

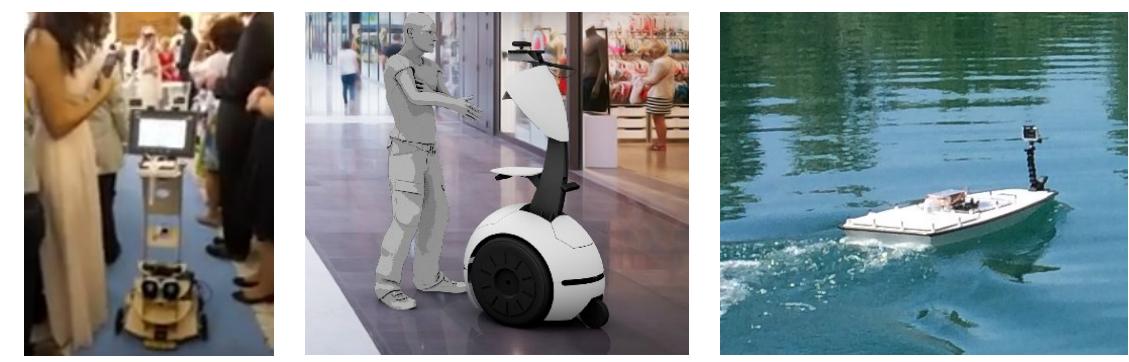
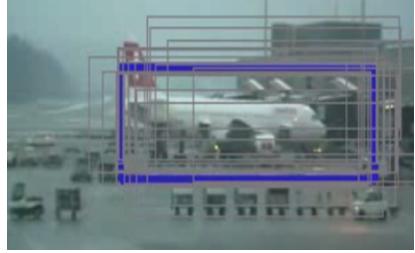
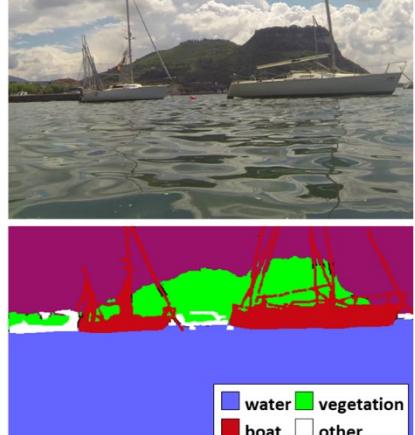
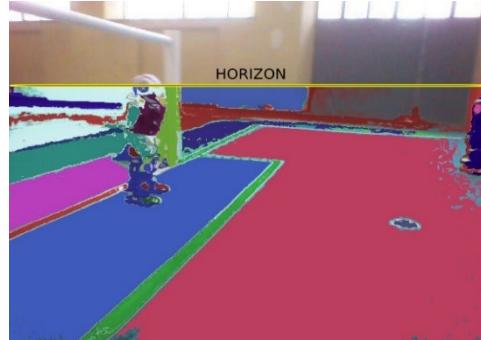


**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Visione e Percezione
A.A. 2019/2020

Navigazione in ROS

Maggio 2020



Il corso

- Home page del corso
<http://web.unibas.it/bloisi/corsi/visione-e-percezione.html>
- Docente: Domenico Daniele Bloisi
- Periodo: Il semestre marzo 2020 – giugno 2020
- Orario:
 - Martedì 17:00-19:00
 - Mercoledì 8:30-10:30

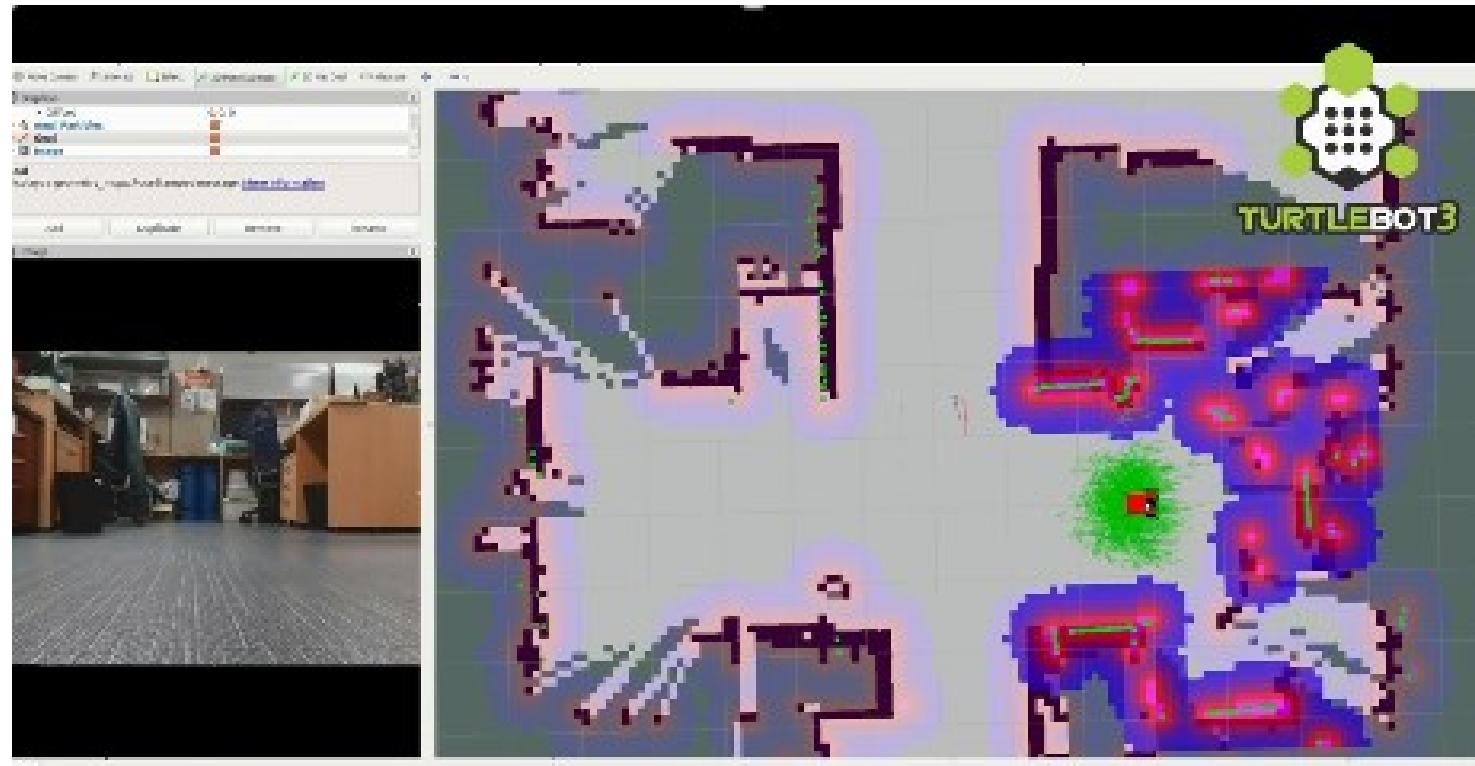
References and credits

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

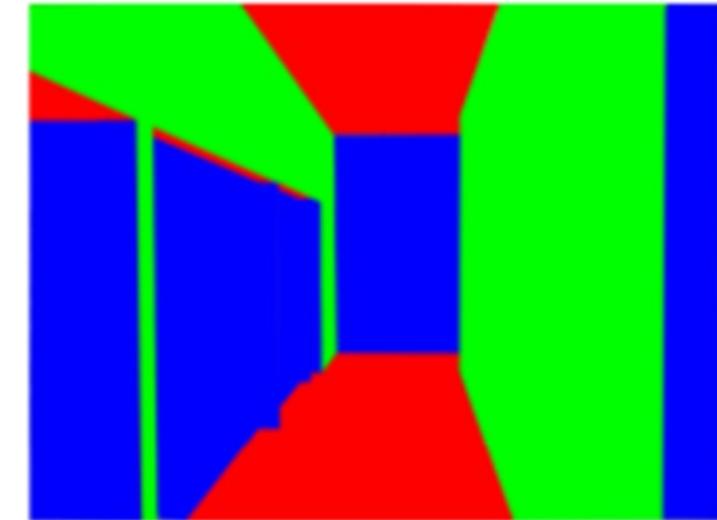
Il compito principale che un robot autonomo mobile deve essere in grado di compiere è quello di saper muoversi nell'ambiente operativo



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

Ambiente operativo

Strutturato



Non strutturato

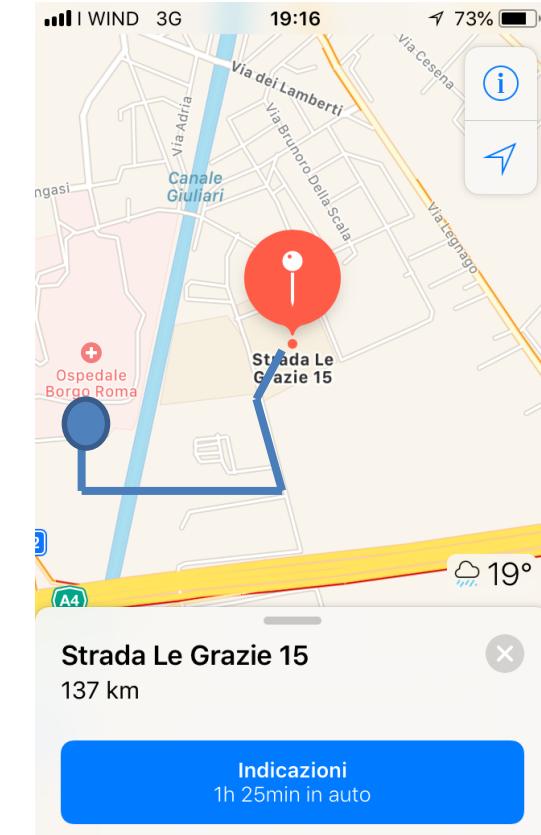


Navigazione GPS

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



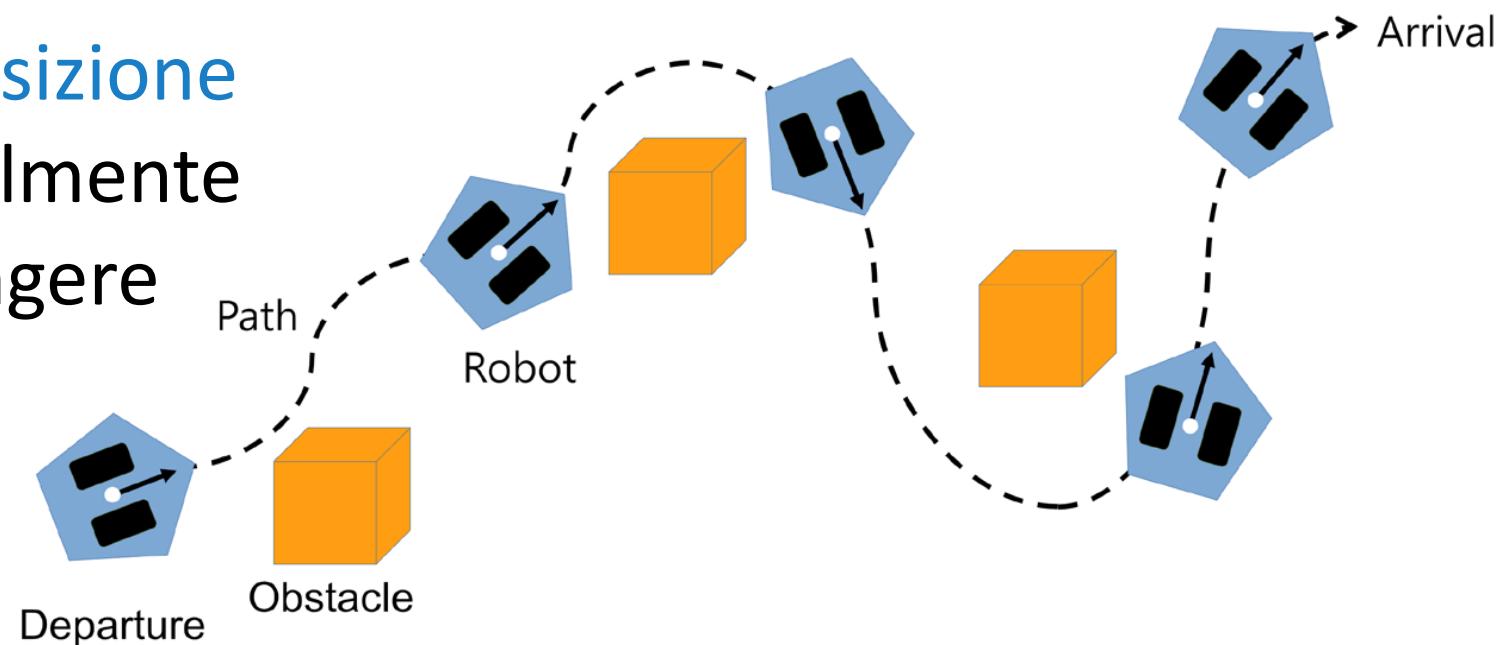
Sono sufficienti?

Navigazione robotica

Con il termine **navigazione** indichiamo il movimento del robot verso una destinazione predefinita

Per poter navigare, un robot ha bisogno di:

1. Avere una **mappa** dell'ambiente
2. Conoscere la propria **posizione**
3. Avere una **rotta** (possibilmente ottimizzata) per raggiungere la destinazione
4. **Evitare gli ostacoli presenti sul percorso**



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**

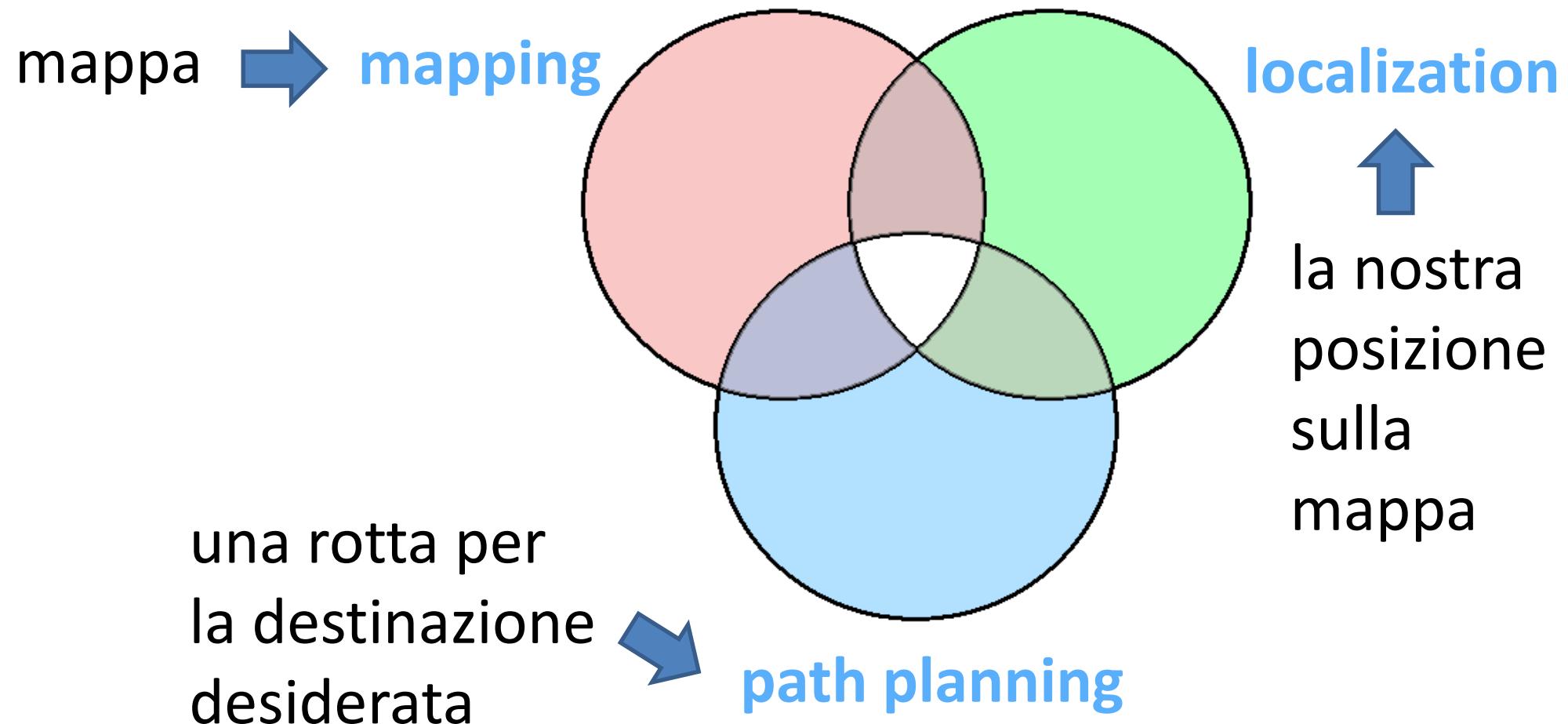
persone e sedie sono esempi di ostacoli mobili

