UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

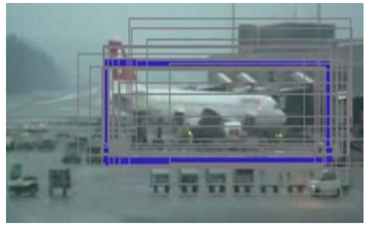
Laurea magistrale in ingegneria e scienze informatiche

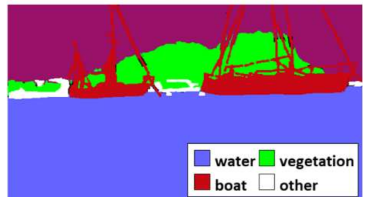



Corso di Robotica
Parte di Laboratorio

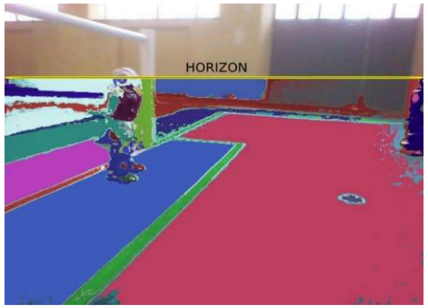
Introduzione alla navigazione in ROS

Docente:
Domenico Daniele Bloisi









Dicembre 2017

References and credits

Alcune delle slide seguenti sono tratte da

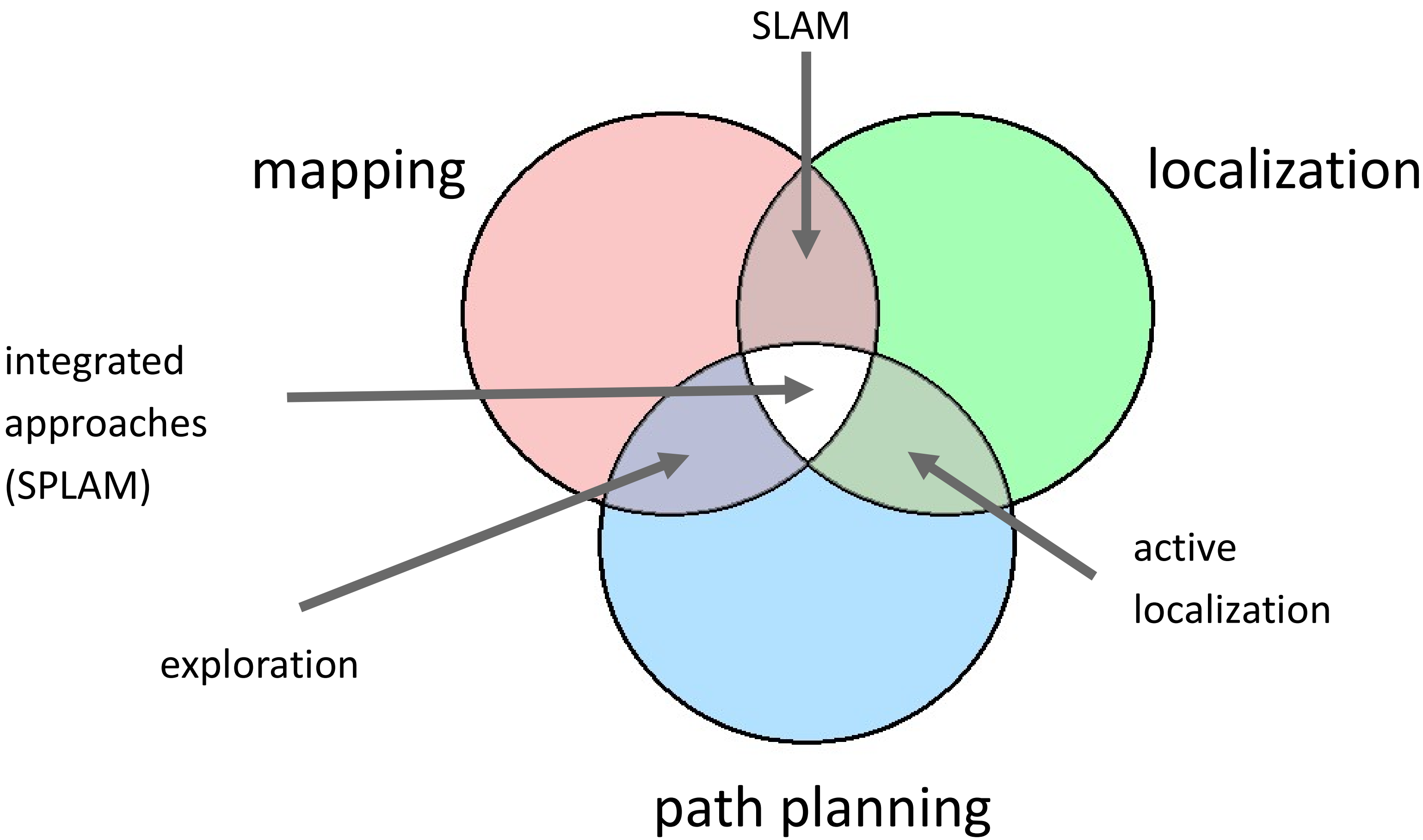
Giorgio Grisetti
Introduction to Navigation using ROS

Giorgio Grisetti
Probabilistic Robotics Course
Introduction

Giorgio Grisetti
Probabilistic Robotics Course
Multi-Pose Registration Graph-SLAM

Learn TurtleBot and ROS (http://learn.turtlebot.com/)
- Creating a Map
- Autonomous Navigation

