base_msgs/MoveBaseActionResult](#)): Result is empty for the move_base action

Sending a goal with move_base

```
typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>
    MoveBaseClient;

//tell the action client that we want to spin a thread by default
MoveBaseClient ac("move_base", true);

//wait for the action server to come up
while(!ac.waitForServer(ros::Duration(5.0))){
    ROS_INFO("Waiting for the move_base action server to come up");
}

// setting the goal
move_base_msgs::MoveBaseGoal goal;
goal.target_pose.header.frame_id = "base_link";
goal.target_pose.header.stamp = ros::Time::now();
goal.target_pose.pose.position.x = 1.0;
goal.target_pose.pose.orientation.w = 1.0;
```

Sending a goal with move_base

```
// sending the goal
ROS_INFO("Sending goal");
ac.sendGoal(goal);

// wait until finish
ac.waitForResult();

// print result
if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
    ROS_INFO("Hooray, the base moved 1 meter forward");
else
    ROS_INFO("The base failed to move forward 1 meter for some reason");
```

Cancelling a goal with move_base

```
typedef
actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>
MoveBaseClient;

MoveBaseClient ac("move_base", true);

...

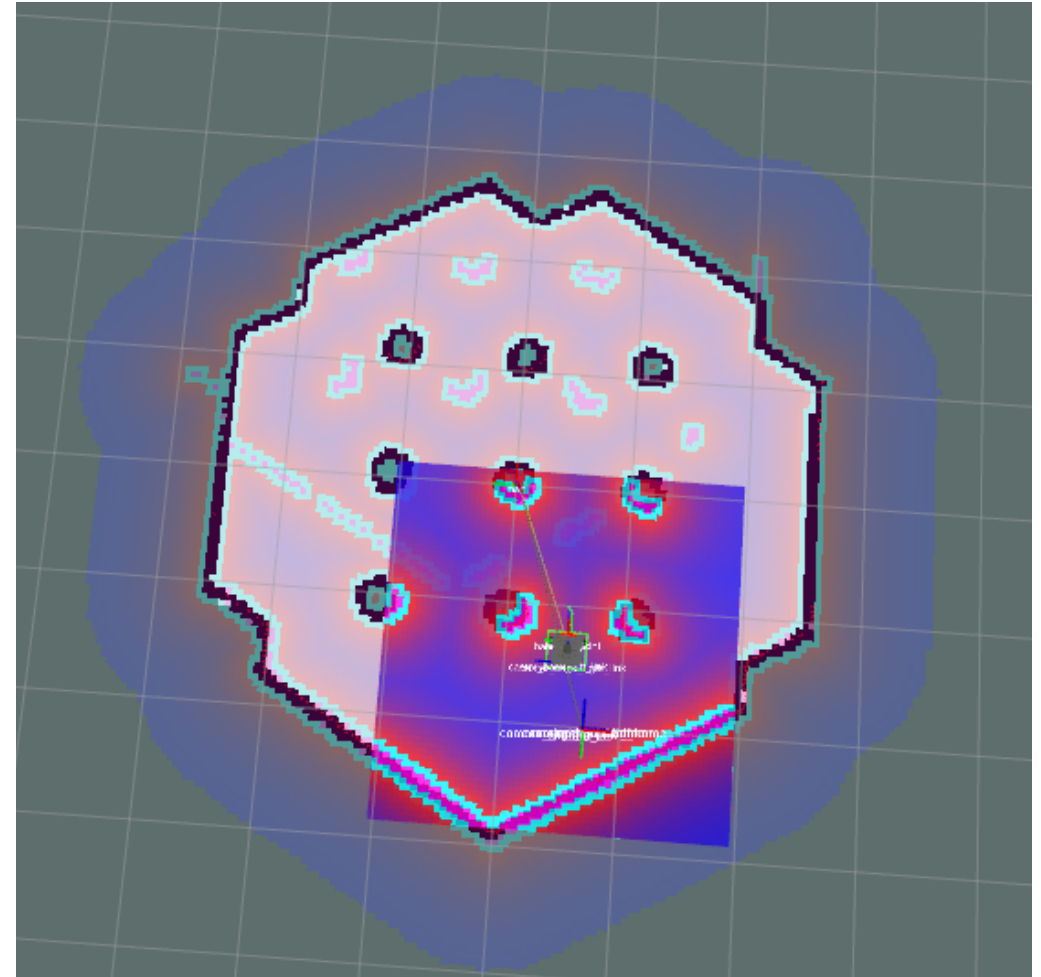
// Cancel all active goals
ac.cancelAllGoals();
```

ActionServer/Client

- [ActionServer](#) and [ActionClient](#) use the complete set of states and transitions
- More difficult to program
- Needed when it is necessary to execute multiple instances of an action at the same time (parallel actions)

Example Turtlebot3

- Use the `SimpleActionClient` to send a navigation goal to the Turtlebot3
- Tell the base to move 1 meter forward in the "base_link" coordinate frame



Package creation

```
$ catkin_create_pkg turtlebot3_navigation_goals  
  move_base_msgs actionlib roscpp
```

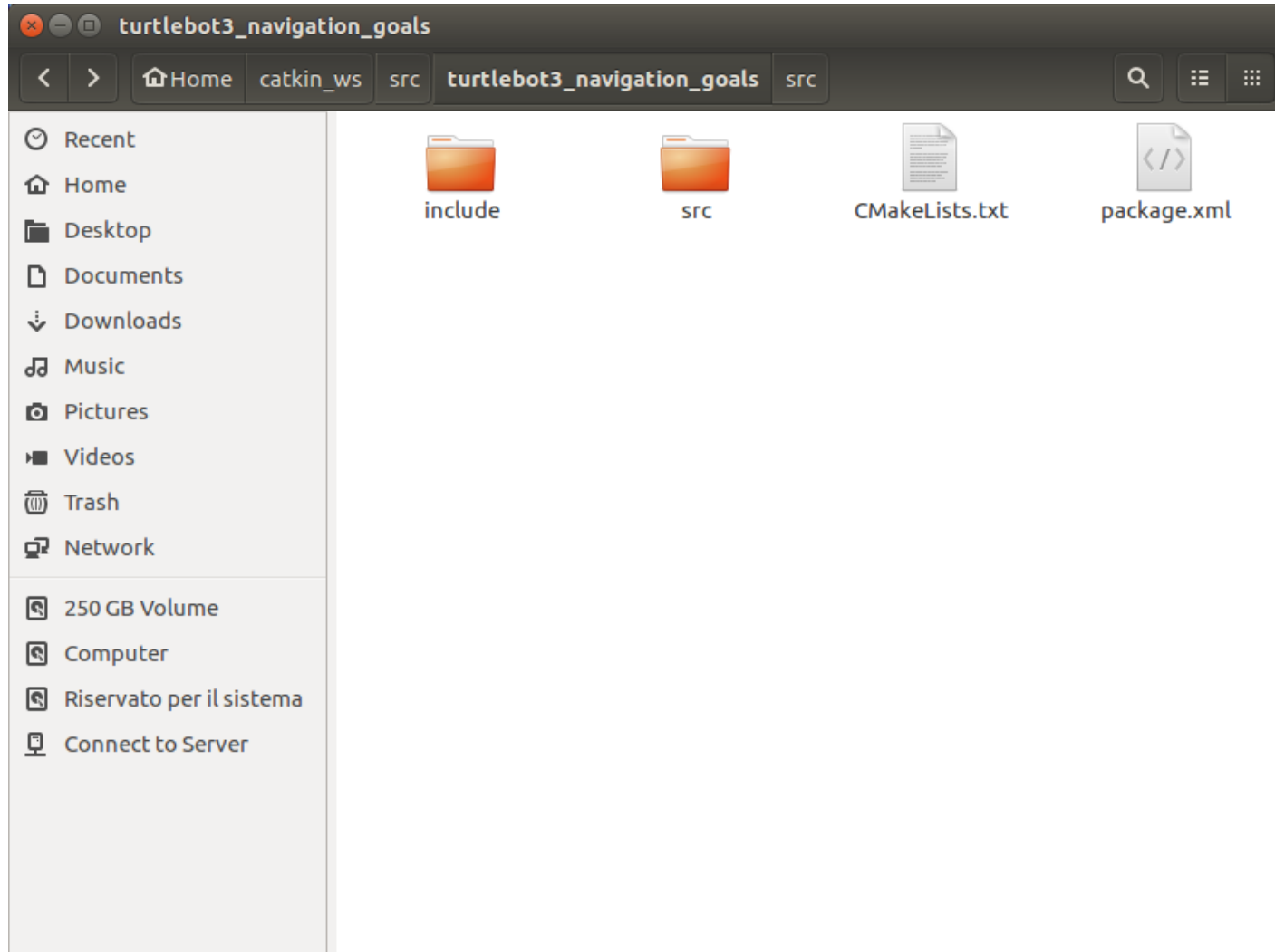
```
bloisi@bloisi-U36SG:~/catkin_ws/src$ catkin_create_pkg turtlebot3_navigation_goa  
ls move_base_msgs actionlib roscpp  
Created file turtlebot3_navigation_goals/CMakeLists.txt  
Created file turtlebot3_navigation_goals/package.xml  
Created folder turtlebot3_navigation_goals/include/turtlebot3_navigation_goals  
Created folder turtlebot3_navigation_goals/src  
Successfully created files in /home/bloisi/catkin_ws/src/turtlebot3_navigation_g  
oals. Please adjust the values in package.xml.
```