Basic features

Affinché il robot sia in grado di navigare autonomamente in un ambiente con ostacoli avremo bisogno di

- ① Map
- ② Pose of Robot
- ③ Sensing
- ④ Path Calculation and Driving

Mapping, localization, planning

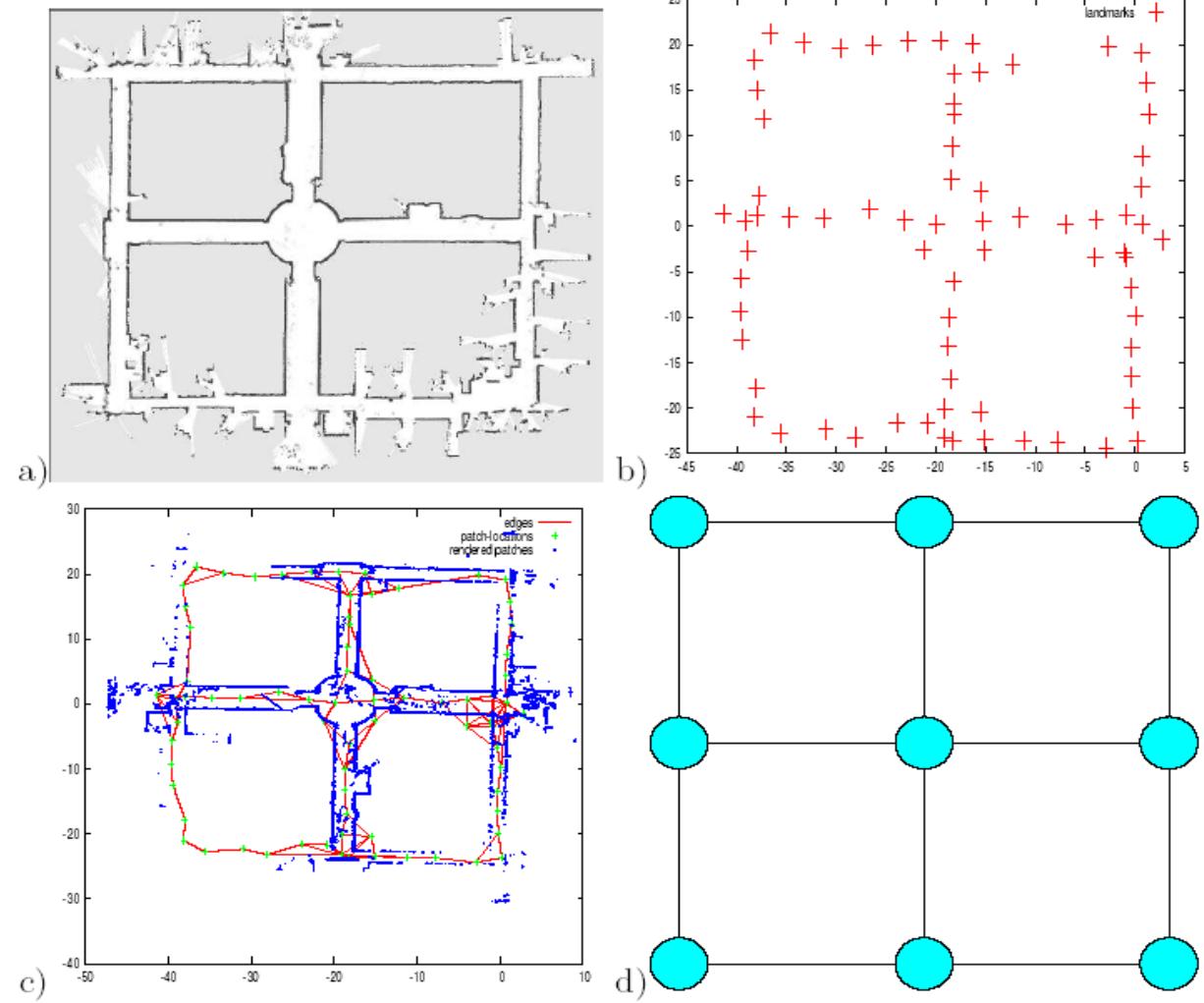


Map

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

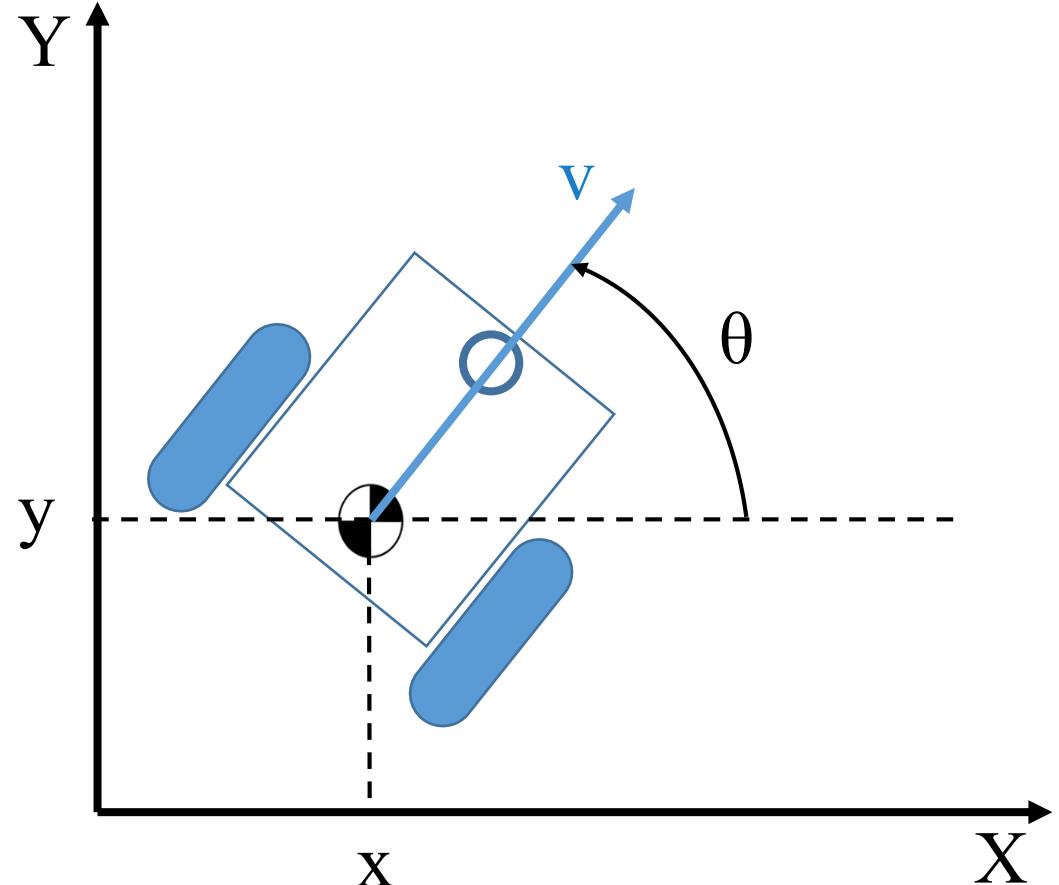
Representations:

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



Robot pose

- 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 $[x, y, \theta]$



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
```

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
```

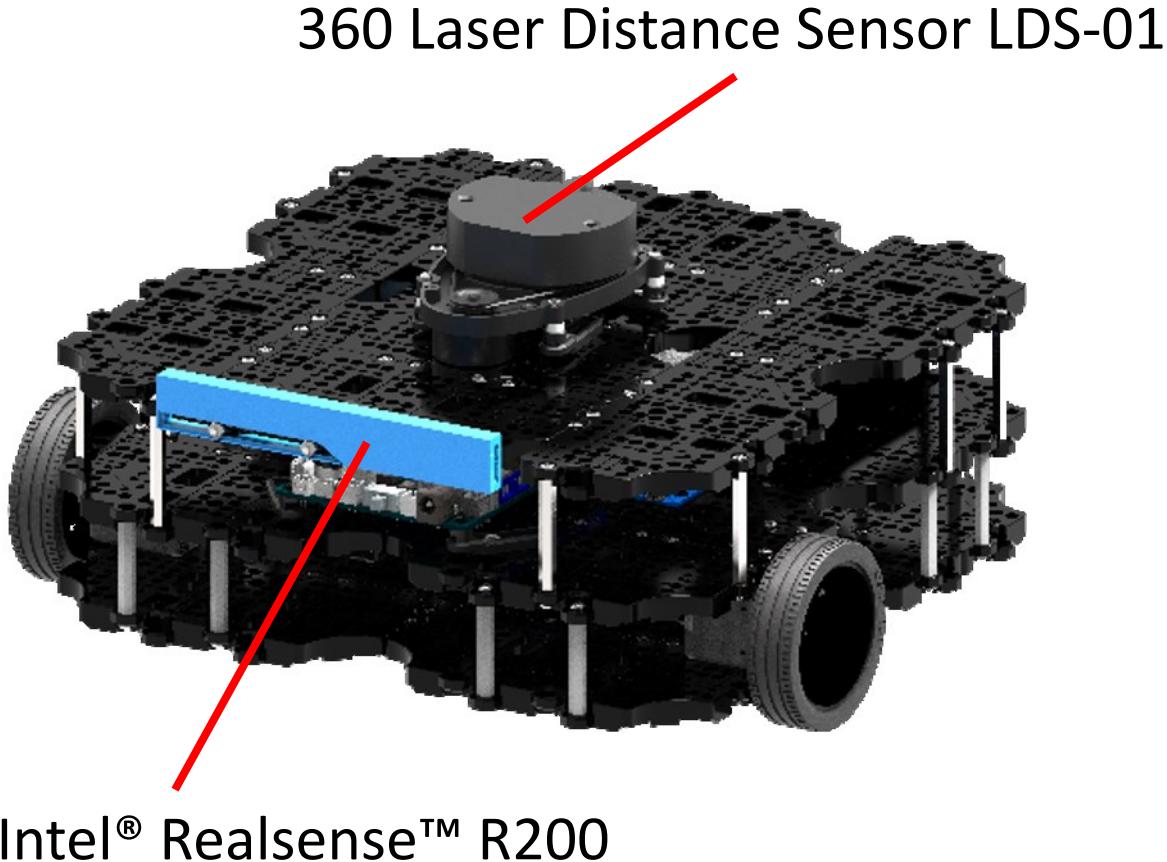
Compact Message Definition

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

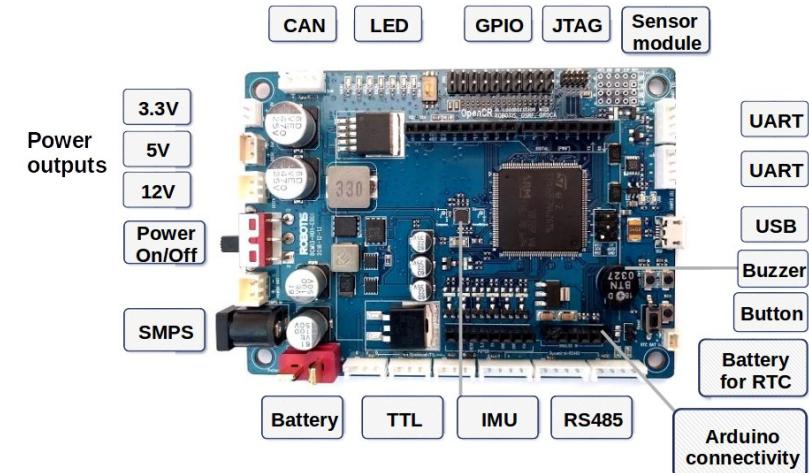
<http://wiki.ros.org/tf2/Tutorials/Quaternions>

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

Sensing – Turtlebot3



OpenCR1.0



Gyroscope 3Axis, Accelerometer 3Axis,
Magnetometer 3Axis

Sensori di distanza

- Sonar
- Laser range finder
- Time of Flight Camera



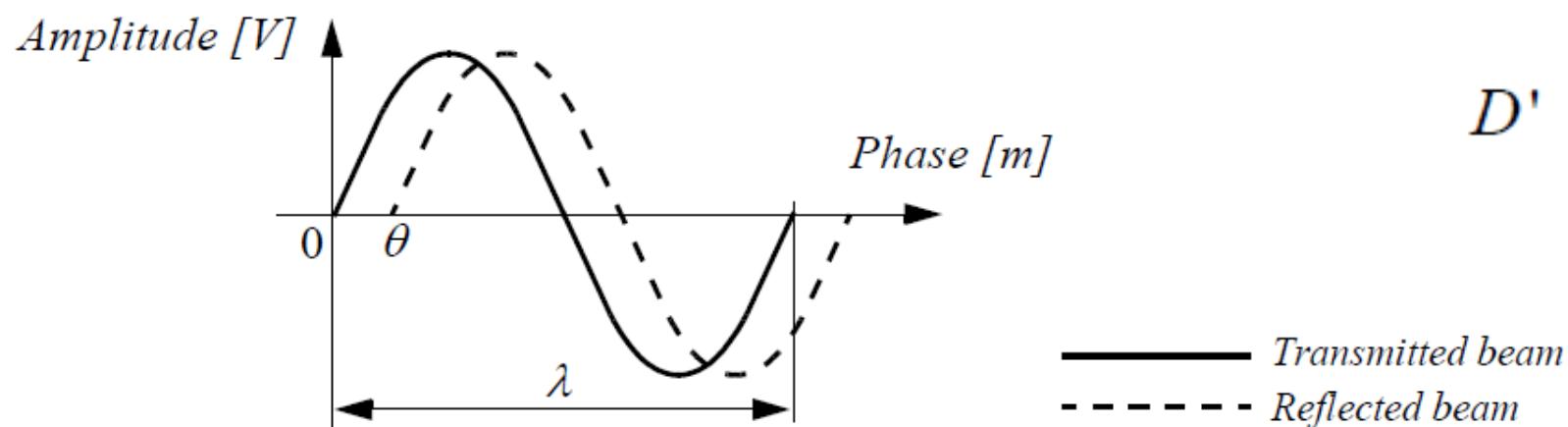
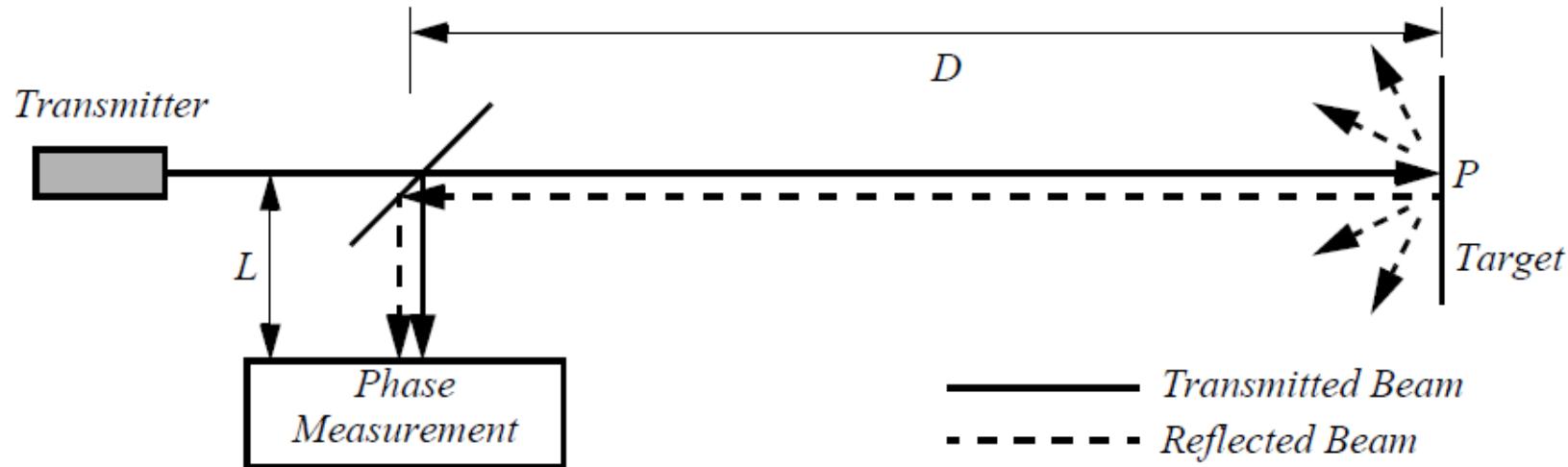
I sensori di distanza basati sul time-of-flight sfruttano la velocità di propagazione del suono o delle onde elettromagnetiche per calcolare la distanza



