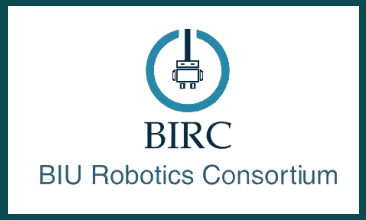
October 2016



ROS – Lecture 7

ROS navigation stack
Costmaps
Localization
Sending goal commands (from rviz)

Lecturer: Roi Yehoshua
roiyeho@gmail.com

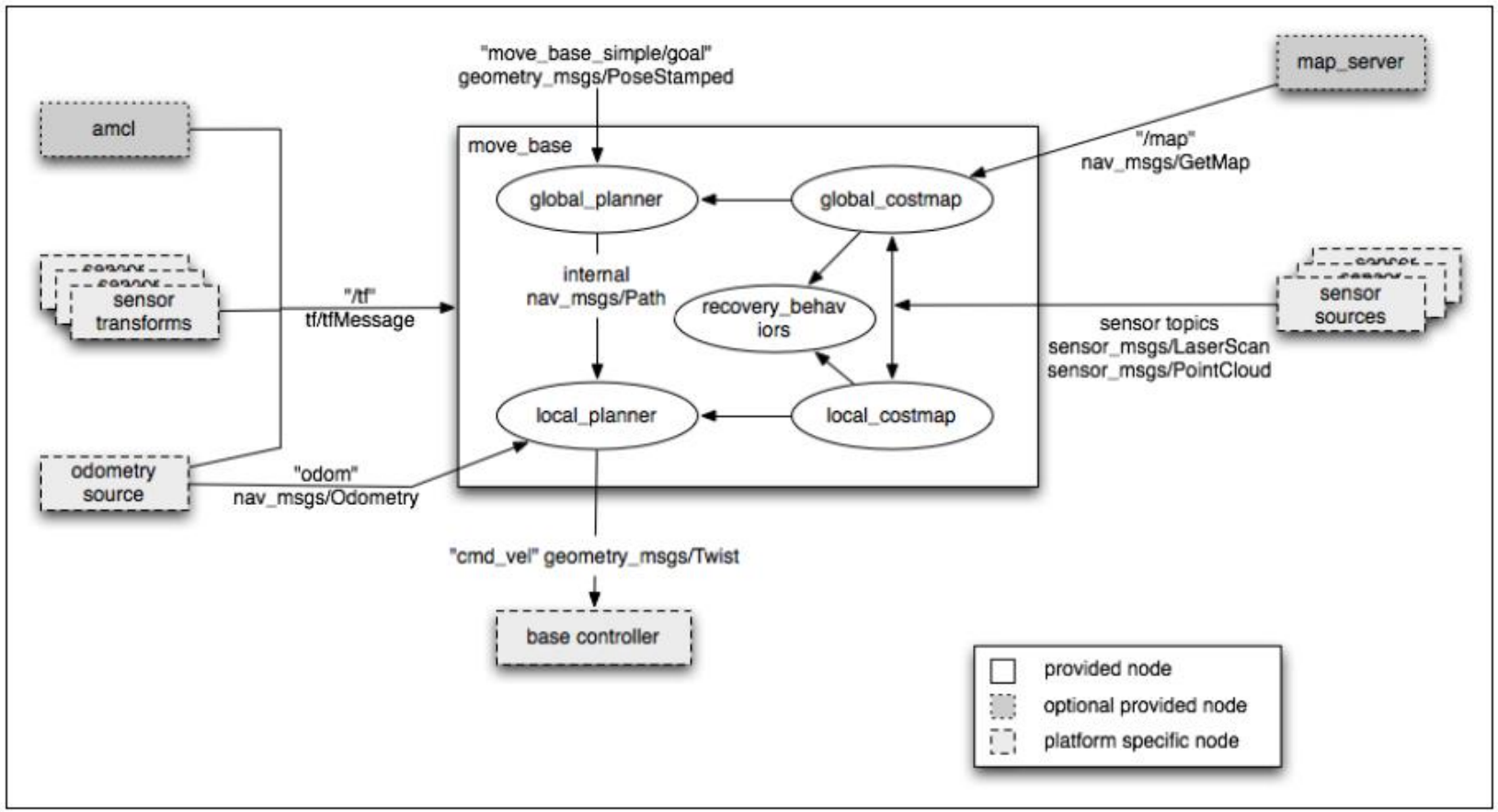
Robot Navigation

- One of the most basic things that a robot can do is to move around the world.
- To do this effectively, the robot needs to know where it is and where it should be going
- This is usually achieved by giving the robot a map of the world, a starting location, and a goal location
- In the previous lesson, we saw how to build a map of the world from sensor data.
- Now, we'll look at how to make your robot autonomously navigate from one part of the world to another, using this map and the ROS navigation packages

ROS Navigation Stack

- http://wiki.ros.org/navigation
- The goal of the navigation stack is to move a robot from one position to another position safely (without crashing or getting lost)
- It takes in information from the odometry and sensors, and a goal pose and outputs safe velocity commands that are sent to the robot
- ROS Navigation Introductory Video

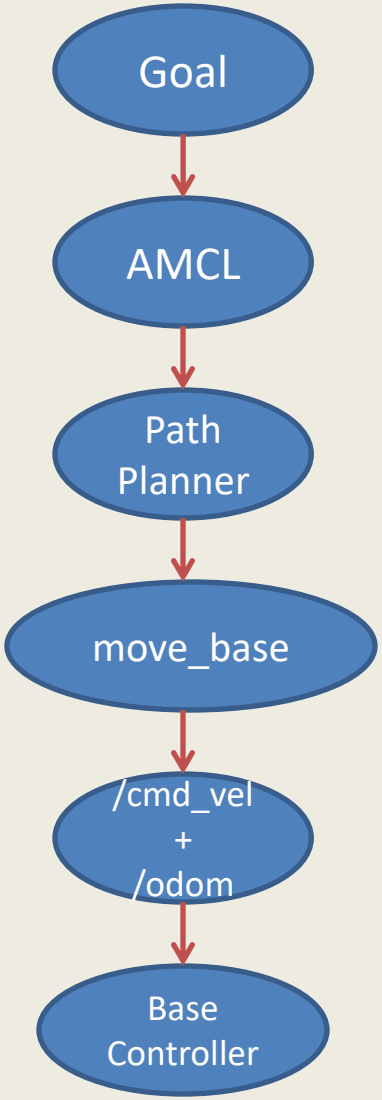
ROS Navigation Stack



Navigation Stack Main Components

Package/Component	Description
map_server	offers map data as a ROS Service
gmapping	provides laser-based SLAM
amcl	a probabilistic localization system
global_planner	implementation of a fast global planner for navigation
local_planner	implementations of the Trajectory Rollout and Dynamic Window approaches to local robot navigation
move_base	links together the global and local planner to accomplish the navigation task

Navigation Main Steps



Install Navigation Stack

- The navigation stack is not part of the standard ROS Indigo installation
- To install the navigation stack type:

```
$ sudo apt-get install ros-indigo-navigation
```

Navigation Stack Requirements

Three main hardware requirements

- The navigation stack can only handle a differential drive and holonomic wheeled robots
  - It can also do certain things with biped robots, such as localization, as long as the robot does not move sideways
- A planar laser must be mounted on the mobile base of the robot to create the map and localization
  - Alternatively, you can generate something equivalent to laser scans from other sensors (Kinect for example)
- Its performance will be best on robots that are nearly square or circular

Navigation Planners

- Our robot will move through the map using two types of navigation—global and local
- The **global planner** is used to create paths for a goal in the map or a far-off distance
- The **local planner** is used to create paths in the nearby distances and avoid obstacles