turtlebot3_navigation_goal

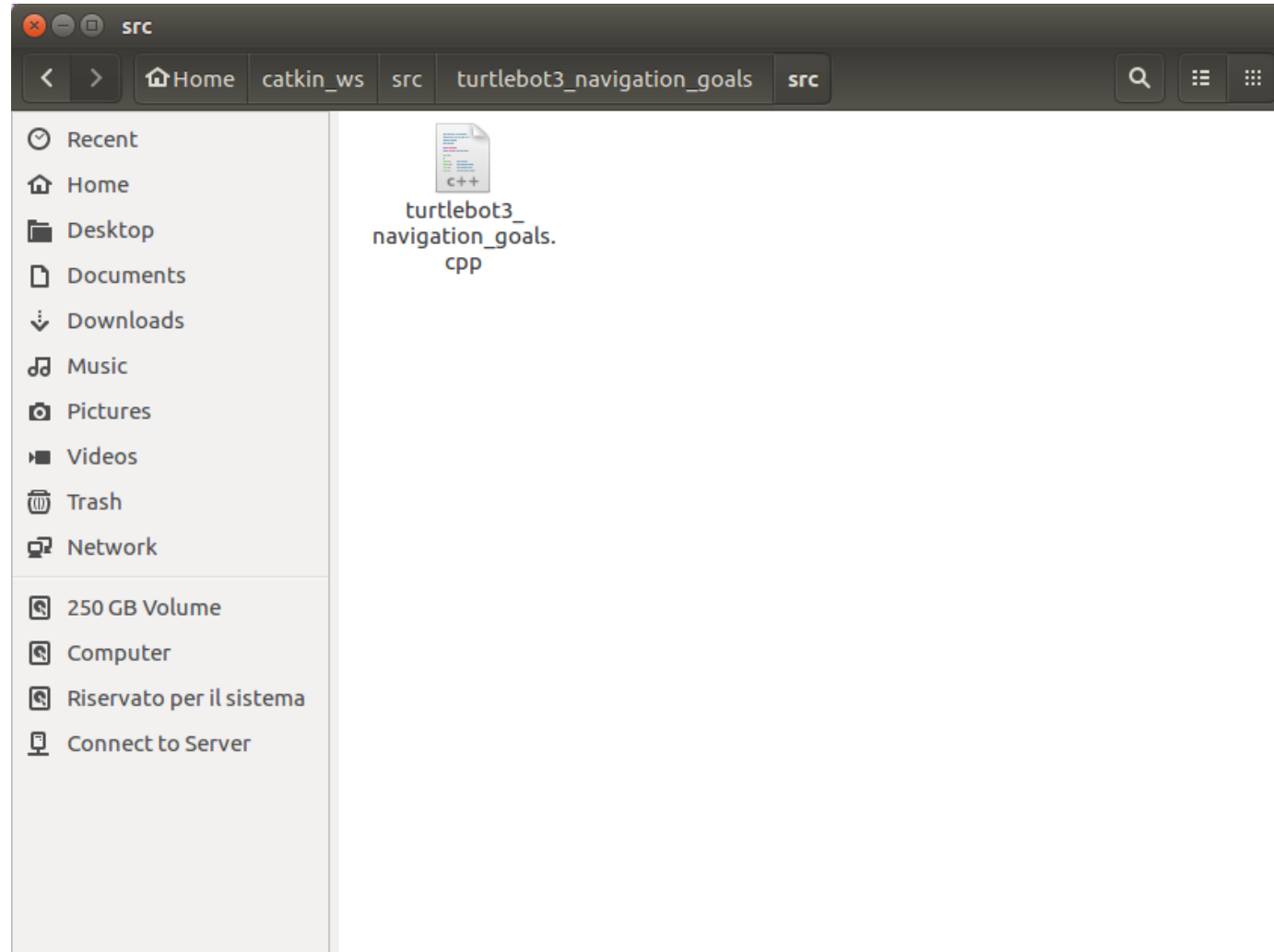
```
$ roscd turtlebot3_navigation_goals
```

```
bloisi@bloisi-U36SG:~/catkin_ws/src$ catkin_create_pkg turtlebot3_navigation_goal
ls move_base_msgs actionlib roscpp
Created file turtlebot3_navigation_goals/CMakeLists.txt
Created file turtlebot3_navigation_goals/package.xml
Created folder turtlebot3_navigation_goals/include/turtlebot3_navigation_goals
Created folder turtlebot3_navigation_goals/src
Successfully created files in /home/bloisi/catkin_ws/src/turtlebot3_navigation_goals.
Please adjust the values in package.xml.
bloisi@bloisi-U36SG:~/catkin_ws/src$ roscd turtlebot3_navigation_goals
```