KINECT
for XBOX 360

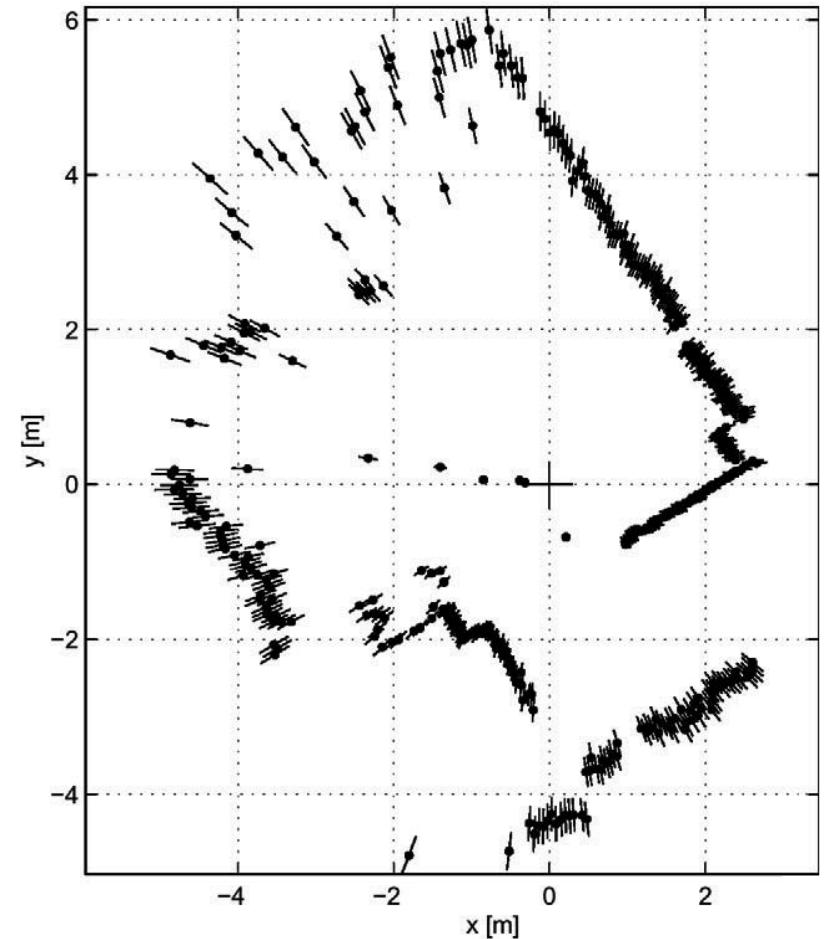
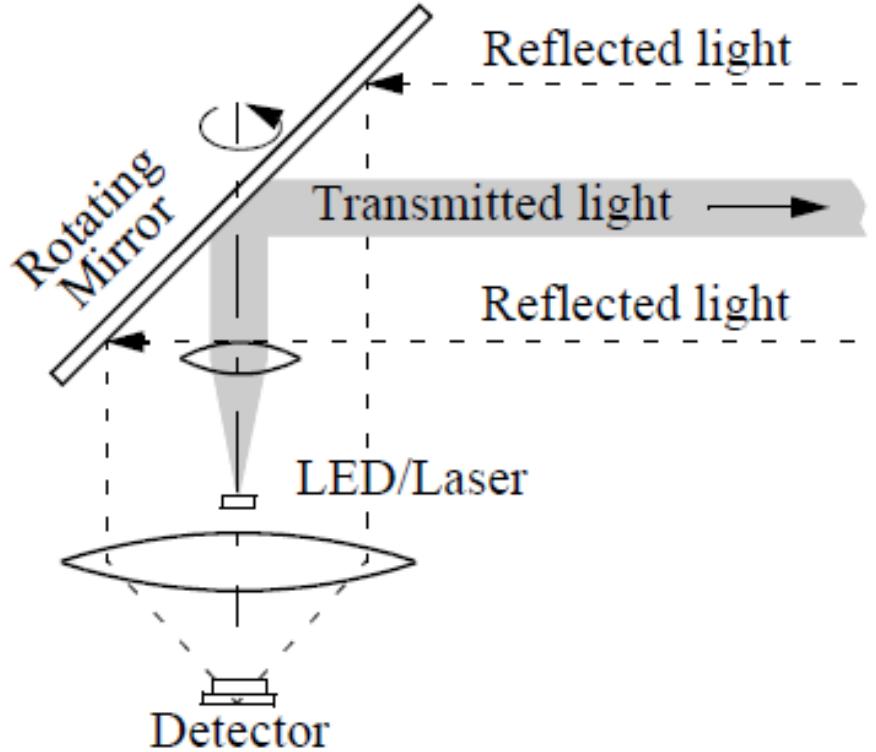


Laser Range Finder



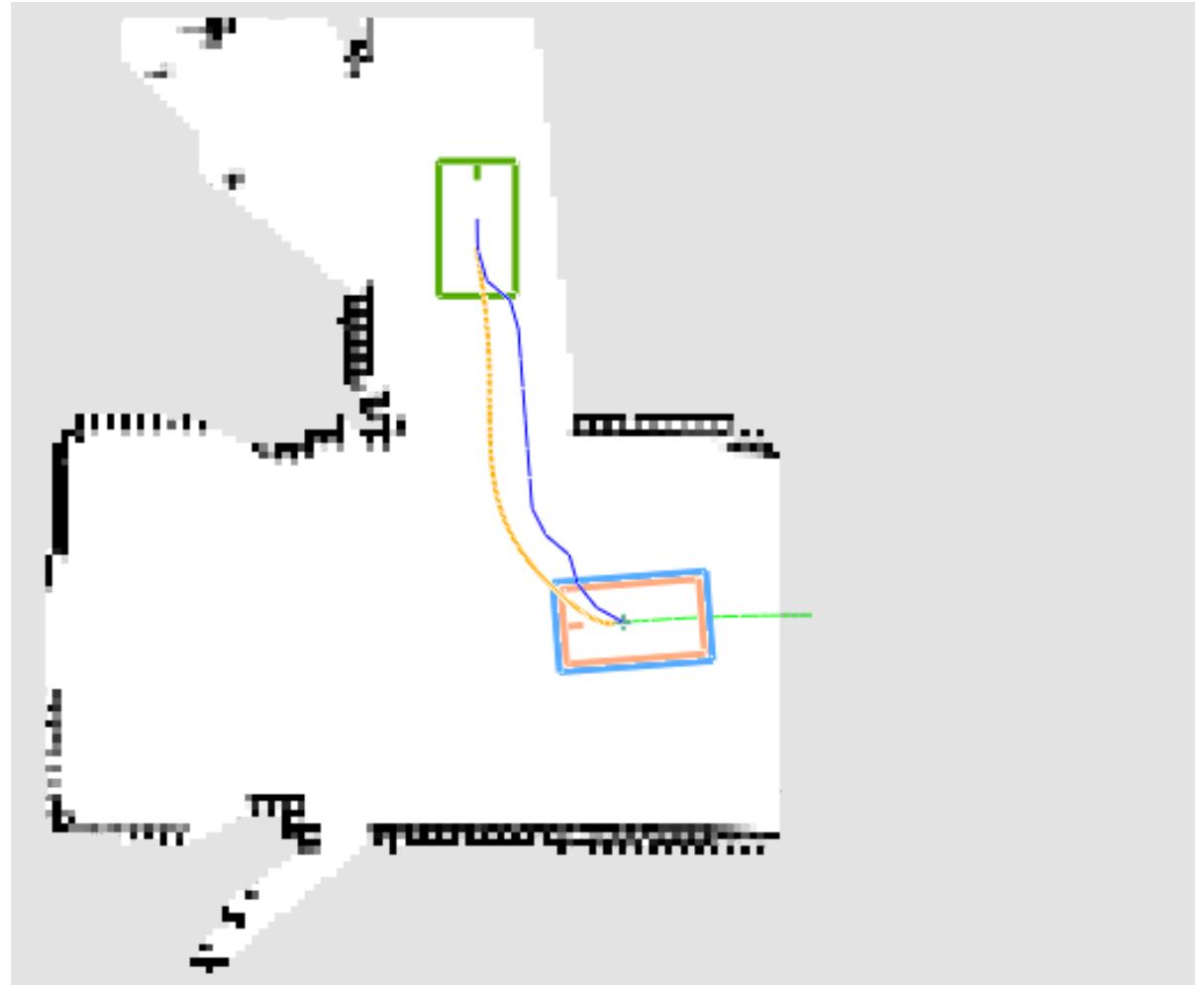
$$D' = L + 2D = L + \frac{\theta}{2\pi}\lambda$$

Laser range sensor with rotating mirror



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 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



We solve a more complex problem:

Simultaneous Localization and Mapping (SLAM)

Localization

Given

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

Determine the current position of the robot



Path planning

Given

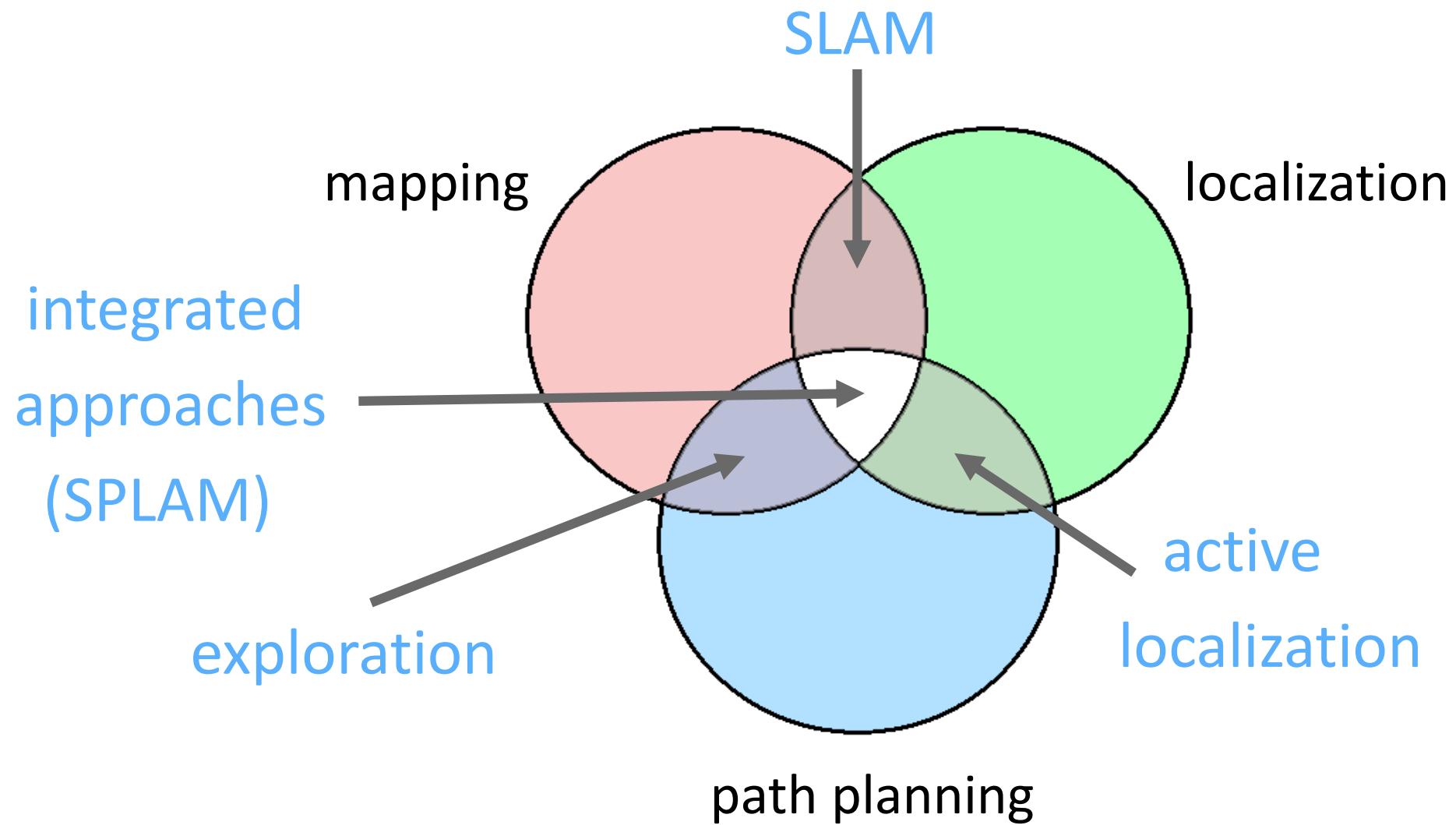
1. a localized robot
2. a map of
traversable regions



Determine (if it exists)
a path to reach a given
goal location



Mapping, localization, planning



Simultaneous Localization and Mapping

Estimate:

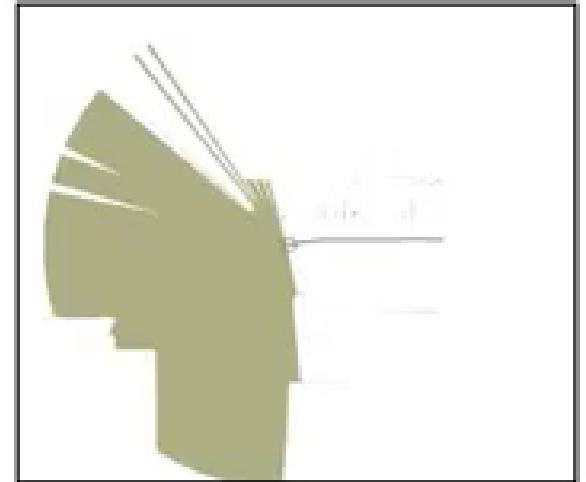
1. the **map** of the environment
 2. the **trajectory** of a moving device
- using a sequence of sensor measurements

these quantities
are correlated



SLAM

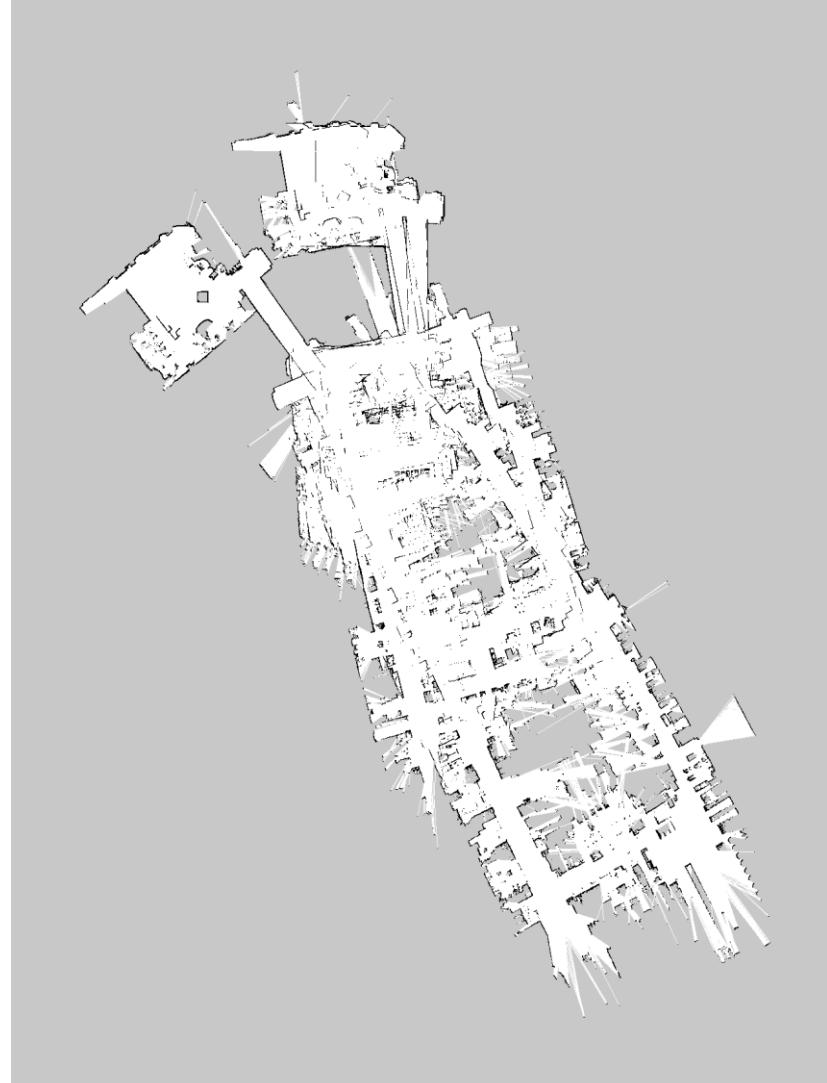
Determine the robot position **AND** the map, based on the sensor measurements