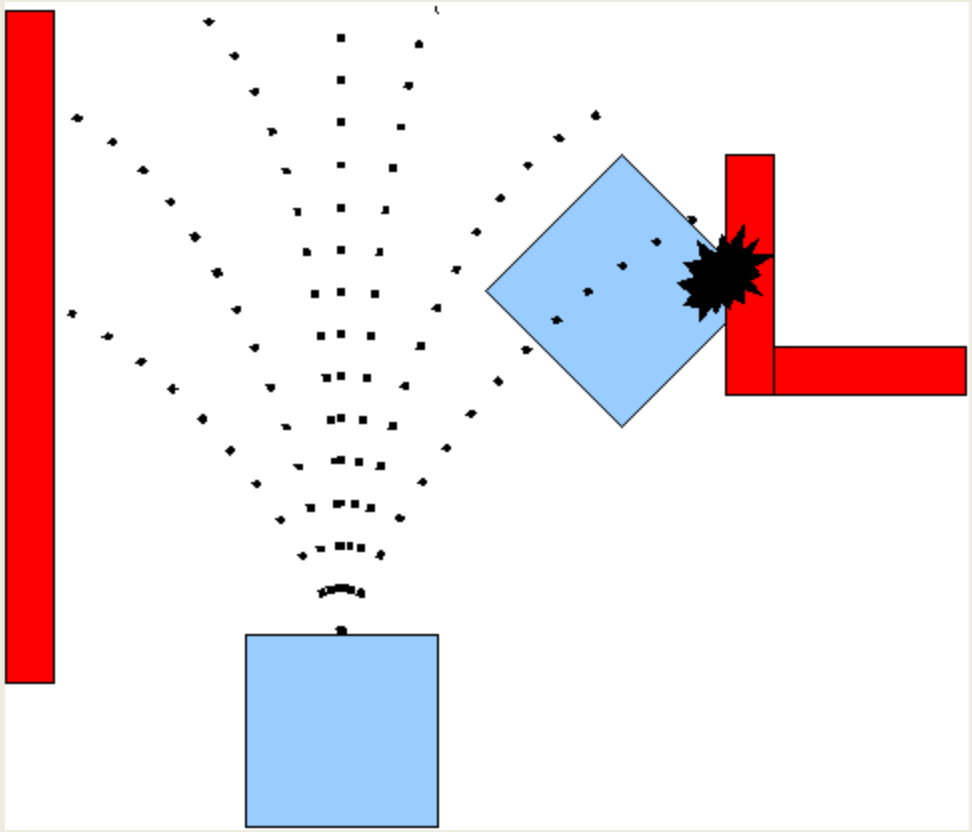
Global Planner

- NavFn provides a fast interpolated navigation function that creates plans for a mobile base
- The global plan is computed before the robot starts moving toward the next destination
- The planner operates on a costmap to find a minimum cost plan from a start point to an end point in a grid, using Dijkstra's algorithm
- The global planner generates a series of waypoints for the local planner to follow

Local Planner

- Chooses appropriate velocity commands for the robot to traverse the current segment of the global path
- Combines sensory and odometry data with both global and local cost maps
- Can recompute the robot's path on the fly to keep the robot from striking objects yet still allowing it to reach its destination
- Implements the **Trajectory Rollout and Dynamic Window** algorithm

Trajectory Rollout Algorithm



Taken from ROS Wiki http://wiki.ros.org/base_local_planner

Trajectory Rollout Algorithm

1. Discretely sample in the robot's control space (dx,dy,dθ)
2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time
3. Evaluate each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed
4. Discard illegal trajectories (those that collide with obstacles)
5. Pick the highest-scoring trajectory and send the associated velocity to the mobile base
6. Rinse and repeat

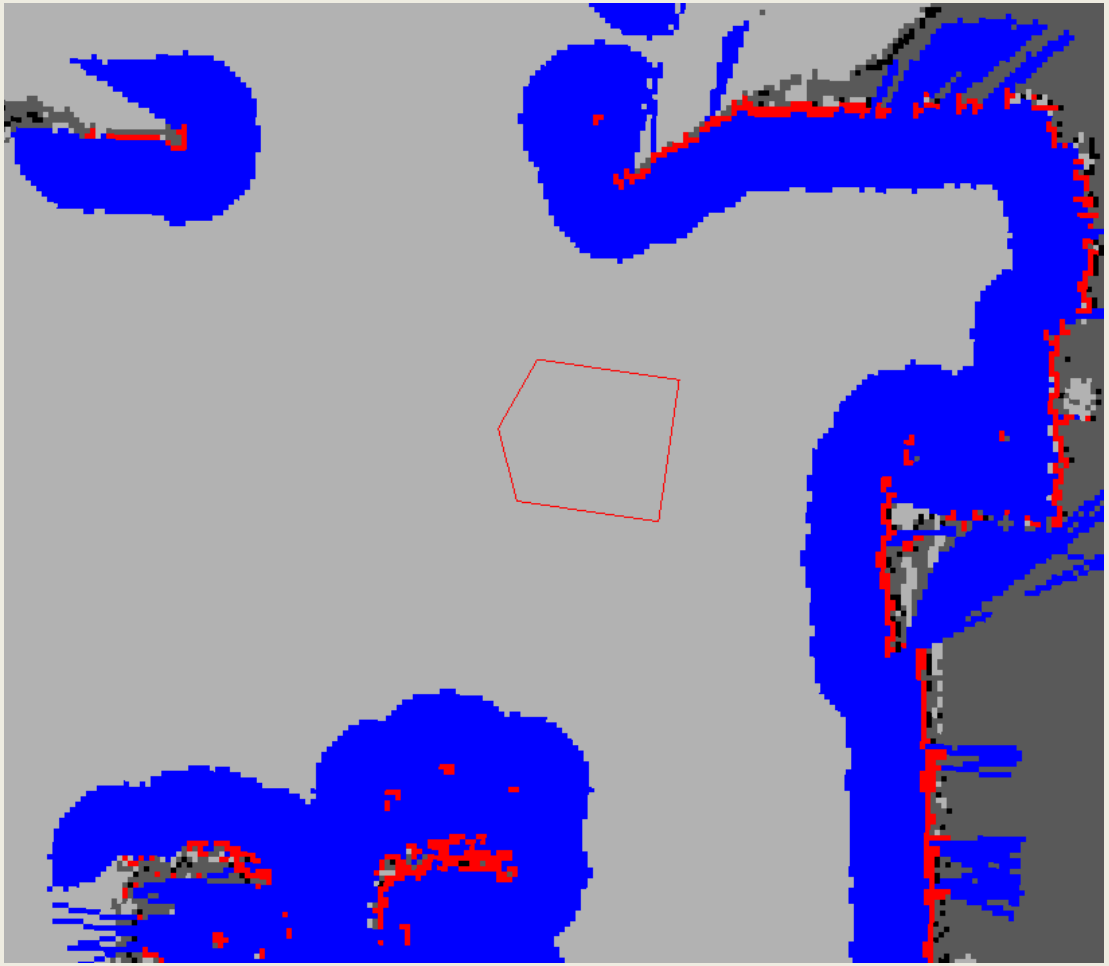
Local Planner Parameters

- The file **base_local_planner.yaml** contains a large number of ROS Parameters that can be set to customize the behavior of the base local planner
- Grouped into several categories:
  - robot configuration
  - goal tolerance
  - forward simulation
  - trajectory scoring
  - oscillation prevention
  - global plan

Costmap

- A data structure that represents places that are safe for the robot to be in a grid of cells
- It is based on the occupancy grid map of the environment and user specified inflation radius
- There are two types of costmaps in ROS:
  - **Global costmap** is used for global navigation
  - **Local costmap** is used for local navigation
- Each cell in the costmap has an integer value in the range [0 (FREE_SPACE), 255 (UNKNOWN)]
- Managed by the costmap_2d package

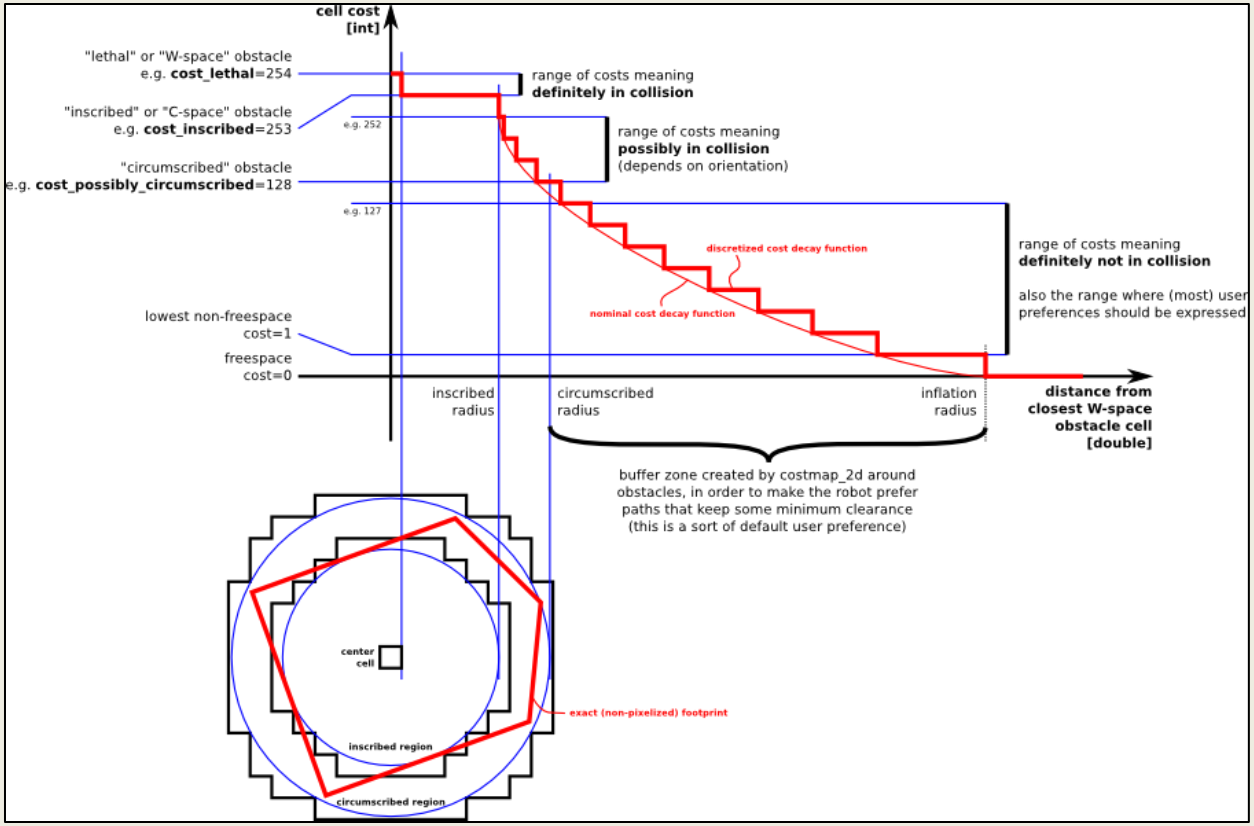
Costmap Example



Taken from ROS Wiki http://wiki.ros.org/costmap_2d

Inflation

- Inflation is the process of propagating cost values out from occupied cells that decrease with distance