Files in the package



src folder

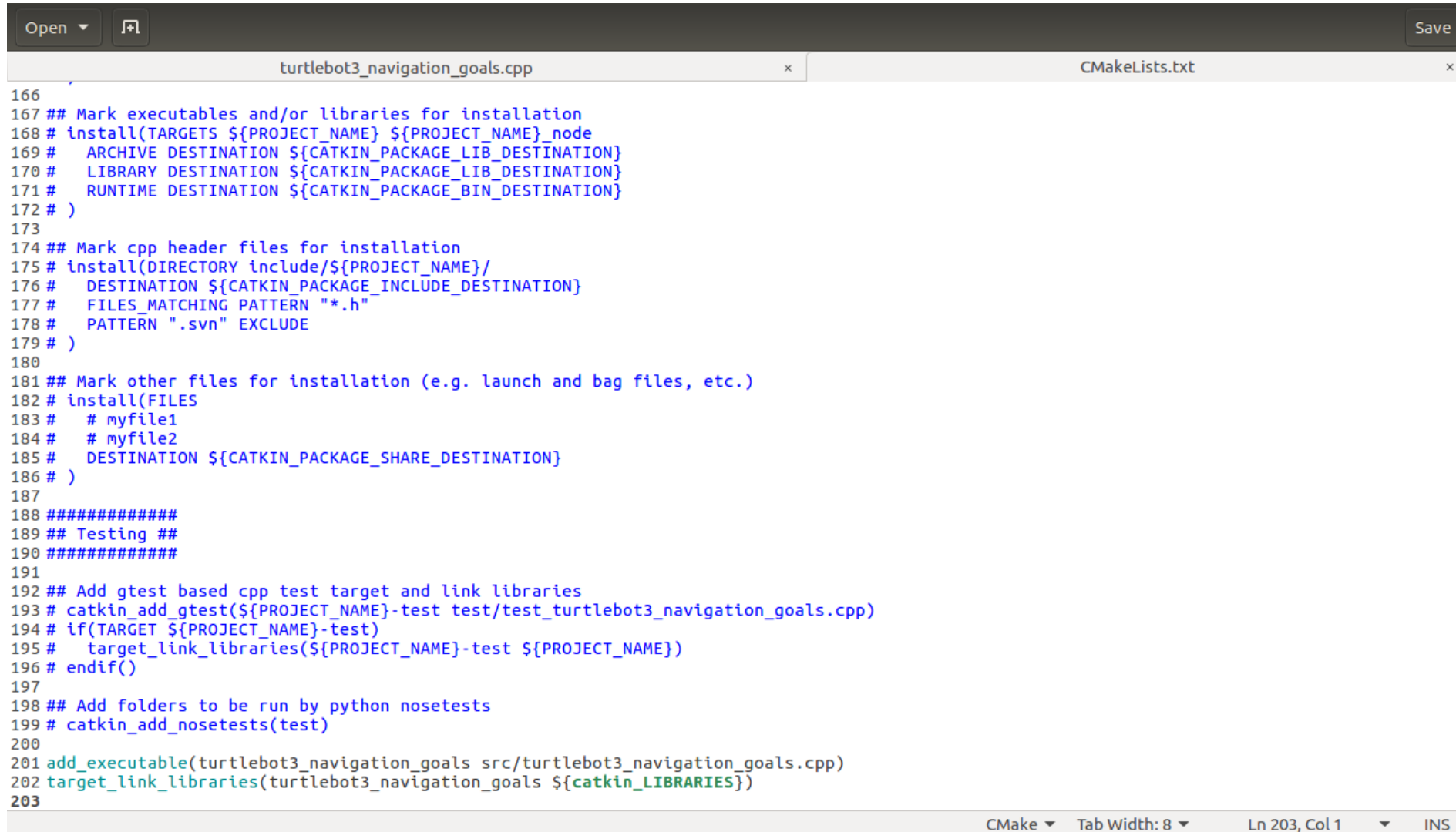


turtlebot3_navigation_goals.cpp

```
Open ▾ [+] Save
1 #include <ros/ros.h>
2 #include <move_base_msgs/MoveBaseAction.h>
3 #include <actionlib/client/simple_action_client.h>
4
5 typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;
6
7 int main(int argc, char** argv){
8   ros::init(argc, argv, "simple_navigation_goals");
9
10  //tell the action client that we want to spin a thread by default
11  MoveBaseClient ac("move_base", true);
12
13  //wait for the action server to come up
14  while(!ac.waitForServer(ros::Duration(5.0))){
15    ROS_INFO("Waiting for the move_base action server to come up");
16  }
17
18  move_base_msgs::MoveBaseGoal goal;
19
20  //we'll send a goal to the robot to move 1 meter forward
21  goal.target_pose.header.frame_id = "base_link";
22  goal.target_pose.header.stamp = ros::Time::now();
23
24  goal.target_pose.pose.position.x = 1.0;
25  goal.target_pose.pose.orientation.w = 1.0;
26
27  ROS_INFO("Sending goal");
28  ac.sendGoal(goal);
29
30  ac.waitForResult();
31
32  if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
33    ROS_INFO("Hooray, the base moved 1 meter forward");
34  else
35    ROS_INFO("The base failed to move forward 1 meter for some reason");
36
37  return 0;
38 }
39
```

C++ ▾ Tab Width: 8 ▾ Ln 39, Col 1 ▾ INS

CmakeLists.txt

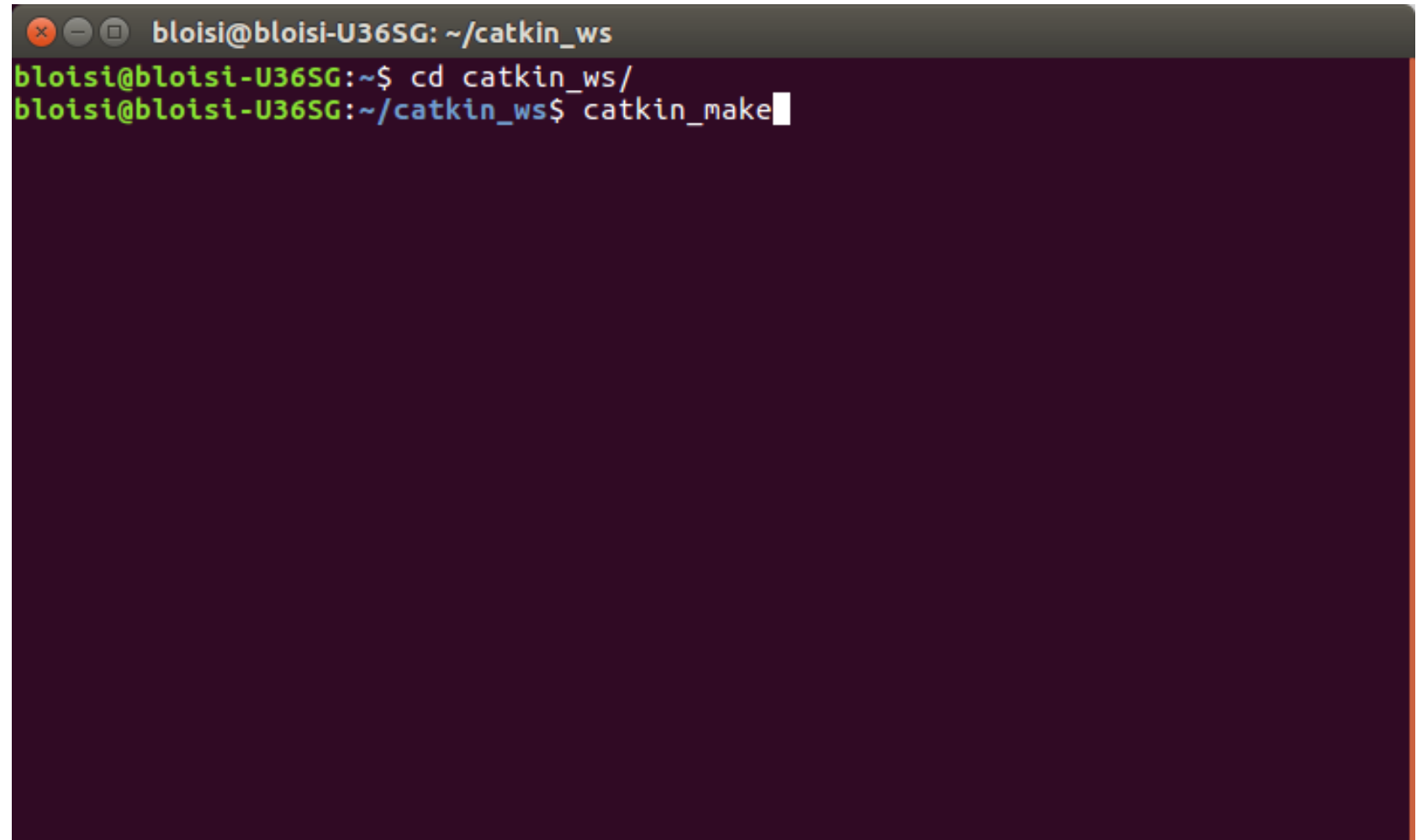


```
166
167 ## Mark executables and/or libraries for installation
168 # install(TARGETS ${PROJECT_NAME} ${PROJECT_NAME}_node
169 #   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
170 #   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
171 #   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
172 # )
173
174 ## Mark cpp header files for installation
175 # install(DIRECTORY include/${PROJECT_NAME}/
176 #   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
177 #   FILES_MATCHING PATTERN "*.h"
178 #   PATTERN ".svn" EXCLUDE
179 # )
180
181 ## Mark other files for installation (e.g. launch and bag files, etc.)
182 # install(FILES
183 #   # myfile1
184 #   # myfile2
185 #   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
186 # )
187
188 #####
189 ## Testing ##
190 #####
191
192 ## Add gtest based cpp test target and link libraries
193 # catkin_add_gtest(${PROJECT_NAME}-test test/test_turtlebot3_navigation_goals.cpp)
194 # if(TARGET ${PROJECT_NAME}-test)
195 #   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
196 # endif()
197
198 ## Add folders to be run by python nosetests
199 # catkin_add_nosetests(test)
200
201 add_executable(turtlebot3_navigation_goals src/turtlebot3_navigation_goals.cpp)
202 target_link_libraries(turtlebot3_navigation_goals ${catkin_LIBRARIES})
203
```