Graph-based SLAM

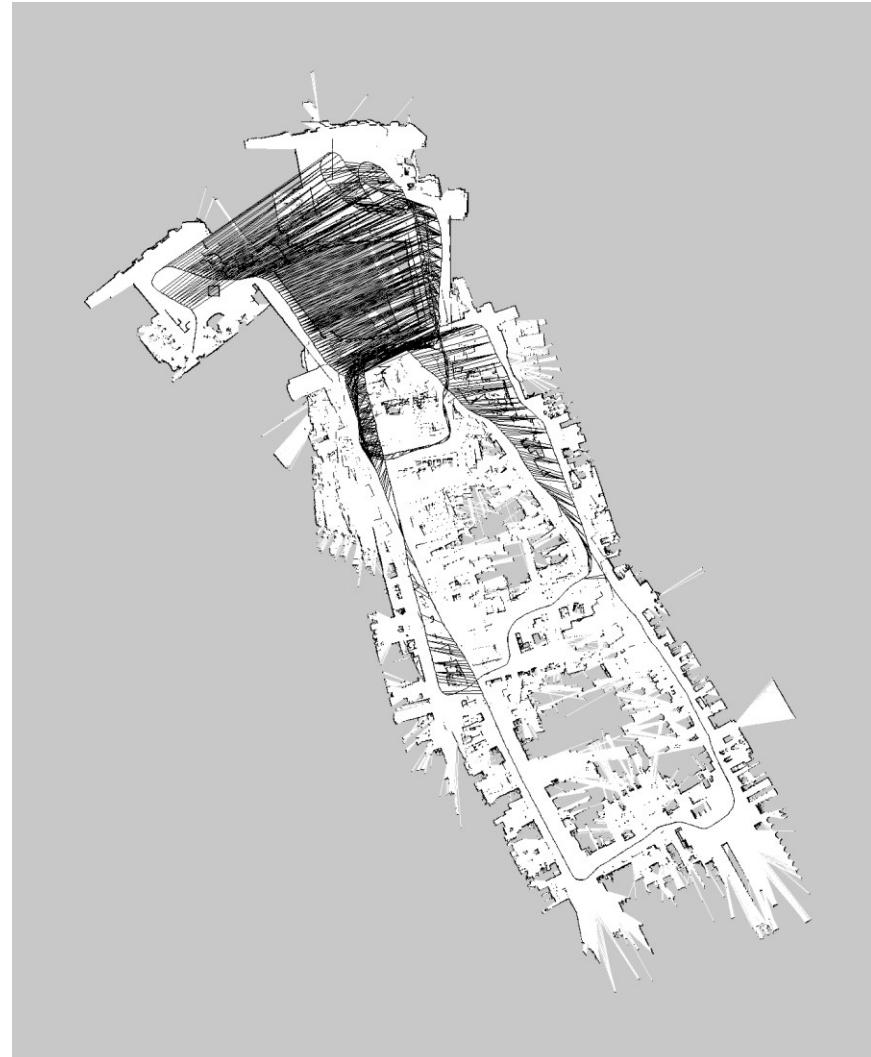
Problem described as a graph

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



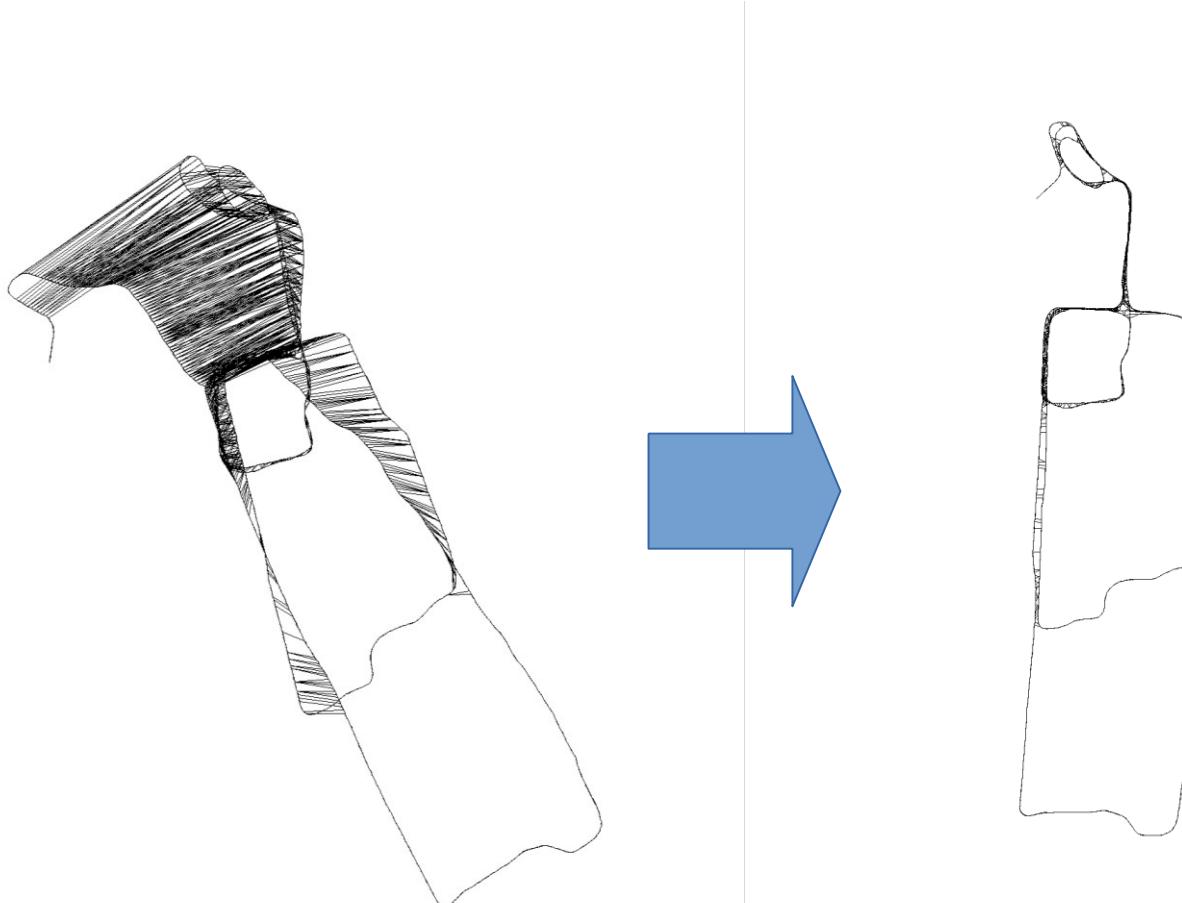
Graph-based SLAM

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



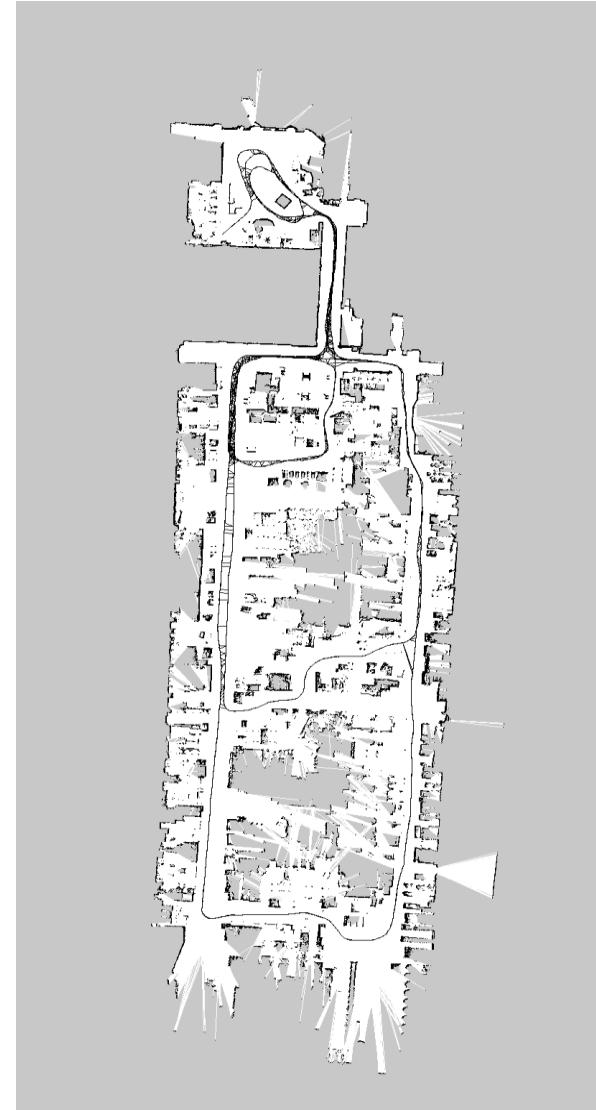
Graph-based SLAM

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



Graph-based SLAM

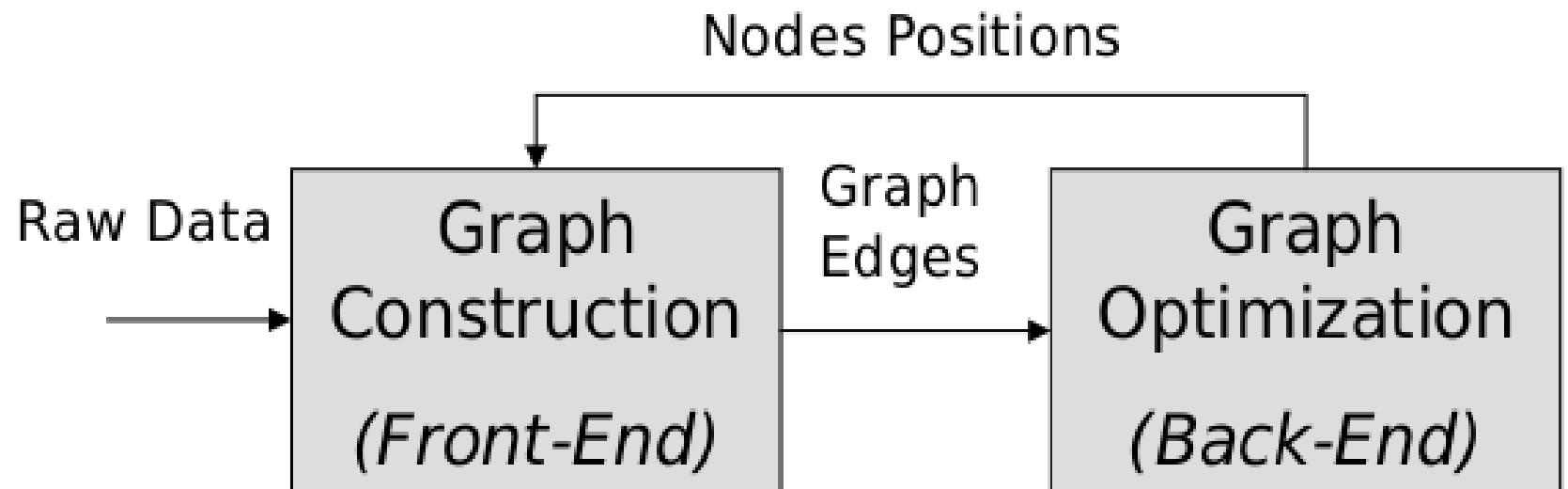
Then, we can render a map based on the known poses



Graph optimization

A general Graph-based SLAM algorithm interleaves the two steps

1. Graph construction
2. Graph optimization

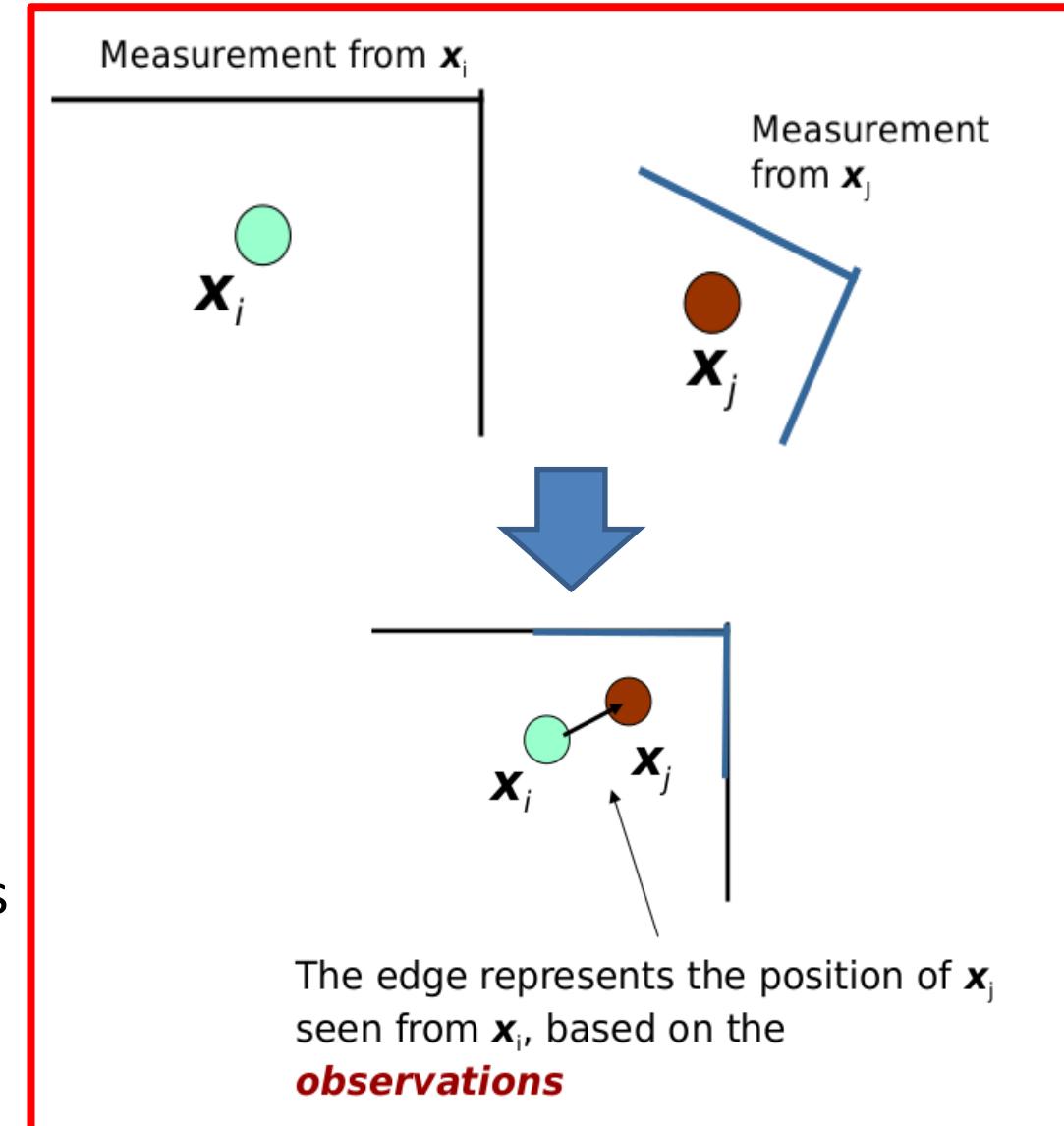


What Does the Graph Look Like?

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

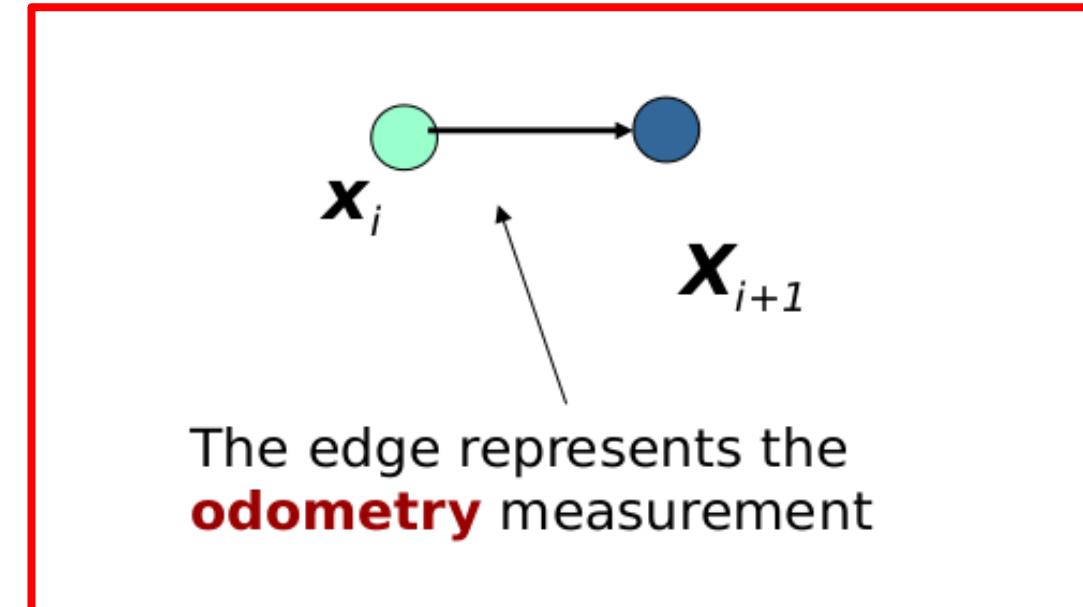


What Does the Graph Look Like?