Map Updates

- The costmap performs map update cycles at the rate specified by the **update_frequency** parameter
- In each cycle:
  - sensor data comes in
  - marking and clearing operations are perfomed in the underlying occupancy structure of the costmap
  - this structure is projected into the costmap where the appropriate cost values are assigned as described above
  - obstacle inflation is performed on each cell with a LETHAL_OBSTACLE value
    - This consists of propagating cost values outwards from each occupied cell out to a user-specified inflation radius

Costmap Parameters Files

- Configuration of the costmaps consists of three files:
  - costmap_common_params.yaml
  - global_costmap_params.yaml
  - local_costmap_params.yaml
- http://wiki.ros.org/costmap_2d/hydro/obstacles

Localization

- Localization is the problem of estimating the pose of the robot relative to a map
- Localization is not terribly sensitive to the exact placement of objects so it can handle small changes to the locations of objects
- ROS uses the **amcl** package for localization

AMCL

- amcl is a probabilistic localization system for a robot moving in 2D
- It implements the adaptive **Monte Carlo localization** approach, which uses a particle filter to track the pose of a robot against a known map
- The algorithm and its parameters are described in the book **Probabilistic Robotics** by Thrun, Burgard, and Fox (http://www.probabilistic-robotics.org/)
- Currently amcl works only with laser scans
  - However, it can be extended to work with other sensors

AMCL

- amcl takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates
- Subscribed topics:
  - scan – Laser scans
  - tf – Transforms
  - initialpose – Mean and covariance with which to (re-) initialize the particle filter
  - map – the map used for laser-based localization
- Published topics:
  - amcl_pose – Robot's estimated pose in the map, with covariance.
  - Particlecloud – The set of pose estimates being maintained by the filter

move_base

- The move_base package lets you move a robot to desired positions using the navigation stack
- The move_base node links together a global and local planner to accomplish its navigation task
- It may optionally perform recovery behaviors when the robot perceives itself as stuck

TurtleBot Navigation

- turtlebot_navigation package includes demos of map building using gmapping and localization with amcl, while running the navigation stack
- In param subdirectory it contains configuration files for TurtleBot navigation

Navigation Configuration Files

Configuration File	Description
global_planner_params.yaml	global planner configuration
navfn_global_planner_params.yaml	navfn configuration
dwa_local_planner_params.yaml	local planner configuration
costmap_common_params.yaml global_costmap_params.yaml local_costmap_params.yaml	costmap configuration files
move_base_params.yaml	move base configuration
amcl.launch.xml	amcl configuration

Autonomous Navigation of a Known Map

- amcl_demo.launch - launch file for navigation demo

```
<launch>
  <!-- Map server -->
  <arg name="map_file" default="$(env TURTLEBOT_GAZEBO_MAP_FILE)"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

  <!-- Localization -->
  <arg name="initial_pose_x" default="0.0"/>
  <arg name="initial_pose_y" default="0.0"/>
  <arg name="initial_pose_a" default="0.0"/>
  <include file="$(find turtlebot_navigation)/launch/includes/amcl.launch.xml">
    <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
    <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
    <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
  </include>

  <!-- Move base -->
  <include file="$(find turtlebot_navigation)/launch/includes/move_base.launch.xml"/>
</launch>
```