CMake ▾ Tab Width: 8 ▾ Ln 203, Col 1 ▾ INS

catkin_make

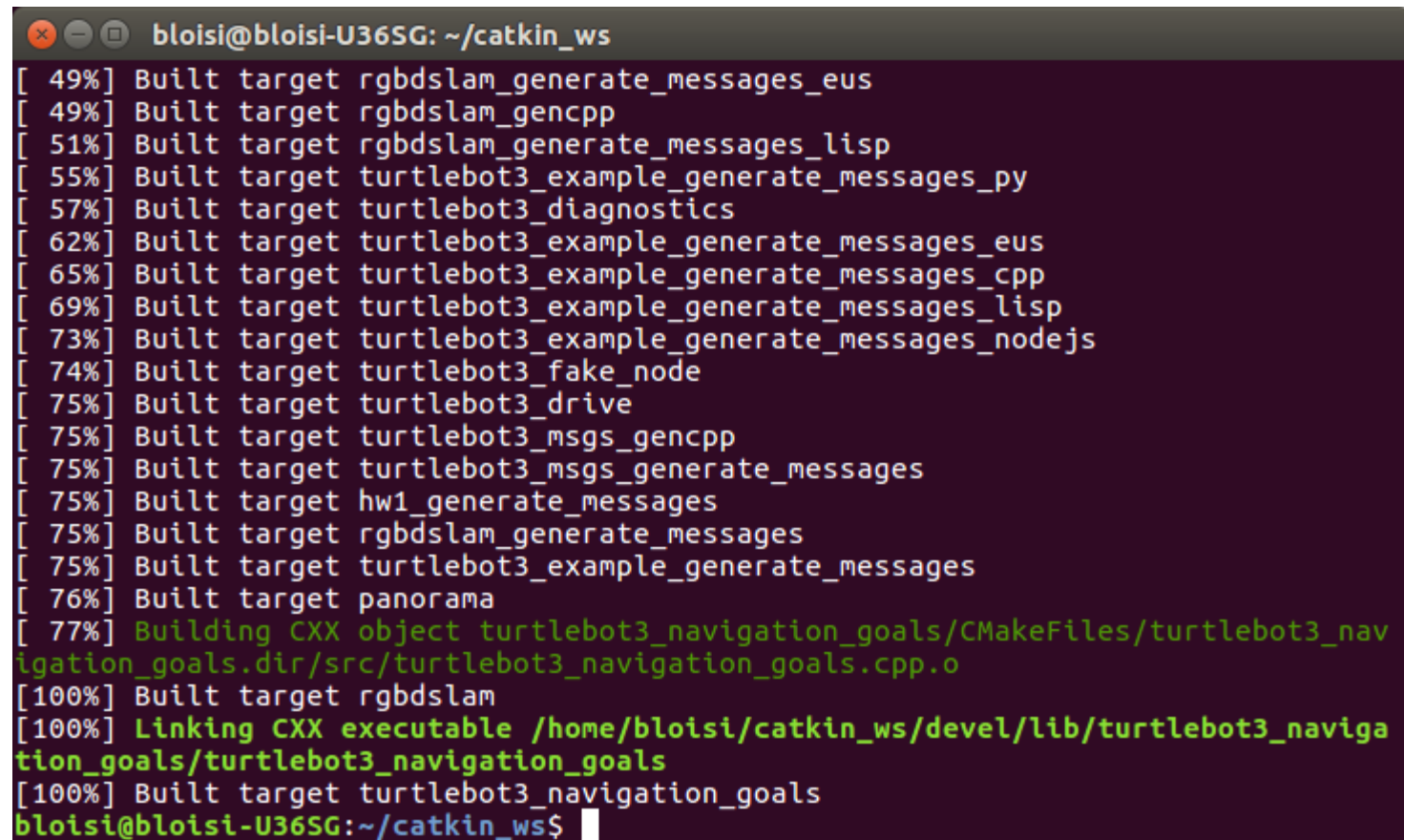
```
$ cd ~/catkin_ws  
$ catkin_make
```

A terminal window with a dark purple background and a grey title bar. The title bar contains the text "bloisi@bloisi-U36SG: ~/catkin_ws". The terminal shows two lines of command execution: "bloisi@bloisi-U36SG:~\$ cd catkin_ws/" and "bloisi@bloisi-U36SG:~/catkin_ws\$ catkin_make". The cursor is at the end of the second line.

```
bloisi@bloisi-U36SG: ~/catkin_ws  
bloisi@bloisi-U36SG:~$ cd catkin_ws/  
bloisi@bloisi-U36SG:~/catkin_ws$ catkin_make
```

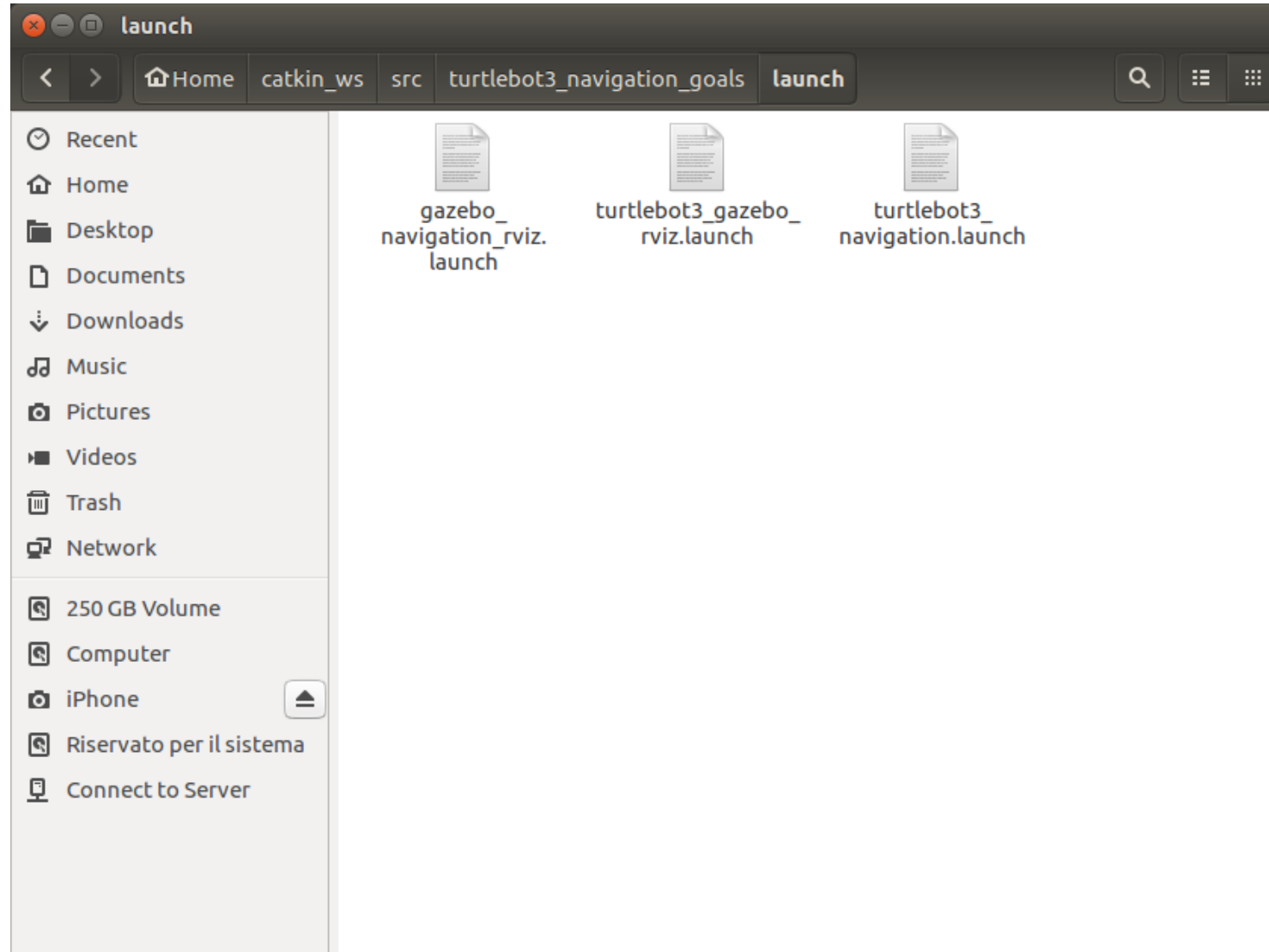

catkin_make - execution

```
$ cd ~/catkin_ws  
$ catkin_make
```



```
bloisi@bloisi-U36SG: ~/catkin_ws  
[ 49%] Built target rgbdslam_generate_messages_eus  
[ 49%] Built target rgbdslam_gencpp  
[ 51%] Built target rgbdslam_generate_messages_lisp  
[ 55%] Built target turtlebot3_example_generate_messages_py  
[ 57%] Built target turtlebot3_diagnostics  
[ 62%] Built target turtlebot3_example_generate_messages_eus  
[ 65%] Built target turtlebot3_example_generate_messages_cpp  
[ 69%] Built target turtlebot3_example_generate_messages_lisp  
[ 73%] Built target turtlebot3_example_generate_messages_nodejs  
[ 74%] Built target turtlebot3_fake_node  
[ 75%] Built target turtlebot3_drive  
[ 75%] Built target turtlebot3_msgs_gencpp  
[ 75%] Built target turtlebot3_msgs_generate_messages  
[ 75%] Built target hw1_generate_messages  
[ 75%] Built target rgbdslam_generate_messages  
[ 75%] Built target turtlebot3_example_generate_messages  
[ 76%] Built target panorama  
[ 77%] Building CXX object turtlebot3_navigation_goals/CMakeFiles/turtlebot3_navigation_goals.dir/src/turtlebot3_navigation_goals.cpp.o  
[100%] Built target rgbdslam  
[100%] Linking CXX executable /home/bloisi/catkin_ws/devel/lib/turtlebot3_navigation_goals/turtlebot3_navigation_goals  
[100%] Built target turtlebot3_navigation_goals  
bloisi@bloisi-U36SG:~/catkin_ws$
```

launch files



gazebo_navigation_rviz.launch

```
<launch>
```

```
  <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_world.launch"/>
```

```
  <include file="$(find turtlebot3_navigation_goals)/launch/turtlebot3_navigation.launch"/>
```

```
  <include file="$(find turtlebot3_navigation_goals)/launch/turtlebot3_gazebo_rviz.launch"/>
```

```
</launch>
```

turtlebot3_navigation.launch

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle]"/>

  <!-- Turtlebot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch" />

  <!-- Map server -->
  <arg name="map_file" default="$(find turtlebot3_navigation_goals)/config/map.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)">
  </node>