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

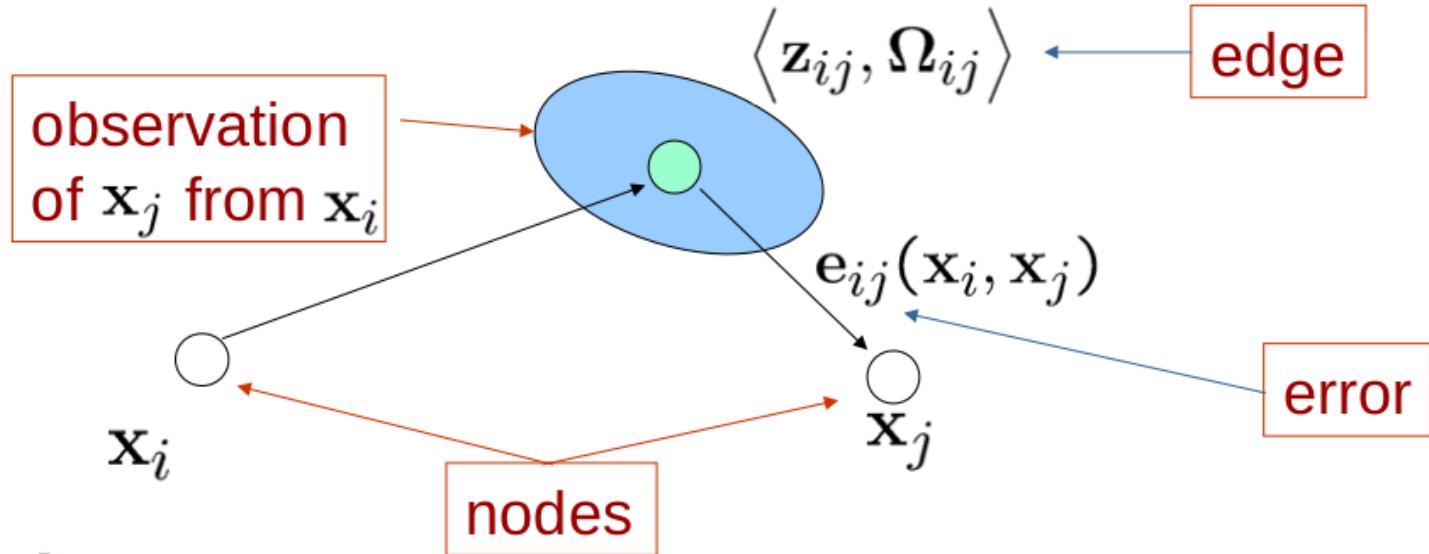
There is a constraint e_{ij} between the node x_i and the node x_j if

- 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:**

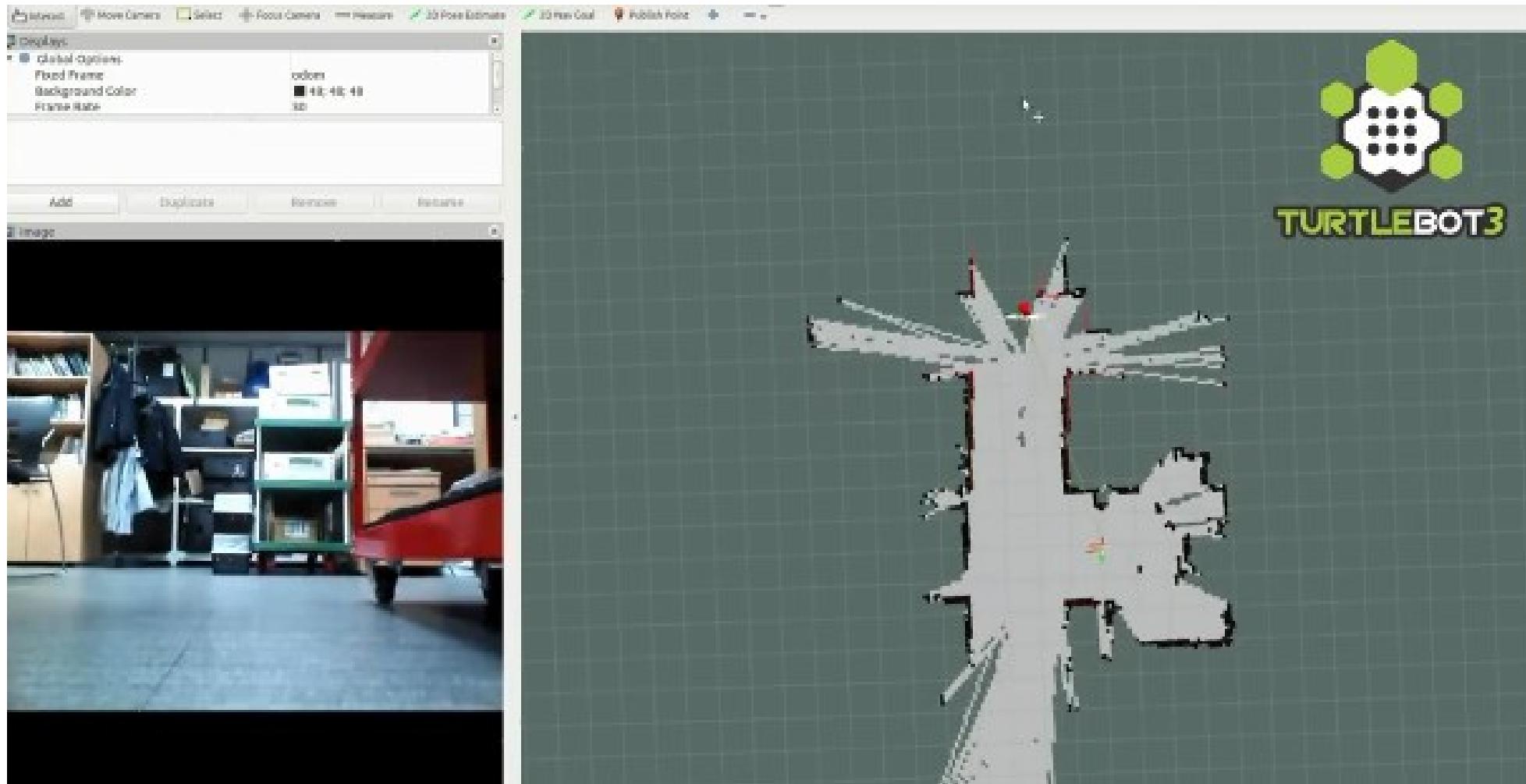
- Find the assignment of poses to the nodes of the graph which minimizes the negative log likelihood of the observations:

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

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

$\boldsymbol{\Omega}_{ij}$ is a matrix to encode the uncertainty of the edge

SLAM – Turtlebot3



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

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

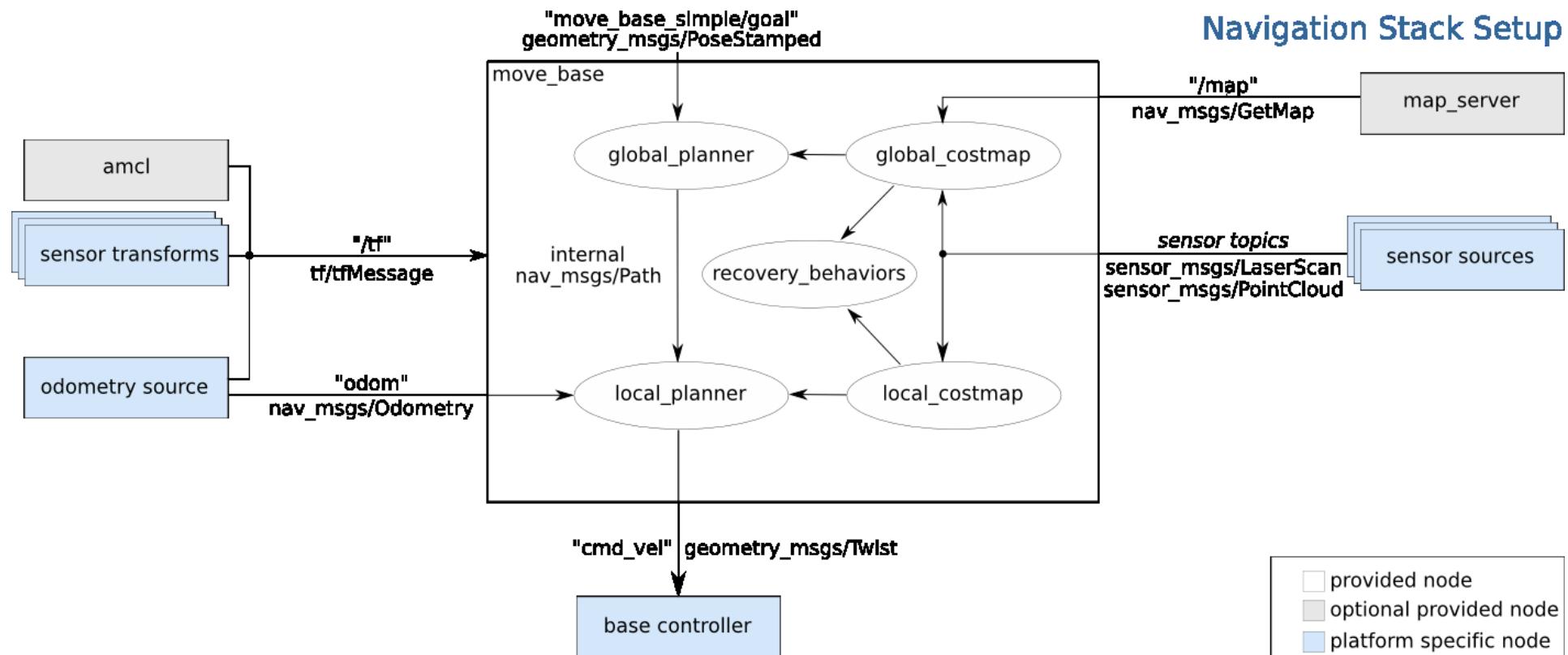
1. The environment changes (e.g., opening/closing doors)
2. It is dynamic (things might appear/disappear from the perception range of the robot)
3. The estimate is “noisy”

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

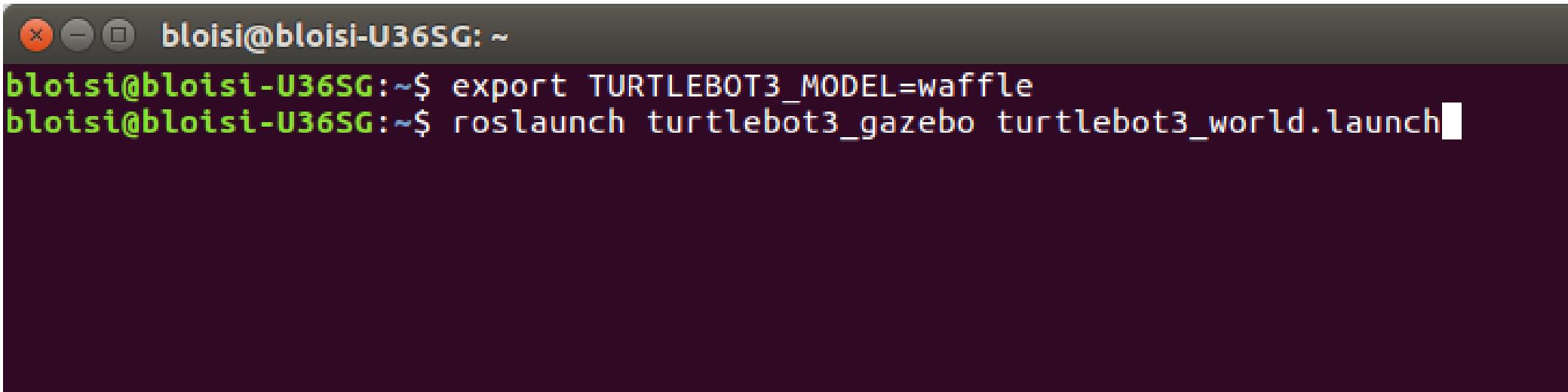
Useremo

- 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

In un primo terminale digitiamo

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

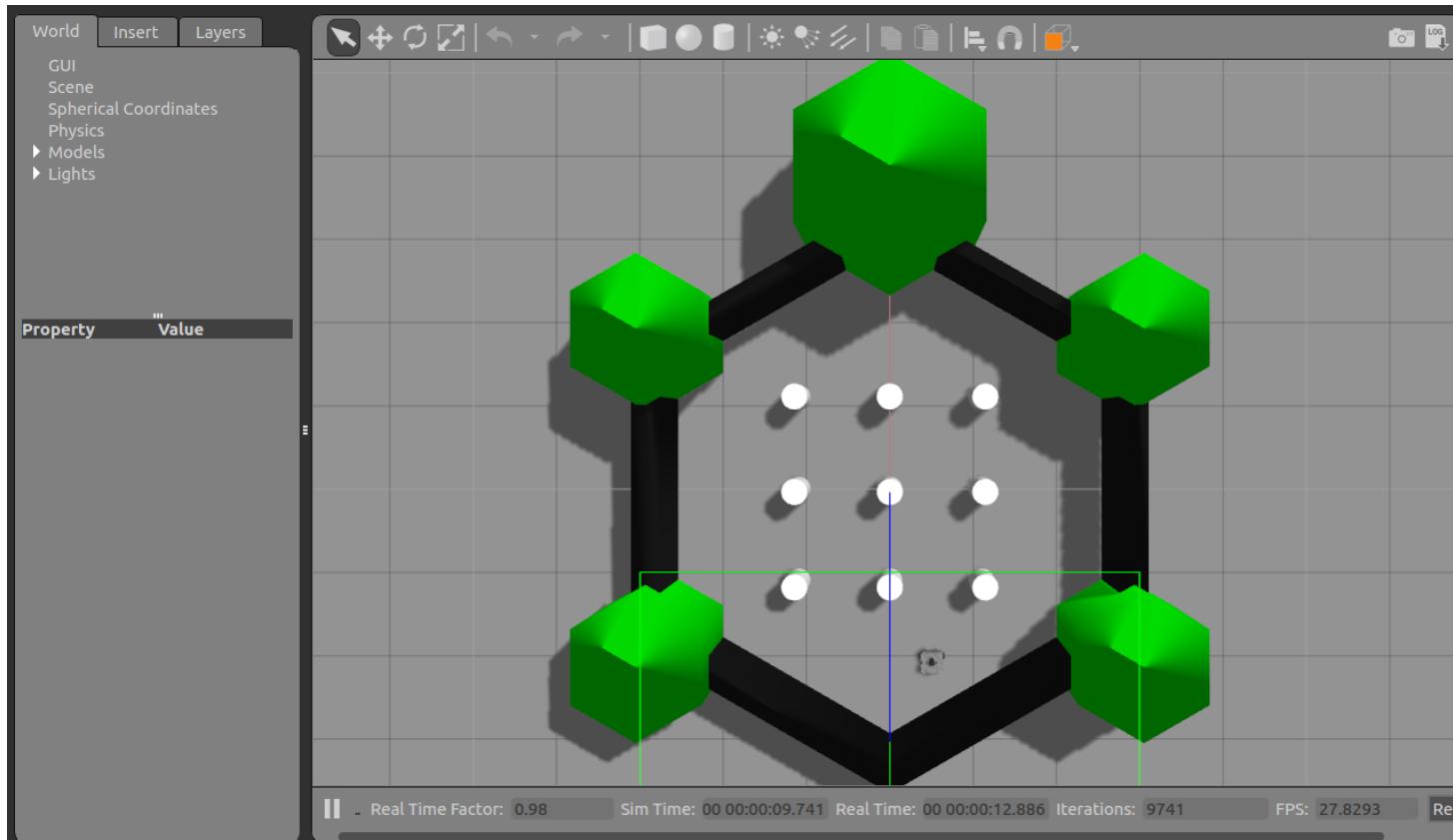


A screenshot of a terminal window titled "bloisi@bloisi-U36SG: ~". The window contains two lines of text: "bloisi@bloisi-U36SG:~\$ export TURTLEBOT3_MODEL=waffle" and "bloisi@bloisi-U36SG:~\$ roslaunch turtlebot3_gazebo turtlebot3_world.launch". The text is in white on a dark background.