Mapping, localization, planning

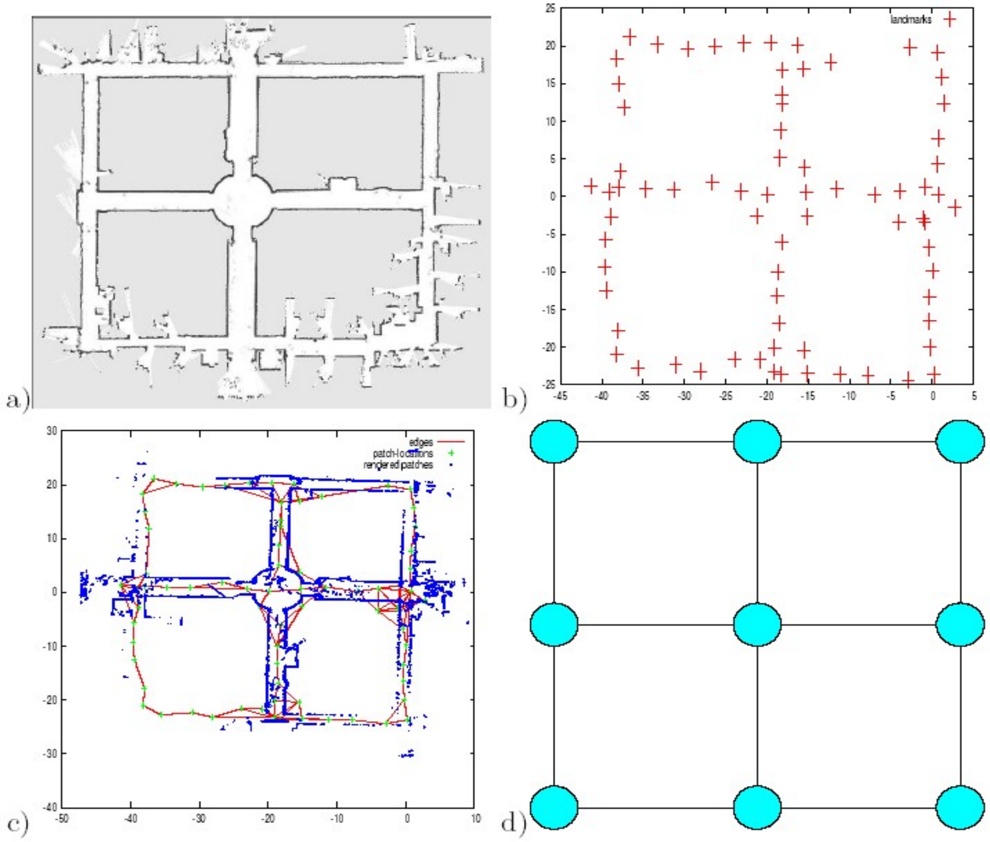


Map

- A map is a representation of the environment where the robot is operating.
- It should contain enough information to accomplish a task of interest.

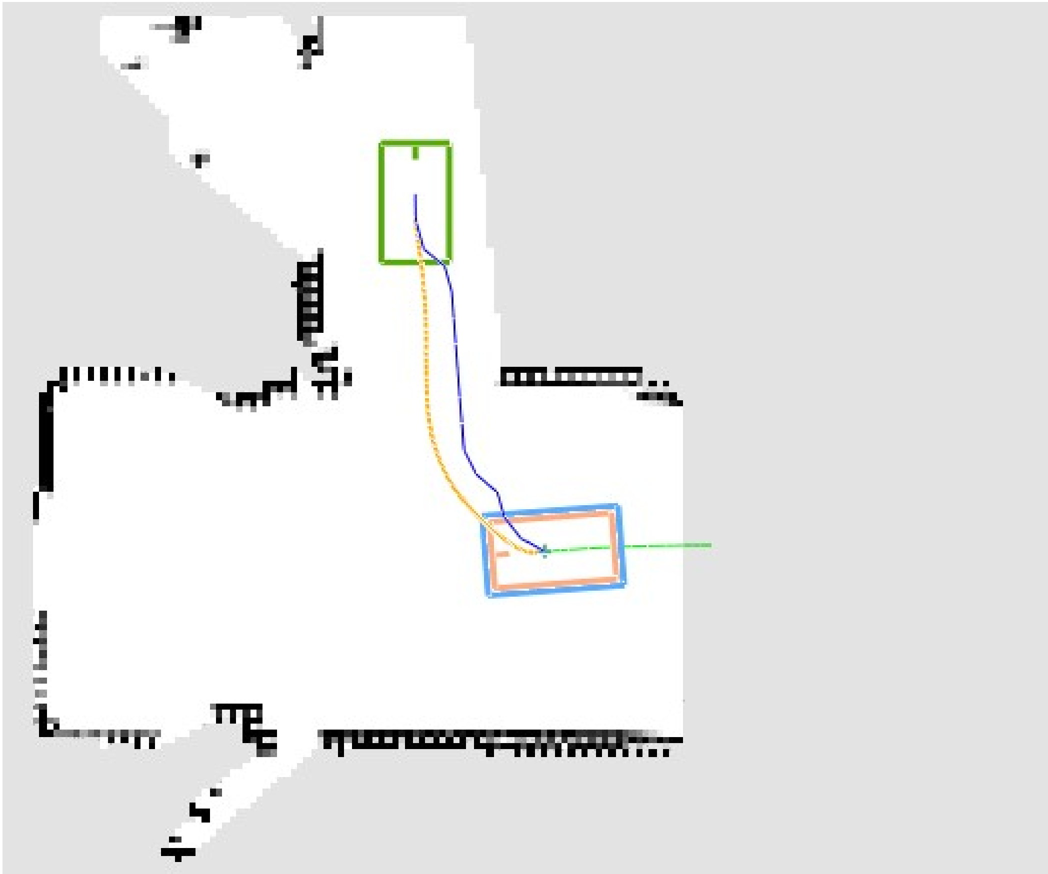
Representations:

- Metric
  - Grid Based
  - Feature Based
  - Hybrid
- Topological
- Hybrid



Robot pose and path

- A metric map defines a reference frame
- To operate in a map, a robot should know its position in that reference frame
- A sequence of waypoints or of actions to reach a goal location in the map is a path



Path planning

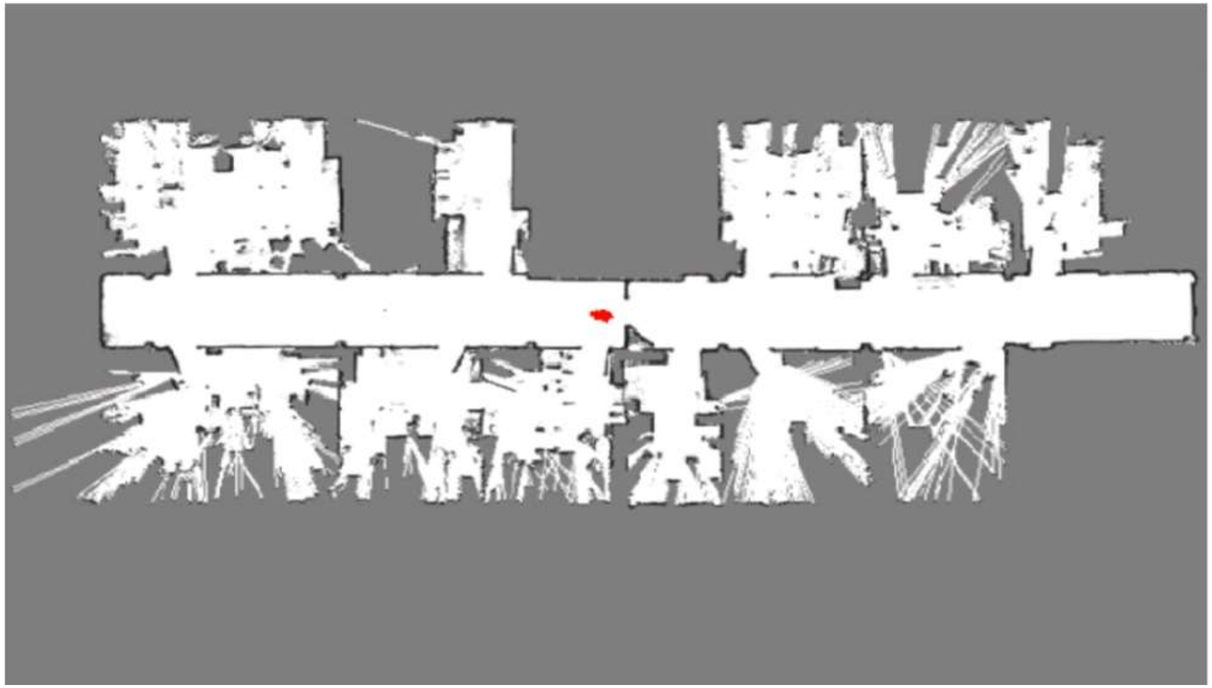
Determine (if it exists) a path to reach a given goal location given a localized robot and a map of traversable regions

