Assignment 6 ROS

Rafal Grasman

**Manual**

The goal in this assignment is to learn using the actionlib from ROS.

To use the package:

1. Extract the package into your catkin workspace
2. Follow the preparation steps of Assignment6.docx
3. Follow <http://wiki.ros.org/catkin/Tutorials/using_a_workspace> to build the package
4. roslaunch assignment6 assignment6.launch

Then use in another terminal that also has access to the catkin workspace:

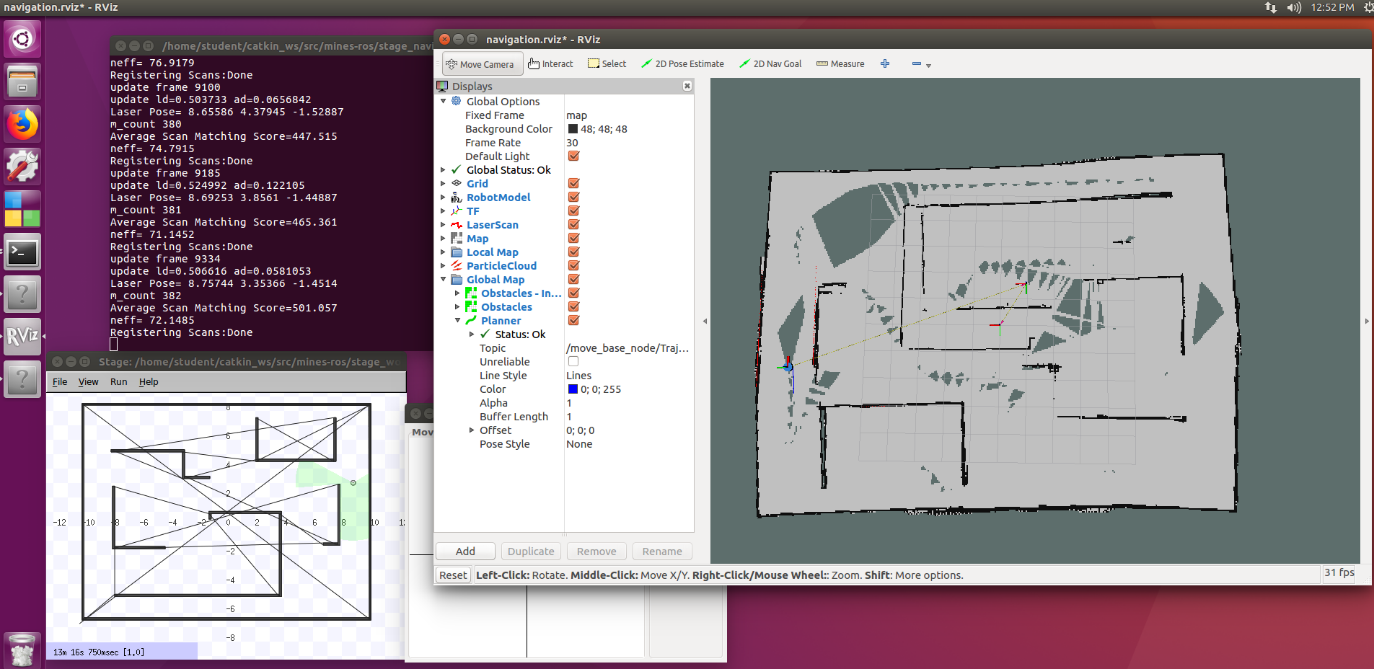
**…**

To <…>

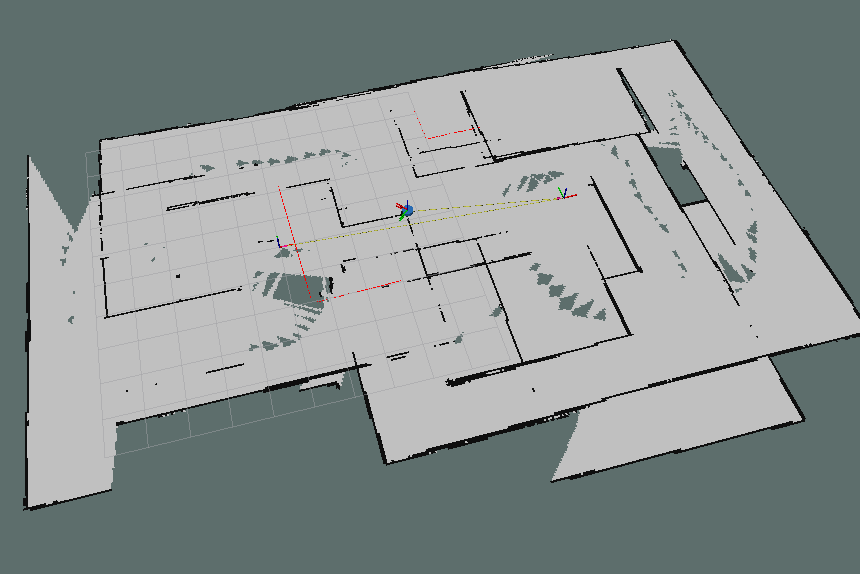
**Documentation**

**2.1**

First the map was generated with a laser robot (but forgot to save the map file.., generated a new one, but it was a bit worse):



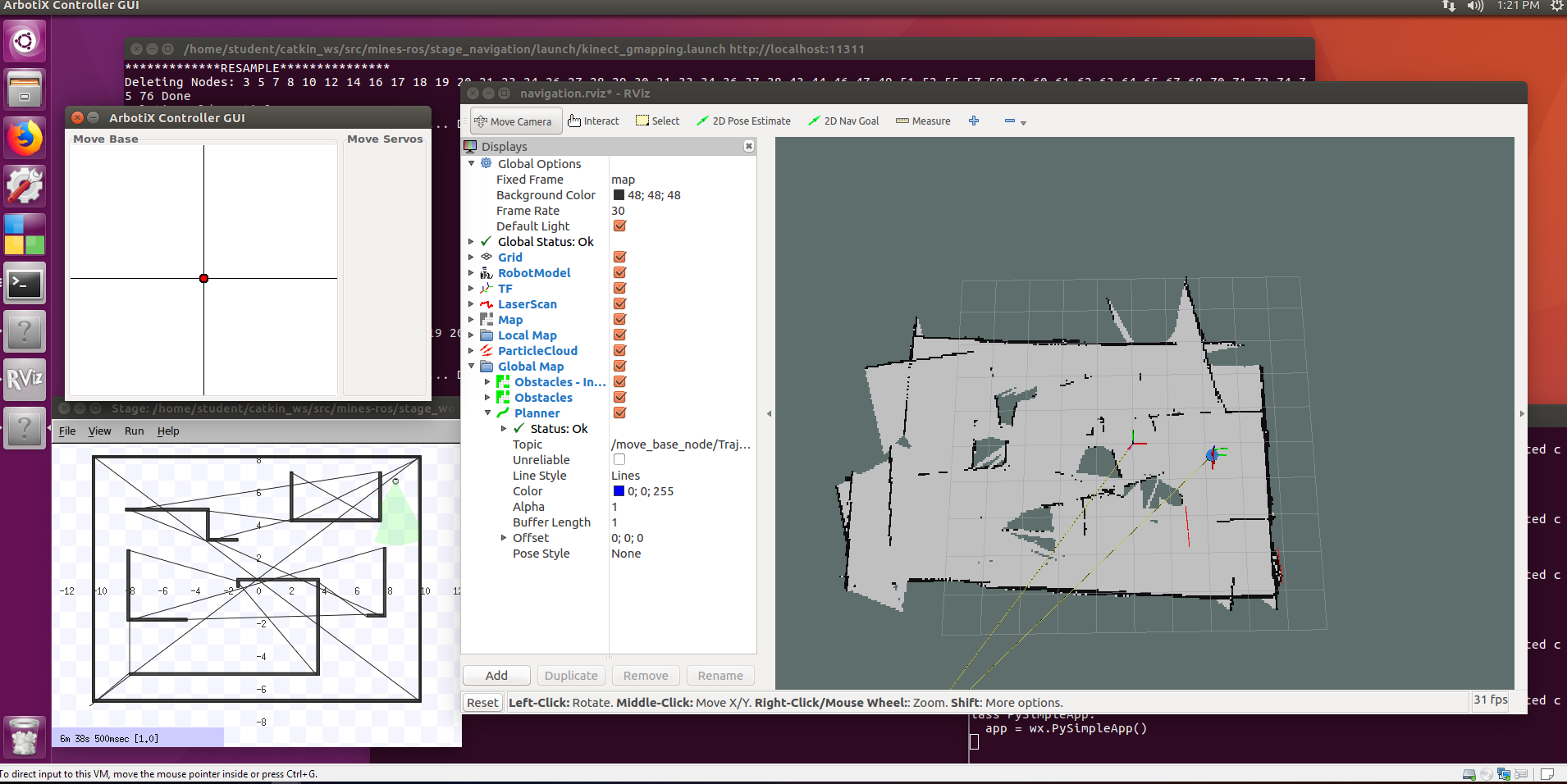
It seems that if the map is generated by driving “faster” with the joystick control it’s less accurate:



What makes is much harder is that the robot “skips” and “jumps” and “lags”.

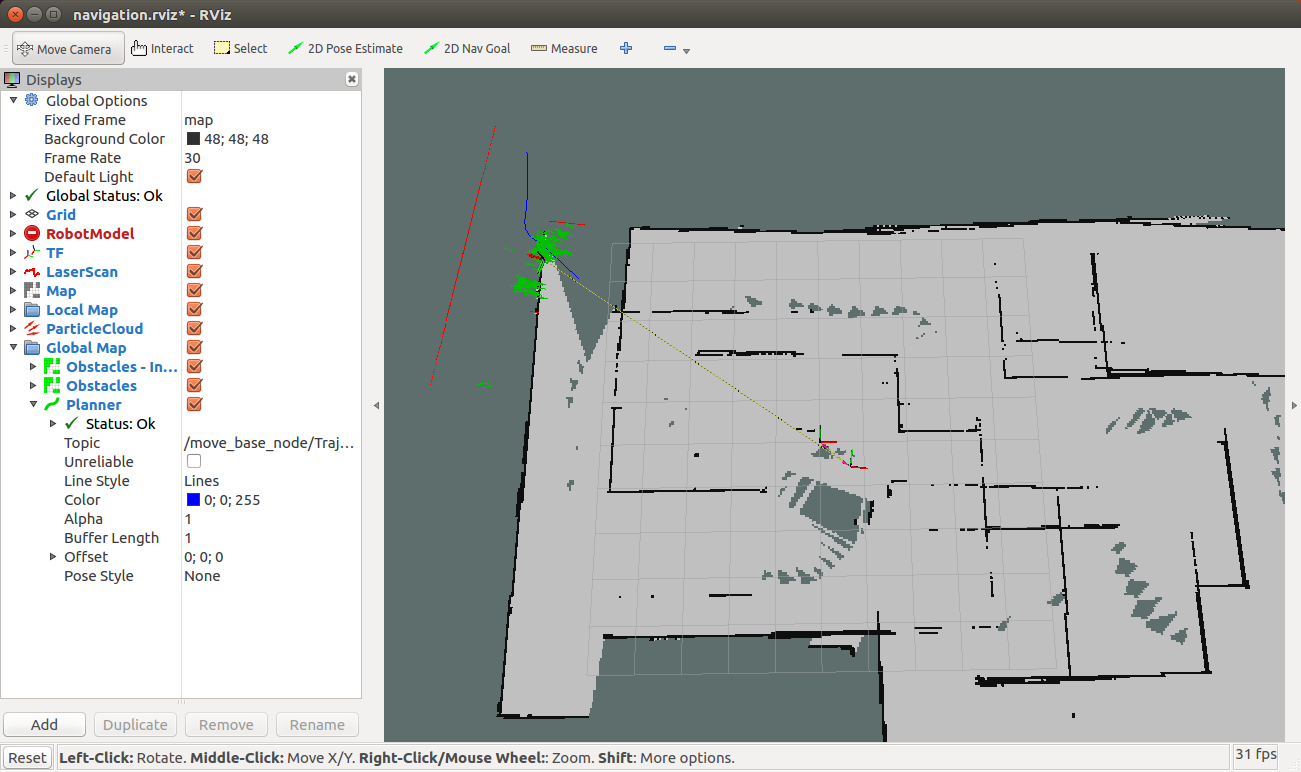
Then with a Kinect robot.

