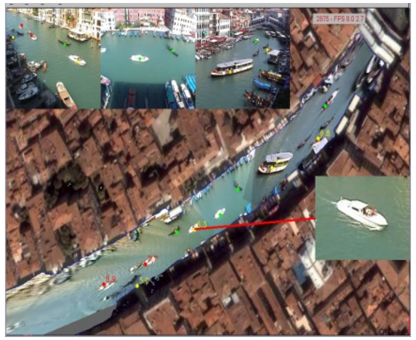
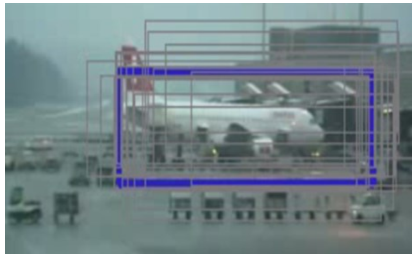
UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

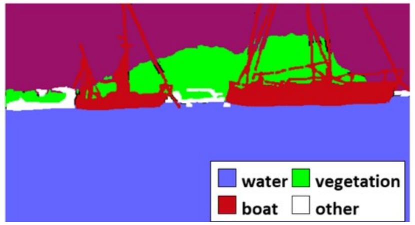
Corso di Laboratorio Ciberfisico
Modulo di Robot Programming with ROS



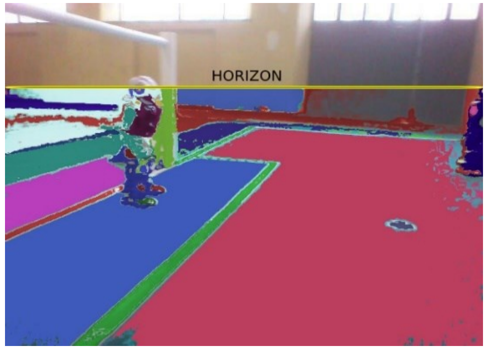


Navigazione in ROS

Docente:
Domenico Daniele Bloisi



Maggio 2018



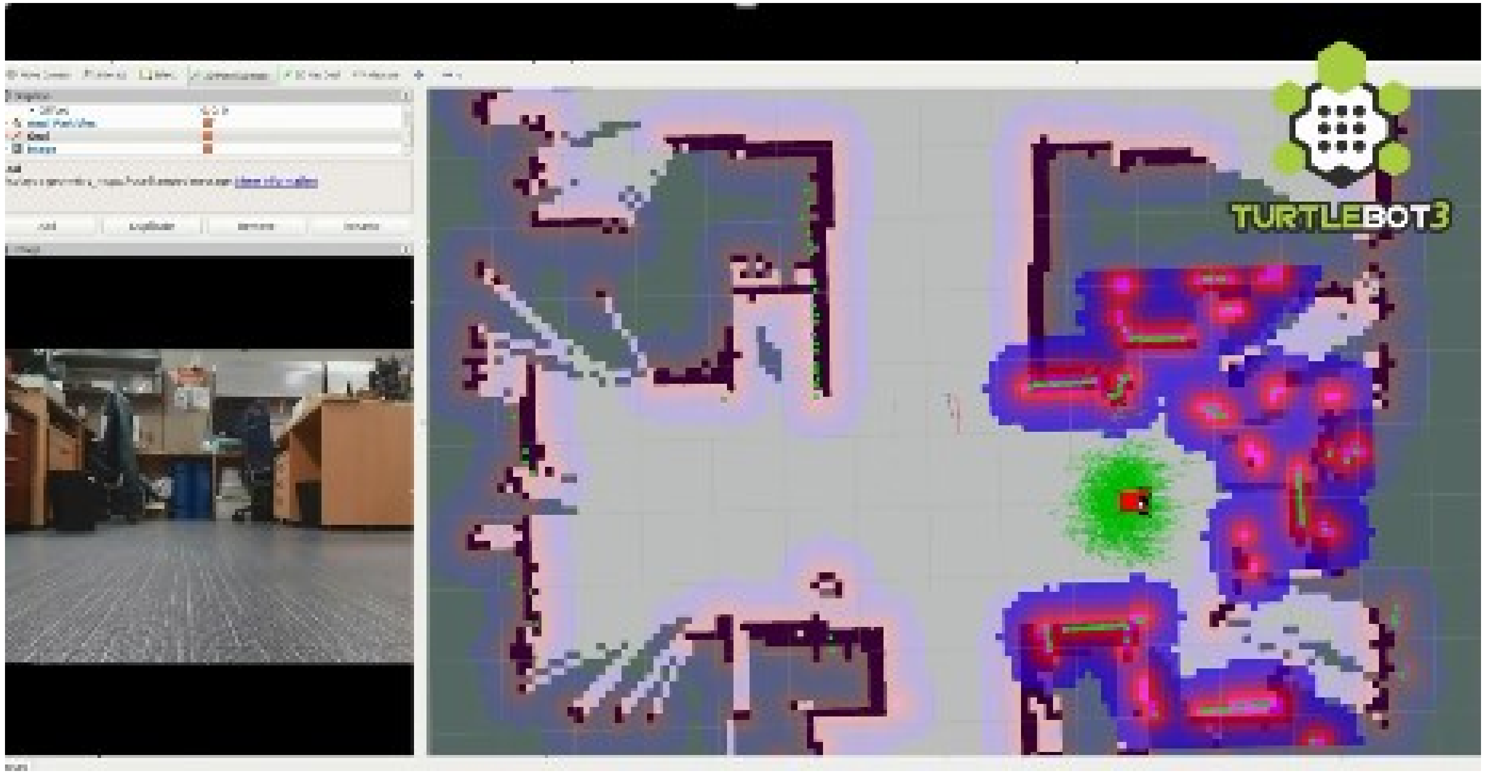
References and credits

Alcune delle slide seguenti sono tratte da

- Giorgio Grisetti, *“Introduction to Navigation using ROS”*
- Giorgio Grisetti, *“Introduction”* in Probabilistic Robotics Course
- Giorgio Grisetti, *“Multi-Pose Registration Graph-SLAM” in* Probabilistic Robotics Course
- YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim, *“ROS Robot Programming - A Handbook Written by TurtleBot3 Developers”* http://www.robotis.com/service/download.php?no=719
- Learn TurtleBot and ROS (http://learn.turtlebot.com/)
  - Creating a Map
  - Autonomous Navigation

Navigazione con robot mobili

Il principale task che un robot autonomo mobile deve essere in grado di compiere è quello di saper muoversi nell’ambiente operativo

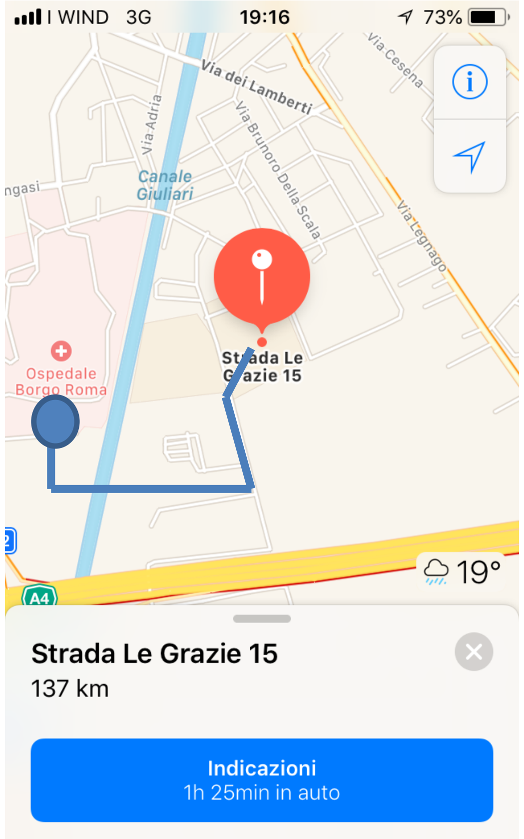


https://www.youtube.com/watch?v=lOZmFC79S6A

Navigazione GPS

Il navigatore, che utilizziamo nella vita di tutti i giorni, ci fornisce tre elementi di base:

1. una mappa
2. la nostra posizione sulla mappa
3. una rotta per la destinazione desiderata

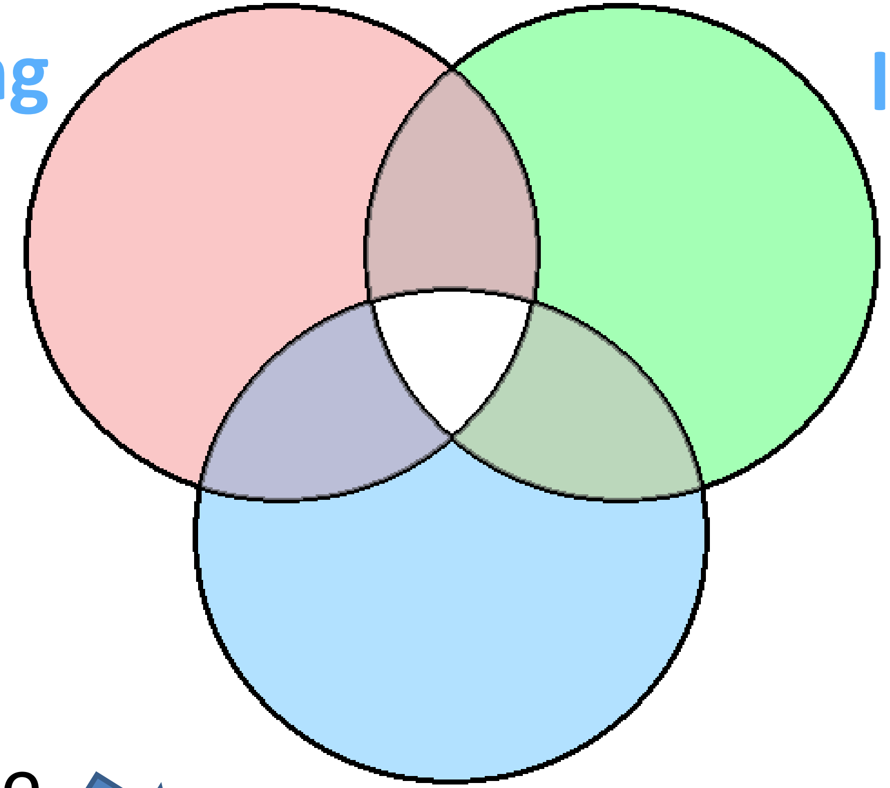


Questi tre elementi sono necessari per muoversi con successo nell’ambiente

Sono sufficienti?

Mapping, localization, planning

mappa → **mapping**



localization

↑

la nostra posizione sulla mappa

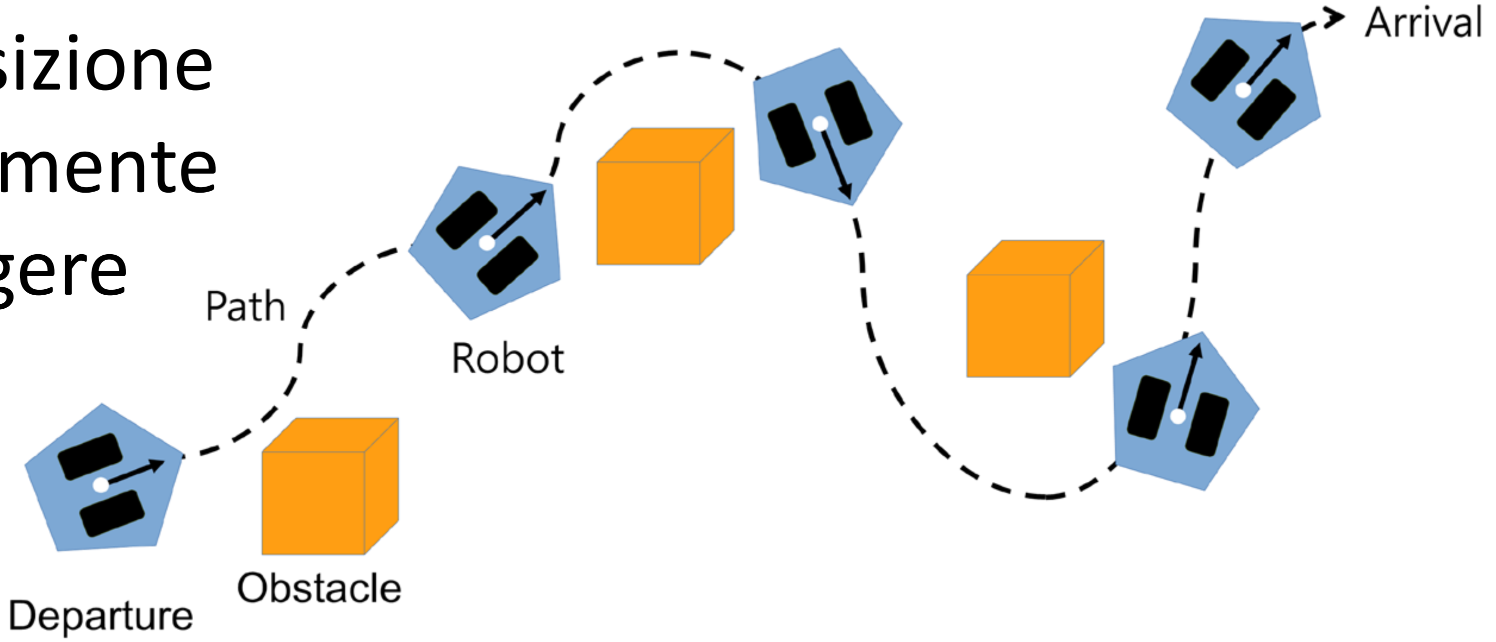
una rotta per la destinazione desiderata → **path planning**

Navigation

Con il termine navigation indichiamo il movimento del robot verso una destinazione predefinita

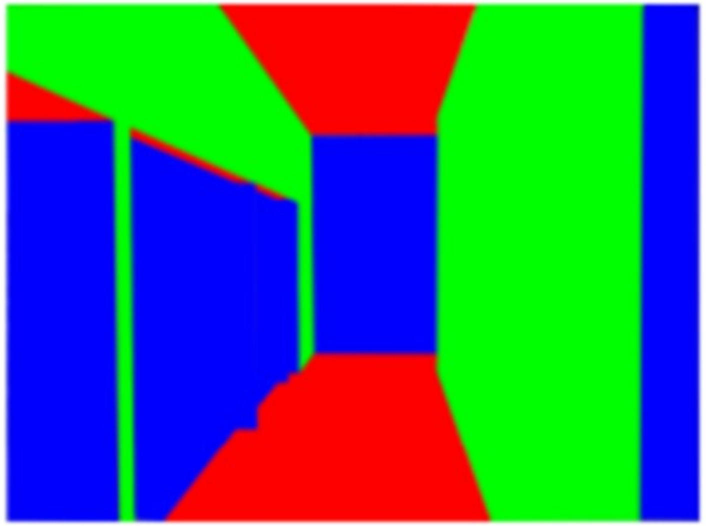
Per poter navigare, un robot ha bisogno di:

- Avere una mappa dell'ambiente
- Conoscere la propria posizione
- Avere una rotta (possibilmente ottimizzata) per raggiungere la destinazione
- Evitare gli ostacoli presenti sul percorso



Ambiente

Strutturato



Non strutturato

Ostacoli fissi e mobili

In base all’ambiente operativo in cui il robot si trova ad agire, si avranno

- ostacoli fissi
  muri e scale sono esempi di ostacoli fissi

- ostacoli mobili
  persone e sedie sono esempi di ostacoli mobili

Basic features

Affinché il robot sia in grado di navigare autonomamente nell’ambiente, avremo bisogno di

❶ Map

❷ Pose of Robot

❸ Sensing

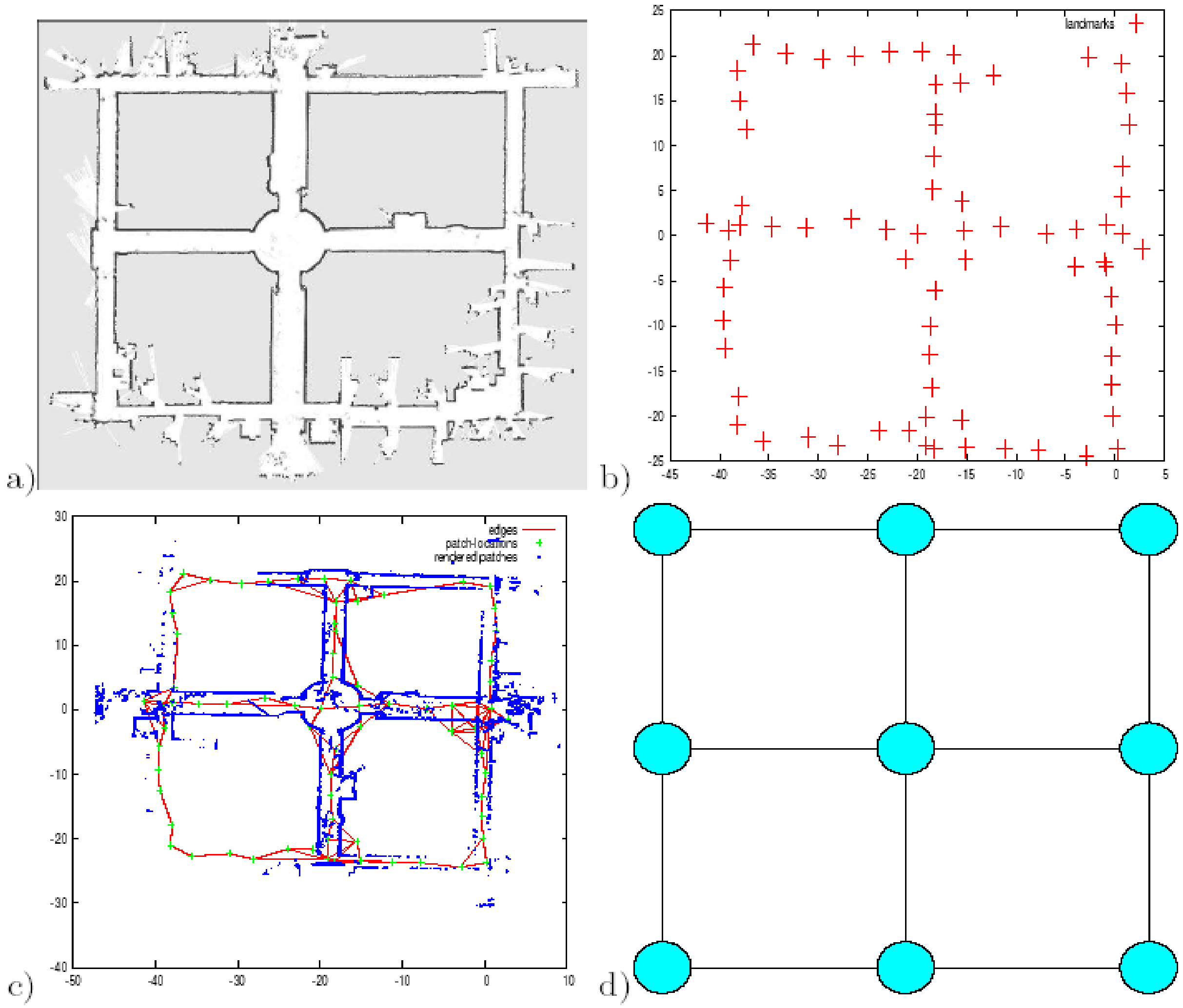
❹ Path Calculation and Driving

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:

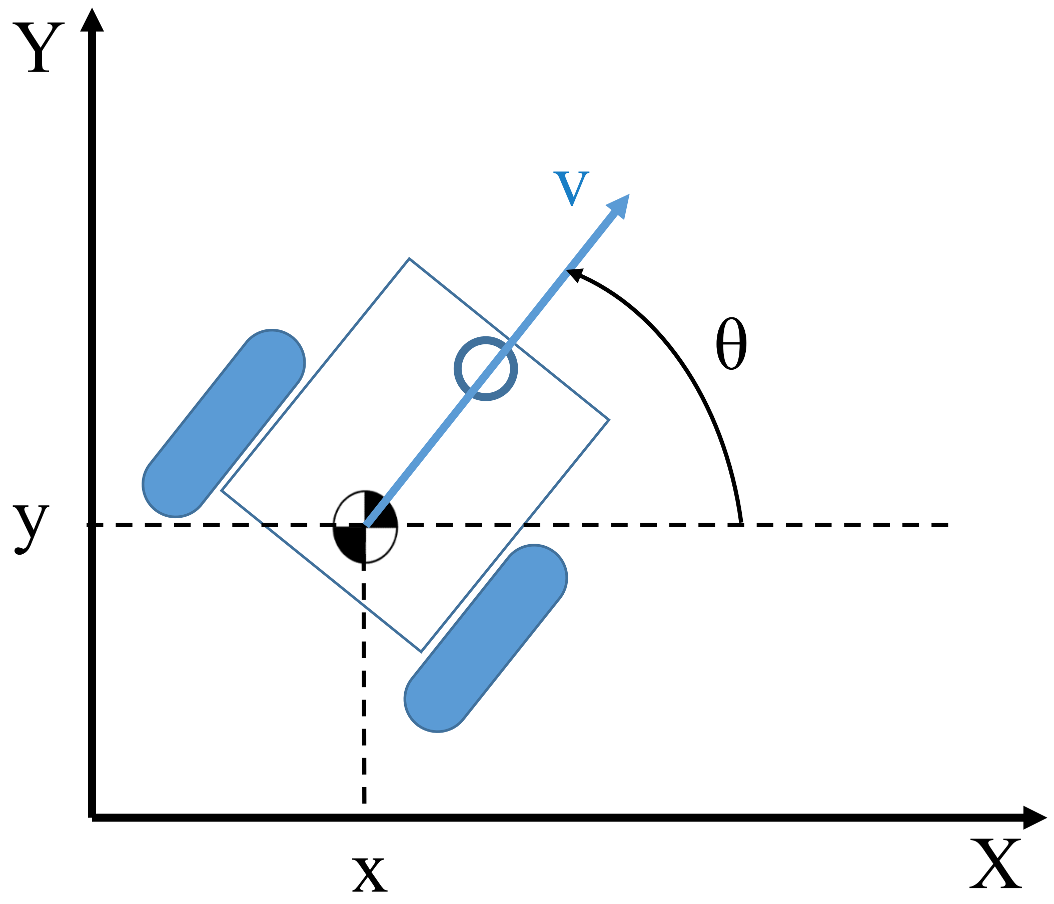
- Metric
  - Grid Based
  - Feature Based
  - Hybrid
- Topological
- Hybrid



Robot pose

- La *robot pose* è definita come la posizione del robot e la sua orientazione in un dato sistema di riferimento

- Per un robot mobile che si muove su un piano, la *pose* è definita dalla tripla [x, y, θ]



Pose in ROS

geometry_msgs/Pose Message

File: geometry_msgs/Pose.msg

Raw Message Definition

```
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

Compact Message Definition

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

http://docs.ros.org/api/geometry_msgs/html/msg/Pose.html

Position in ROS

pose = position + orientation

geometry_msgs/Point Message

File: geometry_msgs/Point.msg

Raw Message Definition

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Compact Message Definition

```
float64 x
float64 y
float64 z
```

Orientation in ROS

pose = position + orientation

geometry_msgs/Quaternion Message

File: geometry_msgs/Quaternion.msg

Raw Message Definition

```
# This represents an orientation in free space in quaternion form.