Localization

Determining the current position of a robot, given

1. The knowledge of the map

2. All sensor measurements up to the current time



Mapping

Given

1. a robot that has a perfect ego-estimate of the position
2. a sequence of measurements

determine the map of the environment

- A perfect estimate of the robot pose is usually not available
- Usually we solve a more complex problem:
  Simultaneous Localization and Mapping (SLAM)

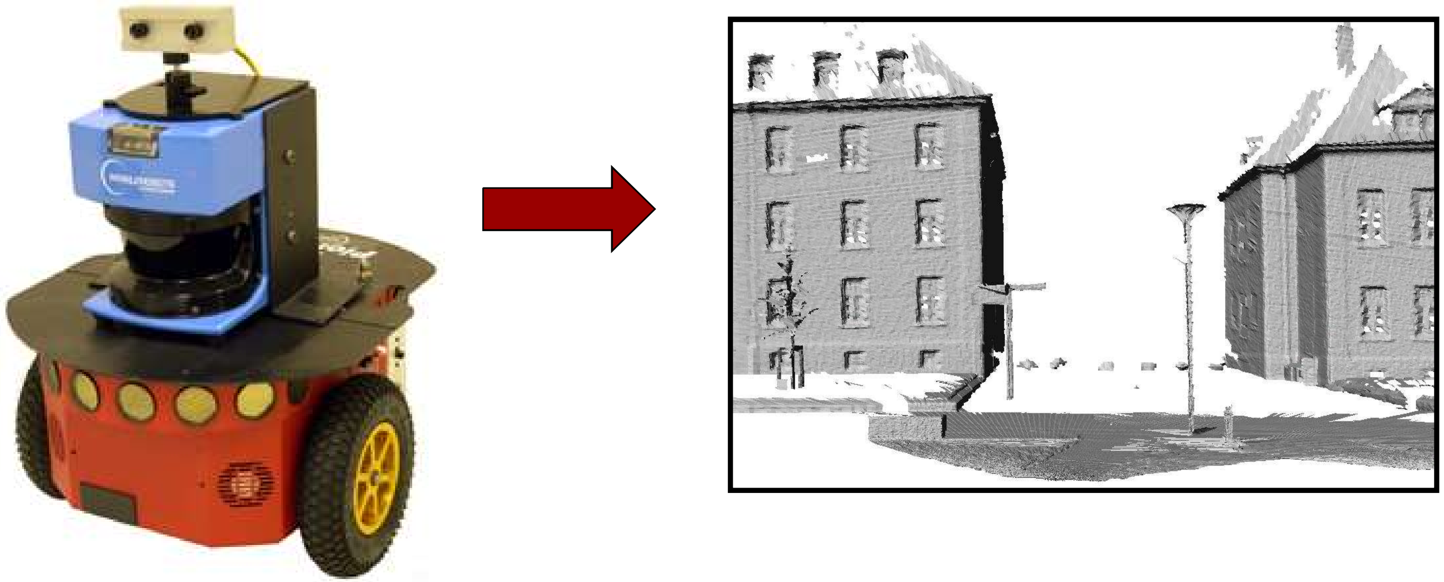
Simultaneous Localization and Mapping

Estimate:

1. the map of the environment
2. the trajectory of a moving device

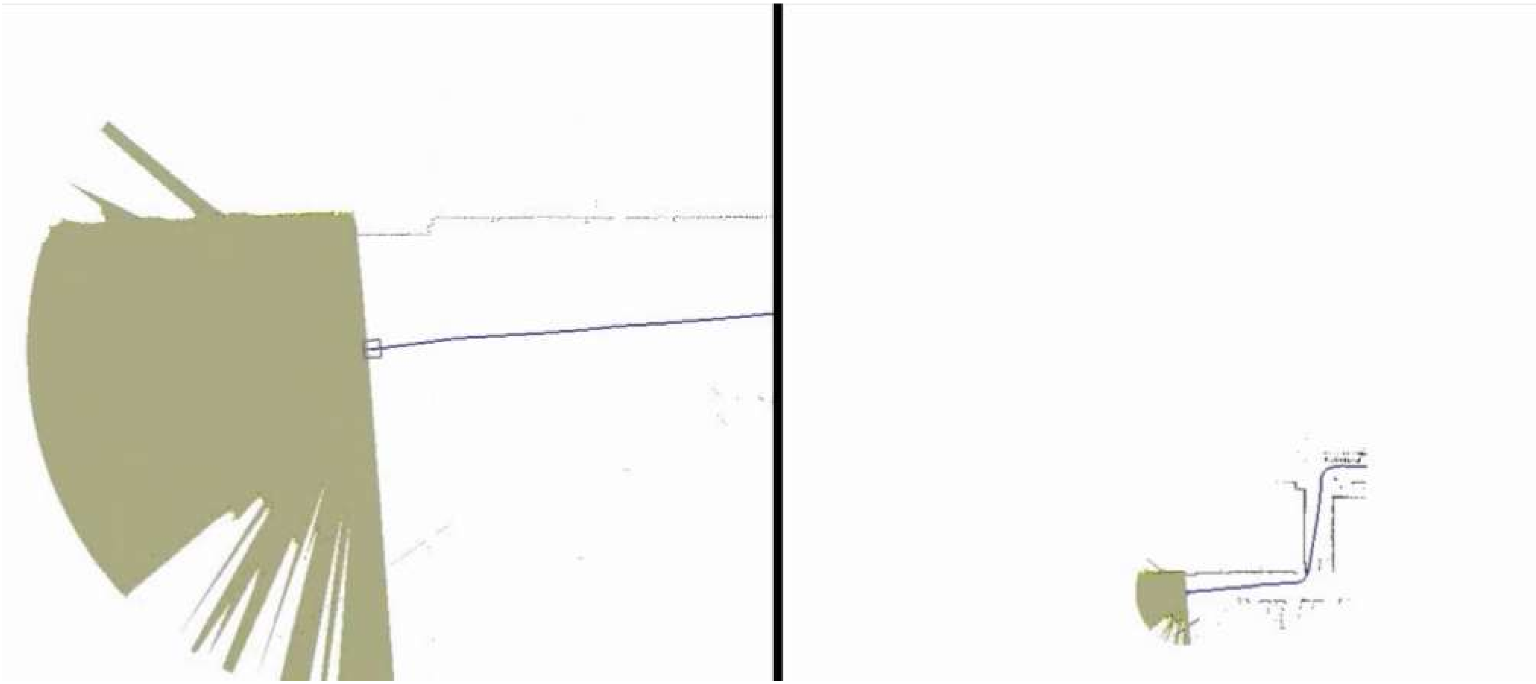
using a sequence of sensor measurements

these quantities are correlated



SLAM

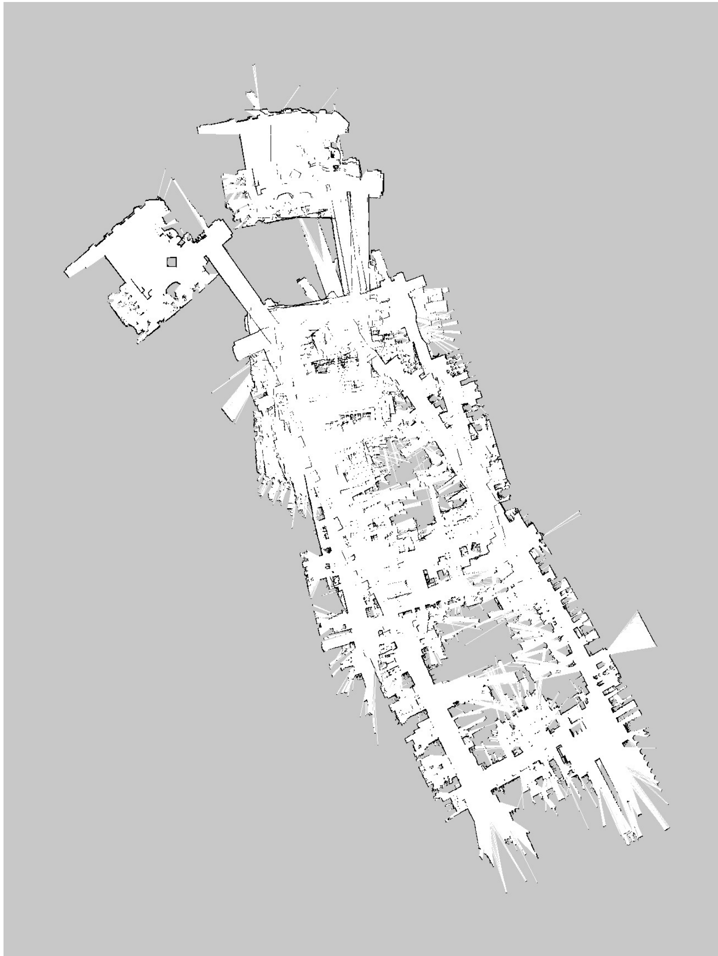
Determine the robot position AND the map, based on the sensor measurements



Graph-based SLAM

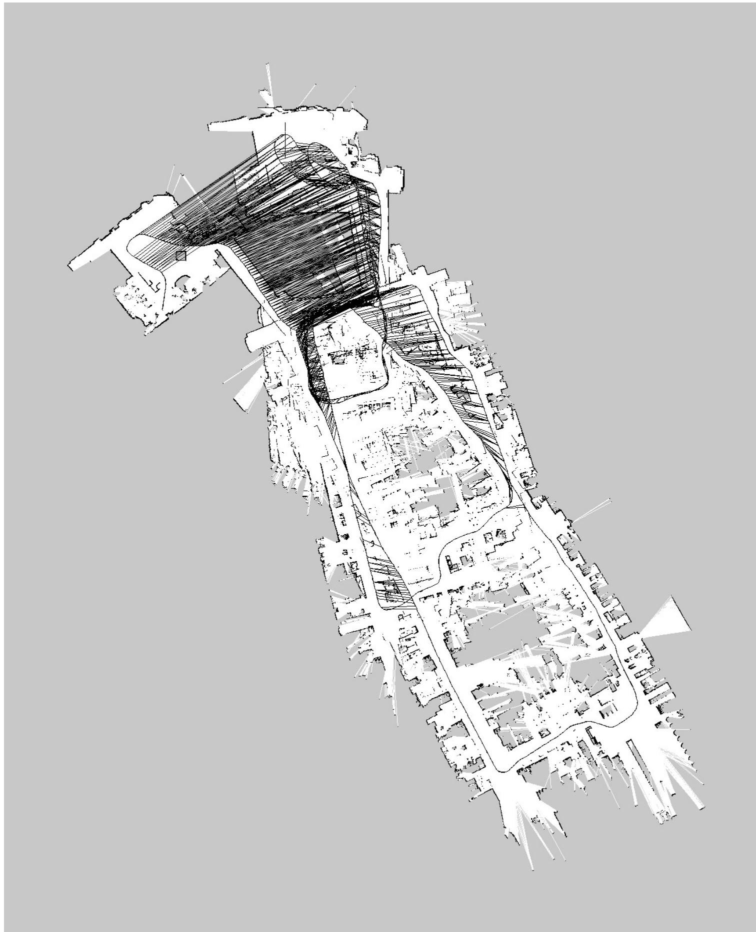
Problem described as a graph

Every node corresponds to a robot position and to a laser measurement



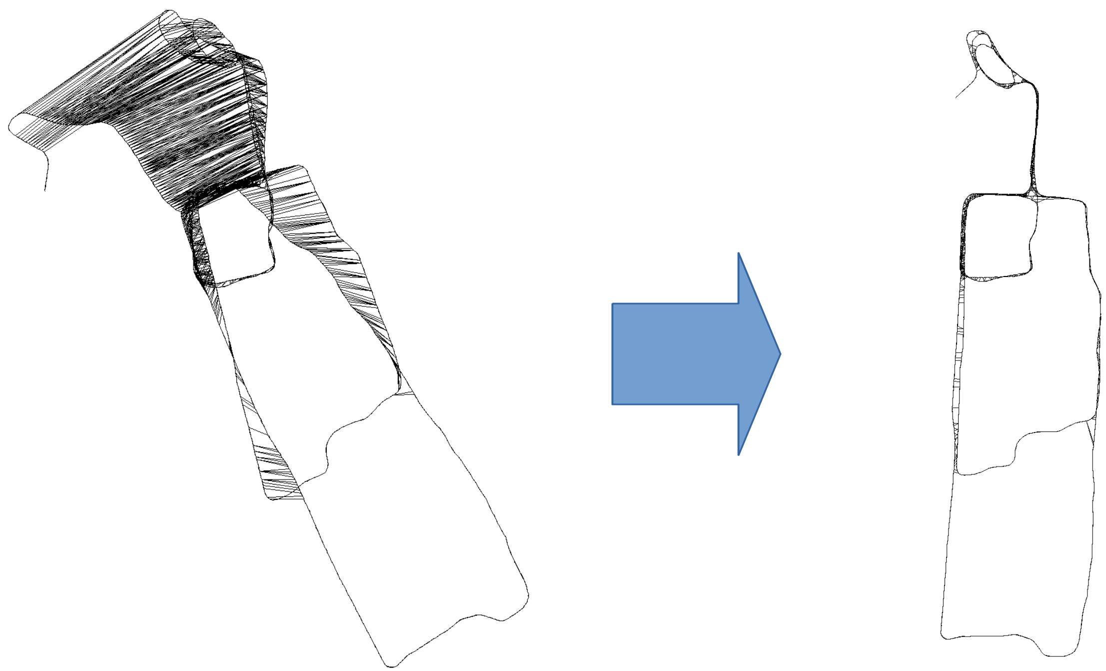
Graph-based SLAM

An edge between two nodes represents a data-dependent spatial constraint between the nodes



