Probabilistic Robotics Lab 1
Introduction to Turtlebot

Rodrigo Caye Daudt

March 8, 2016

1 Introduction

This report describes the work done for lab 1 assignment for the Probabilistic Robotics class. The main objectives of this assignment were to learn how to connect and send commands to the turtlebots, generate maps for navigation automatically, and to program a python script to control the turtlebot's movement.

2 Connexion to turtlebots and Gazebo simulator

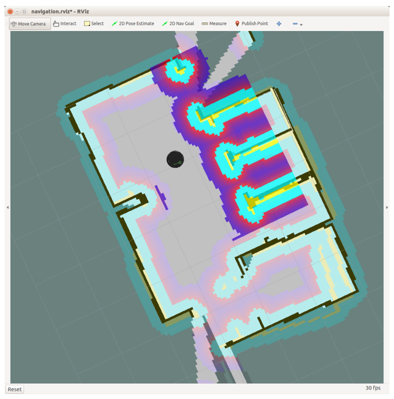
The first part of this lab consisted on configuring the computers to connect to the turtlebots. In the two hours available, it was possible to connect to the turtlebot and use the teleoperation node, but we didn't achieve the mapping or drive to goal tasks on the lab robots.

To accomplish the remaining tasks the computer was reconfigured to run all the operations locally on a simulated turtlebot using the Gazebo simulator. This simulator provides a physical model of the turtlebot with all its working components and can be used for simulations of the turtlebot in many different settings. On the simulator the SLAM node was tested and the "drive to goal" controller was developed.

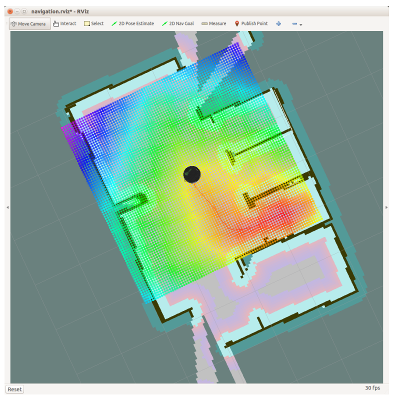
3 Teleoperation, mapping, and navigation

The first task was to control the movement of the turtlebot using teleoperation from the computer's keyboard. This was achieved using a standard turtlebot package called *turtlebot_teleop keyboard_teleop.launch*. This allowed us to move the turtlebot (real or simulated) as desired, which was useful for the mapping step.

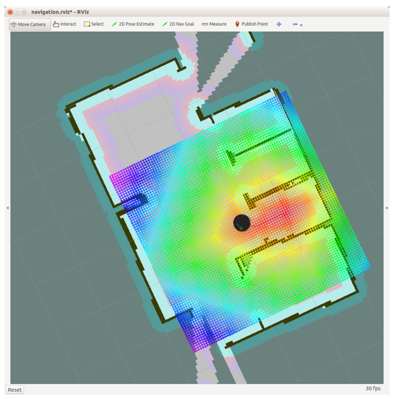
Once we had control over the turtlebot's movement, our next task was to generate a small map of the environment where the robot was located. Using the node *turtlebot_navigation gmapping_demo.launch* in conjunction with the



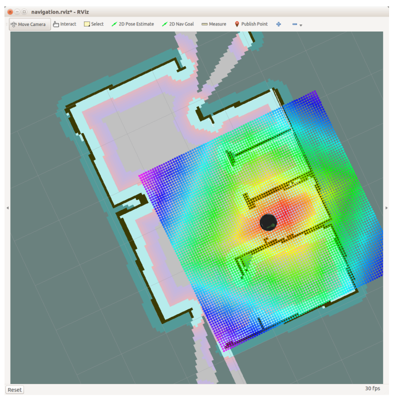
(a) RViz visualisation of map generated by *slam_gmapping* node



(b) Beginning of navigation



(c) Middle of navigation



(d) End of navigation

Figure 1: Movement of turtlebot towards desired goal controlled by *turtlebot_navigation*

teleoperation node we were able to create a map of a small section of the environment. This map was visualised in real time on RViz running on the node *turtlebot_rviz_launchers view_navigation.launch*. An image captured during the process of building the map can be seen in Fig. 1(a).

Once a map had been generated and saved, we used the node *turtlebot_navigation amcl_demo.launch* to navigate the map automatically to a chosen goal. Using RViz we were able to select a point on the map as the goal for the turtlebot, and its trajectory was automatically calculated. Figures 1(b)-1(d) show three different stages of the turtlebot on its way to the goal. The behaviour of the turtlebot during this process was somewhat erratic, but it reached the goal without any problems.

4 Turtlebot driver

The next task was to program a node to control the movement of the turtlebot to move it towards a goal. This can be divided in two main parts: the communication to the turtlebot, and the calculation of the control signals.

Two topics were essential in this operation. The first one was */odom*, whose messages contain information about the turtlebot's position and orientation. The messages published on this node are of type *nav_msgs/Odometry*. A conversion from quaternion orientation was needed for our calculations, and this transformation was done using *tf.transformations.euler_from_quaternion*. The other relevant topic was */cmd_vel_mux/input/navi*, which was used to send messages of type *geometry_msgs/Twist* to the turtlebot containing the desired linear and angular velocities.