float64 x
float64 y
float64 z
float64 w
```

Compact Message Definition

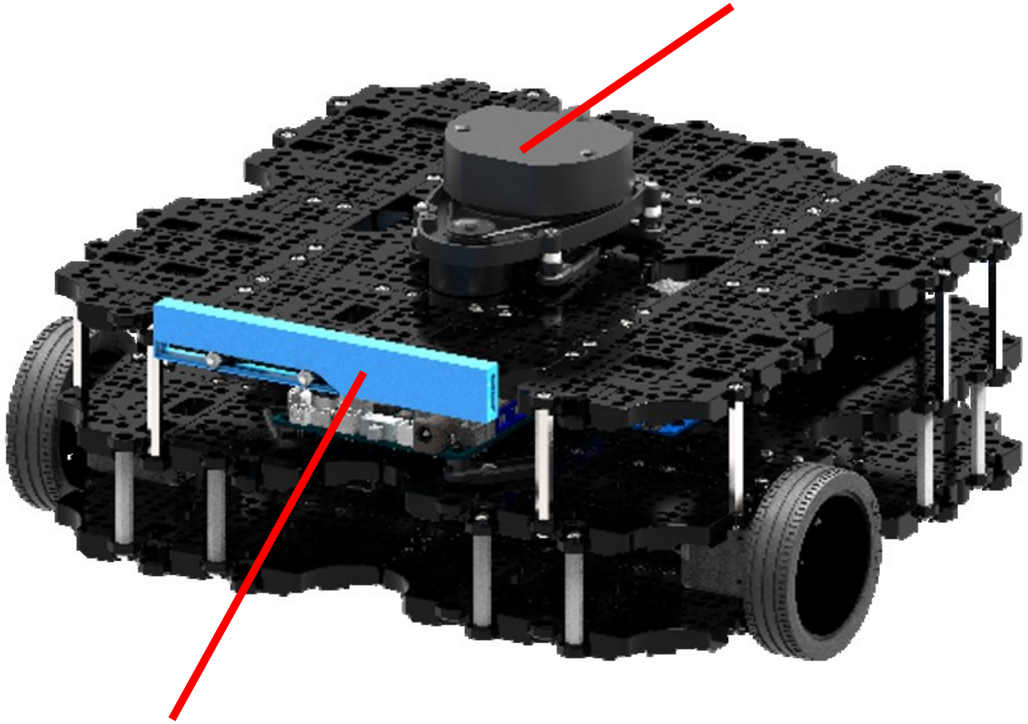
```
float64 x
float64 y
float64 z
float64 w
```

http://wiki.ros.org/tf2/Tutorials/Quaternions

http://run.usc.edu/cs520-s12/quaternions/quaternions-cs520.pdf

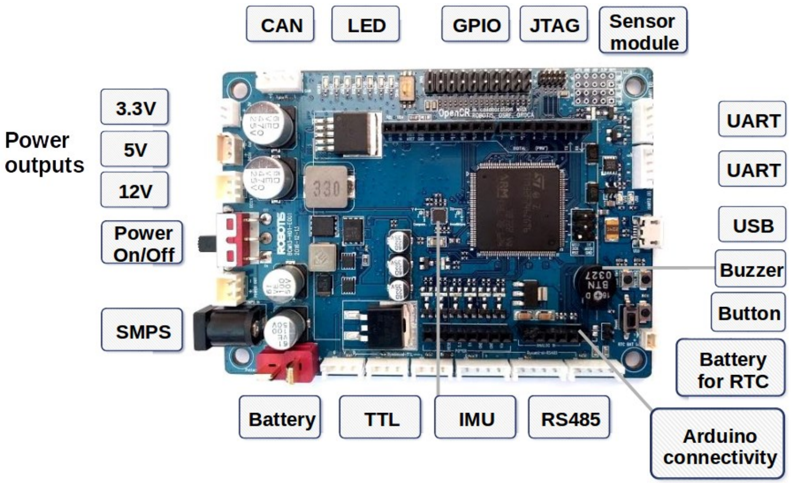
Sensing – Turtlebot3

360 Laser Distance Sensor LDS-01



Intel® Realsense™ R200

OpenCR1.0



Gyroscope 3Axis, Accelerometer 3Axis, Magnetometer 3Axis

http://emanual.robotis.com/docs/en/platform/turtlebot3/hardware_setup/#hardware-setup

Sensori di distanza

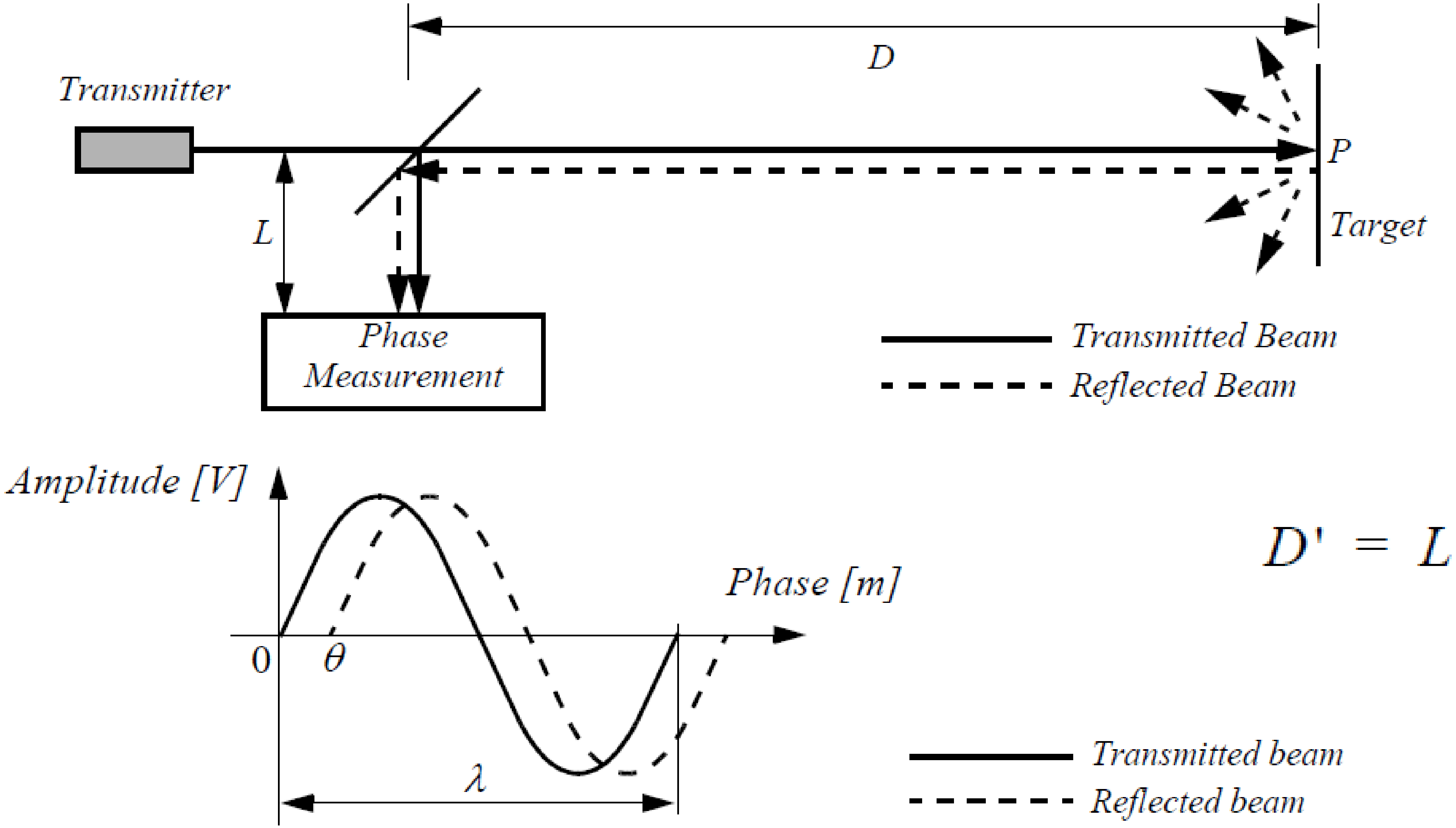
- Sonar

- Laser range finder
- Time of Flight Camera

I sensori di distanza basati sul time-of-flight sfruttano la velocità di propagazione del suono o delle onde elettromagnetiche per calcolare la distanza

Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

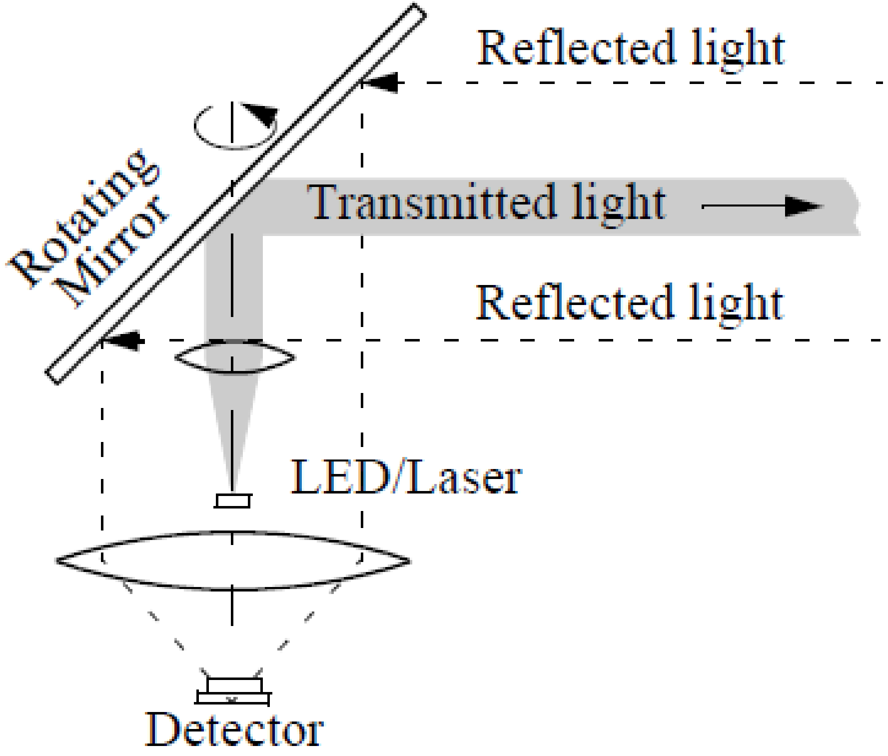
Laser Range Finder

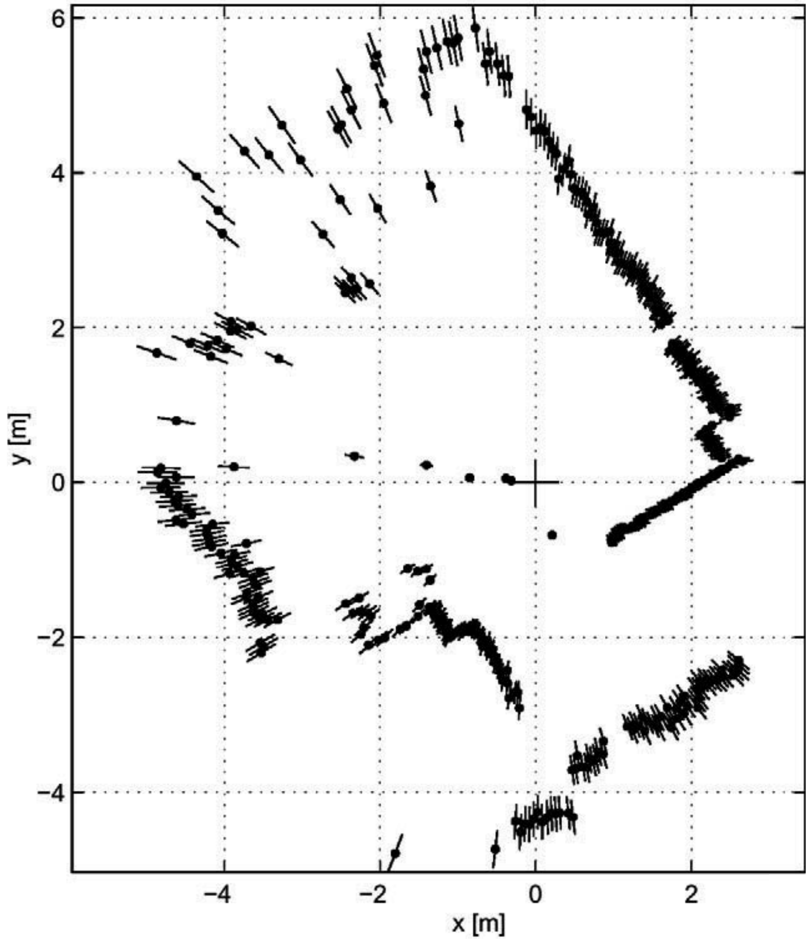


$$D' = L + 2D = L + \frac{\theta}{2\pi}\lambda$$

Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

laser range sensor with rotating mirror





Introduction to Autonomous Mobile Robots
Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza

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

Mapping

Mapping Problem.

Given

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

Determine the map of the environment

However,

- a perfect estimate of the robot pose is usually not available