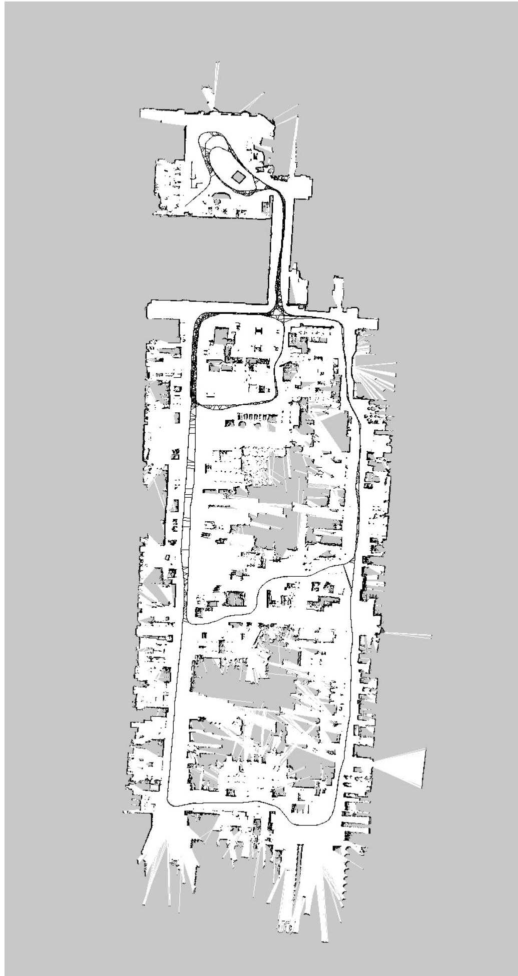
Graph-based SLAM

Once we have the
graph we determin
the most likely ma
by “moving” the
nodes



Graph-based SLAM

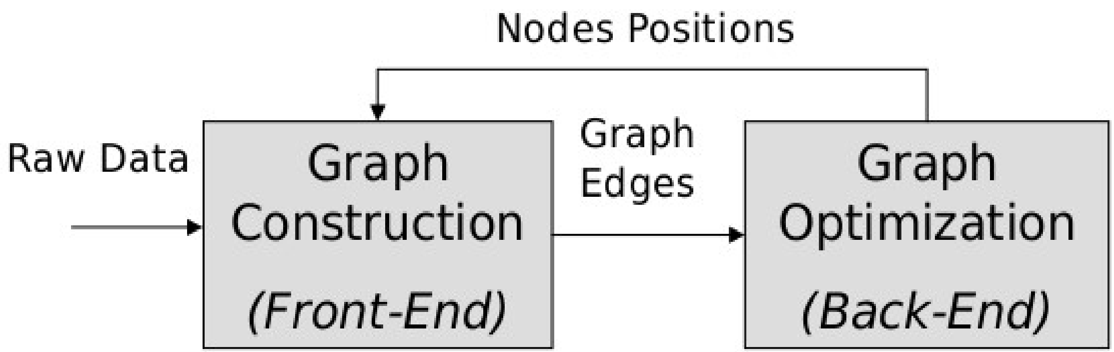
Then, we can render a map based on the known poses



Graph optimization

A general Graph-based SLAM algorithm interleaves the two steps

1. Graph construction
2. Graph optimization

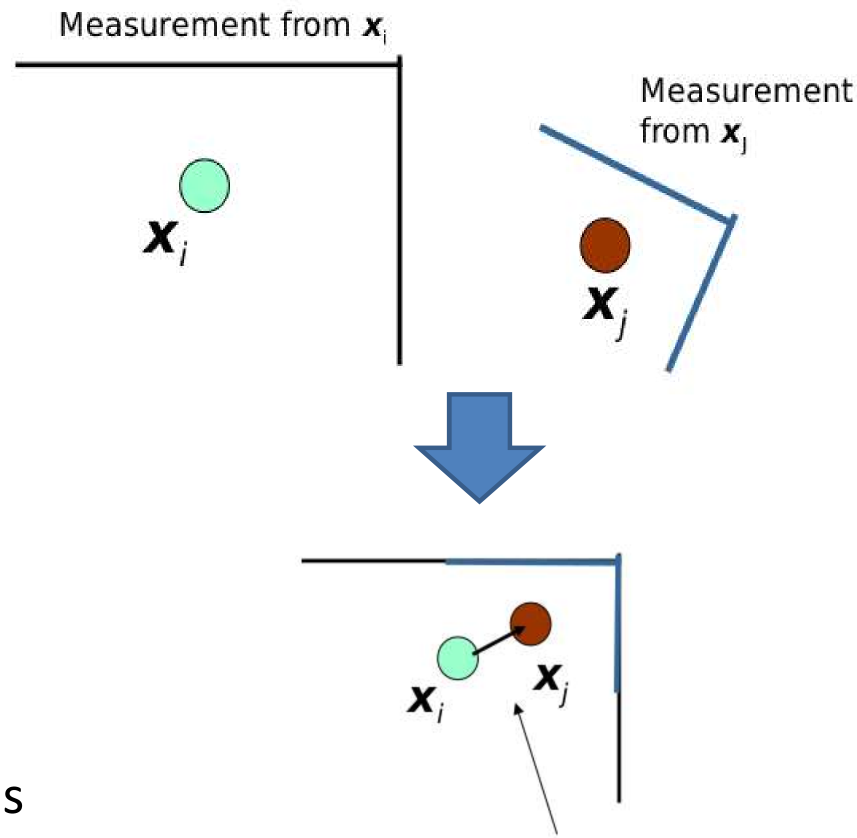


What Does the Graph Look Like?