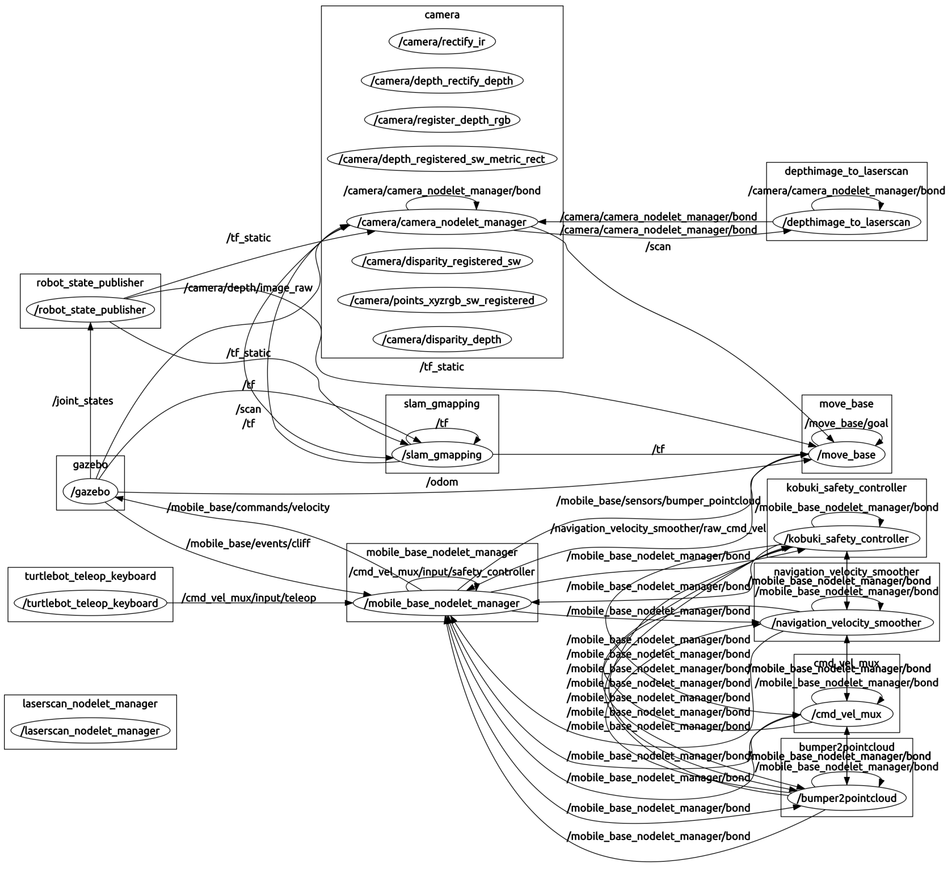
The velocity signals were calculated using a PD controller with some extra non-linear additions. This provided us with a simple and robust solution that could take the robot to any desired goal as long as no objects were on the way. The robot would first go to the desired coordinates, and after it arrived it would turn to the desired orientation. The non-linear additions to the operations served the purpose of decreasing the time the turtlebot took to reach the goal and to improve its stability. For example, the control signals were clipped at some maximum values to avoid unexpected behaviour or the robot, since it was observed that the robot would behave very strangely for large control signals.

5 Difficulties during development

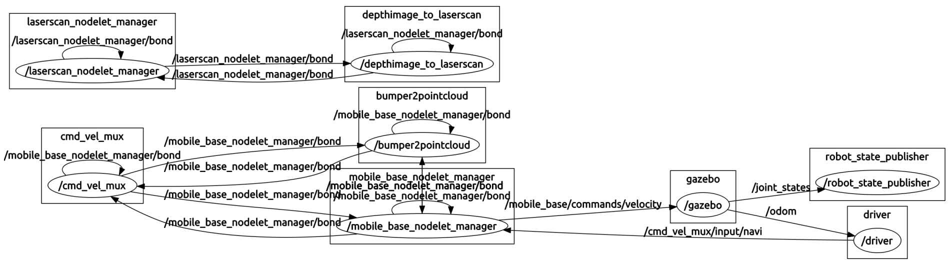
Many configuration and networking issues were faced during the development of this project. Also, at first it was not clear that the reason that the turtlebot was behaving strangely was the large control signals that were being sent. This led to suspecting something was wrong with the code logic, since it was not expected to see the robot spinning indefinitely after a short strong signal was applied. Finally, the documentation for ROS is not very helpful in many cases.

6 Conclusion

In this session we understood the basics of how to control the turtlebot using ROS. The system was tested with a real turtlebot and with a simulated one using Gazebo. Many turtlebot packages were used to achieve teleoperation, mapping and automatic navigation. A ROS package was developed to guide the turtlebot to a list of chosen goals.


camera
/camera/rectify_ir
/camera/depth_rectify_depth
/camera/register_depth_rgb
/camera/depth_registered_sw_metric_rect
/camera/camera_nodelet_manager/bond
/camera/camera_nodelet_manager
/camera/disparity_registered_sw
/camera/points_xyzrgb_sw_registered
/camera/disparity_depth
depthimage_to_laserscan
/depthimage_to_laserscan
/scan
/tf_static
robot_state_publisher
/robot_state_publisher
/camera/depth/image_raw
/tf
/joint_states
slam_gmapping
/slam_gmapping
move_base
/move_base/goal
/move_base
gazebo
/gazebo
/odom
/mobile_base/commands/velocity
/mobile_base/events/cliff
/mobile_base/sensors/bumper_pointcloud
/navigation_velocity_smoother/raw_cmd_vel
kobuki_safety_controller
/kobuki_safety_controller
mobile_base_nodelet_manager
/cmd_vel_mux/input/safety_controller
/mobile_base_nodelet_manager
/mobile_base_nodelet_manager/bond
turtlebot_teleop_keyboard
/turtlebot_teleop_keyboard
/cmd_vel_mux/input/teleop
navigation_velocity_smoother
/navigation_velocity_smoother
cmd_vel_mux
/cmd_vel_mux
laserscan_nodelet_manager
/laserscan_nodelet_manager
bumper2pointcloud
/bumper2pointcloud

(a) rosgraph when performing SLAM



(b) rosgraph when driving to goal

Figure 2: Graphs obtained using rqt_graph