Autonomous Navigation of a Known Map

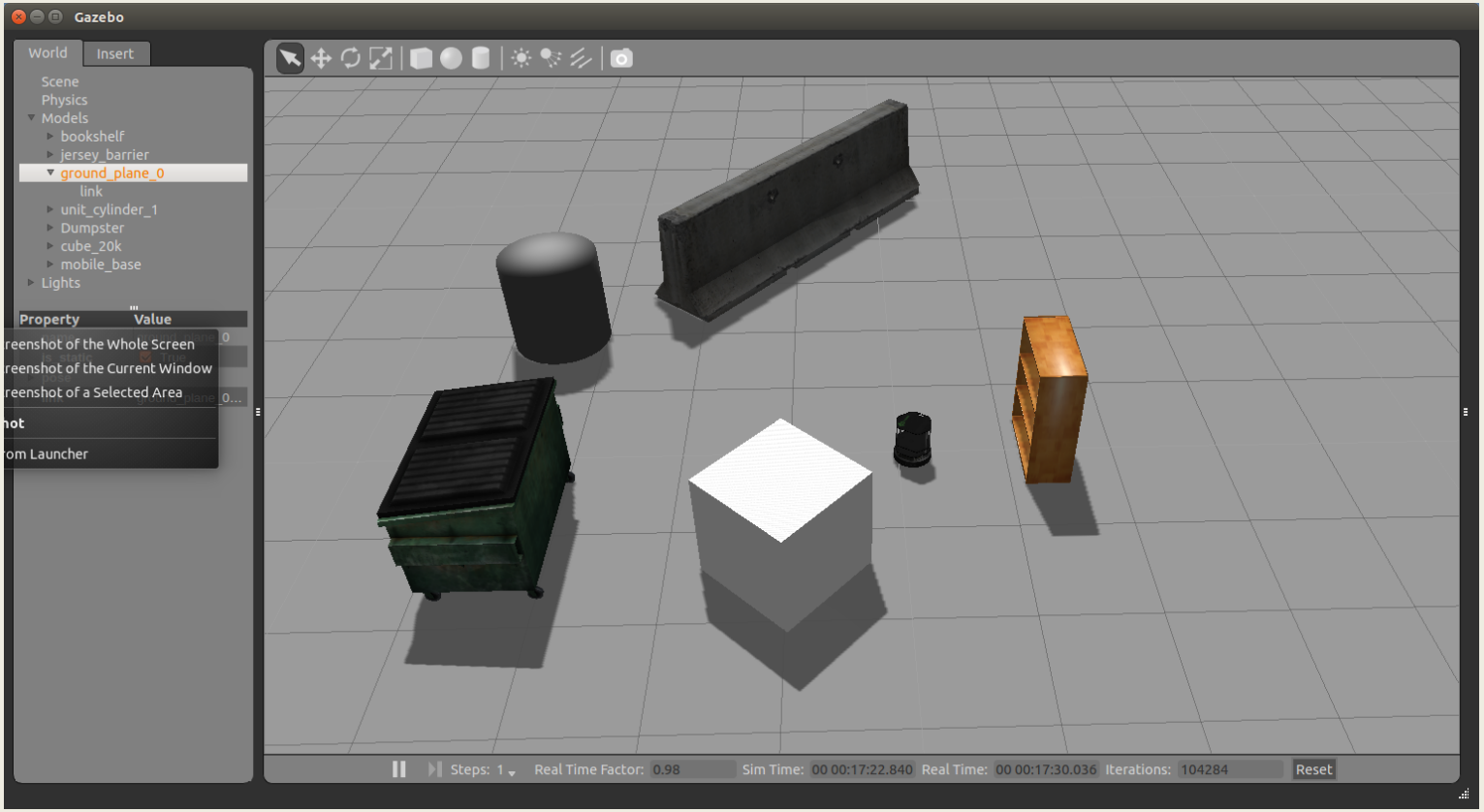
- Launch Gazebo with turtlebot

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

- Run the navigation demo

```
$ roslaunch turtlebot_gazebo amcl_demo.launch
```

Autonomous Navigation of a Known Map

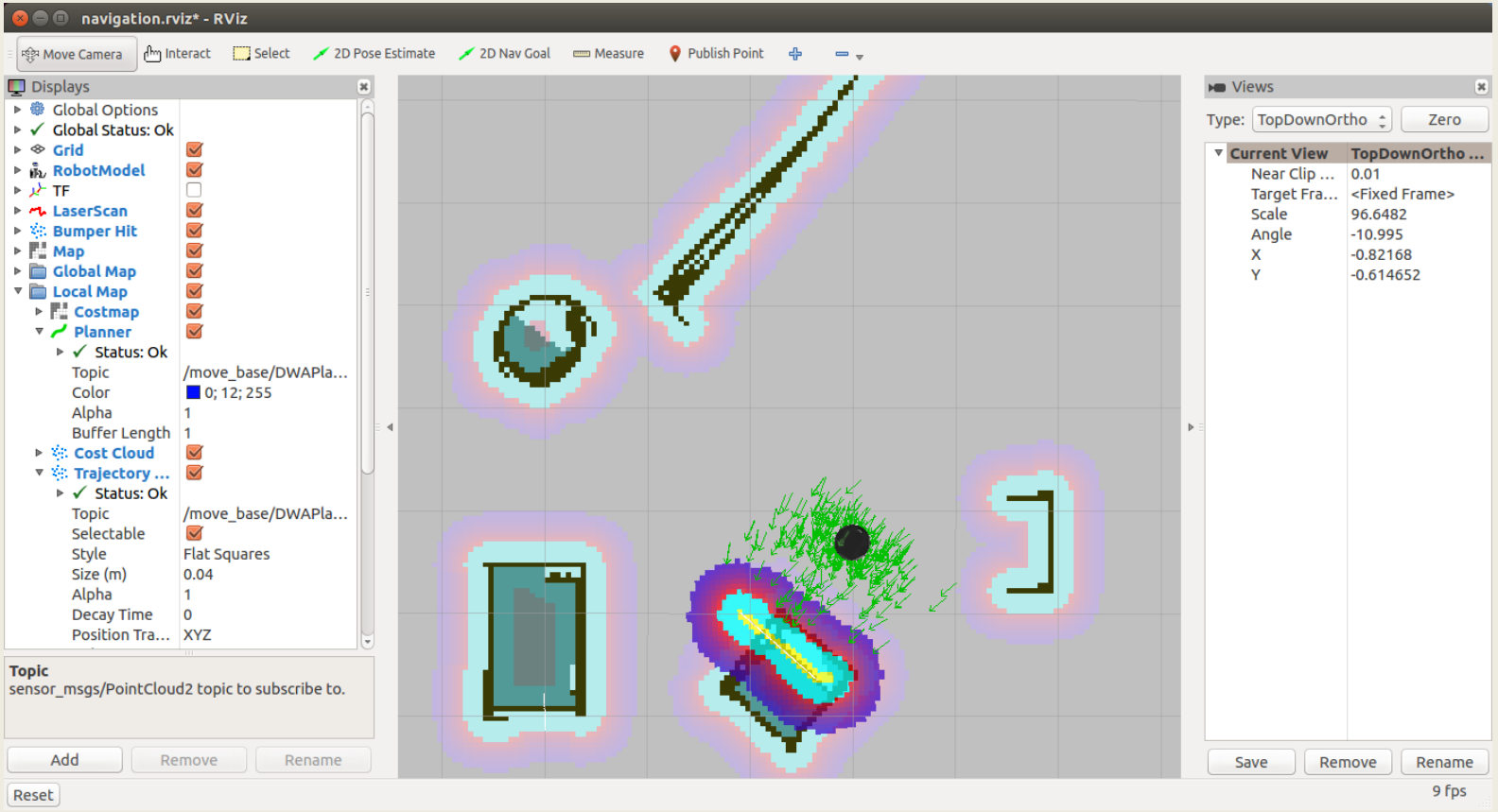


rviz with Navigation Stack

- rviz allows you to:
  - Provide an approximate location of the robot (when starting up, the robot doesn't know where it is)
  - Send goals to the navigation stack
  - Display all the visualization information relevant to the navigation (planned path, costmap, etc.)
- Launch rviz:

```
$ roslaunch turtlebot_rviz_launchers view_navigation.launch
```