  <!-- AMCL -->
  <include file="$(find turtlebot3_navigation)/launch/amcl.launch"/>

  <!-- move_base -->
  <arg name="cmd_vel_topic" default="/cmd_vel" />
  <arg name="odom_topic" default="odom" />
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />

    <rosparam file="$(find turtlebot3_navigation)/param/costmap_common_params_$(arg model).yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find turtlebot3_navigation)/param/costmap_common_params_$(arg model).yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find turtlebot3_navigation)/param/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find turtlebot3_navigation)/param/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find turtlebot3_navigation)/param/move_base_params.yaml" command="load" />
    <rosparam file="$(find turtlebot3_navigation)/param/dwa_local_planner_params_waffle.yaml" command="load" />

    <remap from="cmd_vel" to="$(arg cmd_vel_topic)"/>
    <remap from="odom" to="$(arg odom_topic)"/>
  </node>
</launch>
```

turtlebot3_gazebo_rviz.launch

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle]"/>

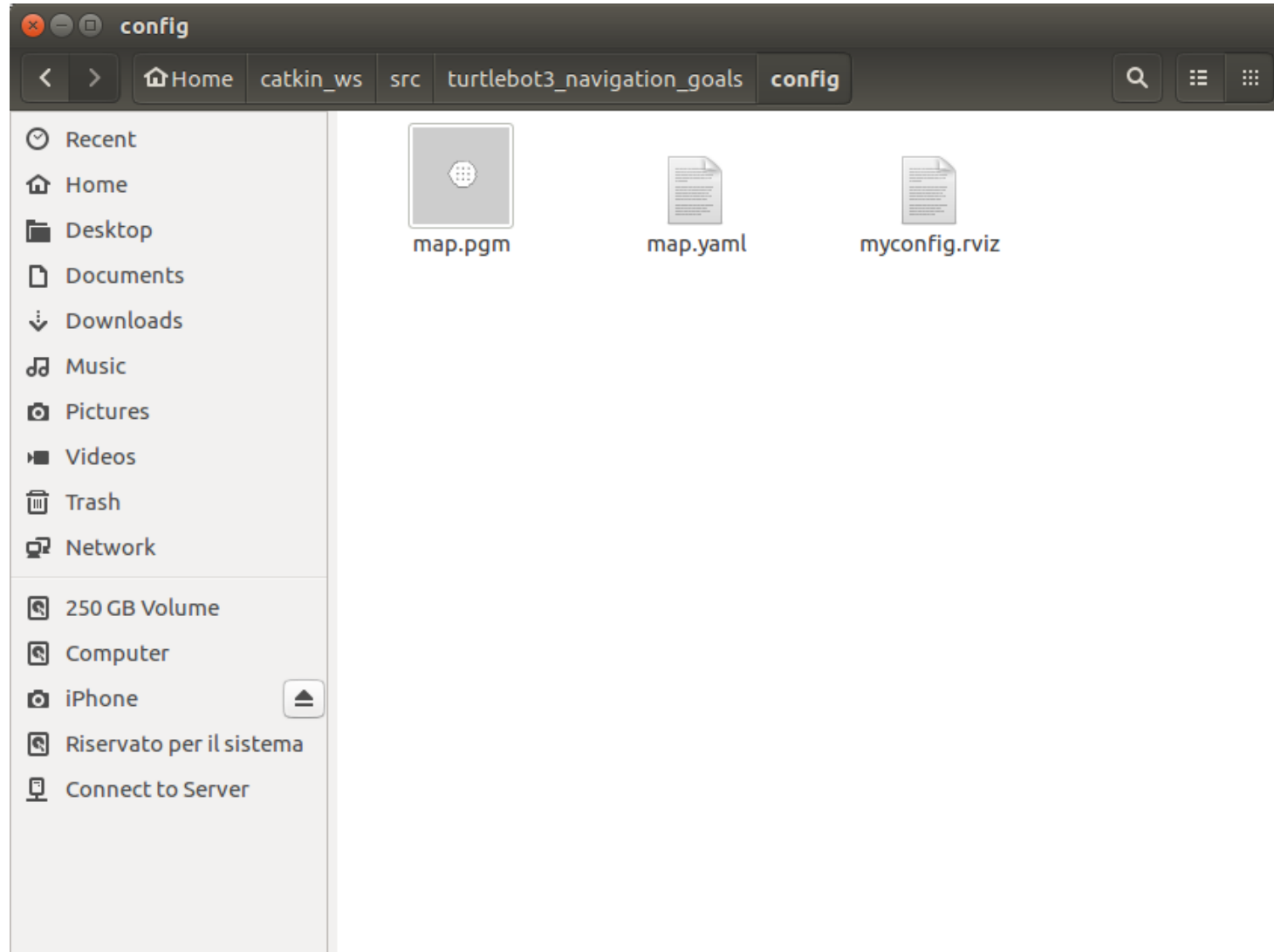
  <include file="$(find turtlebot3_bringup)/launch/includes/description.launch.xml">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- Commented out this node since it is already launched in the navigation launch file or in included files>
  <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher" output="screen">
    <param name="publish_frequency" type="double" value="50.0" />
  </node> -->

  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find turtlebot3_navigation_goals)/config/myconfig.rviz"/>

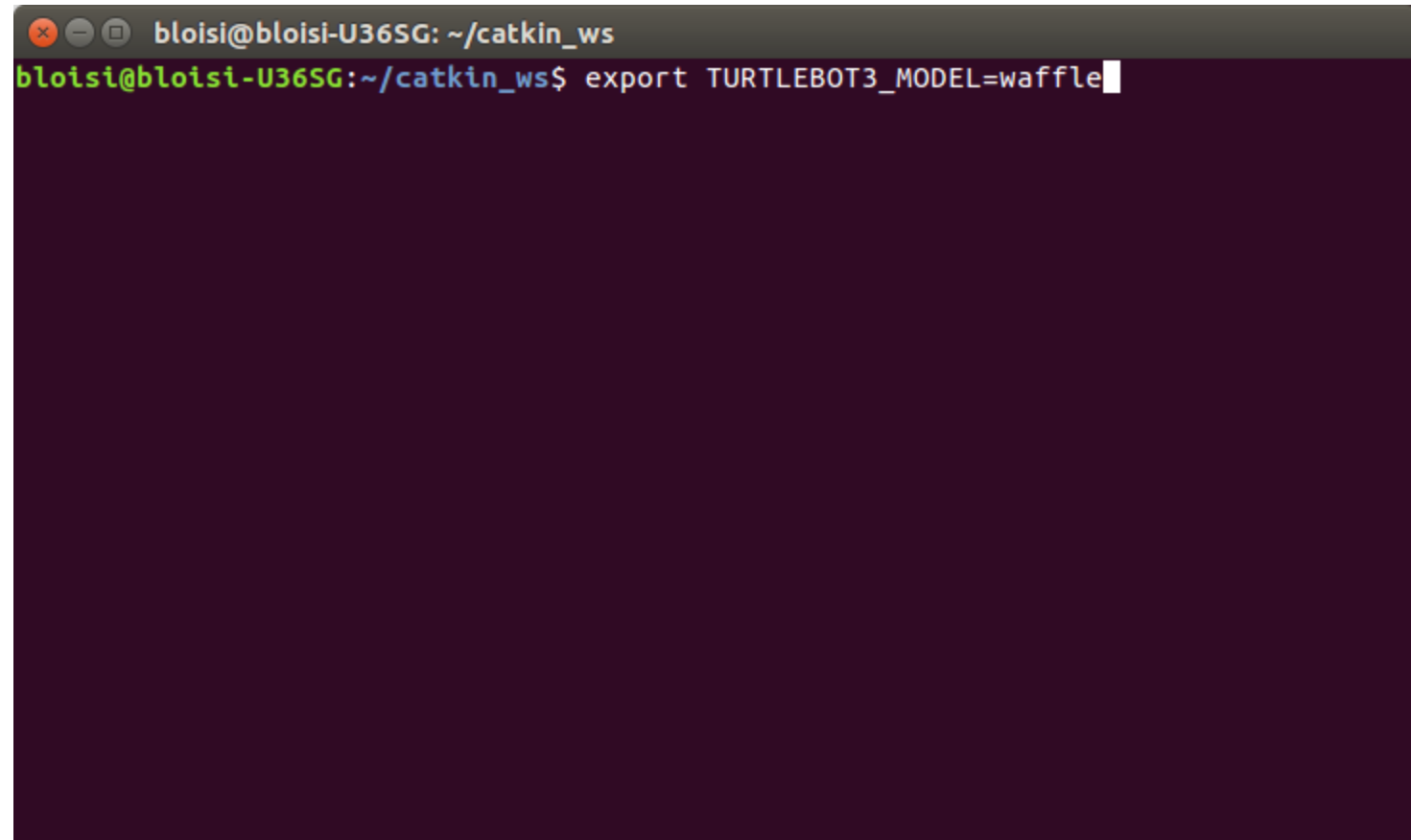
</launch>
```