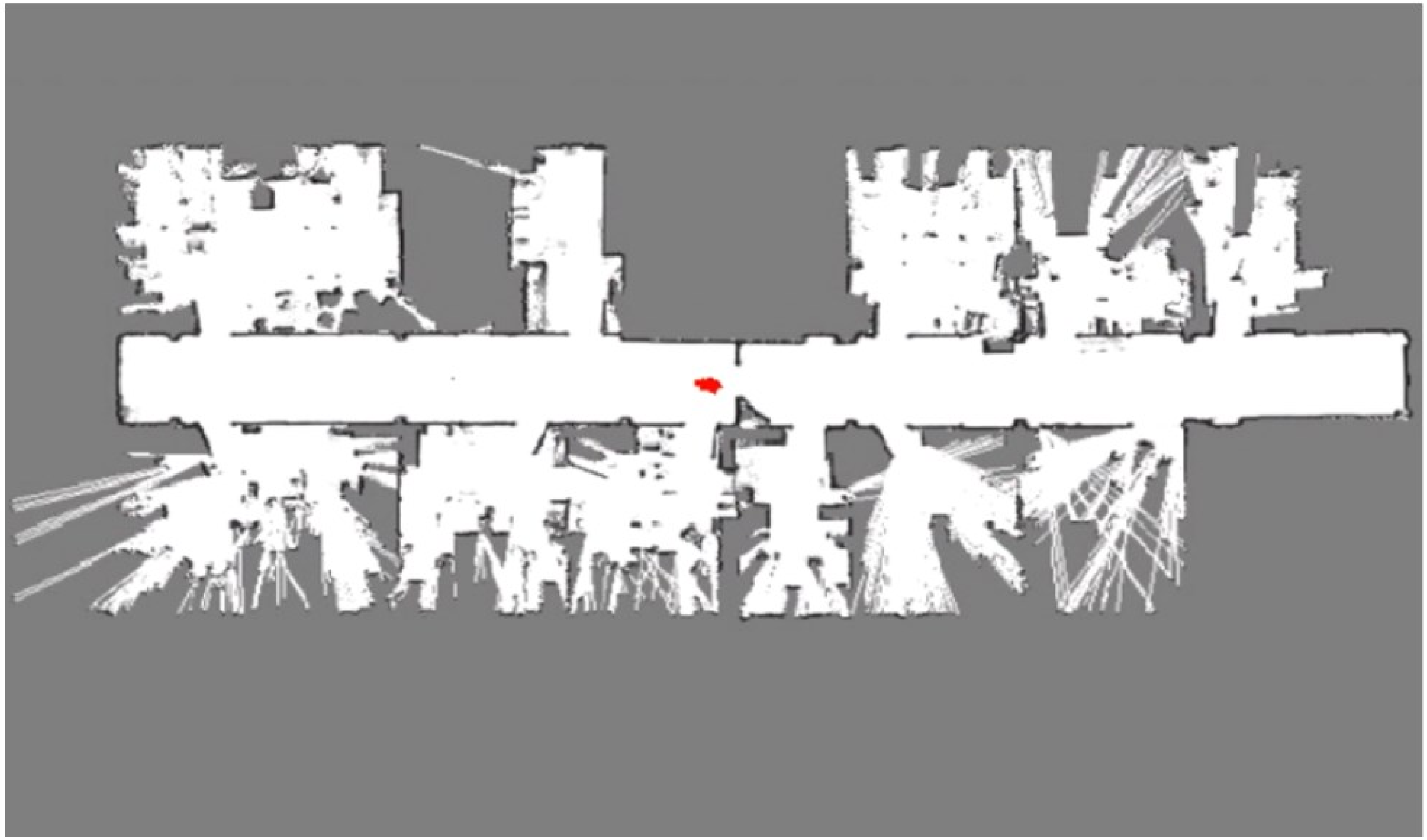
Usually, we solve a more complex problem:

Simultaneous Localization and Mapping (SLAM)

Localization

Determining the current position of a robot, given

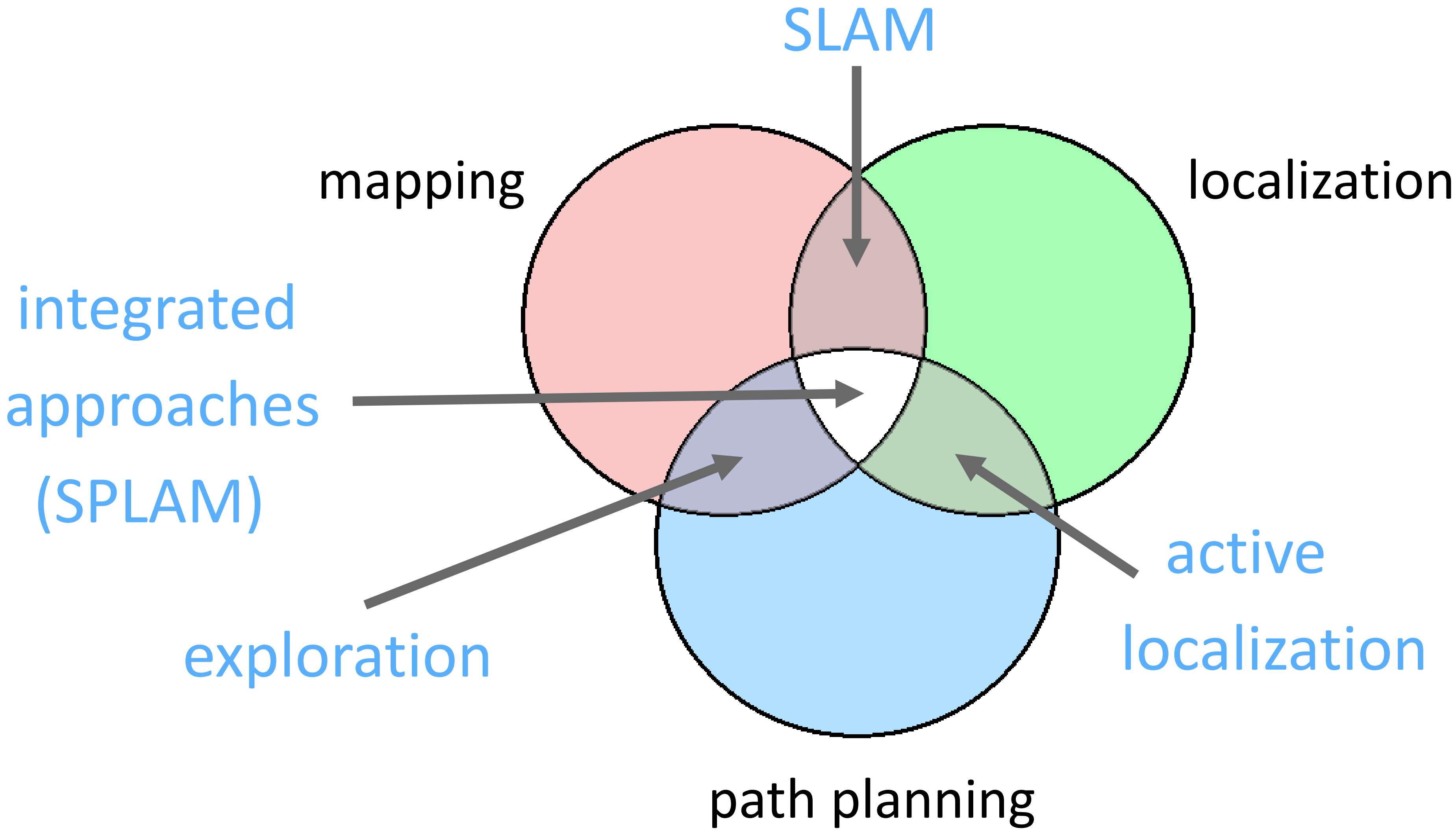
1. The knowledge of the map
2. All sensor measurements up to the current time



Path planning

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

Mapping, localization, planning



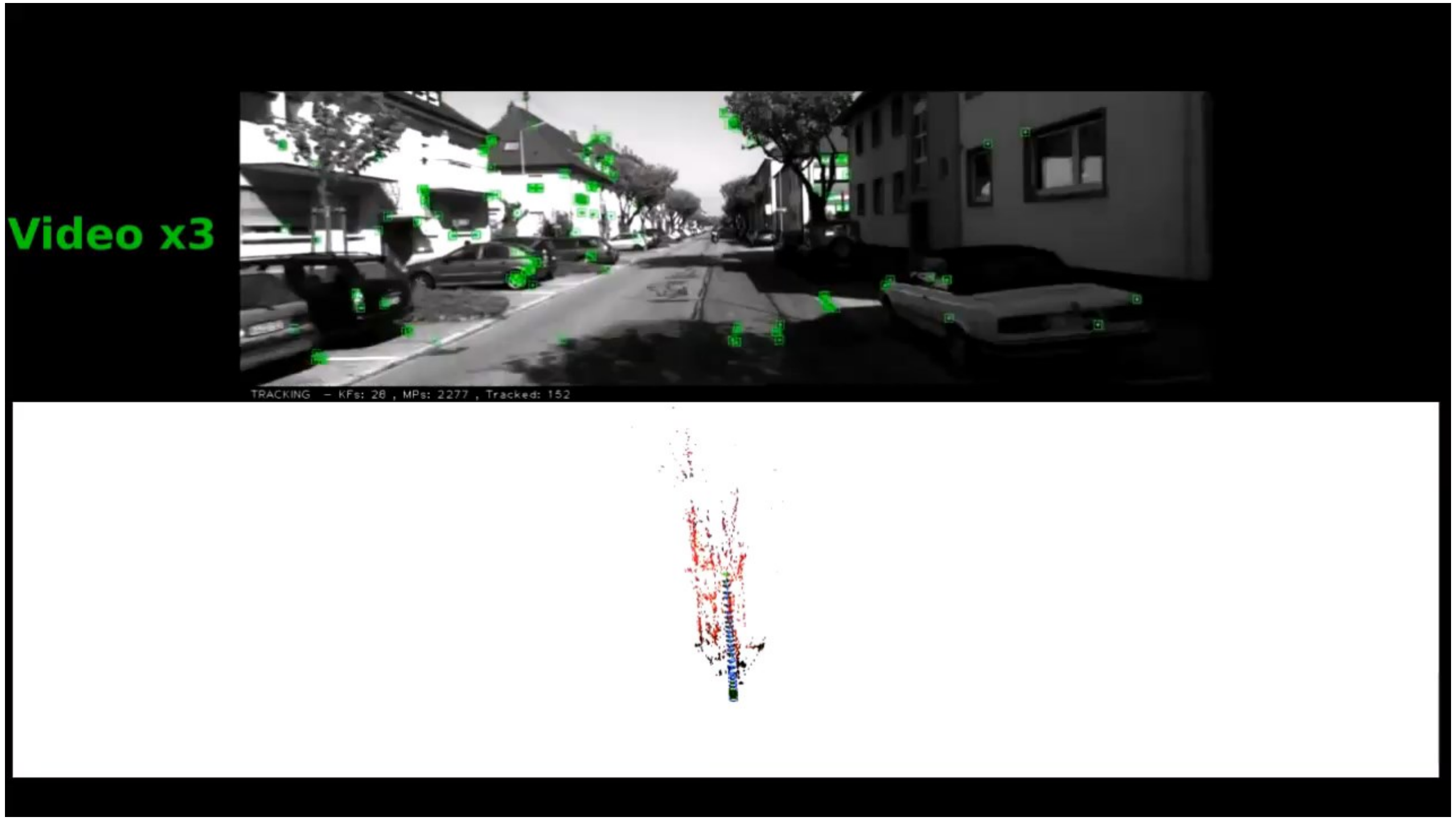
Simultaneous Localization and Mapping

Estimate:

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

using a sequence of sensor measurements

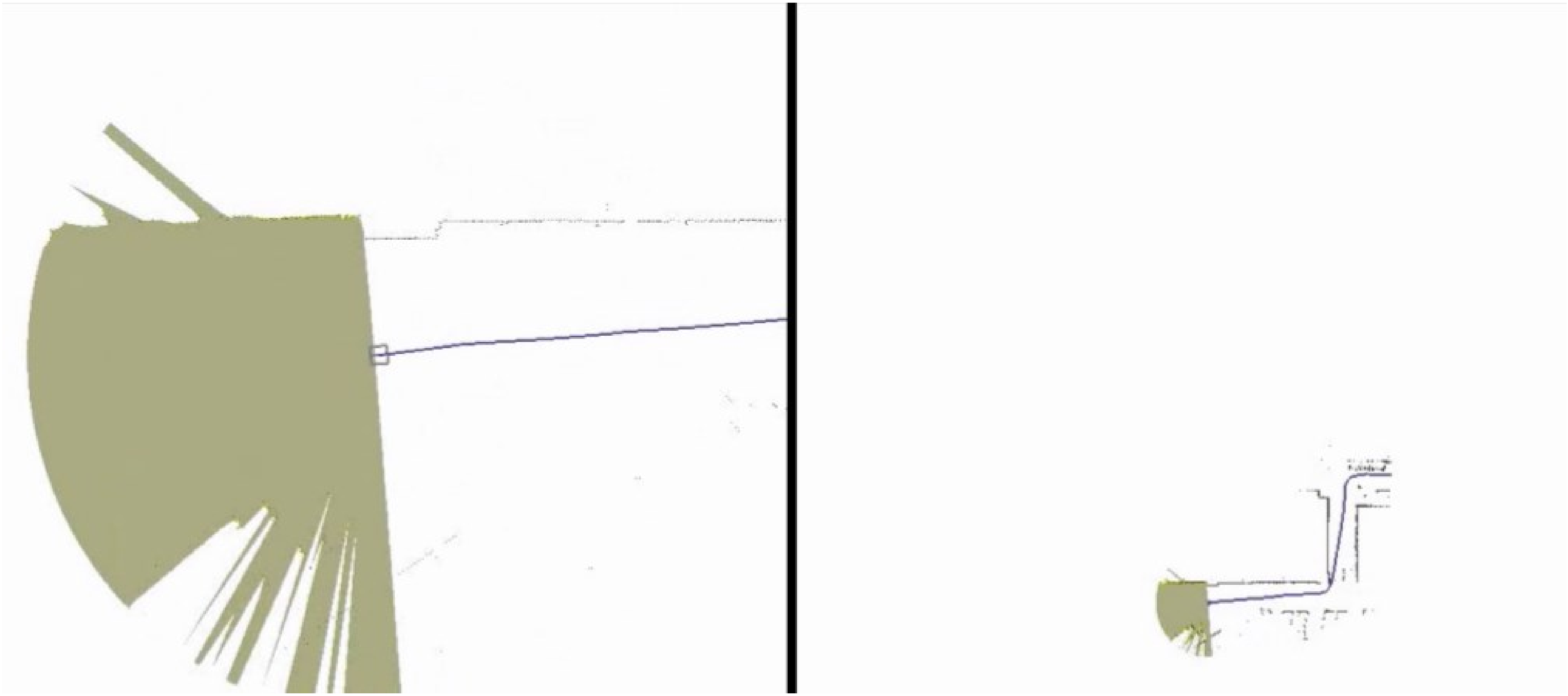
these quantities are correlated



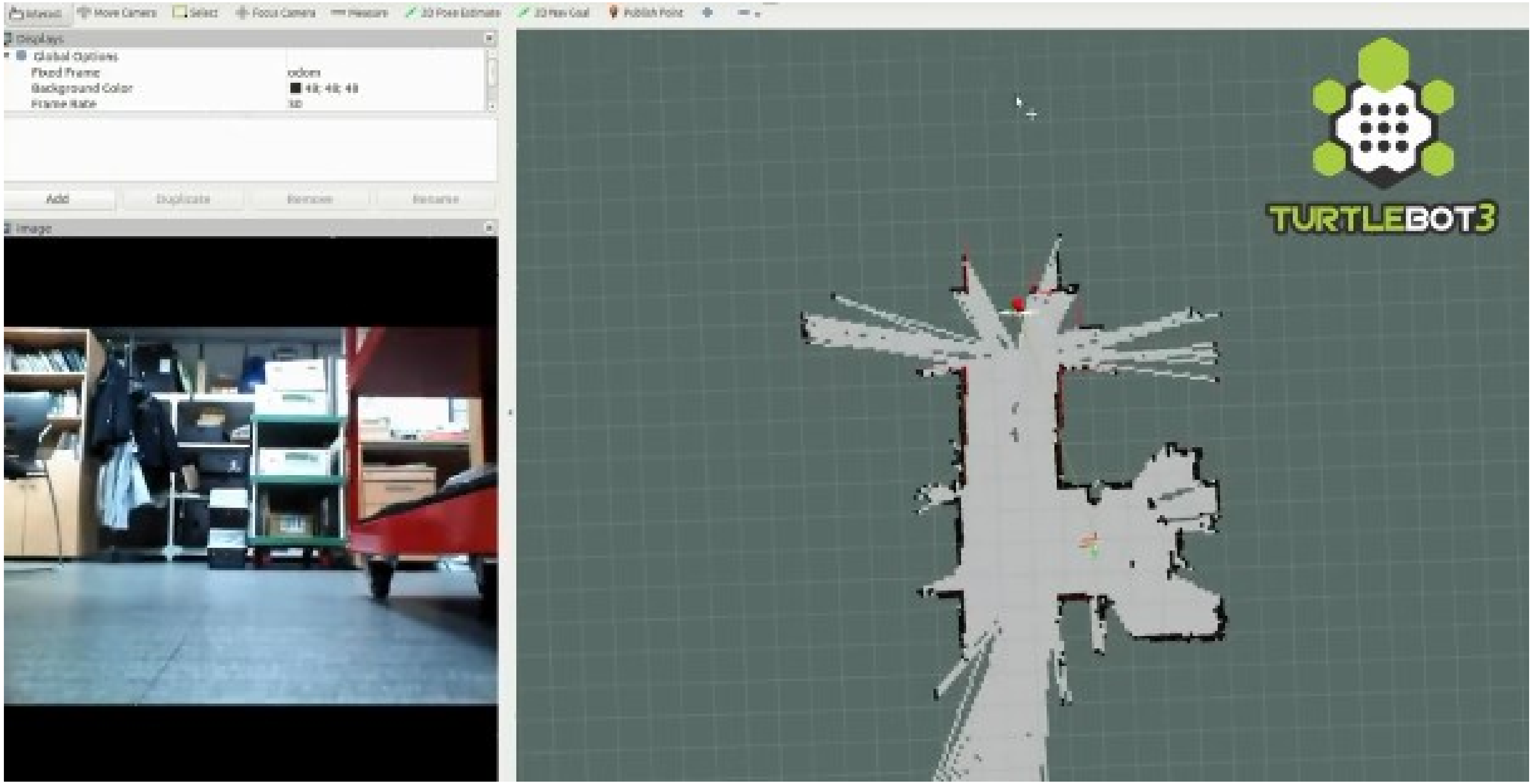
https://www.youtube.com/watch?v=8DISRmsO2YQ

SLAM

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



SLAM – Turtlebot3

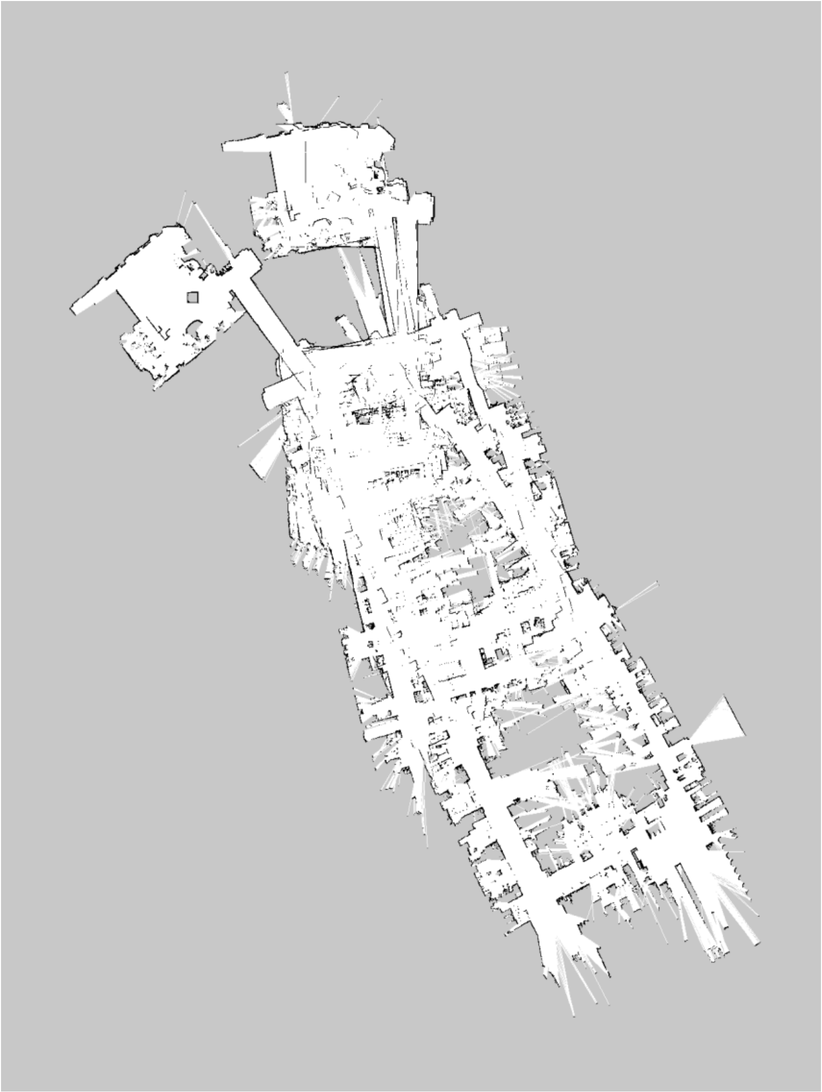


https://www.youtube.com/watch?v=hX6pFcfr29c

Graph-based SLAM

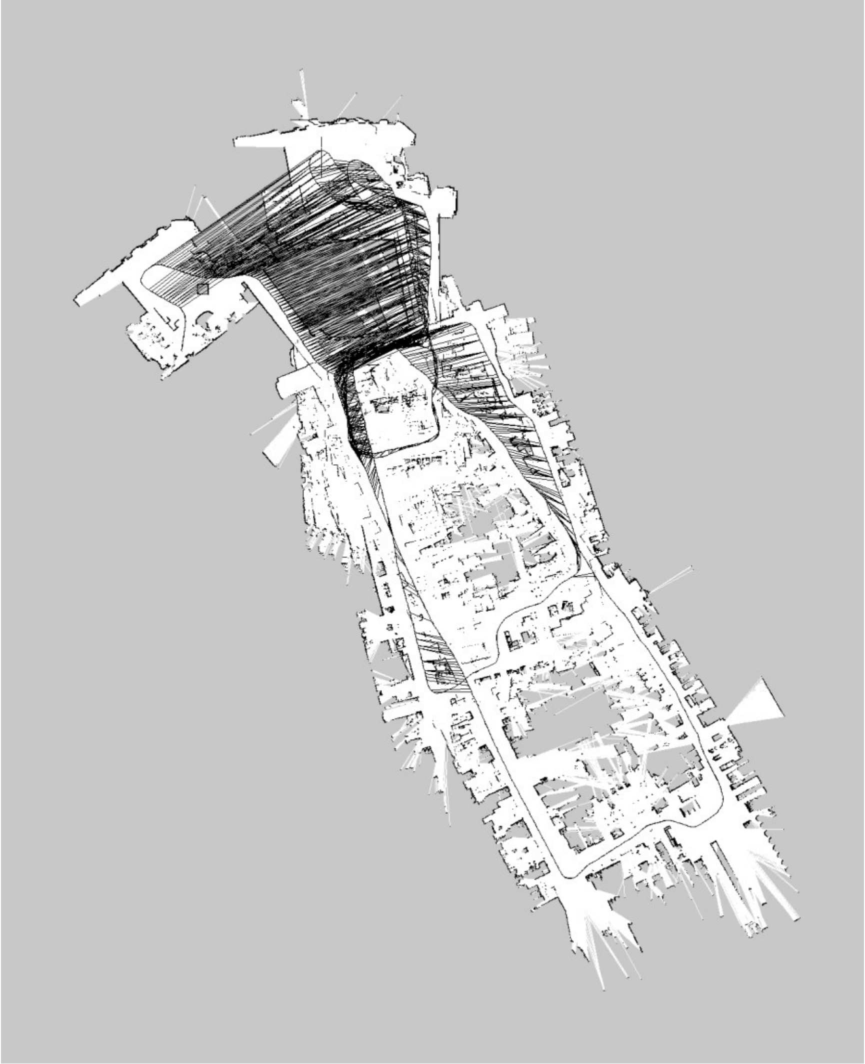
Problem described as a graph

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



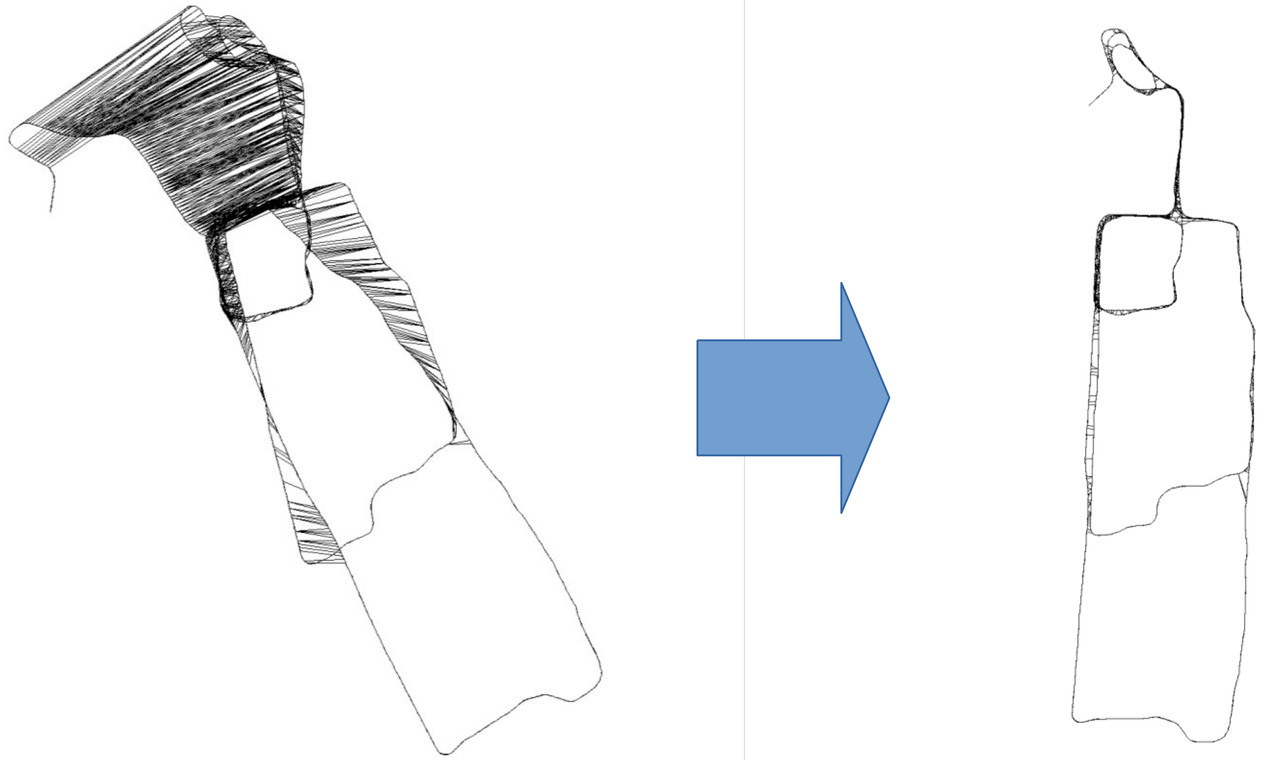
Graph-based SLAM

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



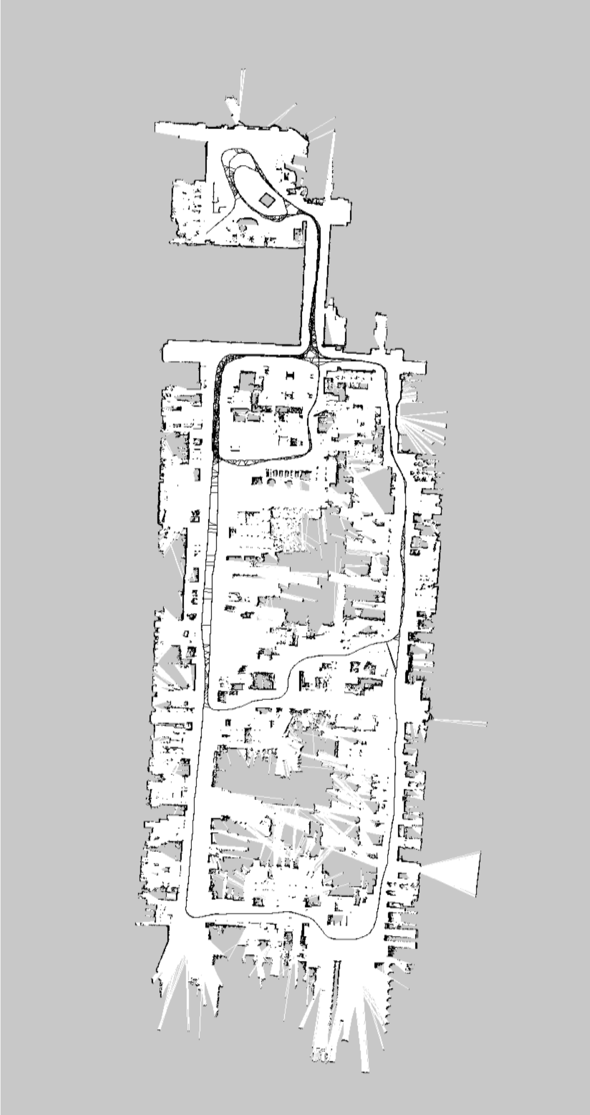
Graph-based SLAM

Once we have the graph we determine the most likely map by “moving” the nodes



Graph-based SLAM

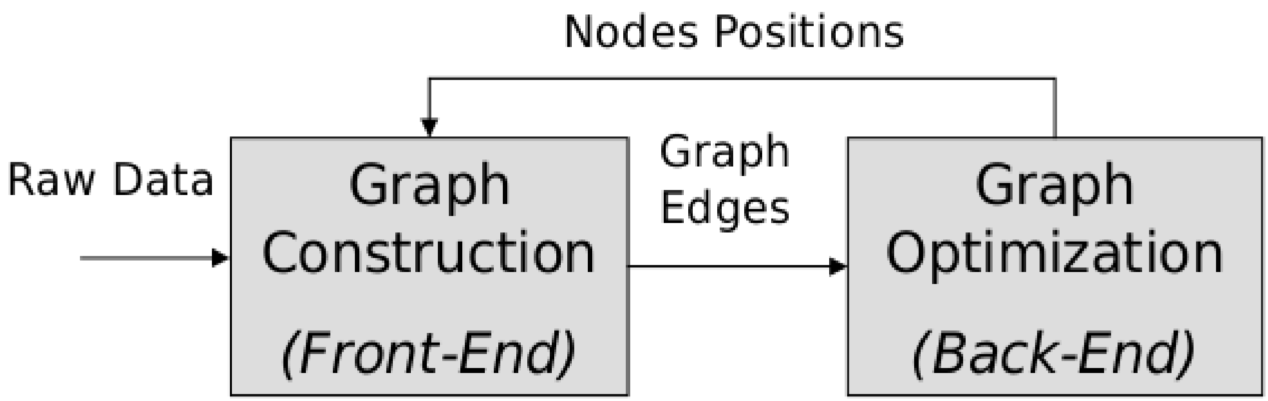
Then, we can render a map based on the known poses



Graph optimization