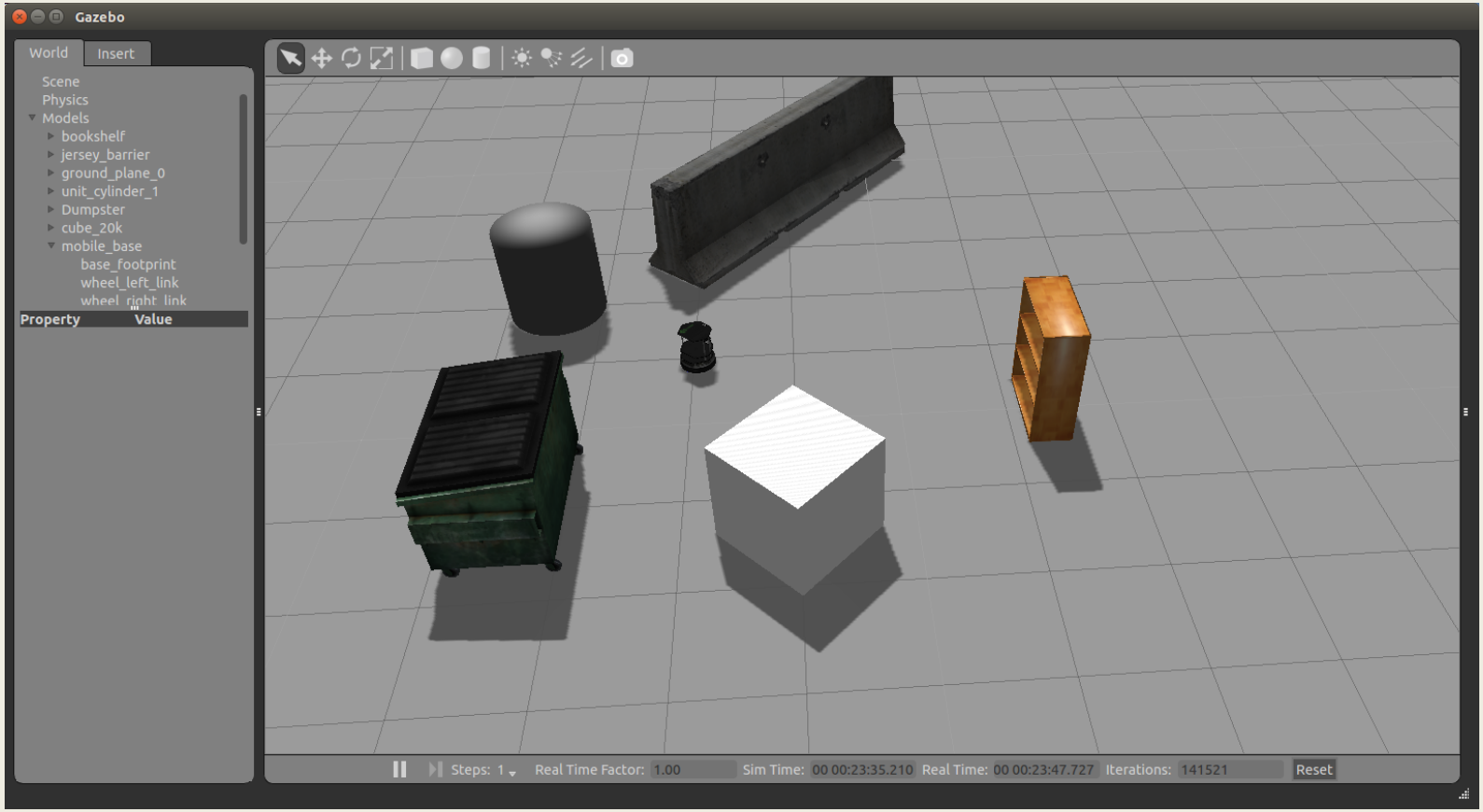
rviz with Navigation Stack



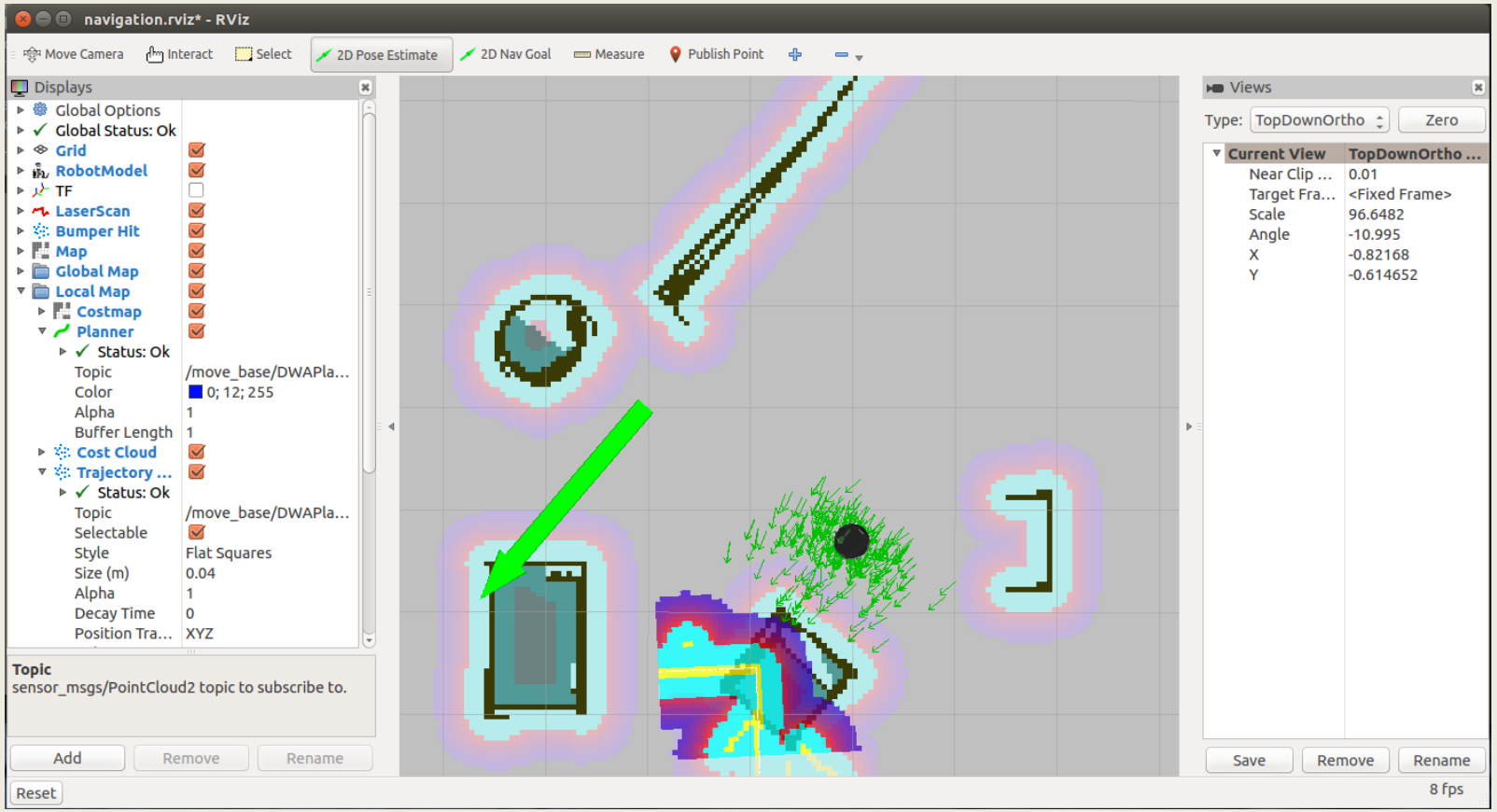
Localize the TurtleBot

- When starting up the TurtleBot doesn't know where it is
- For example, let’s move the robot in Gazebo to (-1,-2)
- Now to provide it its approximate location on the map:
  - Click the "2D Pose Estimate" button
  - Click on the map where the TurtleBot approximately is and drag in the direction the TurtleBot is pointing
- You will see a collection of arrows which are hypotheses of the position of the TurtleBot
- The laser scan should line up approximately with the walls in the map
  - If things don't line up well you can repeat the procedure