config folder



Setting Turtlebot3 model

```
$ export TURTLEBOT3_MODEL=waffle
```

A terminal window with a dark purple background. The title bar shows 'bloisi@bloisi-U36SG: ~/catkin_ws'. The prompt 'bloisi@bloisi-U36SG:~/catkin_ws\$' is followed by the command 'export TURTLEBOT3_MODEL=waffle' and a white cursor. The rest of the terminal is empty.

```
bloisi@bloisi-U36SG: ~/catkin_ws
bloisi@bloisi-U36SG:~/catkin_ws$ export TURTLEBOT3_MODEL=waffle
```

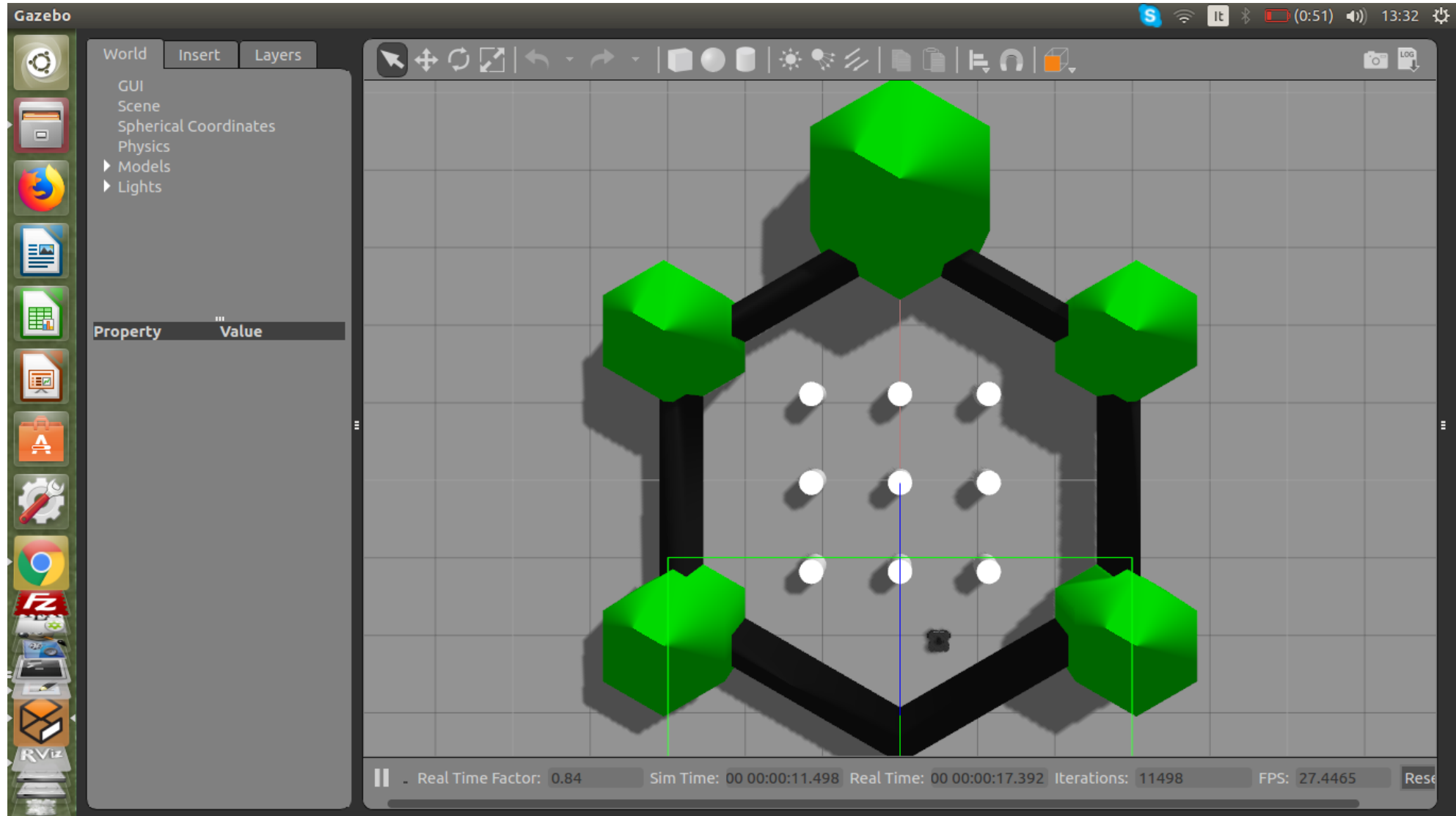
gazebo_navigation_rviz.launch

```
$ roslaunch turtlebot3_navigation_goals gazebo_navigation_rviz.launch
```

A terminal window with a dark purple background and a grey title bar. The title bar contains the text 'bloisi@bloisi-U36SG: ~/catkin_ws'. The terminal shows two lines of command input: 'bloisi@bloisi-U36SG:~/catkin_ws\$ export TURTLEBOT3_MODEL=waffle' and 'bloisi@bloisi-U36SG:~/catkin_ws\$ roslaunch turtlebot3_navigation_goals gazebo_navigation_rviz.launch'. A white cursor is visible at the end of the second line.

```
bloisi@bloisi-U36SG: ~/catkin_ws
bloisi@bloisi-U36SG:~/catkin_ws$ export TURTLEBOT3_MODEL=waffle
bloisi@bloisi-U36SG:~/catkin_ws$ roslaunch turtlebot3_navigation_goals gazebo_navigation_rviz.launch
```


Gazebo



RViz

The screenshot displays the RViz (Robot Visualization) interface. The central 3D view shows a robot's field of view (FOV) as a blue-shaded area on a grid. Inside the FOV, there are several red and yellow markers, likely representing detected objects or targets. The interface is divided into several panels:

- Displays:** A list of visual elements and their properties. It includes:
 - Global Options:** Fixed Frame (map), Background Color (48; 48; 48), Frame Rate (30), Default Light (checked).
 - Global Status: Ok**
 - Grid:** Status: Ok, Reference Frame (<Fixed Frame>), Plane Cell Count (10), Normal Cell Count (0), Cell Size (1), Line Style (Lines), Color (160; 160; 164), Alpha (0,5), Plane (XY), Offset (0; 0; 0).
 - Axes:** Status: Ok, Reference Frame (<Fixed Frame>), Length (0,1), Radius (0,01).
 - LaserScan:** (checked).
- Views:** A panel for configuring the current view. It shows:
 - Type: XYOrbit (rviz)
 - Current View: XYOrbit (rviz)
 - Parameters: Near Clip (0,01), Invert Z Axis (unchecked), Target Fra... (<Fixed Frame>), Distance (11,4363), Focal Shap... (0,05), Focal Shap... (checked), Yaw (3,1504), Pitch (1,3104), Focal Point (9.5367e-07; -4.7...).
- Time:** A panel at the bottom showing system metrics:
 - ROS Time: 34.74
 - ROS Elapsed: 32.82
 - Wall Time: 1527766385.80
 - Wall Elapsed: 46.46
 - Experimental (unchecked)

At the bottom, there is a control bar with a "Reset" button and a keyboard shortcut guide: "Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Move Z. Shift: More options." The current frame rate is shown as "8 fps".