Each node x_i is a 2D or 3D transformation representing the pose of the robot at time t_i

There is a constraint e_{ij} between the node x_i and the node x_j if

- **either**
  the robot observed the same part of the environment from both x_i and x_j and, via this common observation, it constructs a "virtual measurement" about the position of x_j
- or
  the positions are subsequent in time and there is an odometry measurement between the two



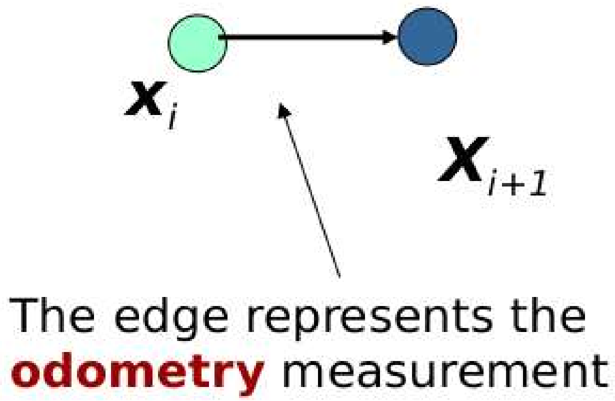
The edge represents the position of $\boldsymbol{x}_j$ seen from $\boldsymbol{x}_i$, based on the ***observations***

What Does the Graph Look Like?

Each node x_i is a 2D or 3D transformation representing the pose of the robot at time t_i

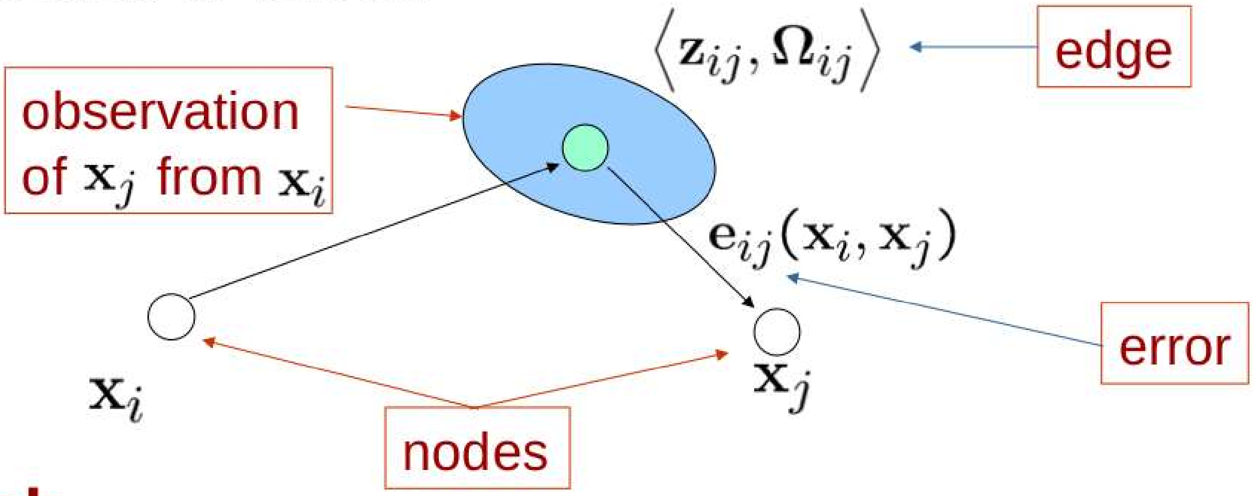
There is a constraint e_{ij} between the node x_i and the node x_j if

- either
  the robot observed the same part of the environment from both x_i and x_j and, via this common observation, it constructs a “virtual measurement” about the position of x_j
- **or**
  the positions are subsequent in time and there is an odometry measurement between the two



Pose graph

- The input for the optimization procedure is a graph annotated as follows:



- **Goal:**
  - Find the assignment of poses to the nodes of the graph which minimizes the negative log likelihood of the observations:

$$\hat{\mathbf{x}} = \operatorname{argmin} \sum_{ij} \mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$$

z_{ij} is a measurement of the robot pose j, performed from robot pose i

Ω_{ij} is a matrix to encode the uncertainty of the edge

Getting started - Navigation

To navigate a robot we need

1. A map
2. A localization module
3. A path planning module

These components are sufficient if

- ✓ The map fully reflects the environment
- ✓ The environment is static
- ✓ There are no errors in the estimate

Getting started - Navigation

However

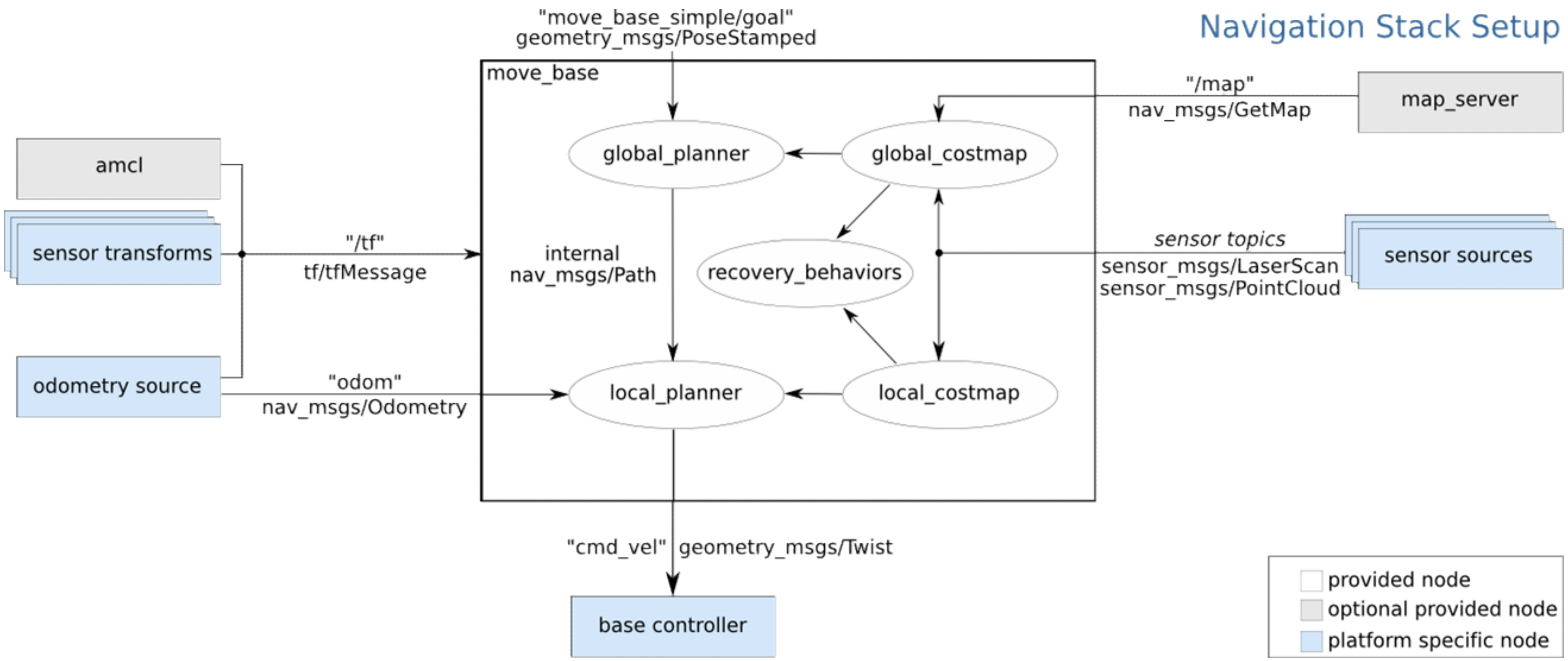
1. The environment changes (e.g. opening/closing doors)
2. It is dynamic (things might appear/disappear from the perception range of the robot)
3. The estimate is "noisy"