The Kinect has a lesser view angle so it takes even more time to generate a map (lots of rotating in place) and the map is less accurate:

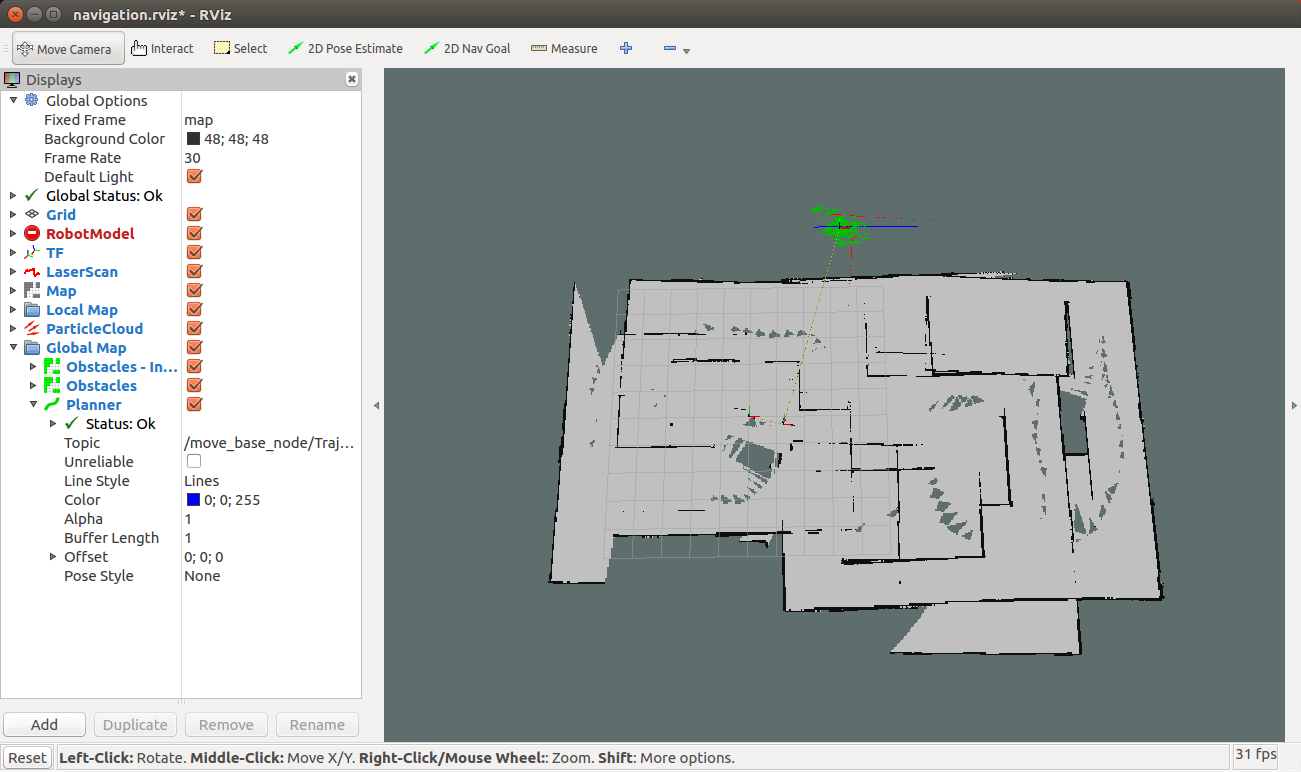


**2.2**

Sending to corners with laser bot:

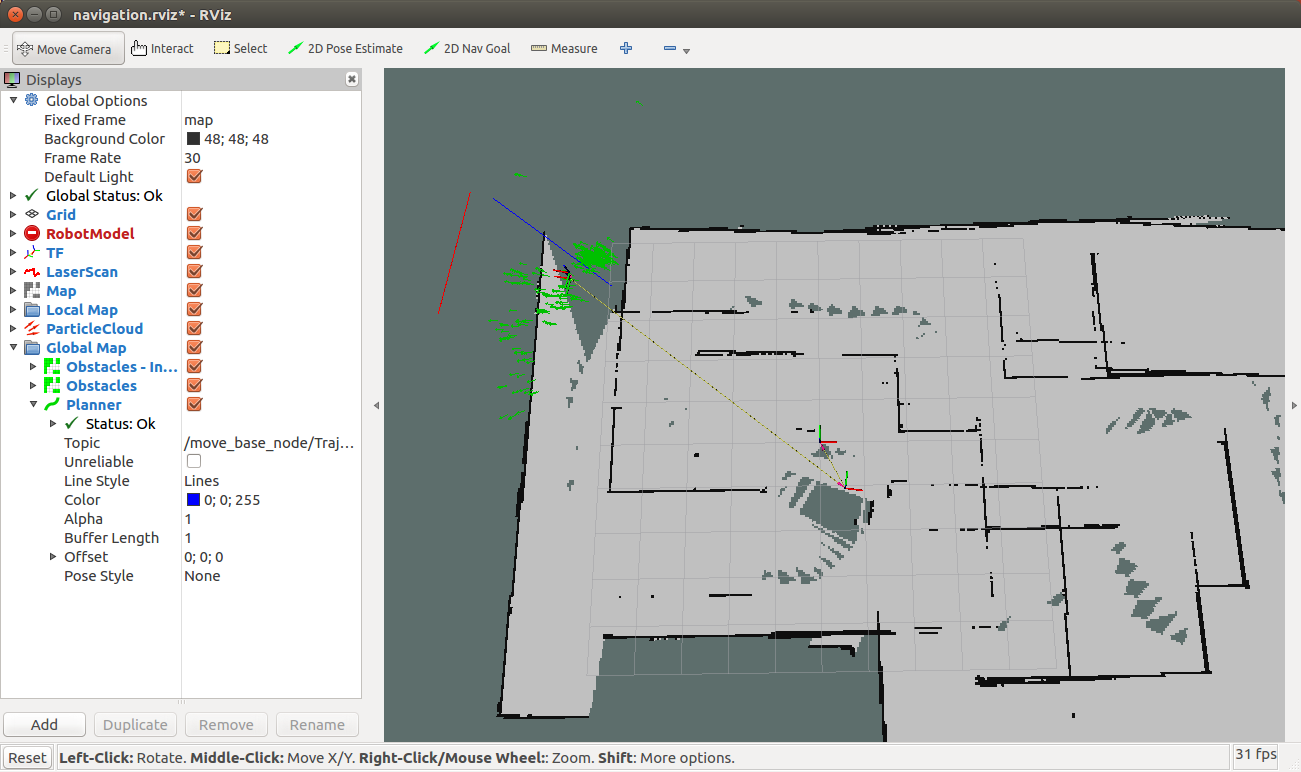


From middle to top-left it was very quick and a straight line but doesn’t match the generated map.



It’s extremely slow, even refusing to move, through narrow paths, erroring out and forcing me to direct another route against it or force it through using manual control. Robot also got stuck on the exact point in the screenshot. Manual control was required. Other things went smoothly and quick.

Sending to corners with Kinect bot:



When nearing turns the kinect bot is very slow, but other than that it goes smoothly. It did get stuck sometimes when nearing the corners in some rotate-recovery behavior. Manual control was required more often. Kinect bot has the same narrow path problems as the laser bot. It also fails hard when the map and kinect data don’t match. Then it’s permanently stuck. The laser bot managed to get out of such situations.

**Solution**

|  |
| --- |
| #include <ros/ros.h>  #include <move\_base\_msgs/MoveBaseAction.h>  #include <actionlib/client/simple\_action\_client.h>  typedef actionlib::SimpleActionClient<move\_base\_msgs::MoveBaseAction> MoveBaseClient;  int main(int argc, char\*\* argv) {  ros::init(argc, argv, "simple\_navigation\_goals");  //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");  }  // isn't big so stack is fine  std::vector <move\_base\_msgs::MoveBaseGoal> goals;  move\_base\_msgs::MoveBaseGoal goal;  // populate goals here  for (auto goal : goals)  {  ROS\_INFO("Sending goal");  goal.target\_pose.header.stamp = ros::Time::now();  ac.sendGoal(goal);  ac.waitForResult();  if(ac.getState() != actionlib::SimpleClientGoalState::SUCCEEDED)  {  ROS\_WARN("The base failed to move");  }  }  return 0;  } |

The application works as expected. The robot drives in the order of the goals added in the program.