Pose estimation

The screenshot displays a ROS2 visualization interface with the following components:

- Top Toolbar:** Includes buttons for Interact, Move Camera, Select, Focus Camera, Measure, **2D Pose Estimate** (circled in red), 2D Nav Goal, and Publish Point.
- Displays Panel (Left):** Contains settings for Global Options, Global Status: Ok, Grid, Axes, and LaserScan.
- Views Panel (Right):** Shows the current view as XYOrbit (rviz) with various camera parameters.
- Time Panel (Bottom):** Displays ROS Time (56.53), ROS Elapsed (54.62), Wall Time (1527766420.83), and Wall Elapsed (81.55).

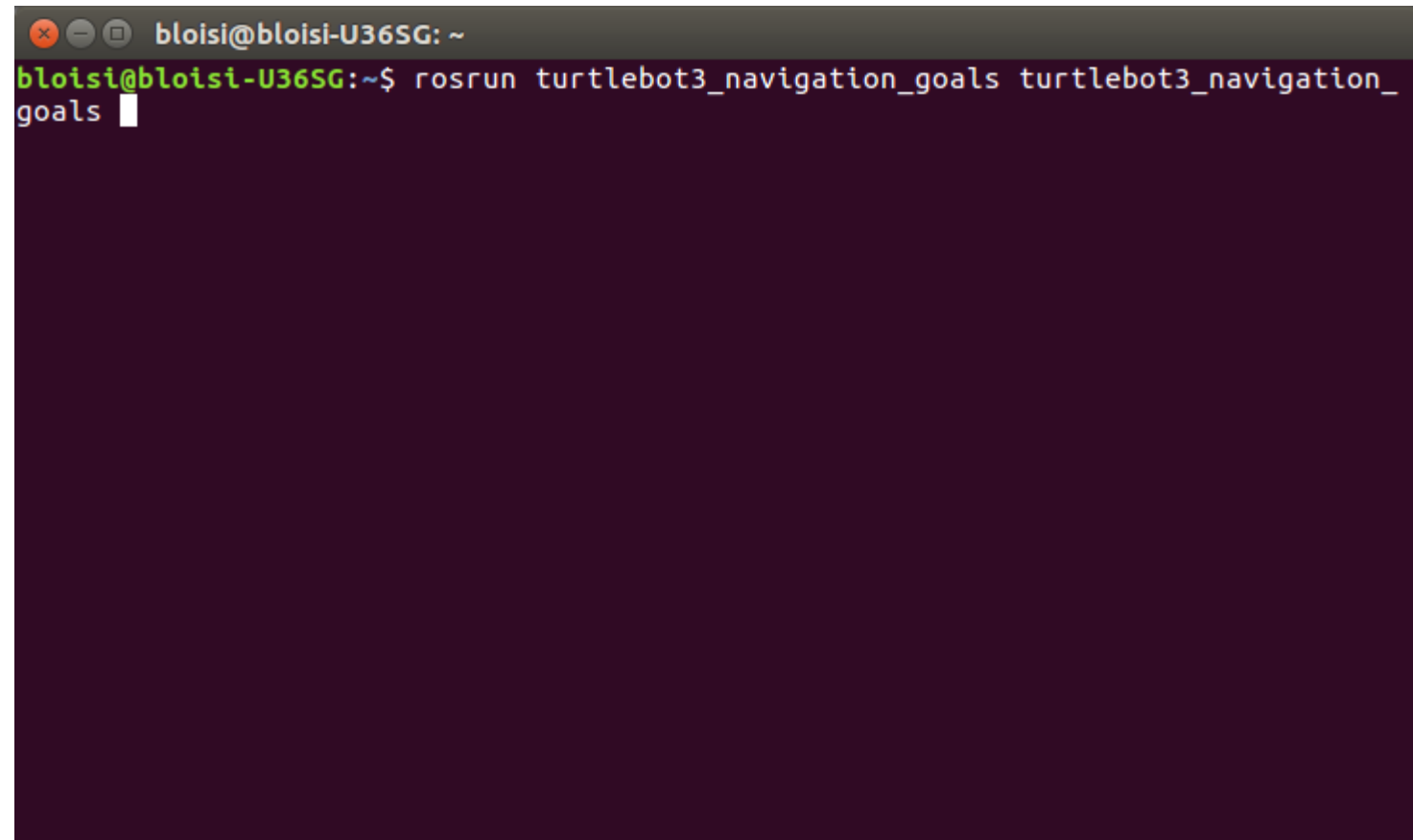
The central 3D view shows a robot (a small blue cube) on a grid. A green arrow points upwards from the robot, indicating its current pose. The robot is surrounded by a large, irregularly shaped area with a color gradient from blue to red, representing a 2D pose estimate or a sensor's field of view.

Parameter	Value
Type	XYOrbit (rviz)
Current View	XYOrbit (rviz)
Near Clip ...	0,01
Invert Z Axis	<input type="checkbox"/>
Target Fra...	<Fixed Frame>
Distance	11,2146
Focal Shap...	0,05
Focal Shap...	<input checked="" type="checkbox"/>
Yaw	3,21194
Pitch	1,40443
Focal Point	-1.8483; 0.04241; 0

Time Metric	Value
ROS Time	56.53
ROS Elapsed	54.62
Wall Time	1527766420.83
Wall Elapsed	81.55

Sending the goal

```
$ rosrun turtlebot3_navigation_goals turtlebot3_navigation_goals
```

A terminal window with a dark purple background. The title bar at the top reads "bloisi@bloisi-U36SG: ~". The prompt "bloisi@bloisi-U36SG:~" is shown in green. The command "roslaunch turtlebot3_navigation_goals turtlebot3_navigation_goals" is entered in white text, with a white cursor at the end of the second line.

```
bloisi@bloisi-U36SG:~$ roslaunch turtlebot3_navigation_goals turtlebot3_navigation_goals
```

Reaching the goal

The screenshot displays a ROS visualization interface with a central 2D map. The map shows a robot (a small green circle) and a goal (a red circle) on a grid. The robot is positioned near the center of the map, and the goal is located to its right. The map is overlaid on a blue grid. The interface includes several panels and a toolbar.

Toolbar: Interact, Move Camera, Select, Focus Camera, Measure, 2D Pose Estimate, 2D Nav Goal, Publish Point, +, -

Displays Panel:

- Global Options
 - Fixed Frame: map
 - Background Color: 48; 48; 48
 - Frame Rate: 30
 - Default Light:
- Global Status: Ok
 - Fixed Frame: OK
- Grid
 - Status: Ok
 - Reference Frame: <Fixed Frame>
 - Plane Cell Count: 10
 - Normal Cell Count: 0
 - Cell Size: 1
 - Line Style: Lines
 - Color: 160; 160; 164
 - Alpha: 0,5
 - Plane: XY
 - Offset: 0; 0; 0
- Axes
 - Status: Ok
 - Reference Frame: <Fixed Frame>
 - Length: 0,1
 - Radius: 0,01
- LaserScan
 -

Views Panel:

Type: XYOrbit (rviz) Zero

Current View	XYOrbit (rviz)
Near Clip ...	0,01
Invert Z Axis	<input type="checkbox"/>
Target Fra...	<Fixed Frame>
Distance	11,2146
Focal Shap...	0,05
Focal Shap...	<input checked="" type="checkbox"/>
Yaw	3,21194
Pitch	1,40443
Focal Point	-1.8483; 0.04241; 0

Time Panel:

ROS Time: 156.88 ROS Elapsed: 154.96 Wall Time: 1527766628.73 Wall Elapsed: 289.34 Experimental

Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Move Z. Shift: More options. 7 fps

Goal reached

```
bloisi@bloisi-U36SG:~/catkin_ws/src/turtlebot3_navigation_goals/launch$ rosrun t
urtlebot3_navigation_goals turtlebot3_navigation_goals
[ INFO] [1528132840.163758075, 71.877000000]: Sending goal
[ INFO] [1528132840.164147895, 71.877000000]: Goal sent, waiting for results
[ INFO] [1528132877.513702526, 87.077000000]: Hooray, the base moved 1 meter for
ward
bloisi@bloisi-U36SG:~/catkin_ws/src/turtlebot3_navigation_goals/launch$
```

Git repo

https://github.com/dbloisi/turtlebot3_navigation_goals.git

The screenshot shows the GitHub interface for the repository 'dbloisi / turtlebot3_navigation_goals'. The repository is described as a 'ROS package for sending navigation goals to the Turtlebot3 robot'. It has 2 commits, 1 branch, 0 releases, and 1 contributor. The 'master' branch is selected. A table of files and their commit history is visible:

File	Commit	Time
config	first commit	4 hours ago
launch	first commit	4 hours ago
src	first commit	4 hours ago
CMakeLists.txt	first commit	4 hours ago
README.md	Initial commit	4 hours ago
package.xml	first commit	4 hours ago

The bottom of the screenshot shows a taskbar with two 'ActionLib.pptx' files open and a 'Show all' button.