Thus we need to complement our ideal design with other components that address these issues, namely

1. Obstacle-Detection/Avoidance
2. Local Map Refinement, based on the most recent sensor reading

ROS navigation stack

- Map provided by a “Map Server”
- Each module is a node
- Planner has a layered architecture (local and global planner)
- Obstacle sensing refined on-line by appropriate modules (local and global costmap)



Building the map in ROS

- ROS uses GMapping, which implements a particle filter to track the robot trajectories

- To build a map you need to
  1. Record a bag with /odom, /scan/ and /tf while driving the robot around in the environment it is going to operate in
  2. Play the bag and the gmapping-node (see the ros wiki), and then save it

- The map is an occupancy map and it is represented as
  1. An image showing the blueprint of the environment
  2. A configuration file (yaml) that gives meta information about the map (origin, size of a pixel in real world)

Localizing the robot

ROS implements the Adaptive Monte Carlo Localization algorithm

1. AMCL uses a particle filter to track the position of the robot
2. Each pose is represented by a particle
3. Particles are
    - Moved according to (relative) movement measured by the odometry
    - Suppressed/replicated based on how well the laser scan fits the map, given the position of the particle

Creating a Map - turtlebot2 example

1. Create a folder for maps.

```
mkdir ~/turtlebot_custom_maps
```

2. Launch Gazebo world.

```
roslaunch turtlebot_gazebo turtlebot_world.launch
```

3. Start map building.

```
roslaunch turtlebot_gazebo gmapping_demo.launch
```

4. Use Rviz to visualize the map building process.

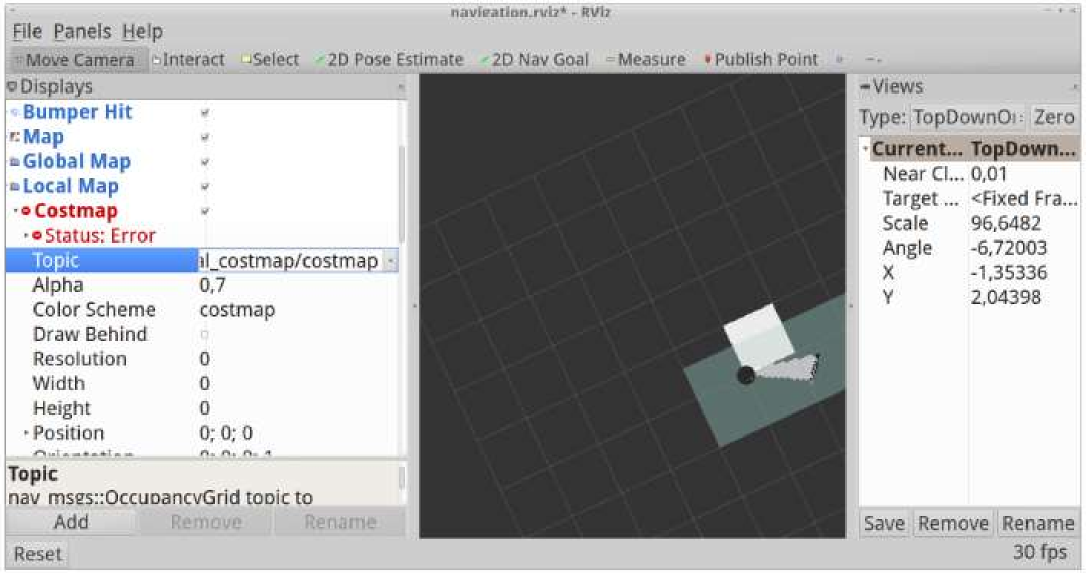
```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

Learn TurtleBot and ROS original web page here

Creating a Map - turtlebot2 example

5. Change the option.

`Local map` -> `Costmap` -> `Topic` (choose `/map` from drop-down list). See on the picture:



6. Change the option.

`Global map` -> `Costmap` -> `Topic` (choose `/map` from drop-down list).

Learn TurtleBot and ROS original web page here

Creating a Map - turtlebot2 example

7. Launch teleop.

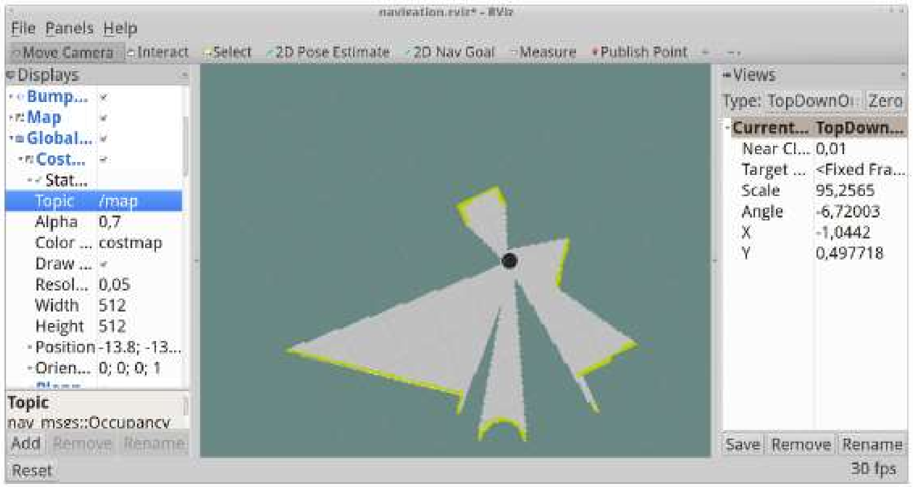
```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

NOTE: *If you want you can use other tools, for example interactive markers, find the information here.*

8. Drive the TurtleBot around.

NOTE: *The terminal with teleop launching has to be active all the time otherwise you won't be able to operate the TurtleBot.*