A general Graph-based SLAM algorithm interleaves the two steps

1. Graph construction
2. Graph optimization

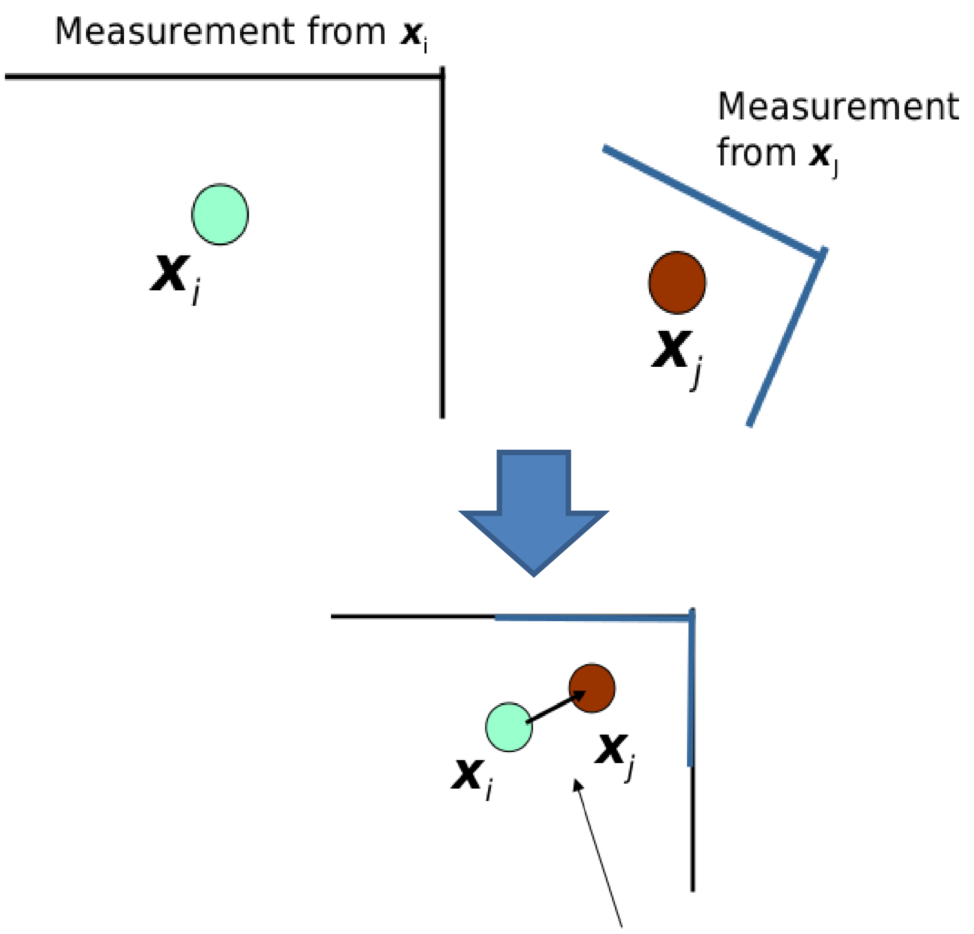


What Does the Graph Look Like?

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



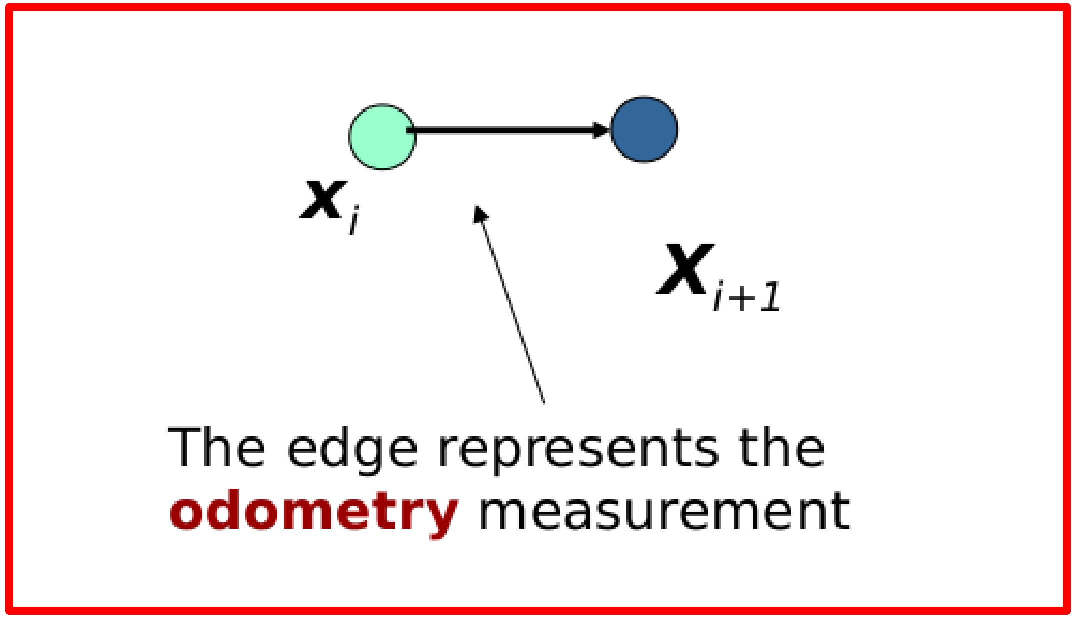
The edge represents the position of $\boldsymbol{x}_j$ seen from $\boldsymbol{x}_i$, based on the ***observations***

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

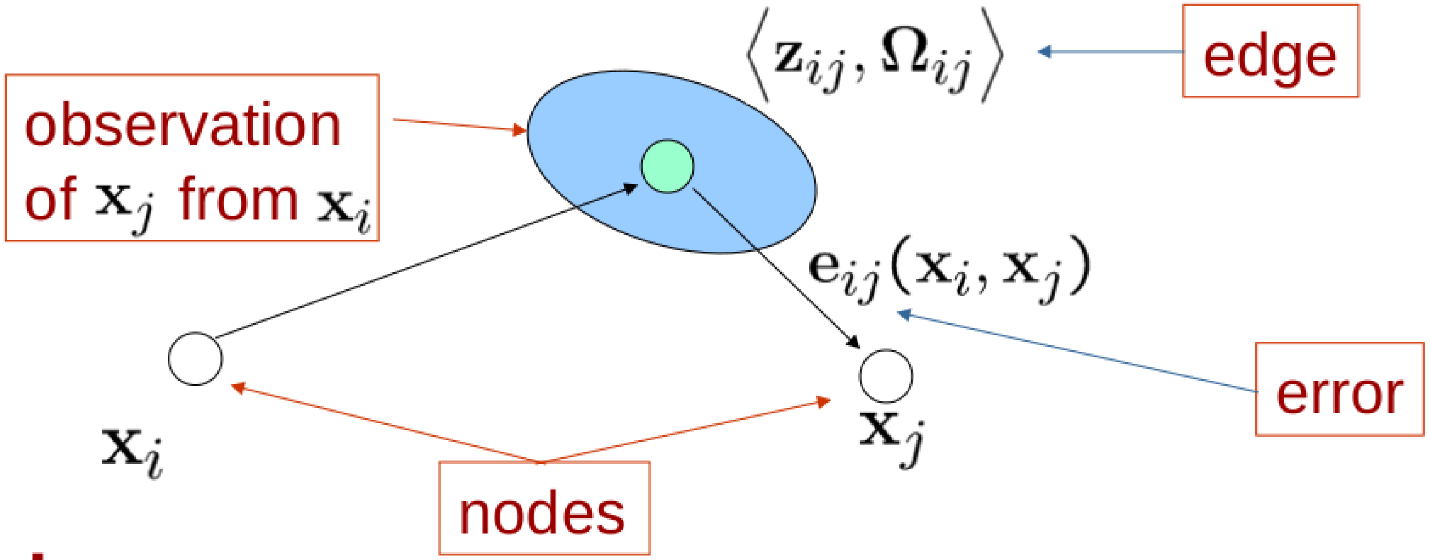
There is a constraint e_{ij} between the node x_i and the node x_j if

- 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

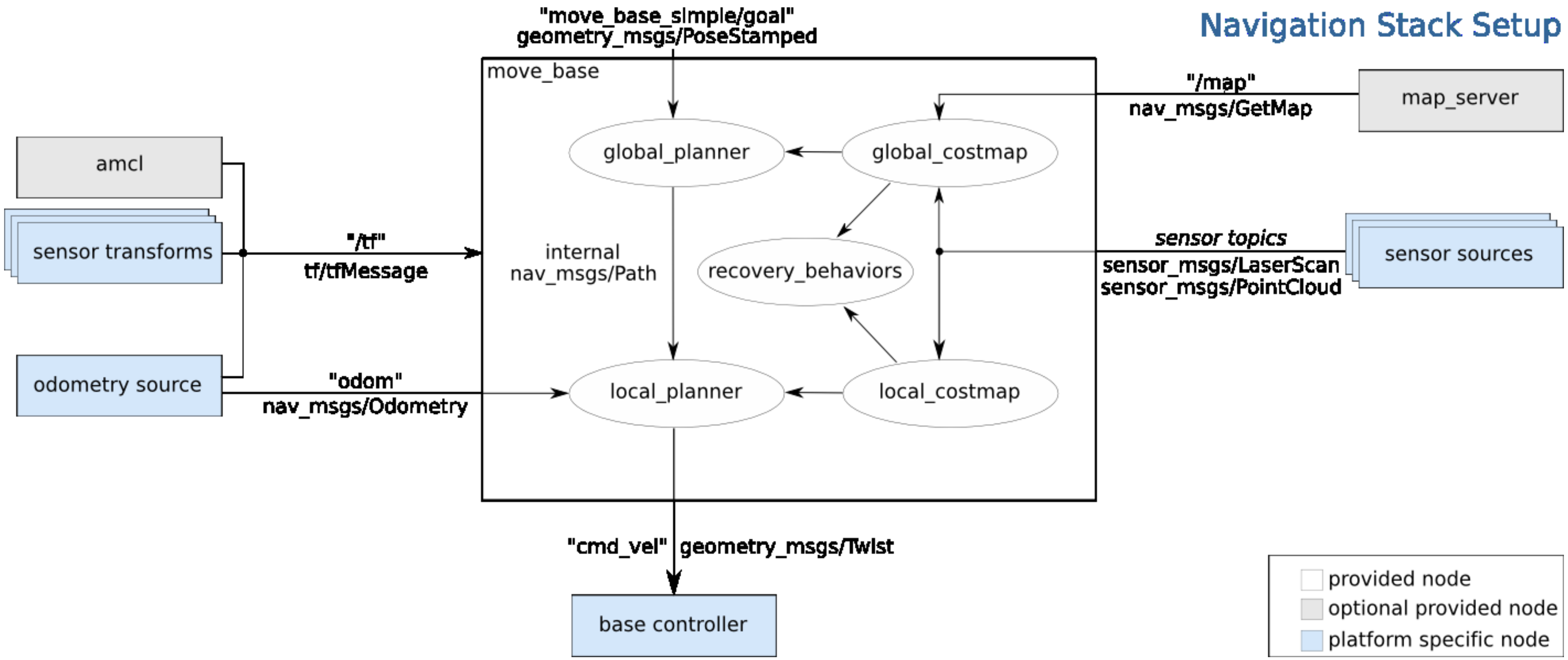
1. The environment changes (e.g., opening/closing doors)
2. It is dynamic (things might appear/disappear from the perception range of the robot)
3. The estimate is “noisy”

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

Virtual SLAM and Navigation

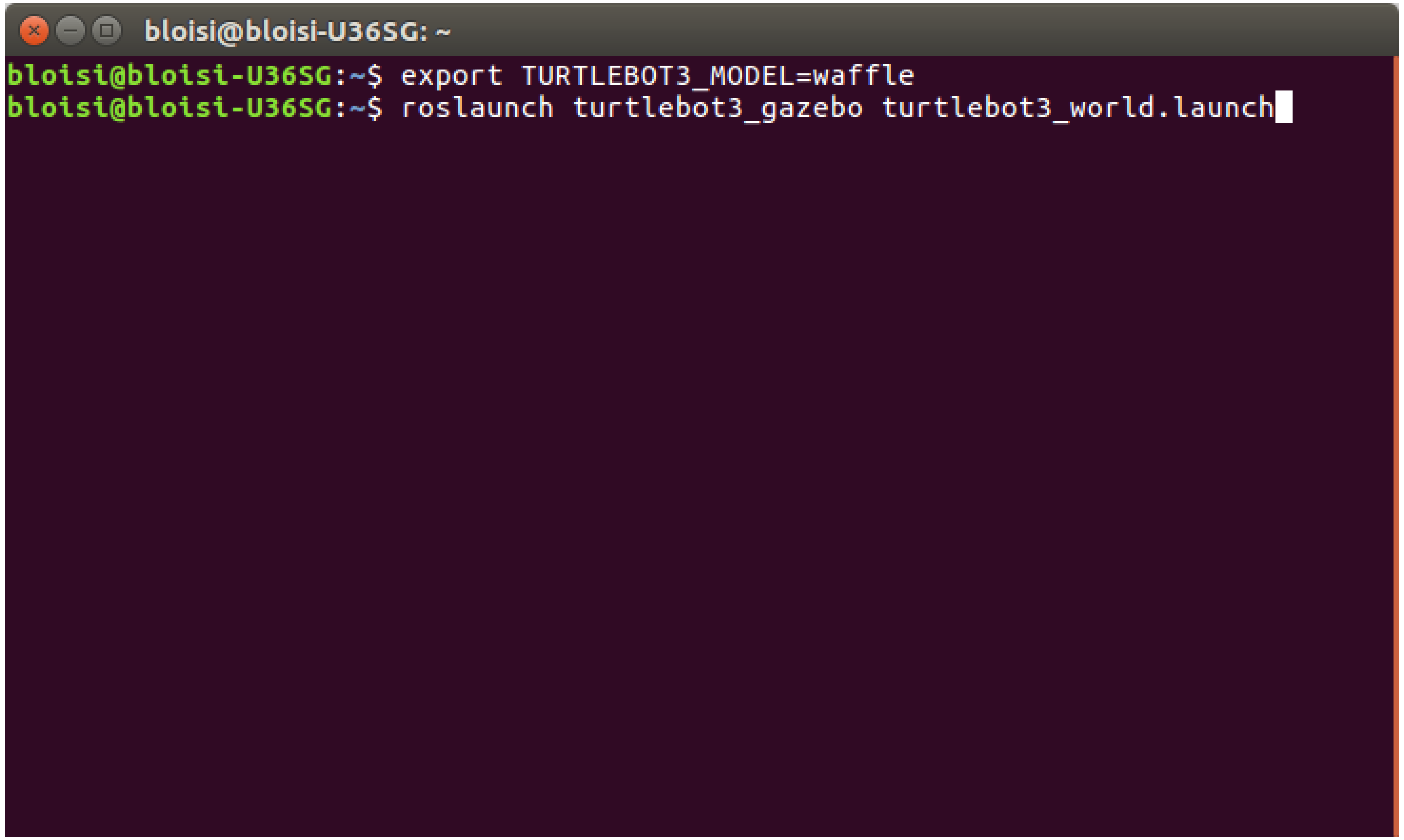
Useremo

- i package ROS per lo SLAM per creare una mappa di un mondo simulato tramite Gazebo

- lo stack ROS per la navigazione per far muovere il TurtleBot3 verso una destinazione sulla mappa

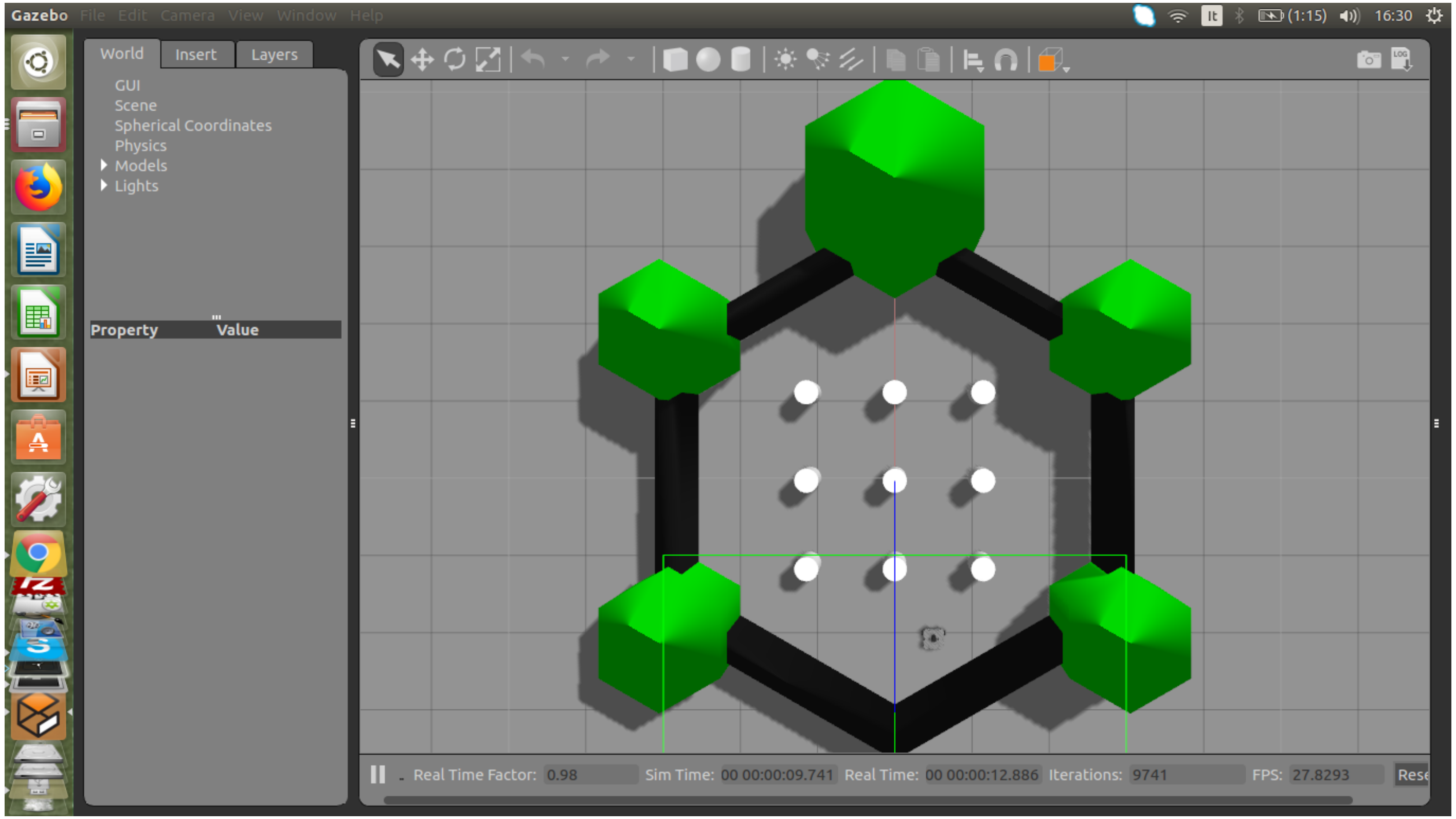
Launch Gazebo

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```



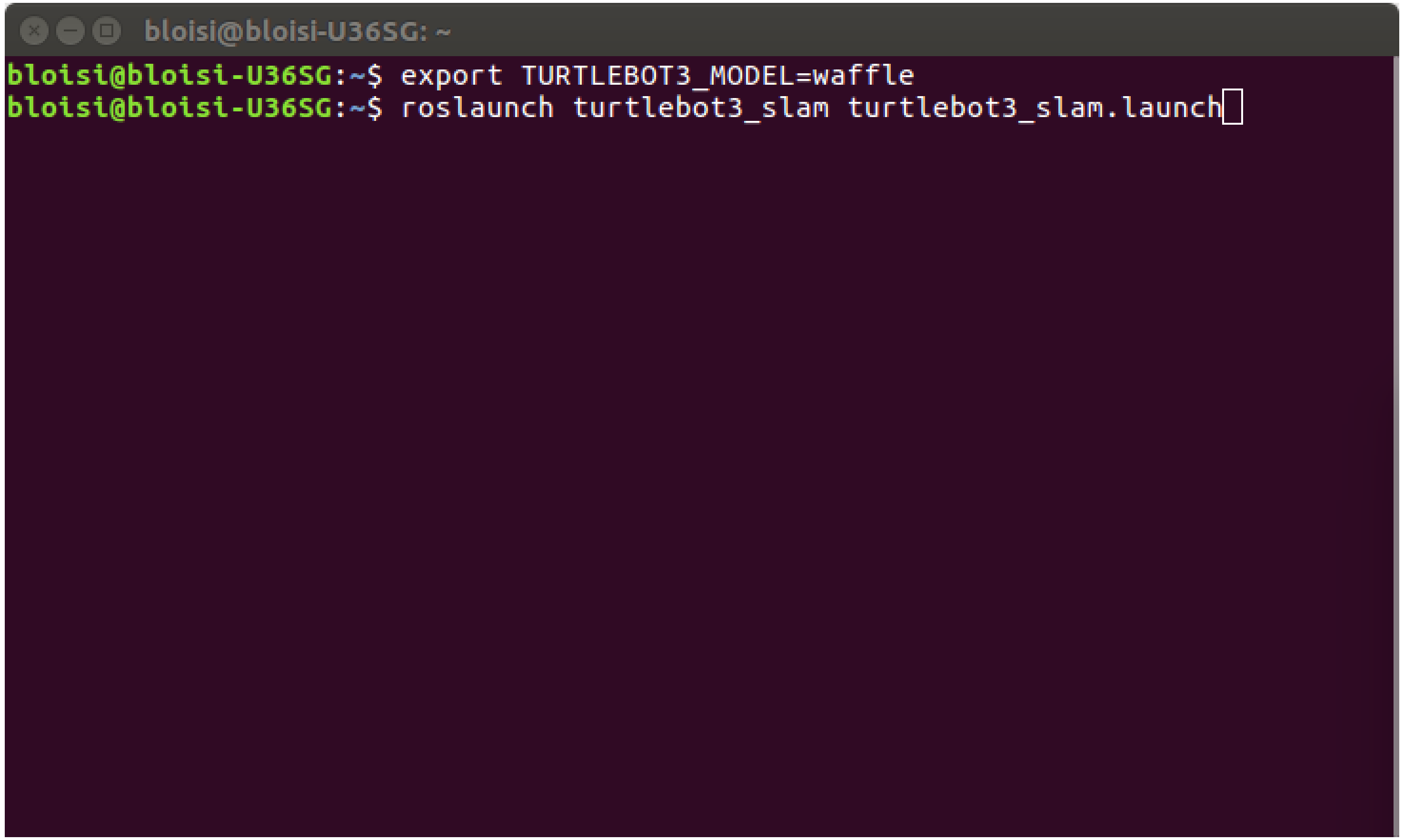
Launch Gazebo

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```



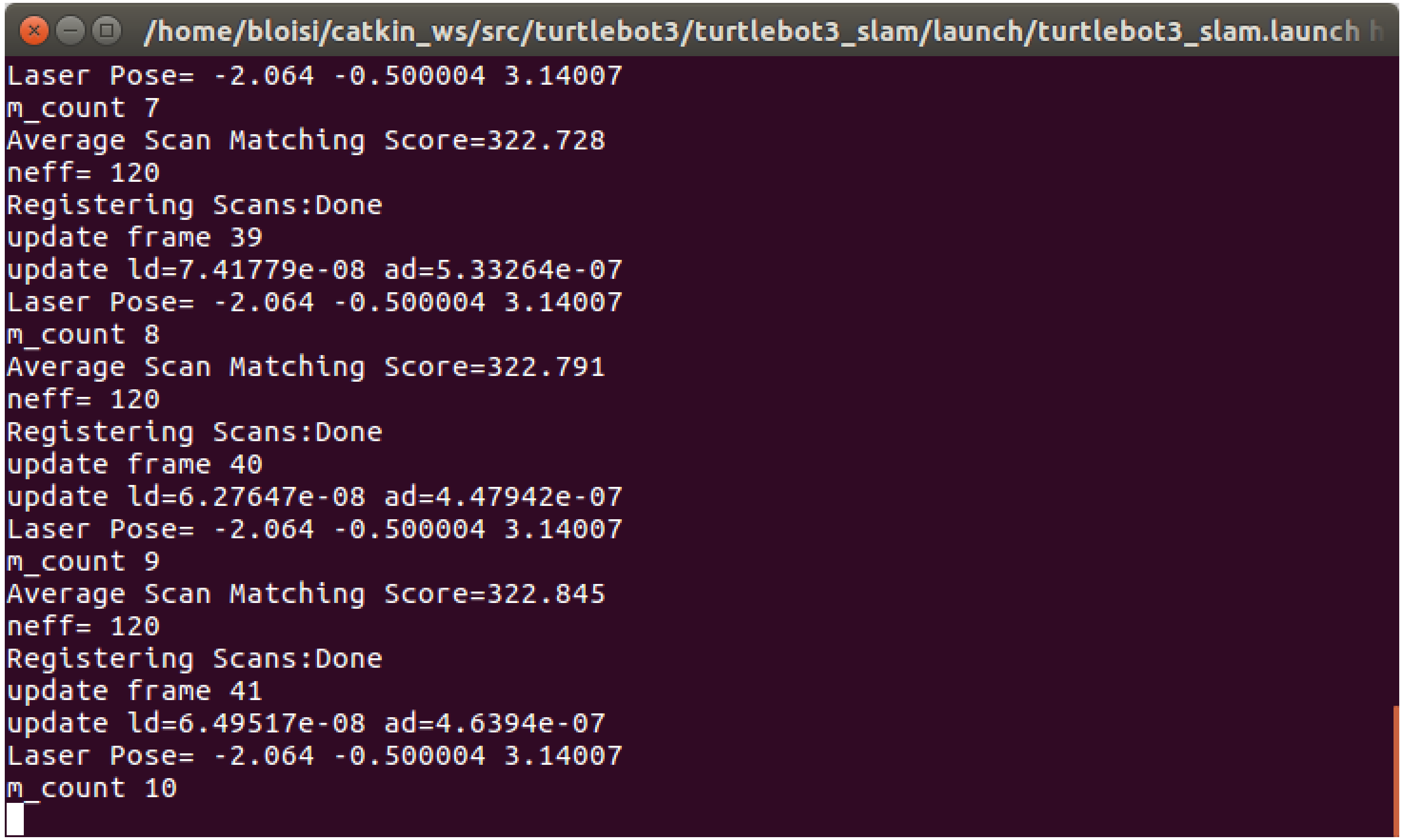
Launch SLAM

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
```