Esercizio 1

Modificare il codice del package `turtlebot3_navigation_goals` in modo che siano inviati più task al robot invece di un singolo task

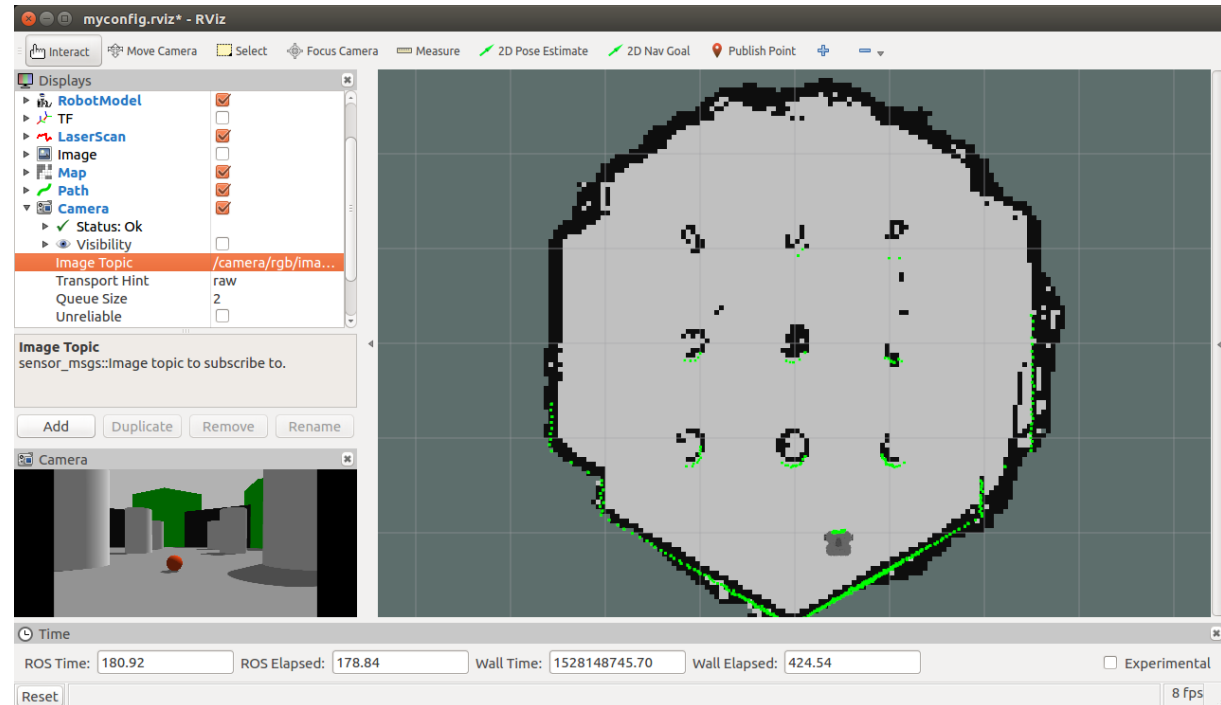
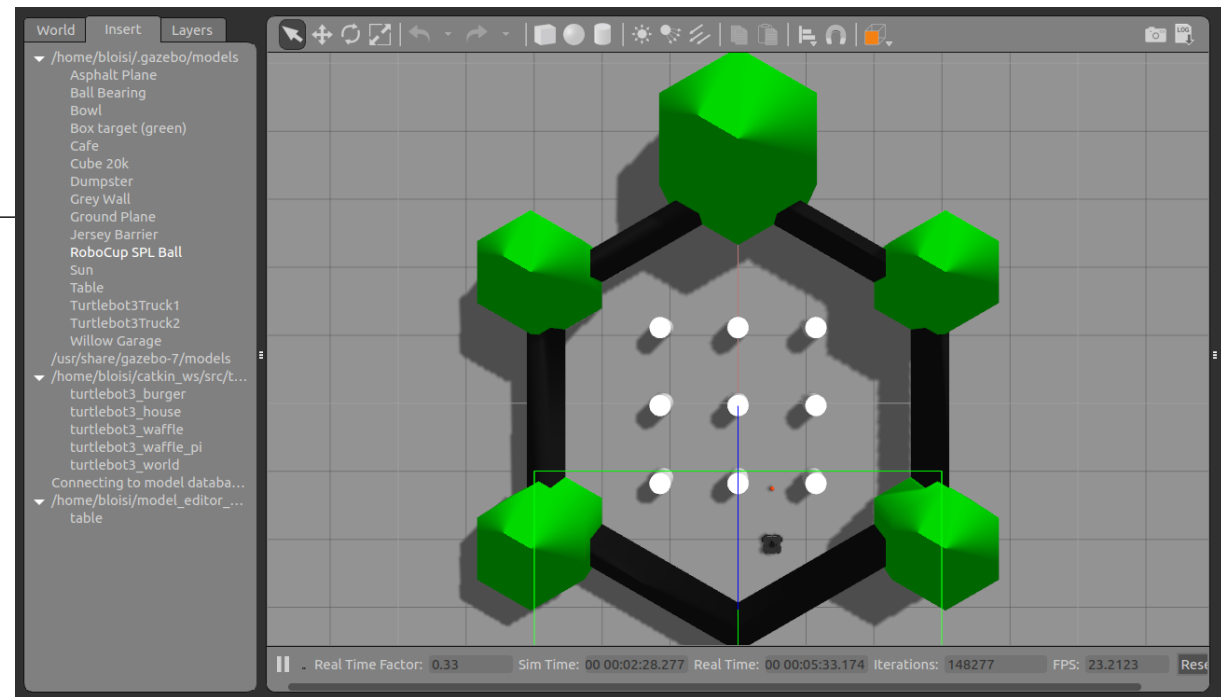
Esercizio 2

Eseguire il package
turtlebot3_navigation_goal utilizzando un
mondo virtuale Gazebo diverso da quello
TurtleBot3 world

Esercizio 2

Scrivere un nuovo package denominato `turtlebot3_visual_goal` in grado di

1. individuare la RoboCup SPL ball nella scena attraverso la camera del robot
2. utilizzare la posizione della palla come goal facendo navigare il robot verso di essa





UNIVERSITÀ
di **VERONA**

Dipartimento
di **INFORMATICA**

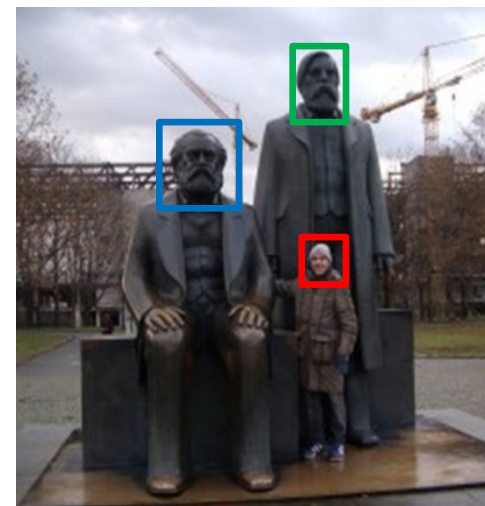
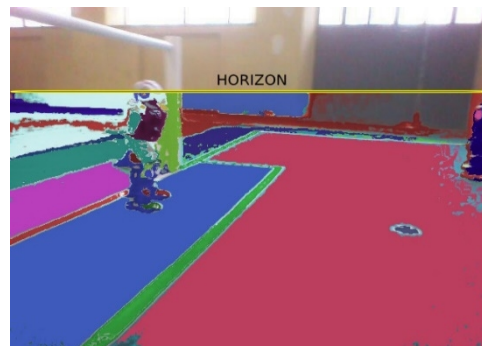
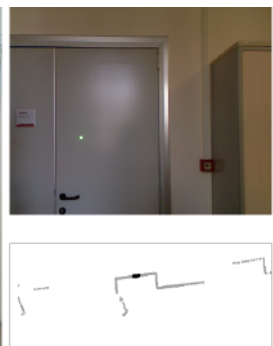
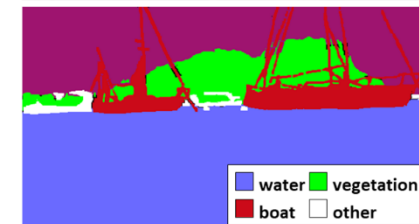
*Corso di Laboratorio Ciberfisico
Modulo di Robot Programming with ROS*



actionlib

ROS

Docente:
**Domenico Daniele
Bloisi**



Giugno 2018