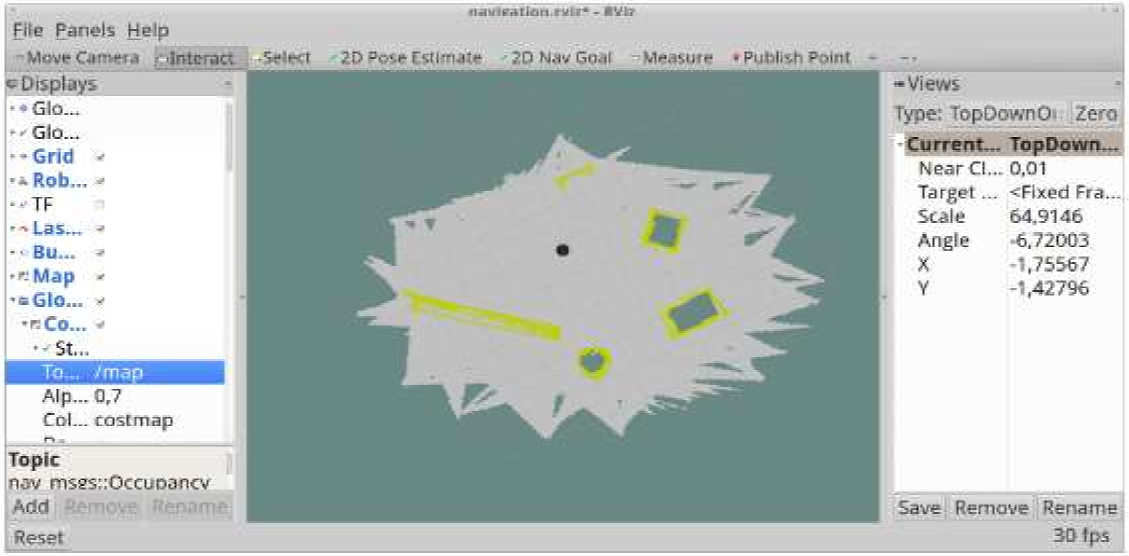
This is a picture of 360-degrees turn:



Learn TurtleBot and ROS original web page here

Creating a Map - turtlebot2 example

9. Save a map when your picture is good enough (like this).



Map

```
rosrun map_server map_saver -f /home/<user_name>/turtlebot_custom_maps/tutoria
```

10. Interrupt processes and close the terminals.

Learn TurtleBot and ROS original web page here

Navigation - turtlebot2 example

1. Launch Gazebo.

```
roslaunch turtlebot_gazebo turtlebot_world.launch
```

If you want to launch your own world run this command.

```
roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=<full path to the w
```

2. Run the navigation demo.

```
roslaunch turtlebot_gazebo amcl_demo.launch
```

If you have launched your own world or you want to use the map which you created in the previous lesson, specify a map file.

```
roslaunch turtlebot_gazebo amcl_demo.launch map_file:=<full path to map yaml file
```

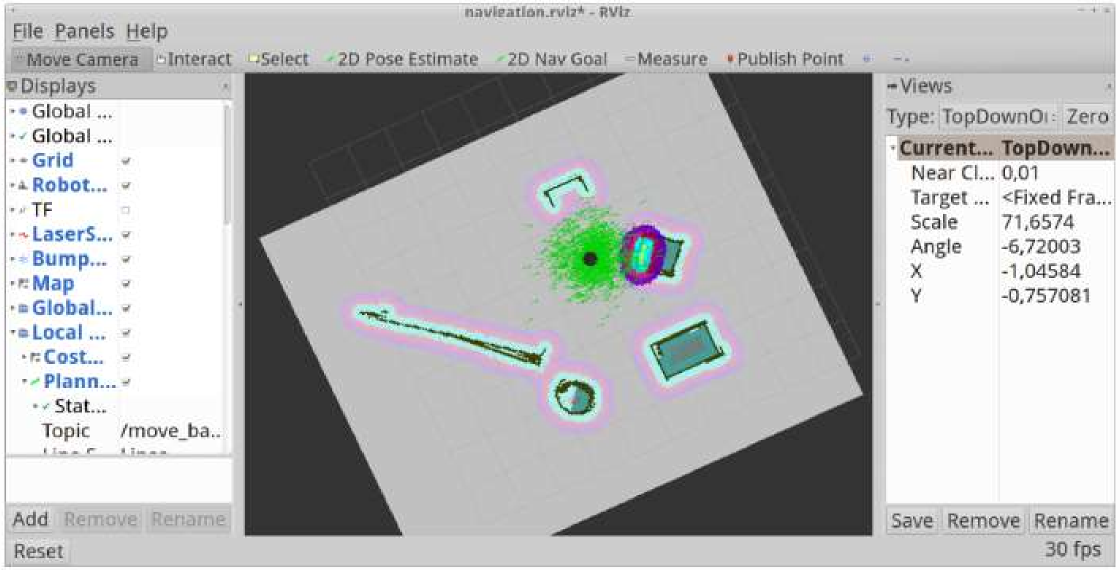
Learn TurtleBot and ROS original web page here

Navigation - turtlebot2 example

3. Launch Rviz.

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

You can see this picture.



Location of the TurtleBot on the map is already known. You will see a collection of arrows which show the position of the Turtlebot.

Learn TurtleBot and ROS original web page here

Navigation - turtlebot2 example

4. Send a navigation goal. Click the `2D Nav Goal` button.

5. Click on the map where you want the TurtleBot to drive and drag in the direction the Turtlebot should be pointing at the end.

NOTE: *If the path or goal is blocked it can fail.*

NOTE: *If you want to stop the TurtleBot before it reaches it's goal, send it a goal at it's current location.*

6. Interrupt processes and close the terminals.

Learn TurtleBot and ROS
original web page here

Esercizi

1. Provare a creare una mappa dell'ambiente cyber_lab scaricabile da https://github.com/dbloisi/cyber_lab_gazebo

2. Utilizzare il turtlebot2 per navigare atonomamente nel mondo cyber_lab

UNIVERSITÀ di VERONA

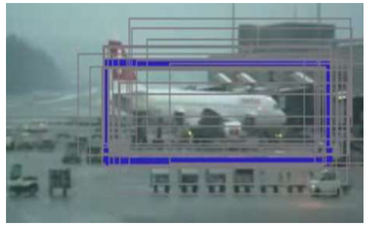
Dipartimento di INFORMATICA

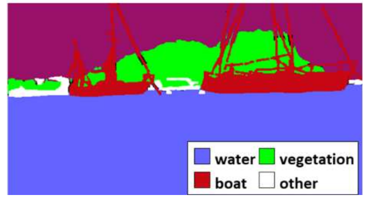
Laurea magistrale in ingegneria e scienze informatiche

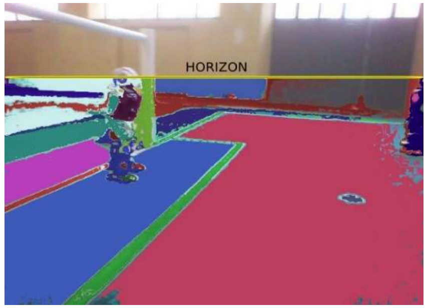
Introduzione alla navigazione in ROS

Corso di Robotica
Parte di Laboratorio

Docente:
Domenico Daniele Bloisi







Dicembre 2017