Launch SLAM

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
```

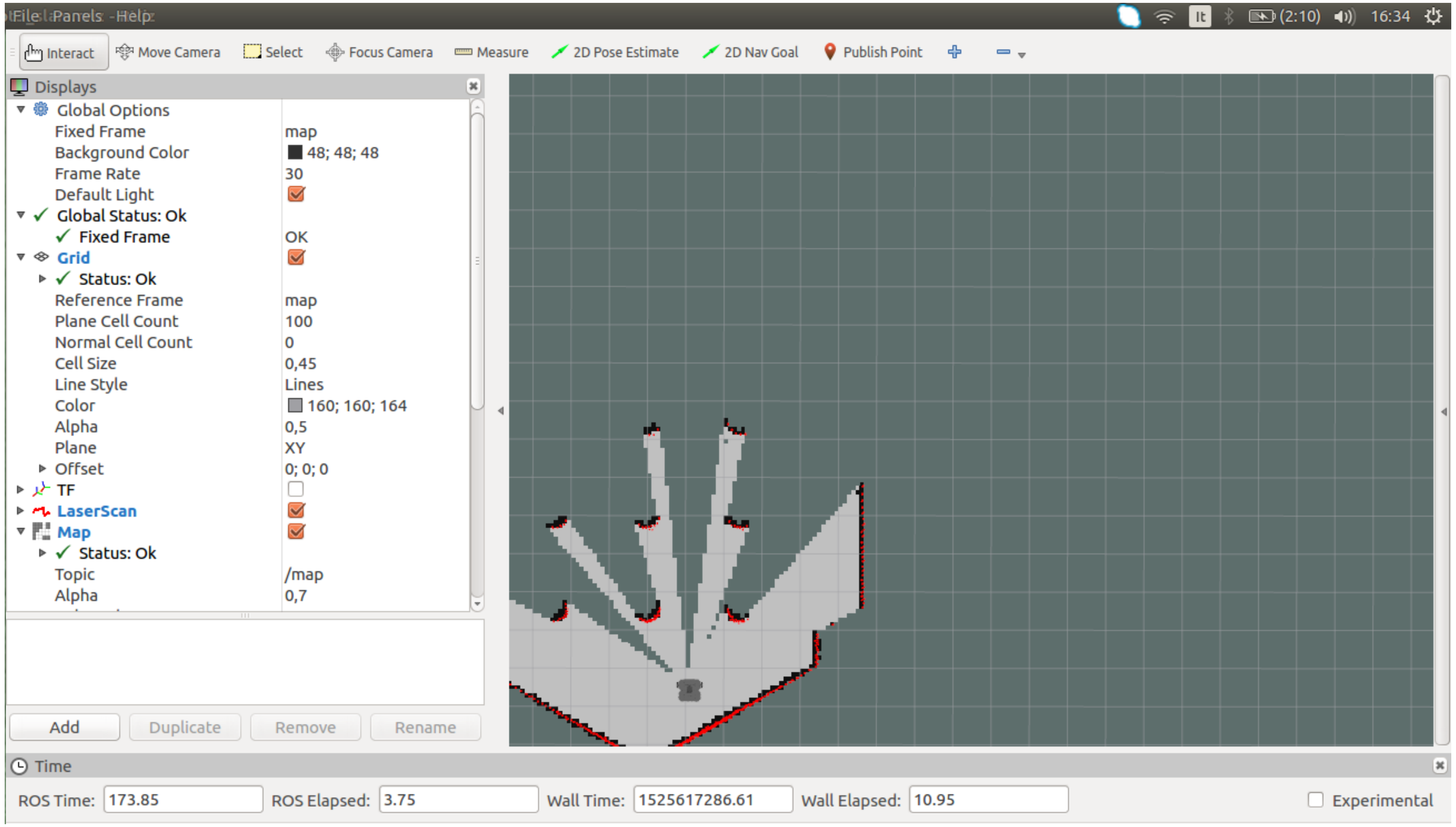


Execute RViz

```
$ export TURTLEBOT3_MODEL=waffle
$ rosrun rviz rviz -d `rospack find turtlebot3_slam`/rviz/turtlebot3_slam.rviz
```

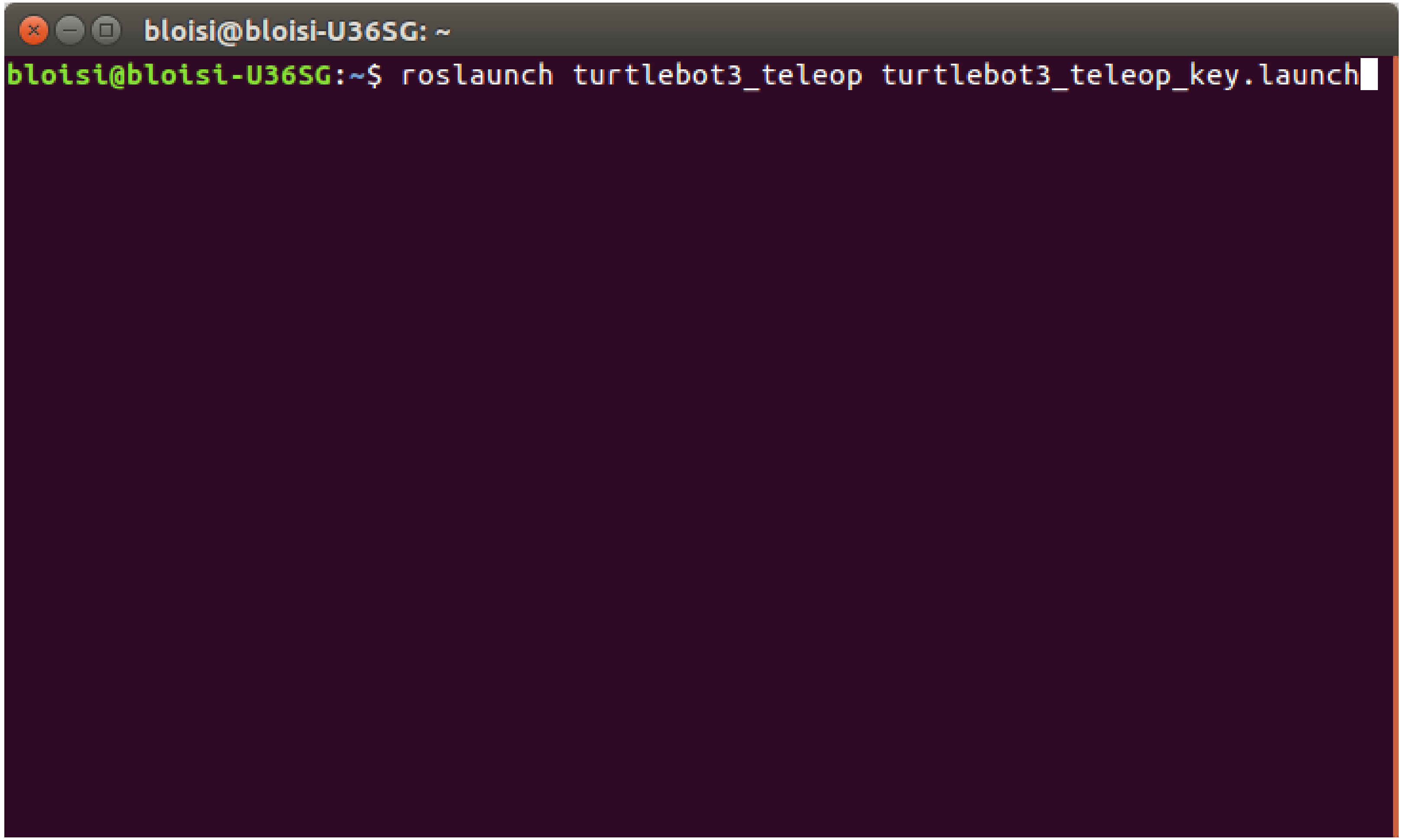
Execute RViz

```
$ export TURTLEBOT3_MODEL=waffle
$ rosrun rviz rviz -d `rospack find turtlebot3_slam`/rviz/turtlebot3_slam.rviz
```



Remotely Control Turtlebot3

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

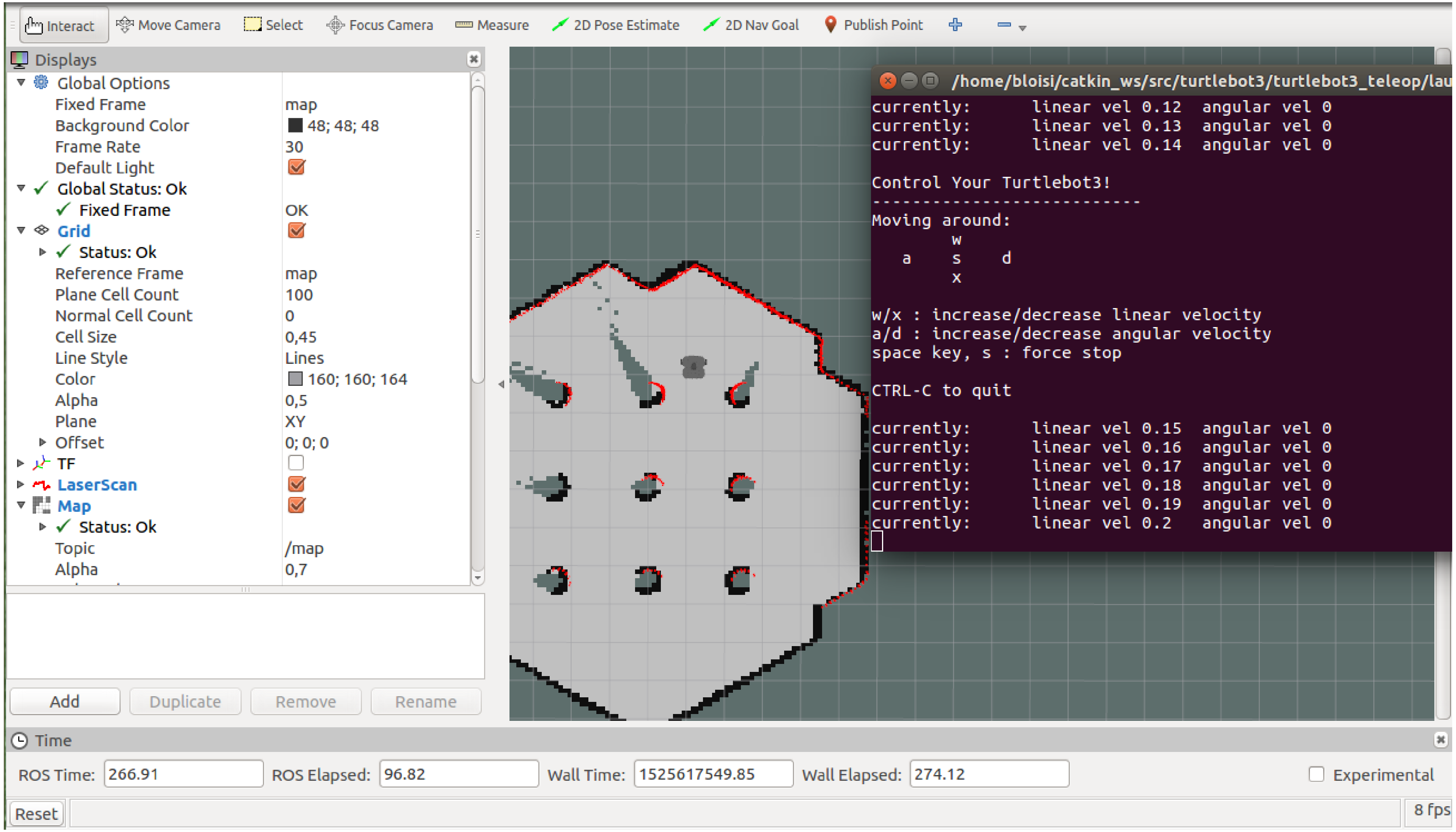


Remotely Control Turtlebot3

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

```
/home/bloisi/catkin_ws/src/turtlebot3/turtlebot3_teleop/launch/turtlebot3_teleop_key
 * /rosversion: 1.12.13

NODES
  /
    turtlebot3_teleop_keyboard (turtlebot3_teleop/turtlebot3_teleop_key)

ROS_MASTER_URI=http://localhost:11311

process[turtlebot3_teleop_keyboard-1]: started with pid [6305]

Control Your Turtlebot3!
---------------------------
Moving around:
        w
   a    s    d
        x

w/x : increase/decrease linear velocity
a/d : increase/decrease angular velocity
space key, s : force stop