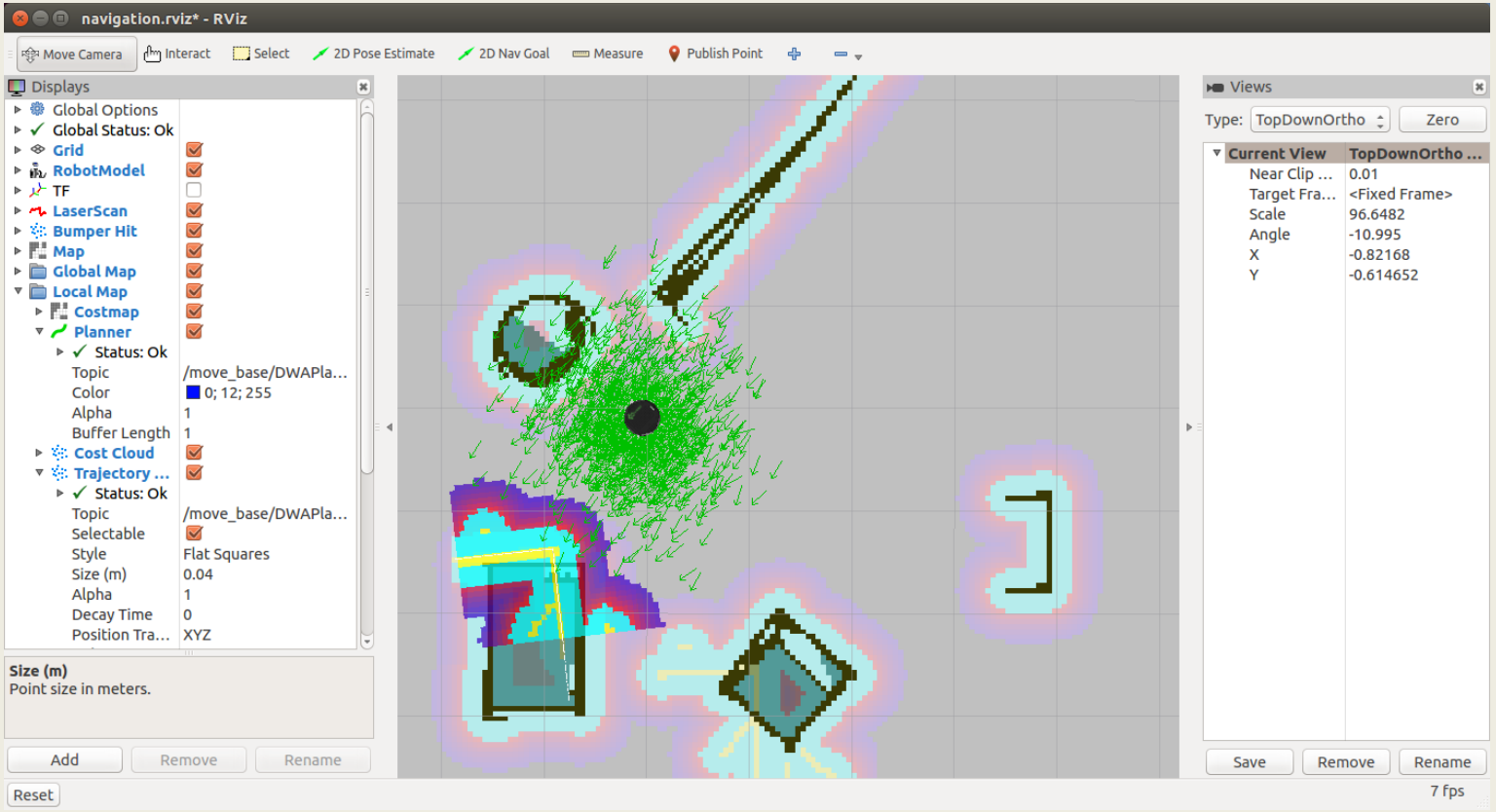
Localize the TurtleBot



Localize the TurtleBot

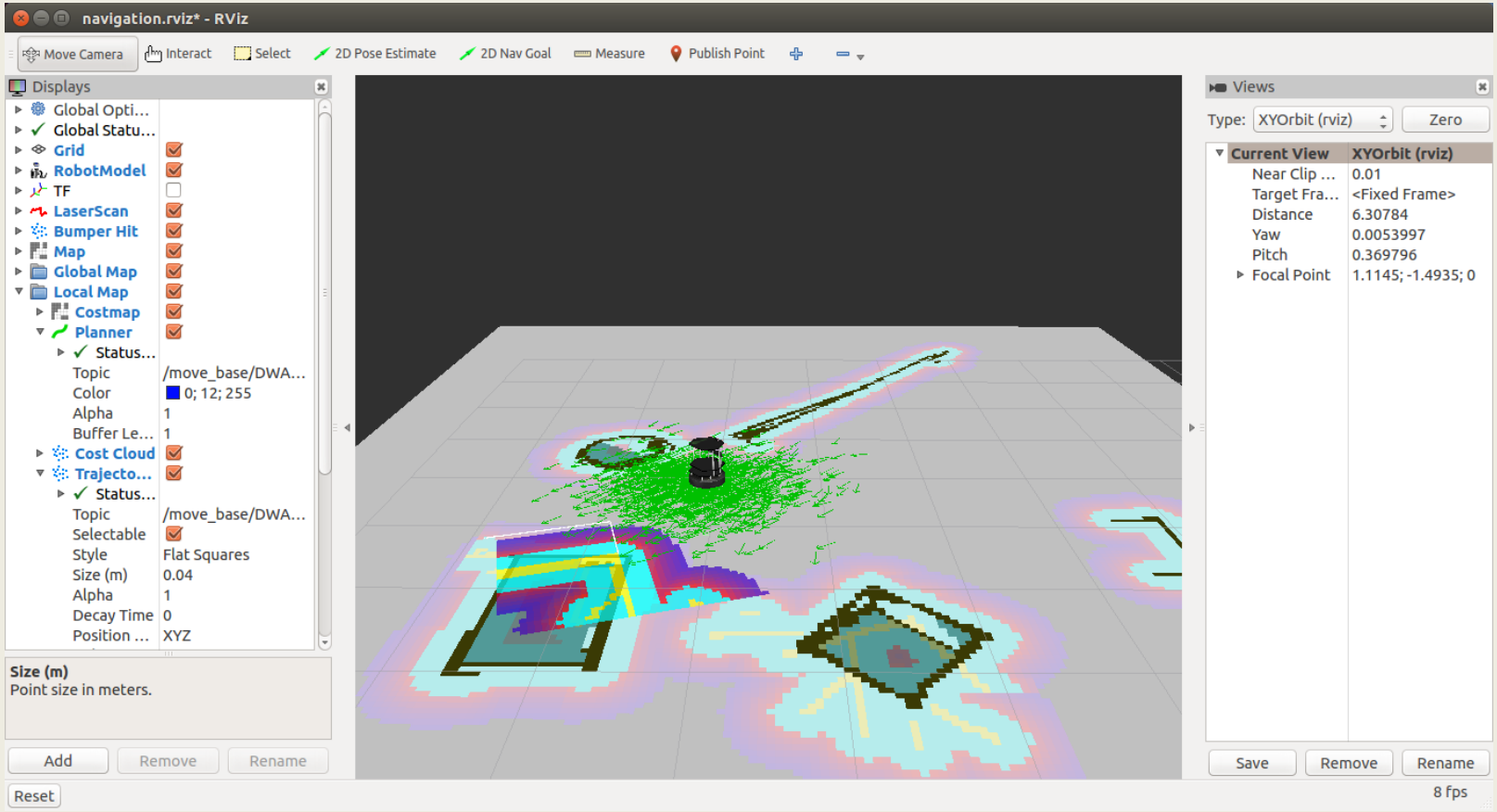


Localize the TurtleBot



Localize the TurtleBot

- You can change the current view (on right panel):



Particle Cloud in rviz

- The **Particle Cloud** display shows the particle cloud used by the robot's localization system
- The spread of the cloud represents the localization system's uncertainty about the robot's pose
- As the robot moves about the environment, this cloud should shrink in size as additional scan data allows amcl to refine its estimate of the robot's position and orientation
- To watch the particle cloud in rviz:
  - Click Add Display and choose Pose Array
  - Set topic name to /particlecloud

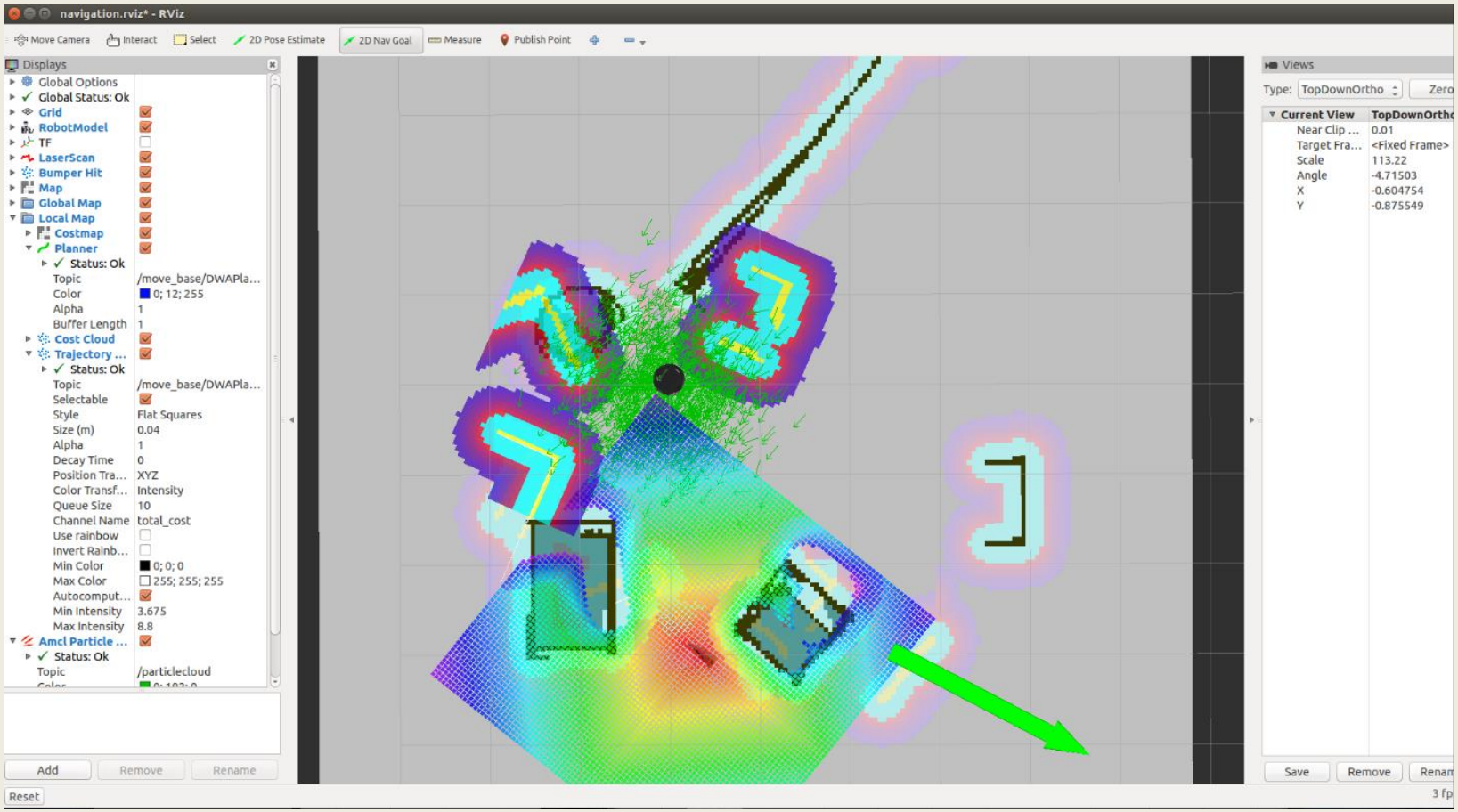
Teleoperation

- The teleoperation can be run simultaneously with the navigation stack
- It will override the autonomous behavior if commands are being sent
- It is often a good idea to teleoperate the robot after seeding the localization to make sure it converges to a good estimate of the position