Launch Gazebo

In un secondo terminale digitiamo

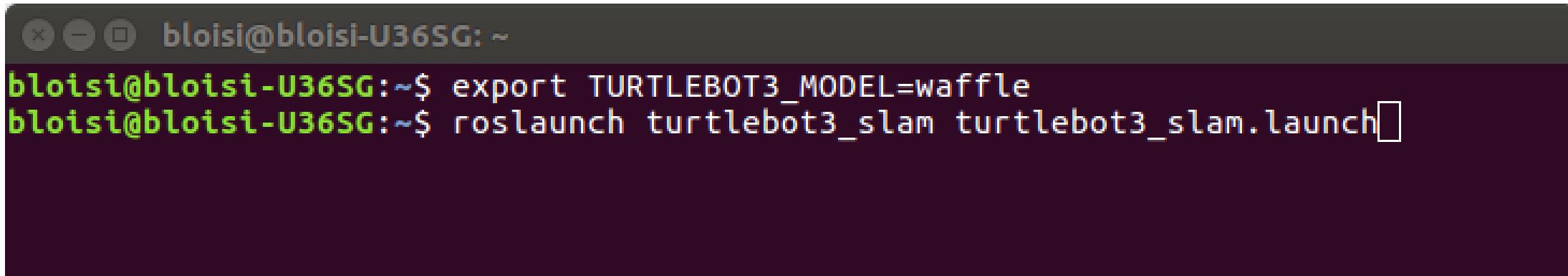
```
export TURTLEBOT3_MODEL=waffle  
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```



Launch SLAM

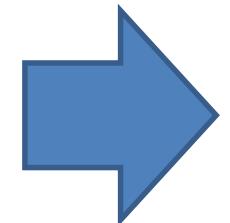
In un terzo terminale digitiamo

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



A screenshot of a terminal window titled "bloisi@bloisi-U36SG: ~". The window contains two commands: "export TURTLEBOT3_MODEL=waffle" and "roslaunch turtlebot3_slam turtlebot3_slam.launch". The text is in white on a dark background.

```
bloisi@bloisi-U36SG:~$ export TURTLEBOT3_MODEL=waffle  
bloisi@bloisi-U36SG:~$ roslaunch turtlebot3_slam turtlebot3_slam.launch
```

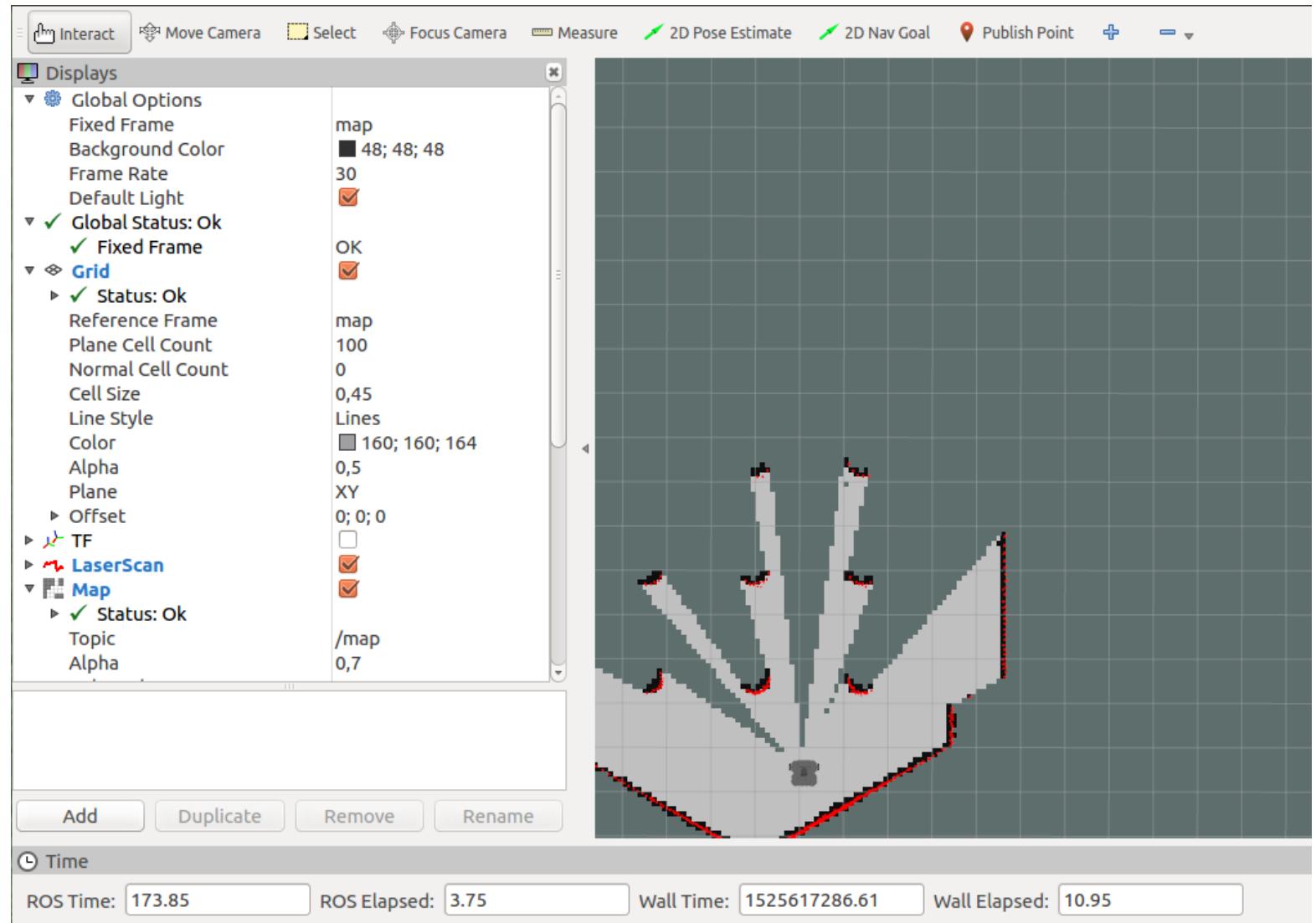


Launch SLAM

```
× - ⊞ /home/bloisi/catkin_ws/src/turtlebot3/turtlebot3_slam/launch/turtlebot3_slam.launch h
Laser Pose= -2.064 -0.500004 3.14007
m_count 7
Average Scan Matching Score=322.728
neff= 120
Registering Scans:Done
update frame 39
update ld=7.41779e-08 ad=5.33264e-07
Laser Pose= -2.064 -0.500004 3.14007
m_count 8
Average Scan Matching Score=322.791
neff= 120
Registering Scans:Done
update frame 40
update ld=6.27647e-08 ad=4.47942e-07
Laser Pose= -2.064 -0.500004 3.14007
m_count 9
Average Scan Matching Score=322.845
neff= 120
Registering Scans:Done
update frame 41
update ld=6.49517e-08 ad=4.6394e-07
Laser Pose= -2.064 -0.500004 3.14007
m_count 10
```

RViz

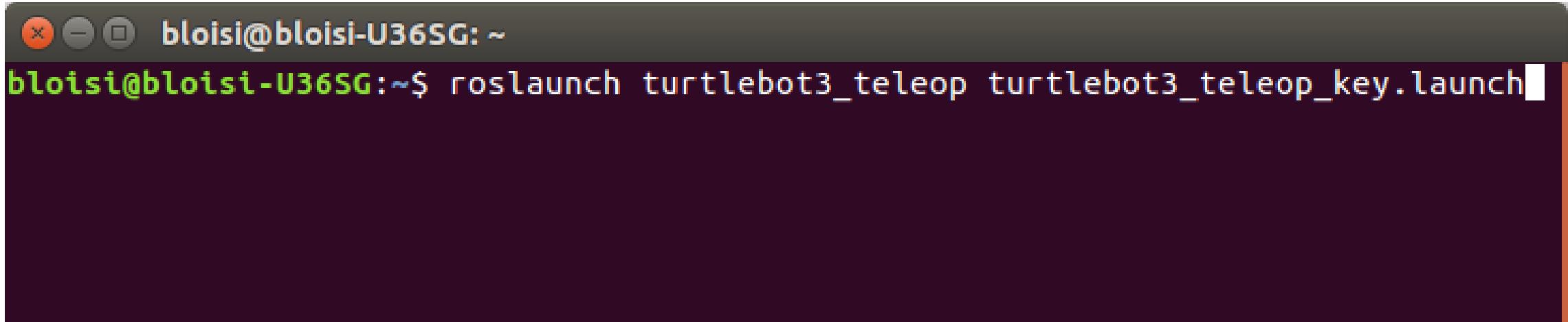
Lanciando Rviz
potremo
visualizzare i dati
che i sensori del
robot stanno
inviando



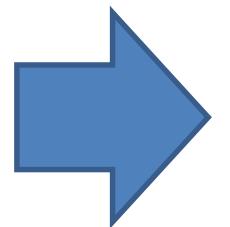
Remotely Control Turtlebot3

In un quarto terminale digitiamo

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```



A screenshot of a terminal window with a dark background and light-colored text. The window title bar shows the user's name and host: "bloisi@bloisi-U36SG: ~". The main area of the terminal contains the command "roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch" in green text, which is the command to launch the key-based teleop node for a Turtlebot3 robot.



Remotely Control Turtlebot3

```
/home/bloisi/catkin_ws/src/turtlebot3/turtlebot3_teleop/launch/turtlebot3_teleop_keyboard
* /rosversion: 1.12.13

NODES
/
  turtlebot3_teleop_keyboard (turtlebot3_teleop/turtlebot3_teleop_key)

ROS_MASTER_URI=http://localhost:11311

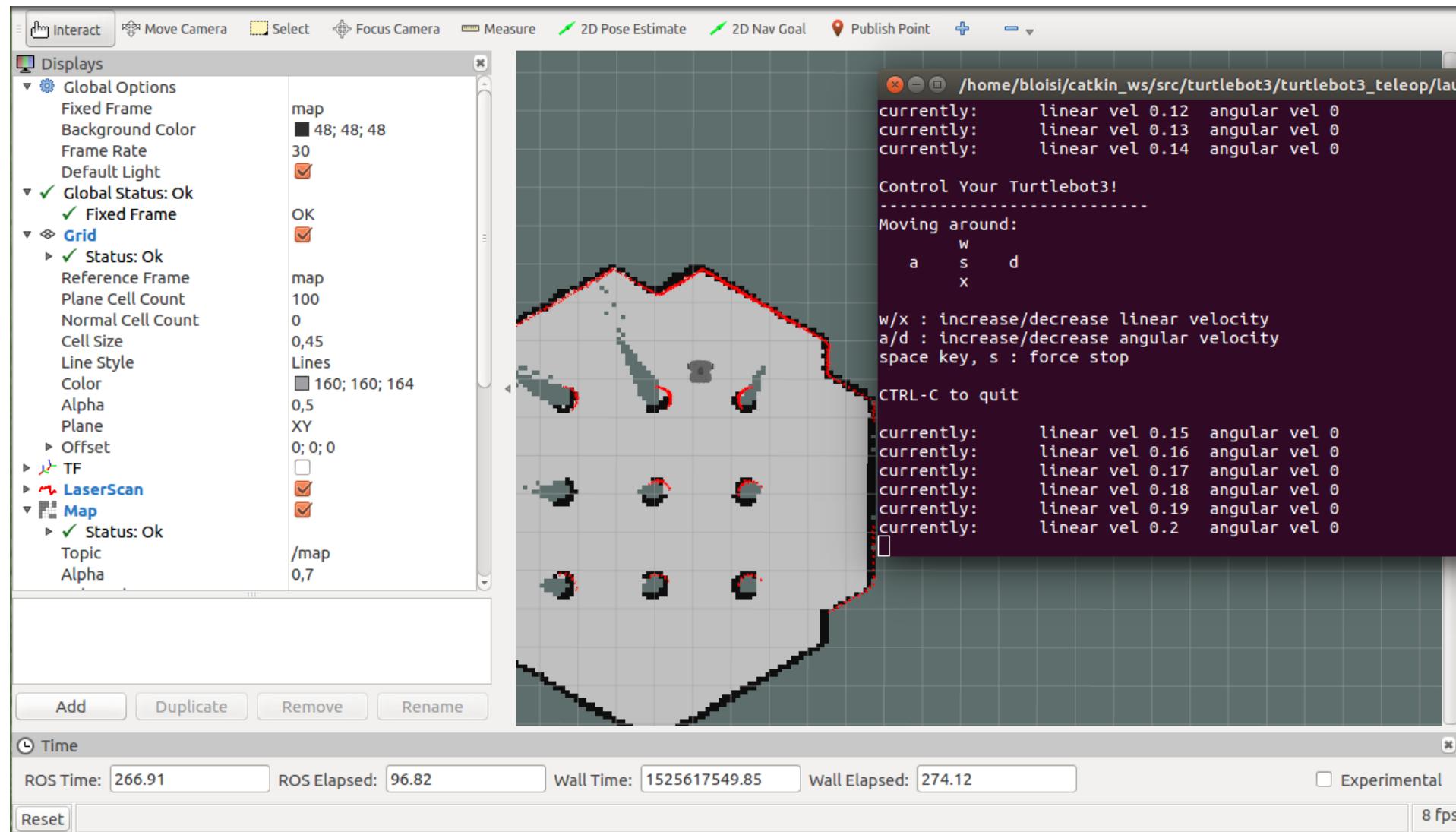
process[turtlebot3_teleop_keyboard-1]: started with pid [6305]