CTRL-C to quit
```

Exploration



Exploration

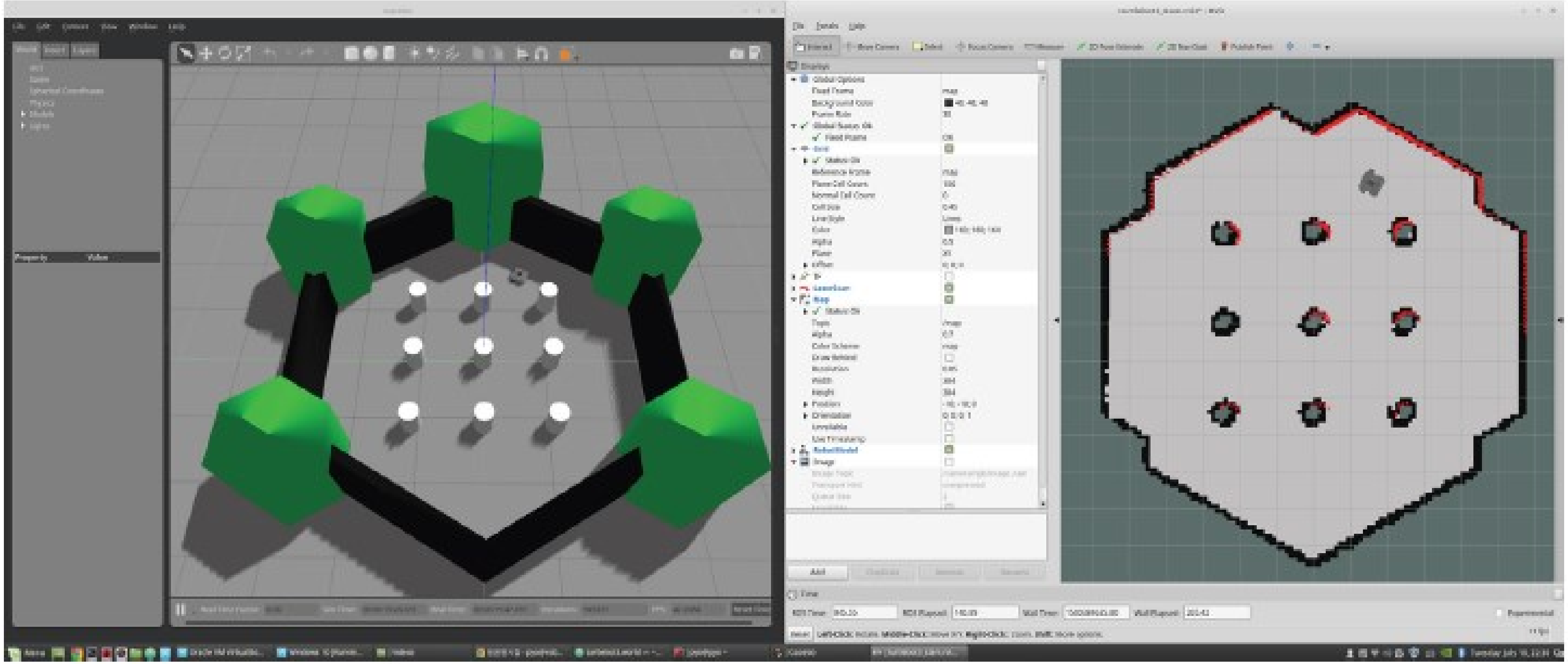


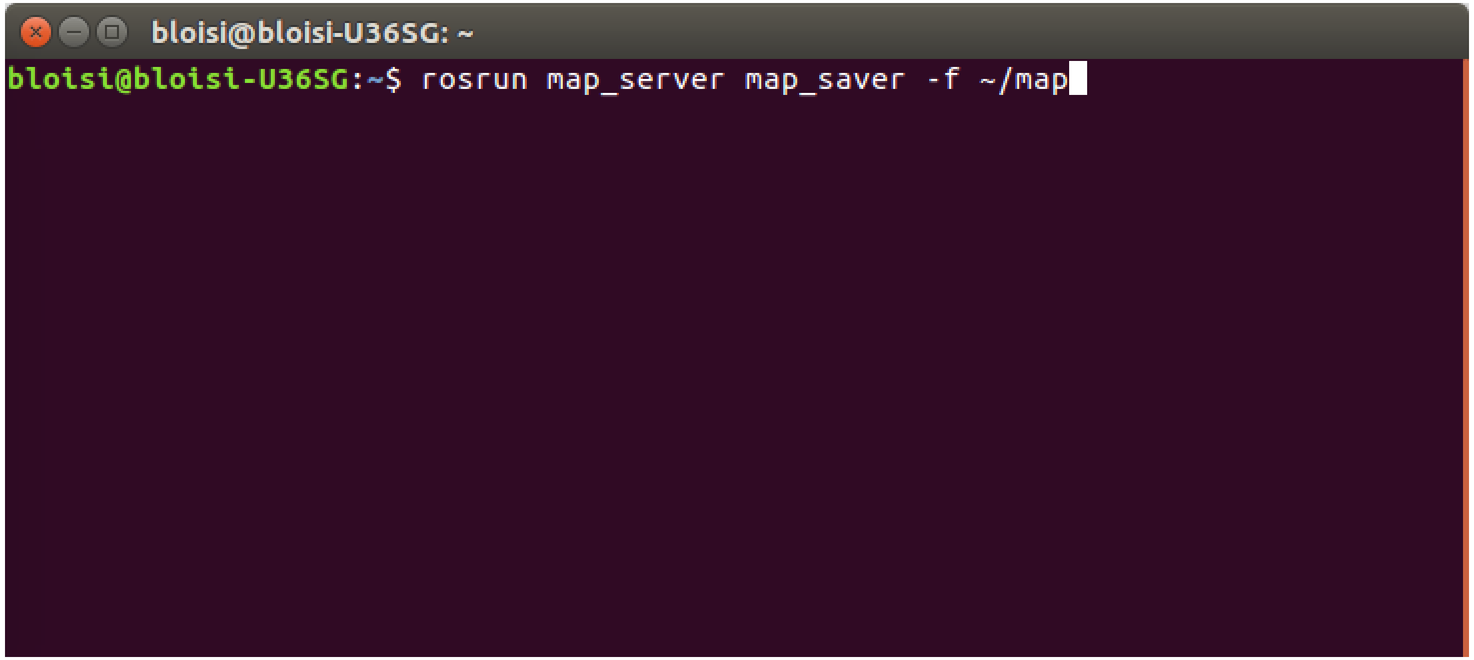
FIGURE 10-20 Running SLAM on Gazebo (Left: Gazebo, Right: RViz)

YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim, *“ROS Robot Programming - A Handbook Written by TurtleBot3 Developers”*

Save the Map

Terminata l'esplorazione, possiamo salvare la mappa che è stata generata con il map_server

```
$ rosrun map_server map_saver -f ~/map
```



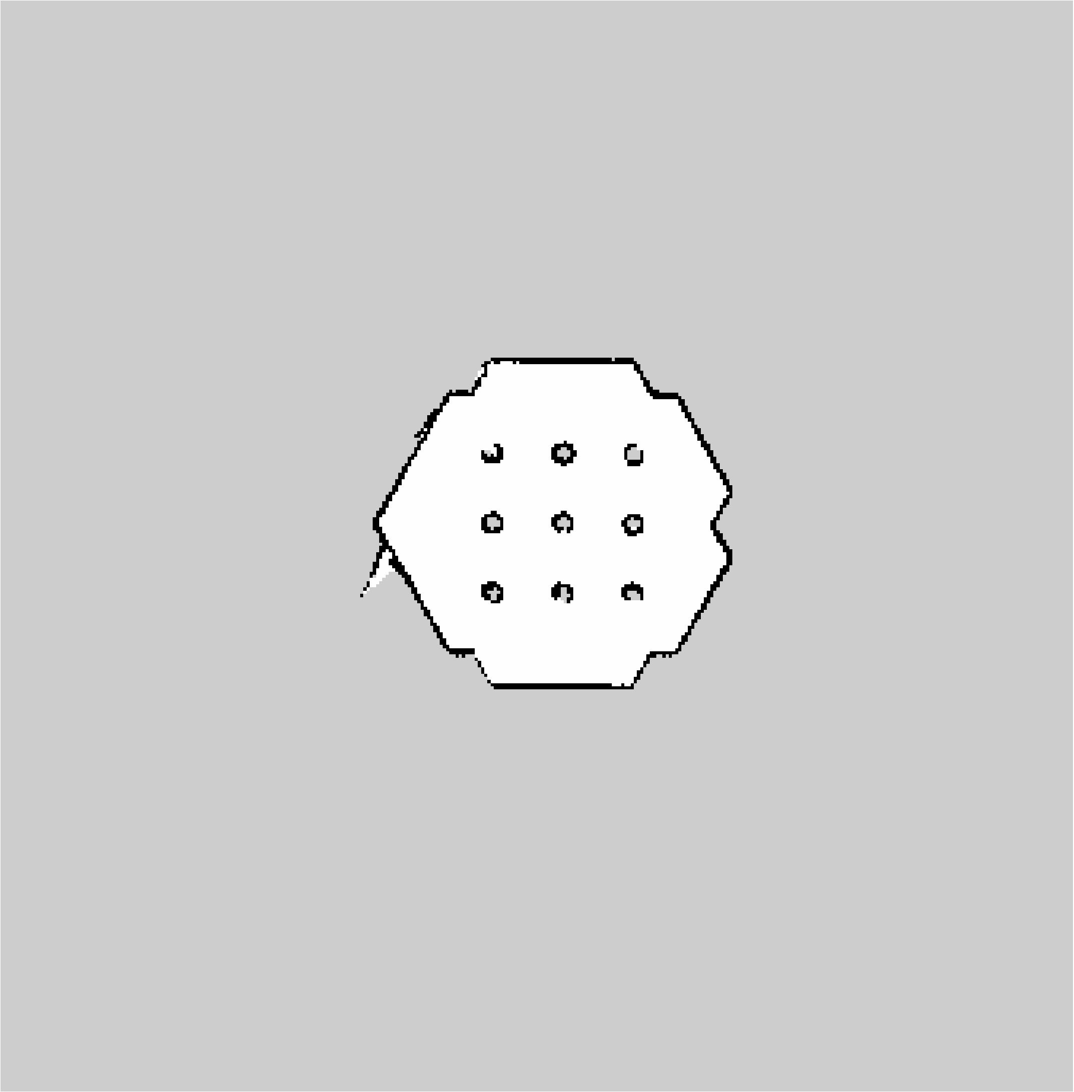
Save the Map

Terminata l'esplorazione, possiamo salvare la mappa che è stata generata con il map_server

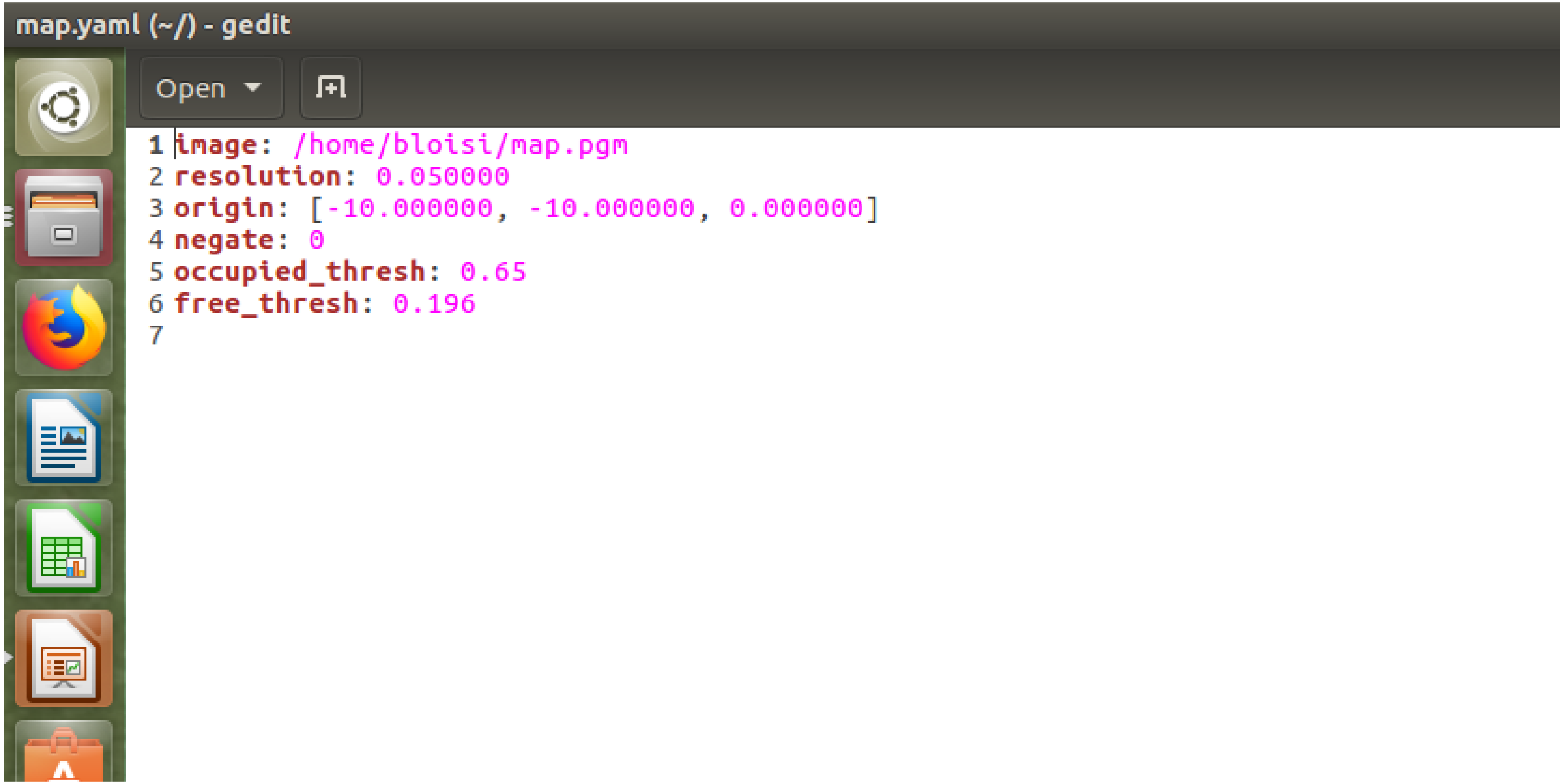
```
$ rosrun map_server map_saver -f ~/map
```

```
bloisi@bloisi-U36SG: ~
bloisi@bloisi-U36SG:~$ rosrun map_server map_saver -f ~/map
[ INFO] [1525617922.440250407]: Waiting for the map
[ INFO] [1525617922.705725024, 410.589000000]: Received a 384 X 384 map @ 0.050
m/pix
[ INFO] [1525617922.706387150, 410.589000000]: Writing map occupancy data to /ho
me/bloisi/map.pgm
[ INFO] [1525617922.715131989, 410.590000000]: Writing map occupancy data to /ho
me/bloisi/map.yaml
[ INFO] [1525617922.716040700, 410.590000000]: Done

bloisi@bloisi-U36SG:~$ 
```

map.pgm



map.yaml


map.yaml (~/) - gedit
Open
image: /home/bloisi/map.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196

Navigazione

Per poter procedere con la navigazione

1. Terminare tutti i processi attivi
2. Lanciare in un terminale

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

3. Aprire un secondo terminale e digitare

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_navigation
turtlebot3_navigation.launch map_file:=$HOME/map.yaml
```