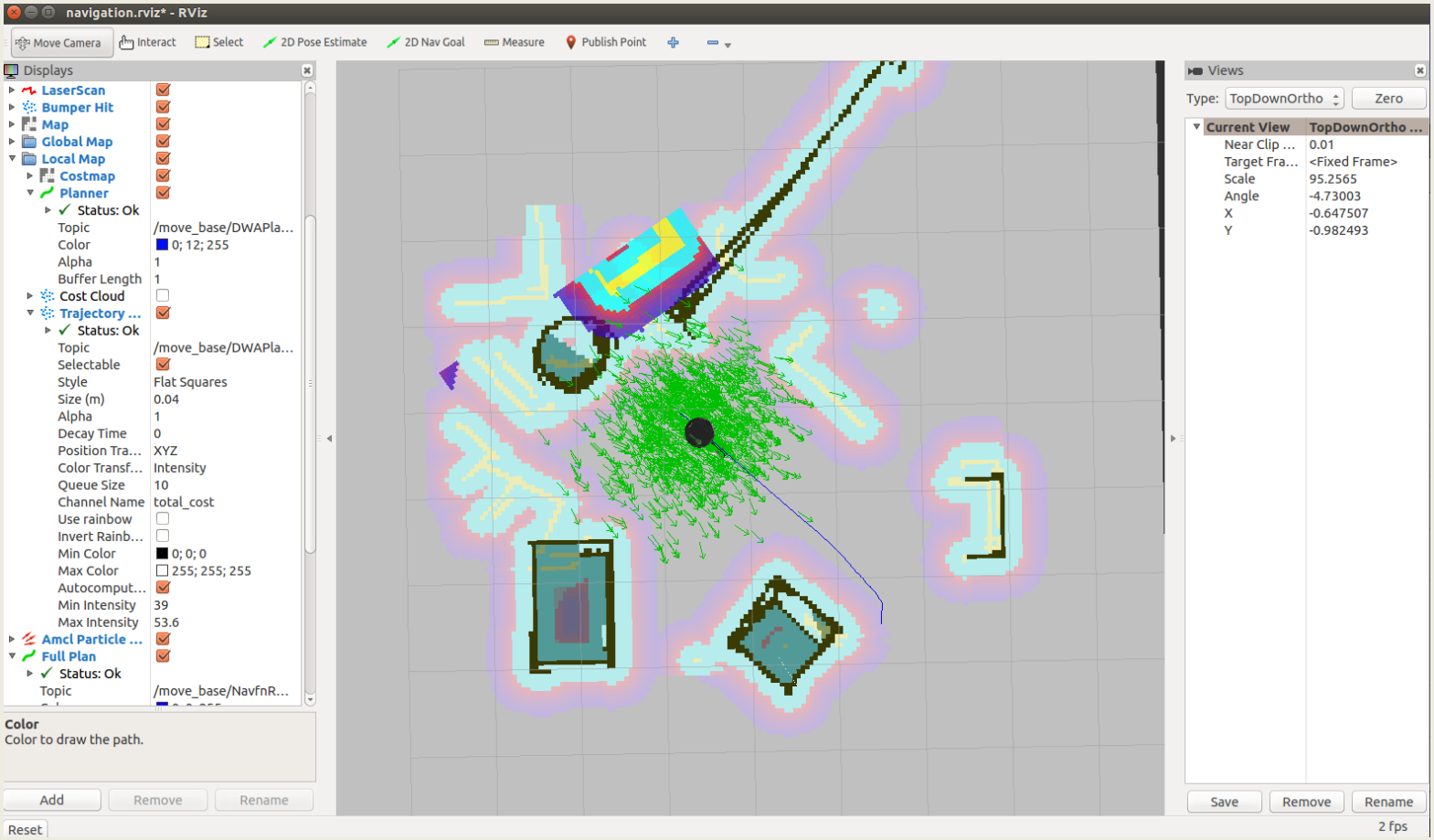
Send a Navigation Goal

- With the TurtleBot localized, it can then autonomously plan through the environment
- To send a goal:
  - Click the "2D Nav Goal" button
  - Click on the map where you want the TurtleBot to drive and drag in the direction where it should be pointing at the end
- If you want to stop the robot before it reaches it's goal, send it a goal at it's current location