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
```

Costruiamo la mappa



Gazebo + RViz views

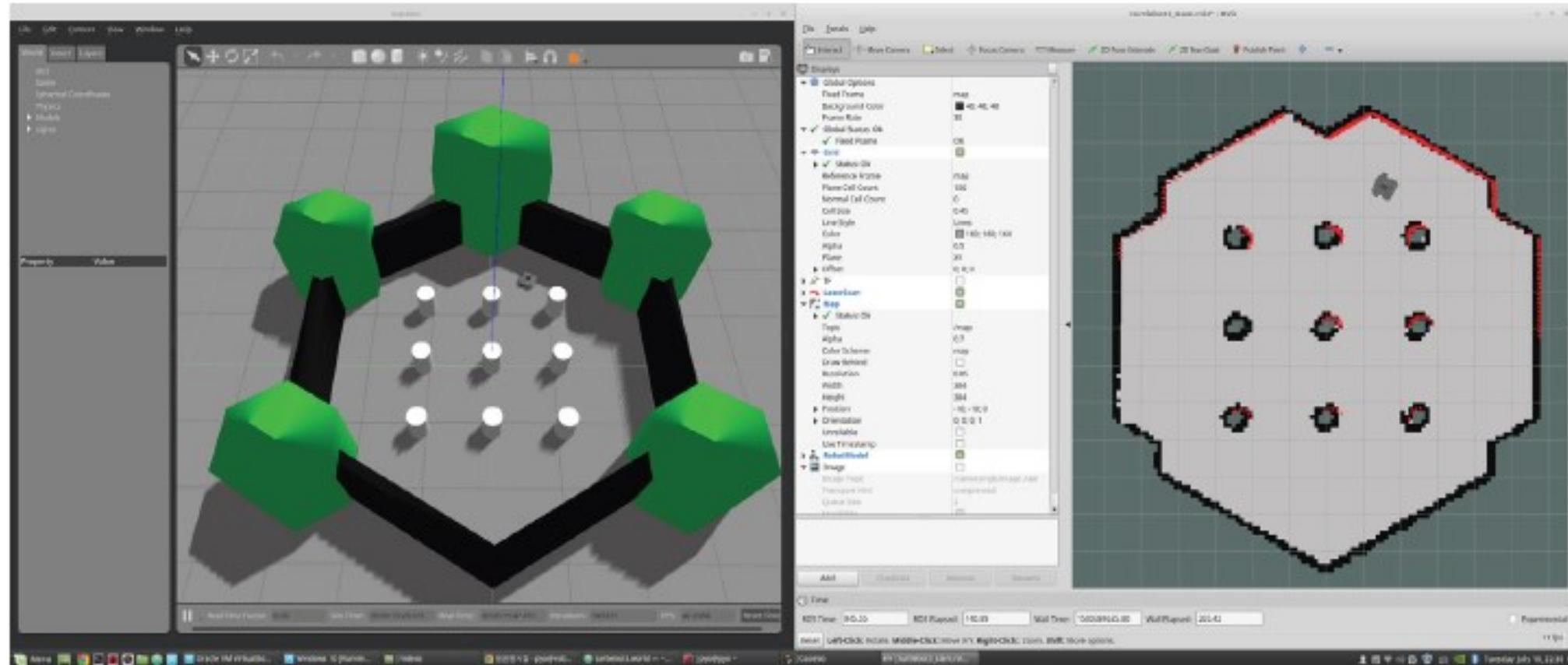
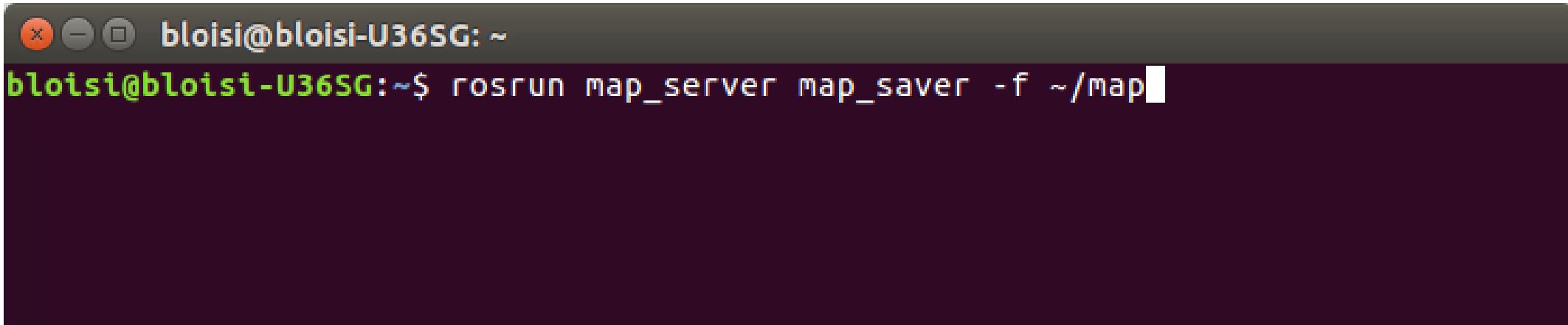


FIGURE 10-20 Running SLAM on Gazebo (Left: Gazebo, Right: RViz)

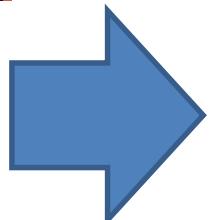
Save the Map

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

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



A screenshot of a terminal window titled "bloisi@bloisi-U36SG: ~". The user has typed the command "rosrun map_server map_saver -f ~/map" and is pressing the Enter key. The terminal has a dark background and light-colored text.

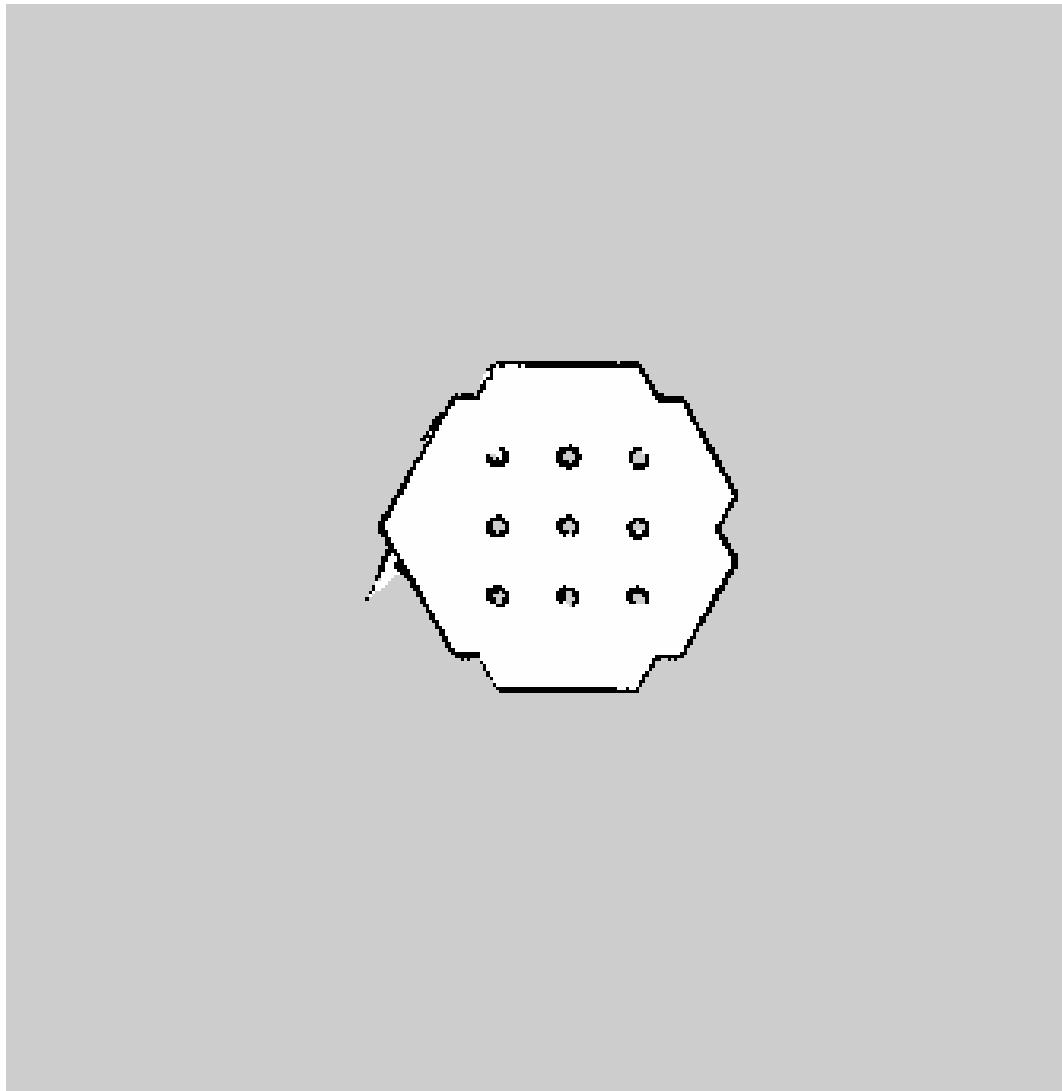


Save the Map

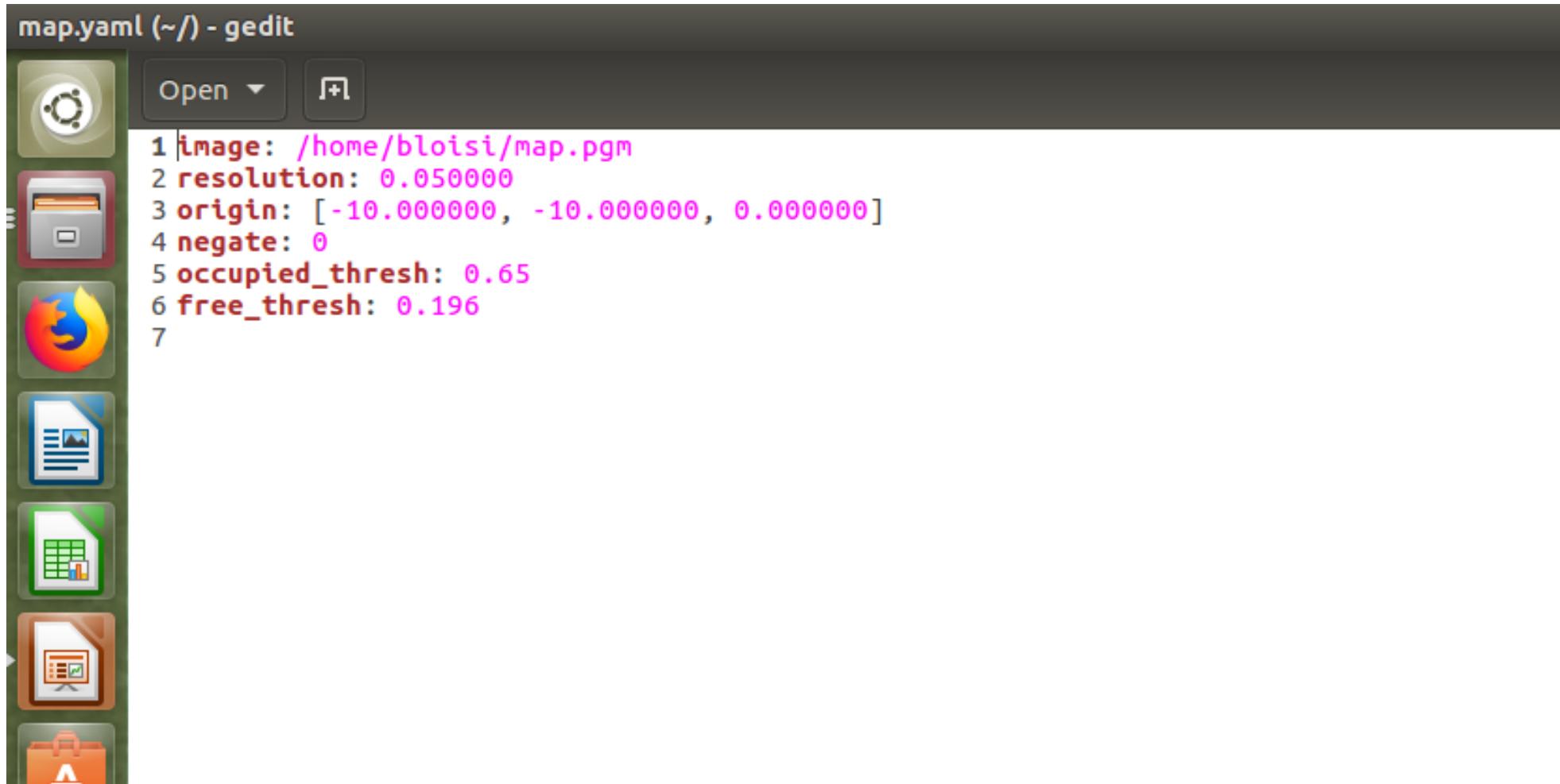
```
bloisi@bloisi-U36SG: ~$ rosruncat map_server map_saver -f ~/map
[ INFO] [1525617922.440250407]: Waiting for the map
[ INFO] [1525617922.705725024, 410.589000000]: Received a 384 X 384 map @ 0.050
m/pix
[ INFO] [1525617922.706387150, 410.589000000]: Writing map occupancy data to /ho
me/bloisi/map.pgm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



The screenshot shows a Gedit text editor window titled "map.yaml (~/) - gedit". The window contains the following YAML configuration:

```
1 image: /home/bloisi/map.pgm
2 resolution: 0.050000
3 origin: [-10.000000, -10.000000, 0.000000]
4 negate: 0
5 occupied_thresh: 0.65
6 free_thresh: 0.196
7
```

Navigazione con Turtlebot3

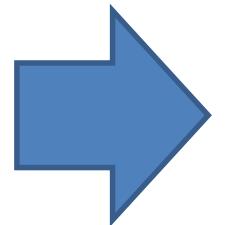
Per poter procedere con la navigazione

1. Terminare tutti i processi attivi
2. Digitare in un primo terminale

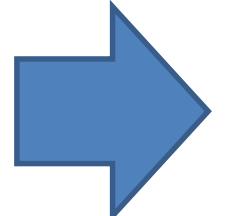
```
export TURTLEBOT3_MODEL=waffle  
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

3. Aprire un secondo terminale e digitare

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

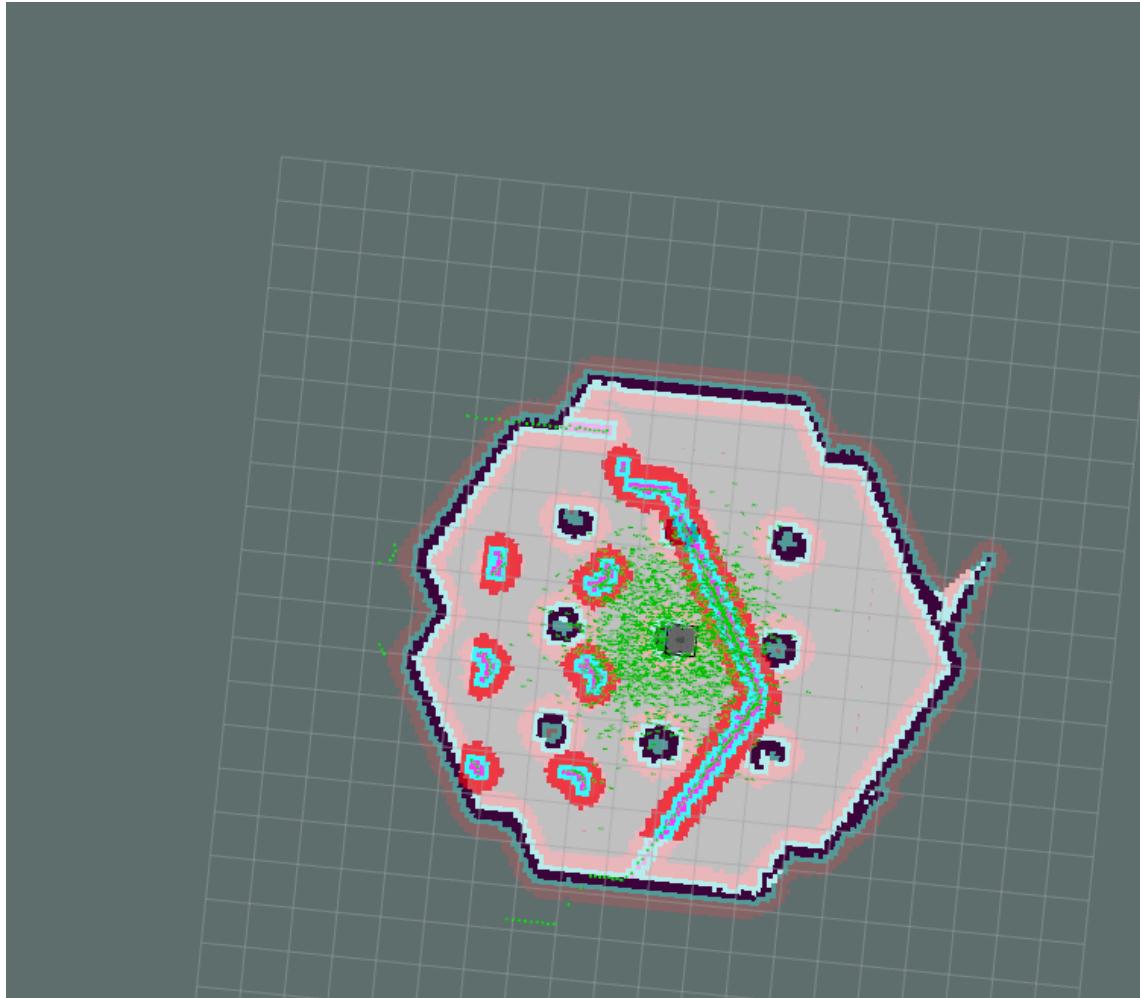


Navigazione con Turtlebot3

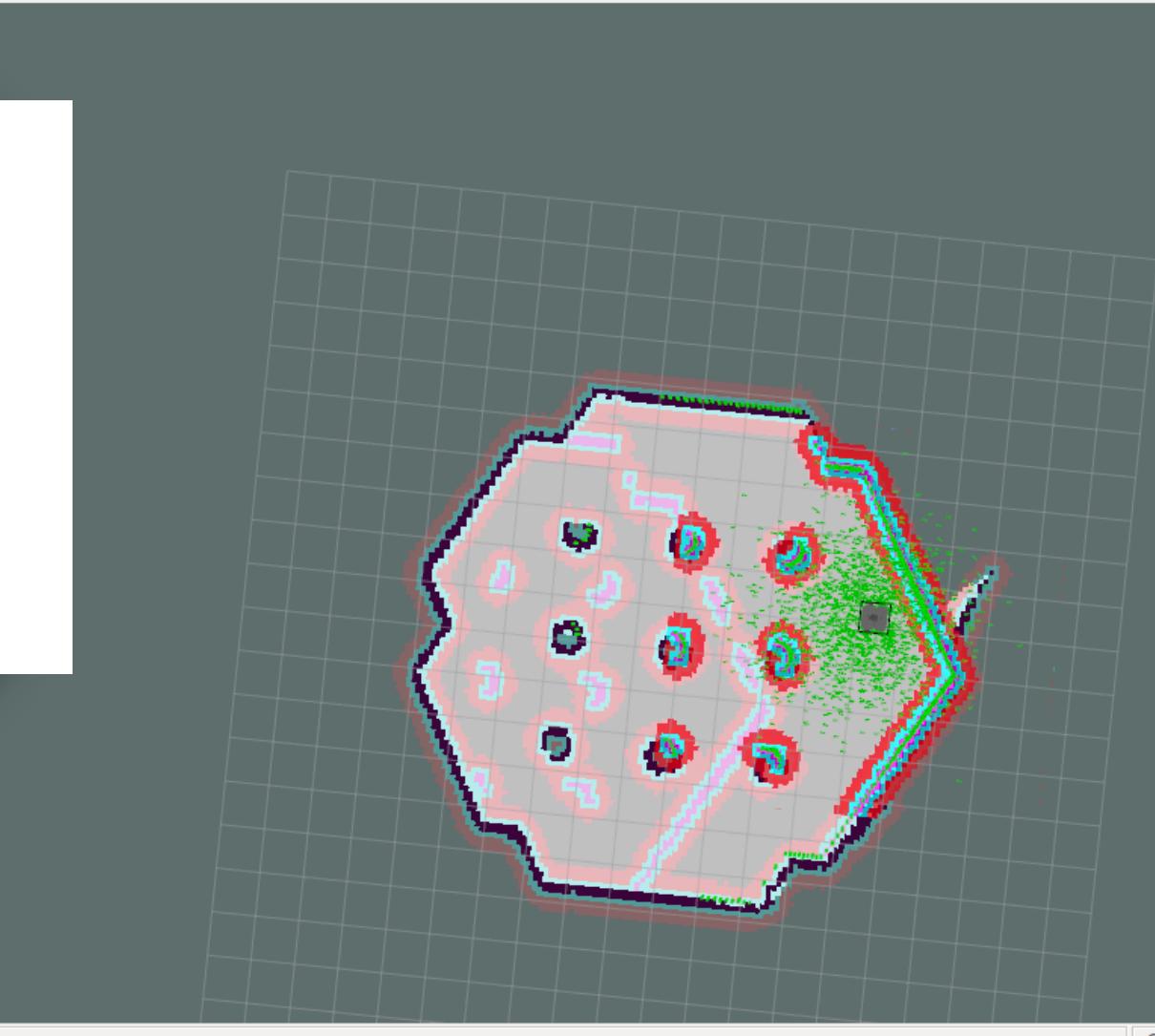
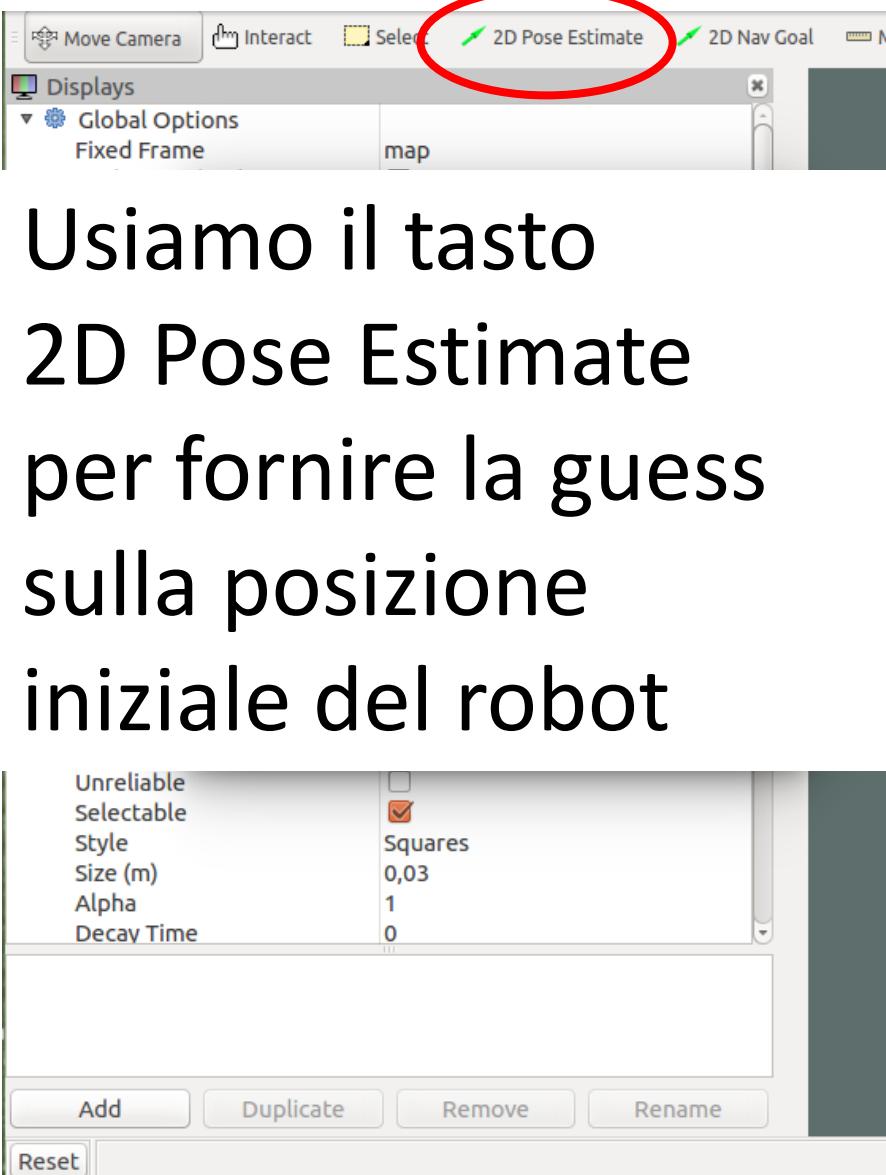