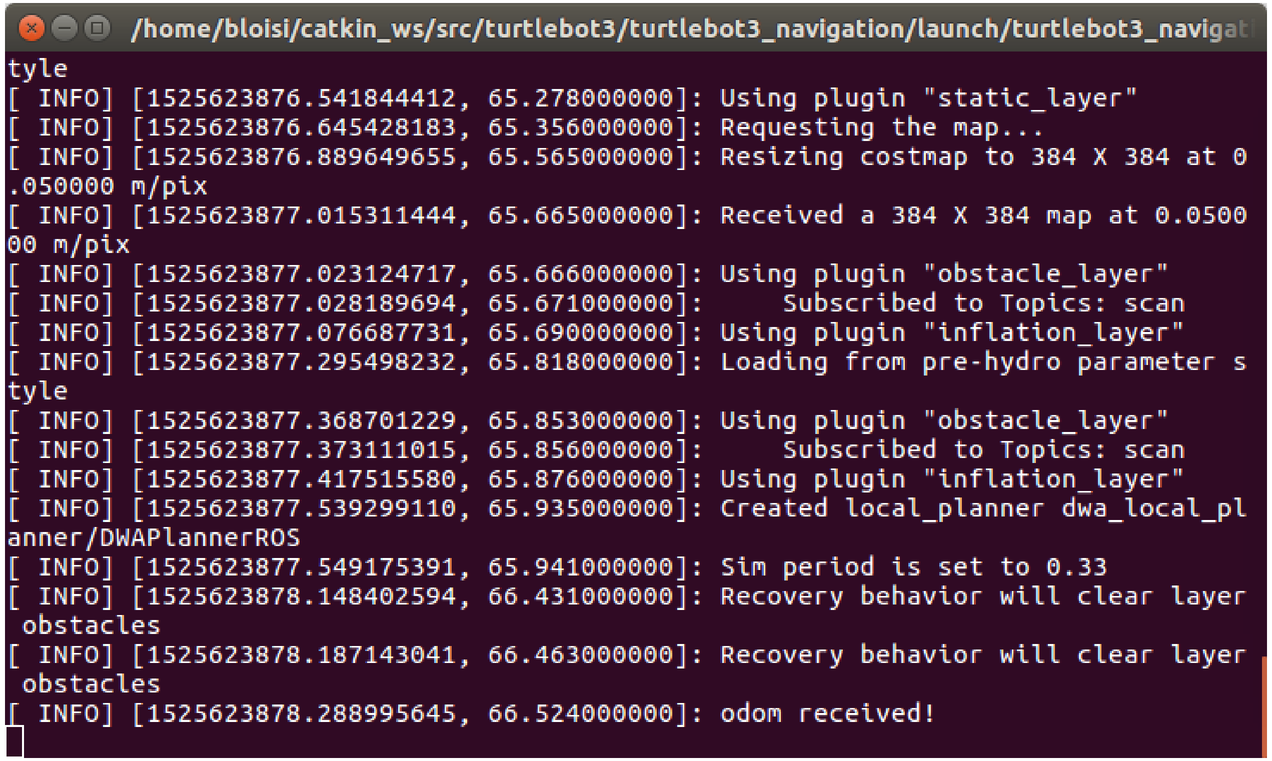
map_file

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch
map_file:=$HOME/map.yaml
```

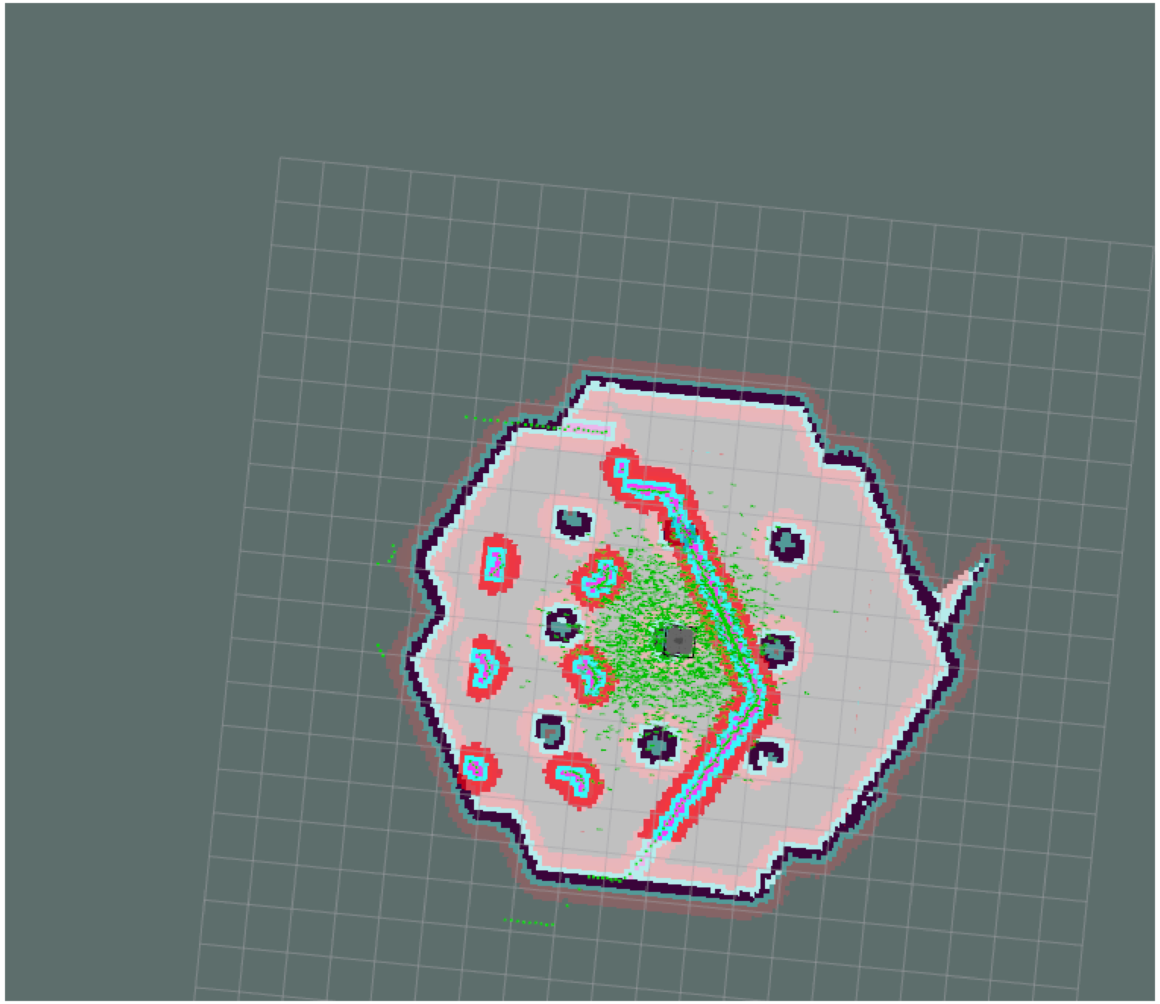
Navigazione - RViz

4. Aprire un terzo terminale e digitare

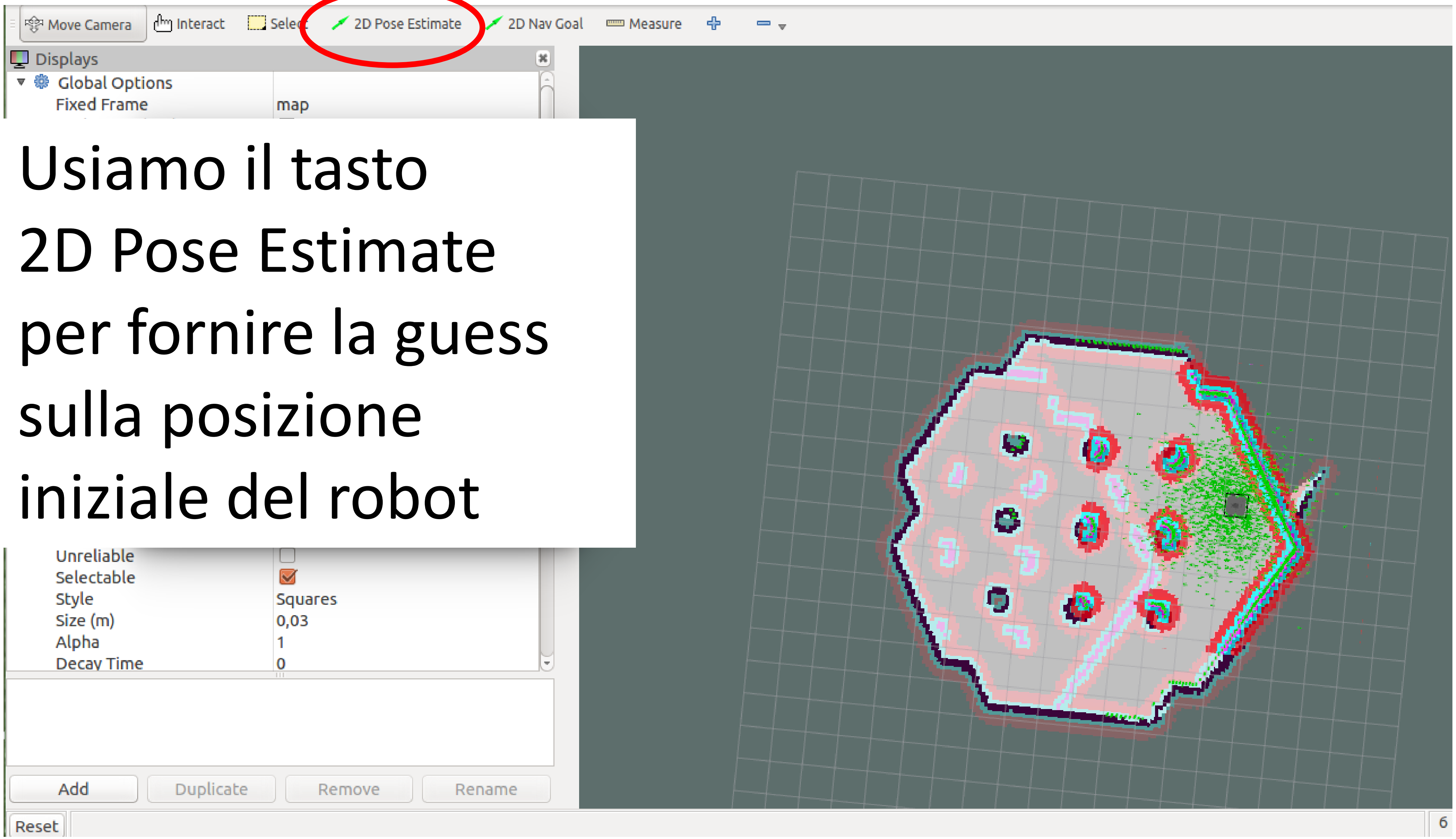
```
$ export TURTLEBOT3_MODEL=waffle
$ rosrun rviz rviz -d `rospack find turtlebot3_navigation`/rviz/turtlebot3_nav.rviz
```



Navigazione - RViz

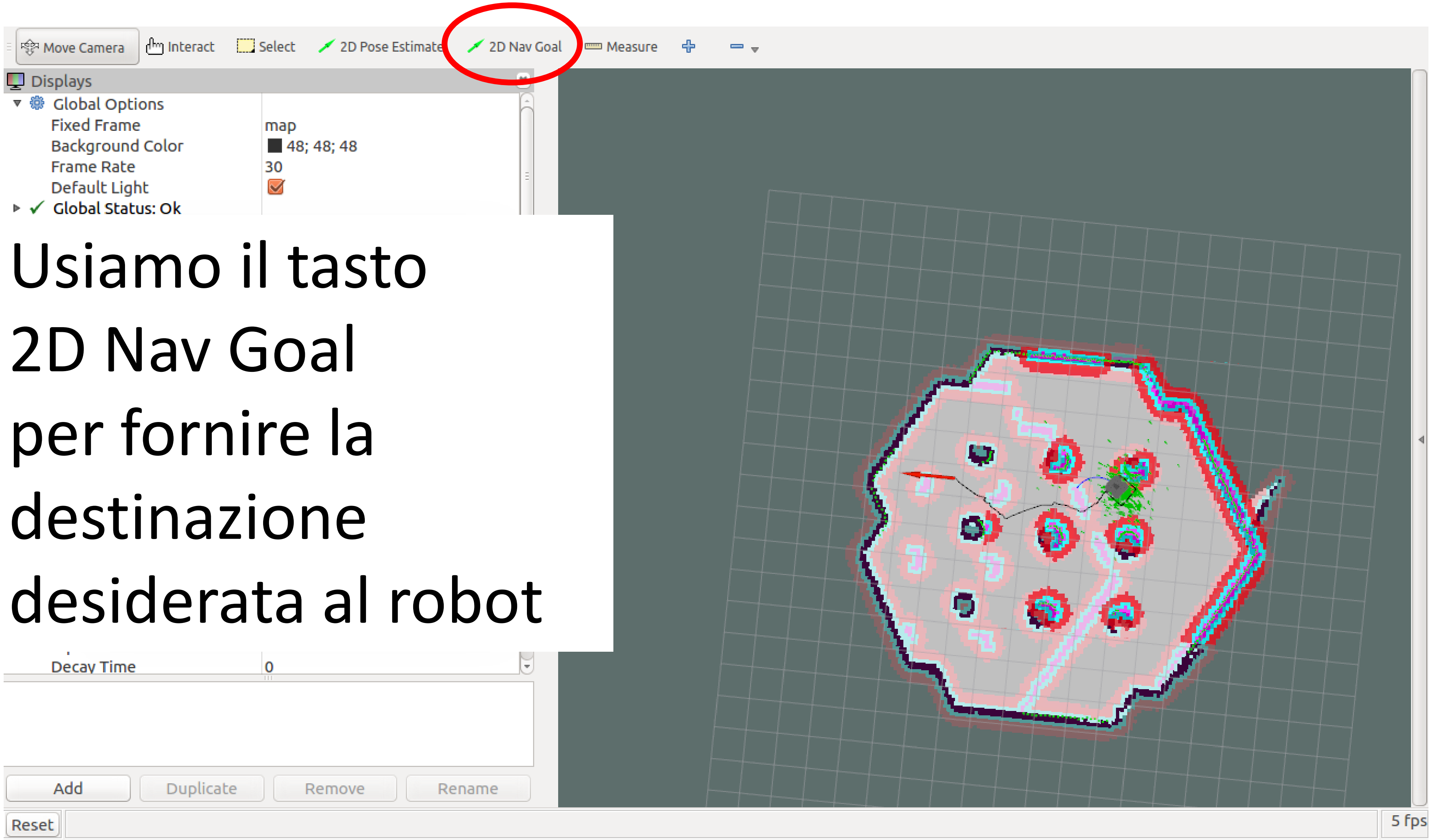


Pose Estimate



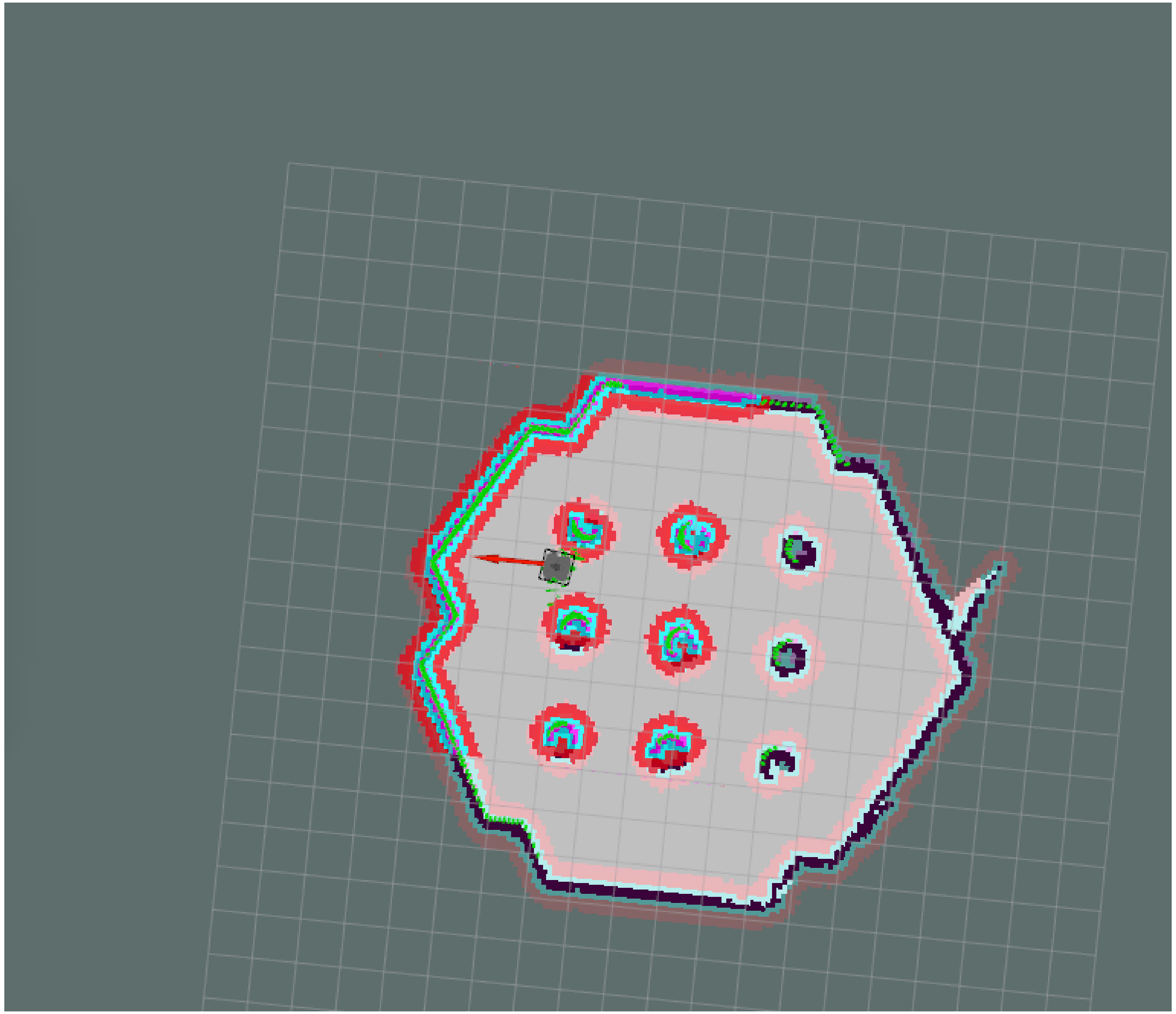
Usiamo il tasto
2D Pose Estimate
per fornire la guess
sulla posizione
iniziale del robot

Navigation Goal



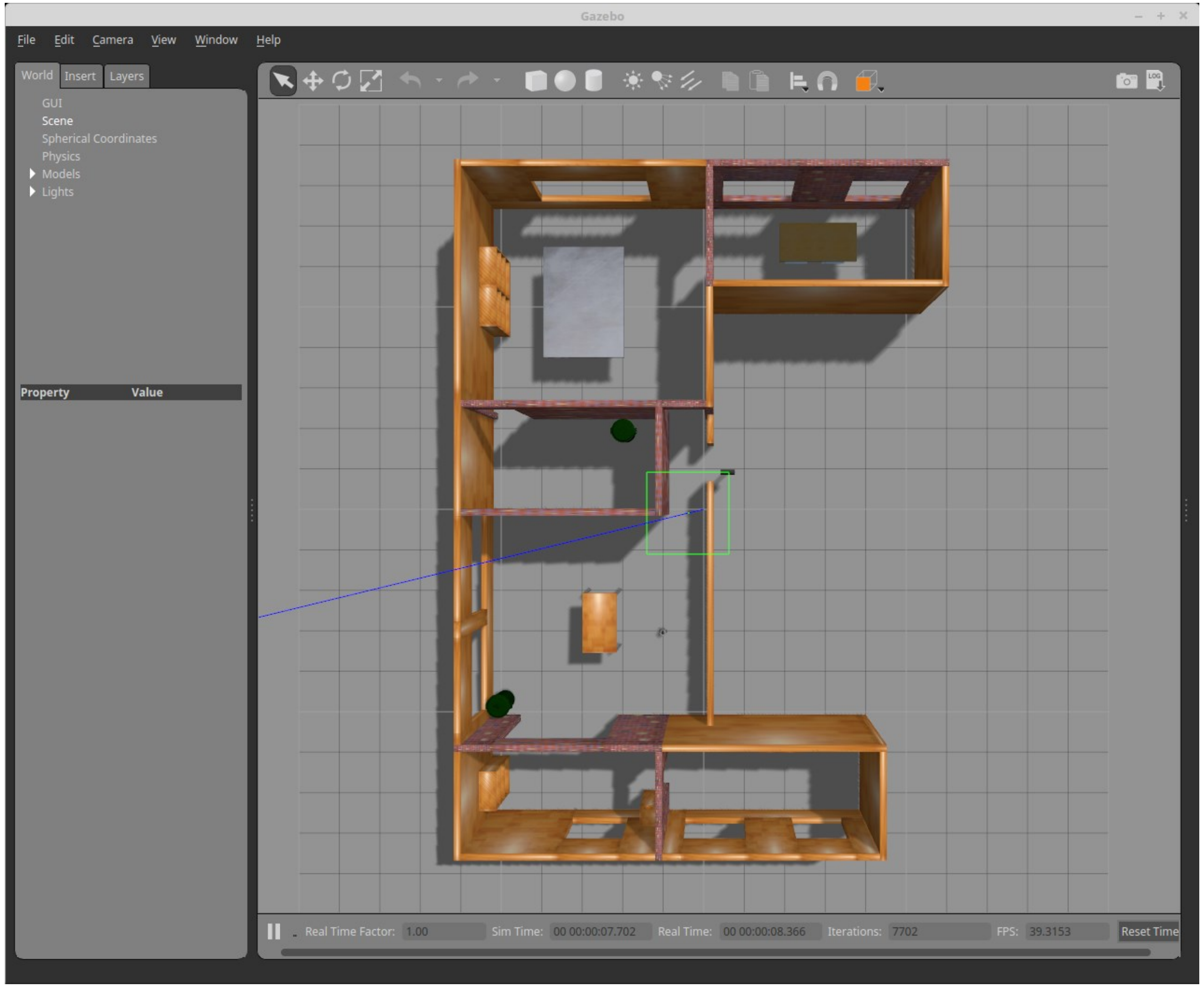
Usiamo il tasto
2D Nav Goal
per fornire la
destinazione
desiderata al robot

Goal raggiunto



Esercizio 1

Creare una mappa dell'ambiente Turtlebot3 House e utilizzarla per far navigare il robot



http://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/

Esercizio 2

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

2. Utilizzare il turtlebot3 per navigare autonomamente nel mondo cyber_lab

Corso di Laboratorio Ciberfisico

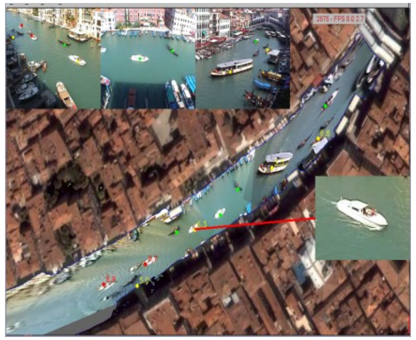
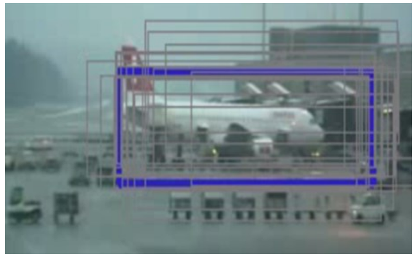
Modulo di Robot Programming with ROS

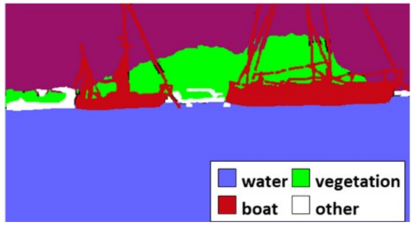


Navigazione in ROS

Docente:

Domenico Daniele Bloisi



Maggio 2018