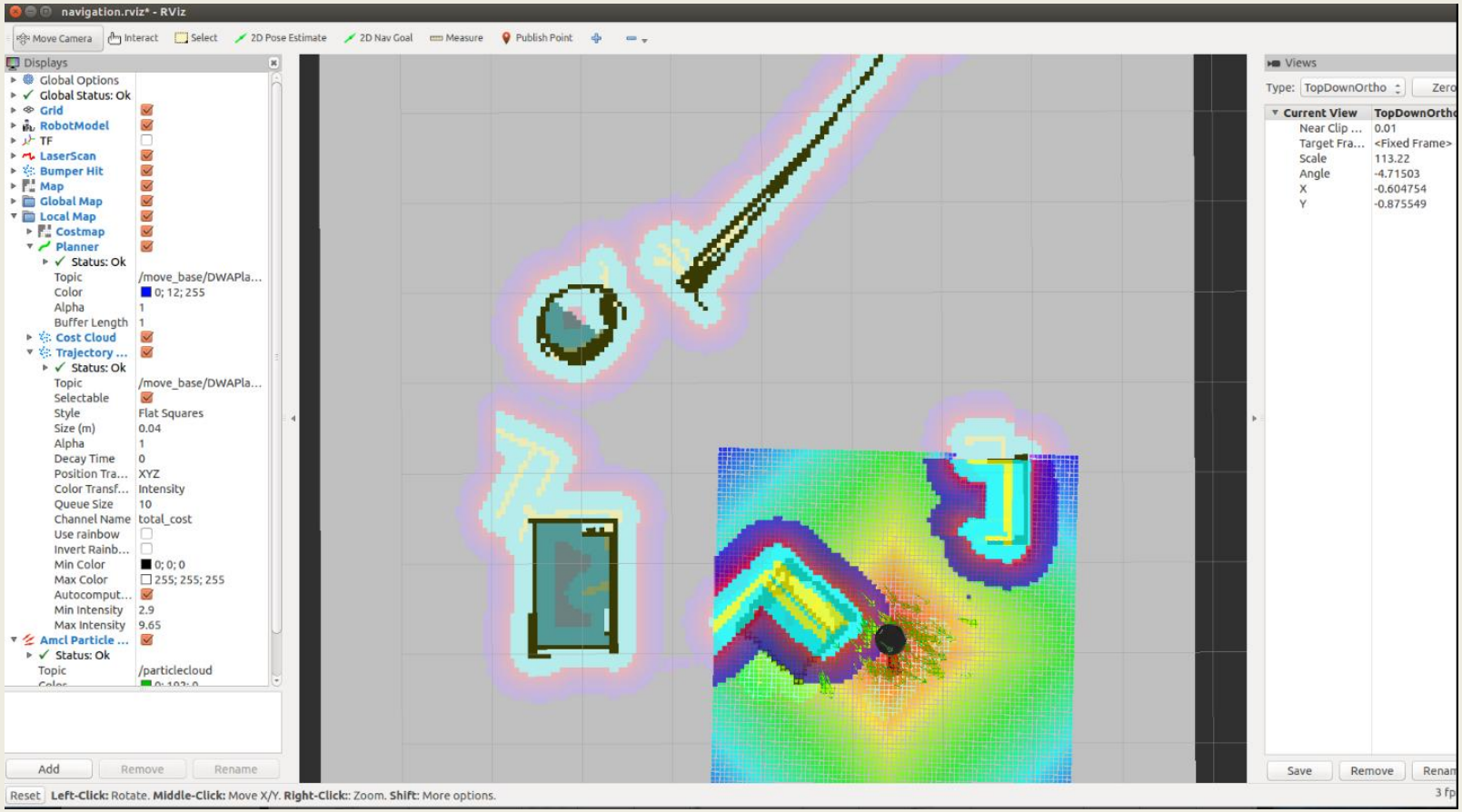
Send a Navigation Goal



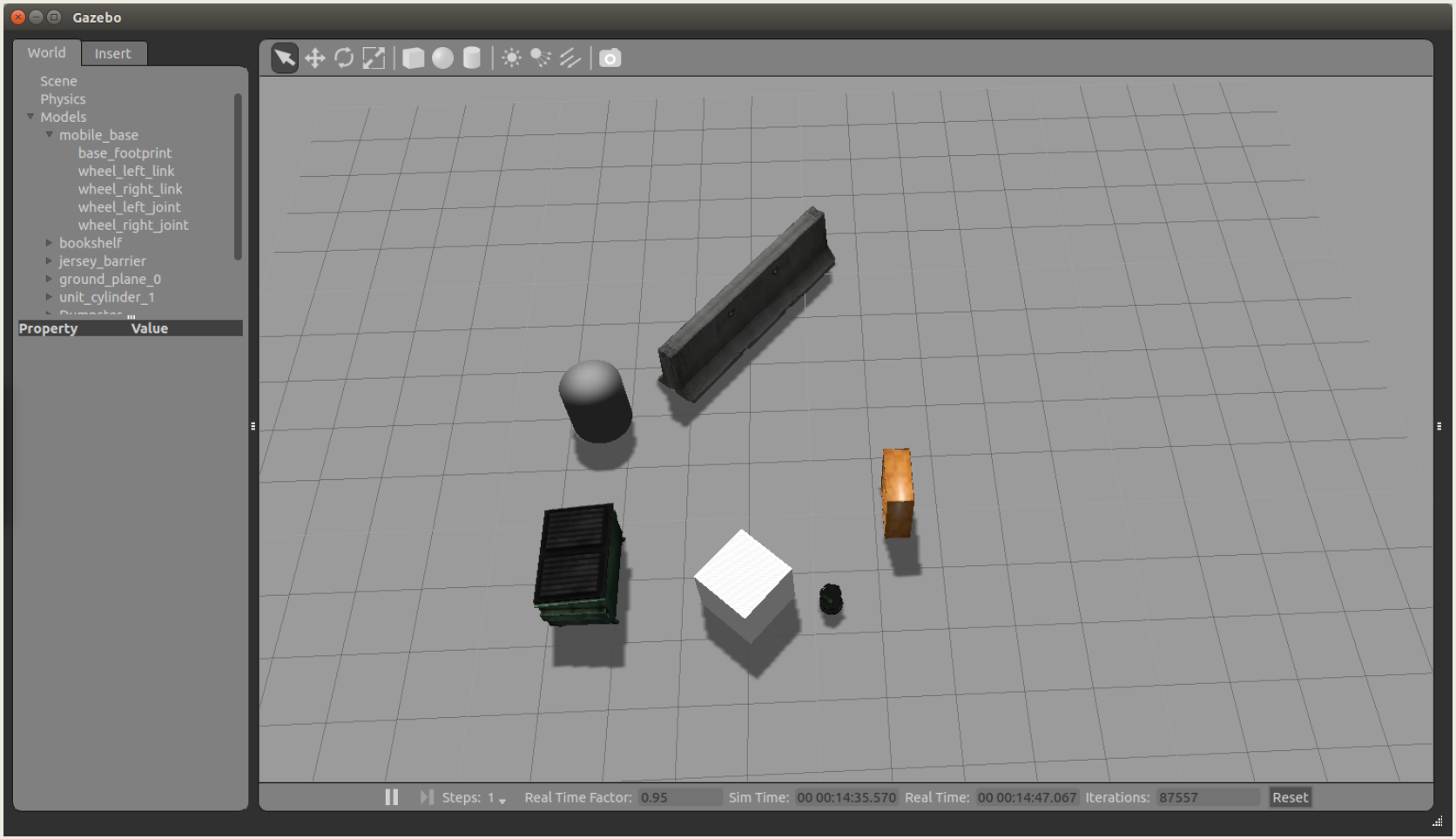
Robot Moves to Destination



Final Pose



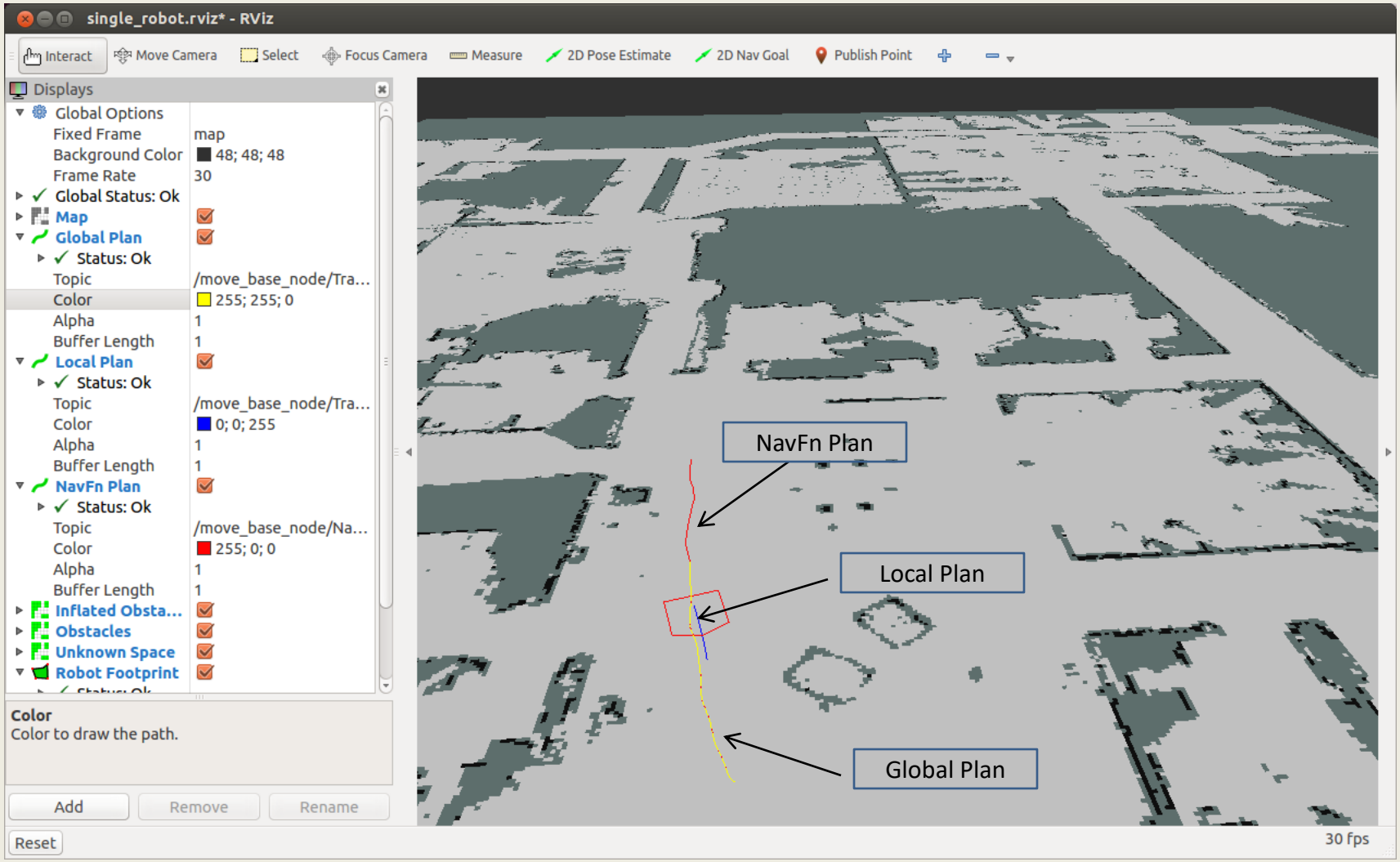
Final Pose In Gazebo



Navigation Plans in rviz

- **NavFn Plan**
  - Displays the full plan for the robot computed by the global planner
  - Topic: /move_base_node/NavfnROS/plan
- **Global Plan**
  - Shows the portion of the global plan that the local planner is currently pursuing
  - Topic: /move_base_node/TrajectoryPlannerROS/global_plan
- **Local Plan**
  - Shows the trajectory associated with the velocity commands currently being commanded to the base by the local planner
  - Topic: /move_base_node/TrajectoryPlannerROS/local_plan

Navigation Plans in rviz



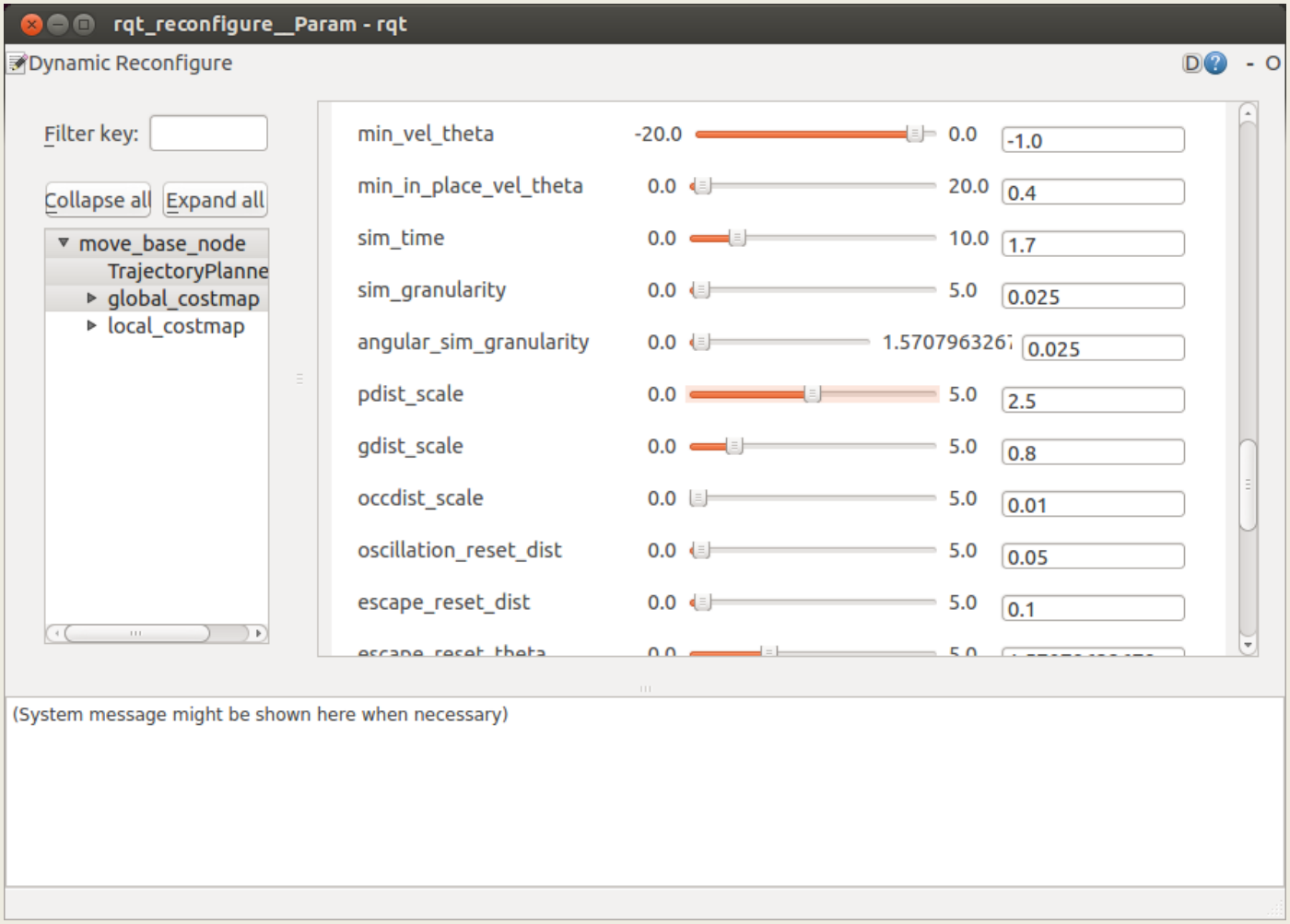
rqt_reconfigure

- A tool for changing dynamic configuration values
- To launch rqt_reconfigure, run:

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

- The navigation stack parameters will appear under move_base_node

rqt_reconfigure



Ex. 7

- Implement a simple navigation algorithm (based on A*) that will use the grid representation of the map to compute a route for the robot from its current location to a given goal location