```
bloisi@bloisi-U36SG: ~
bloisi@bloisi-U36SG:~$ export TURTLEBOT3_MODEL=waffle
bloisi@bloisi-U36SG:~$ roslaunch turtlebot3_navigation turtlebot3_navigation.lau
nch map_file:=$HOME/map.yaml
```

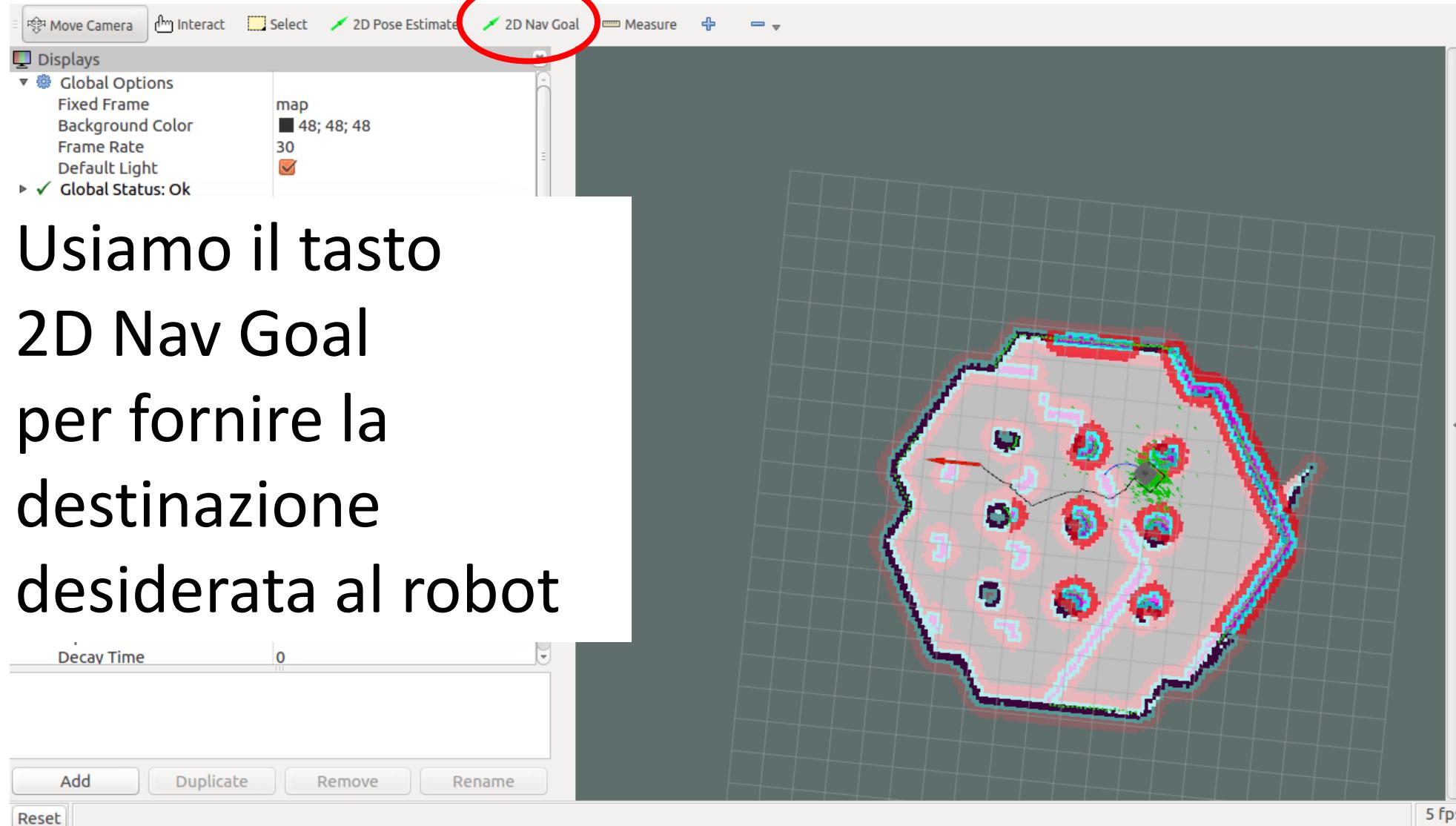
Navigazione con Turtlebot3



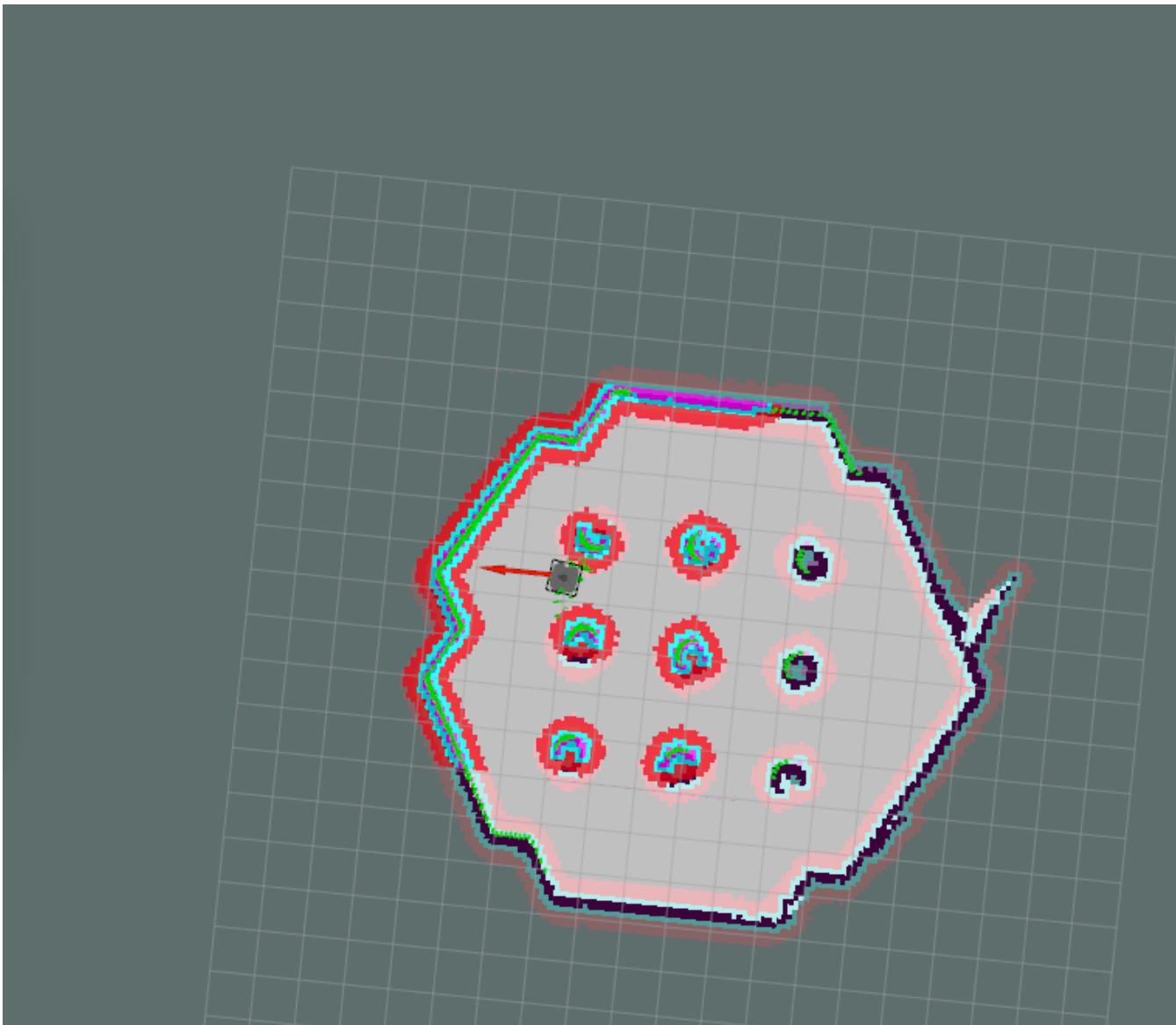
Pose Estimate



Navigation Goal

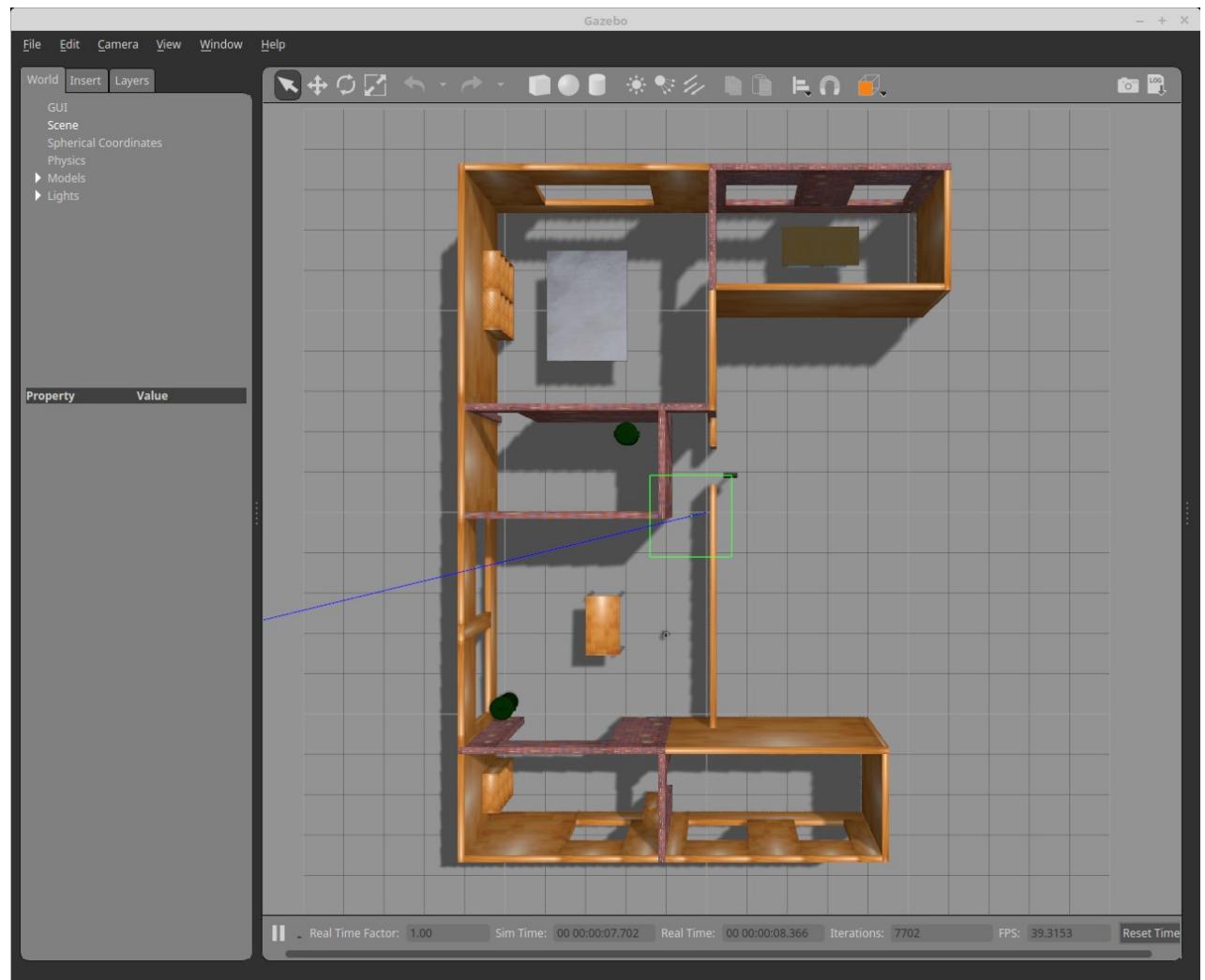


Goal raggiunto



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



**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Visione e Percezione
A.A. 2019/2020

Navigazione in ROS

Maggio 2020

