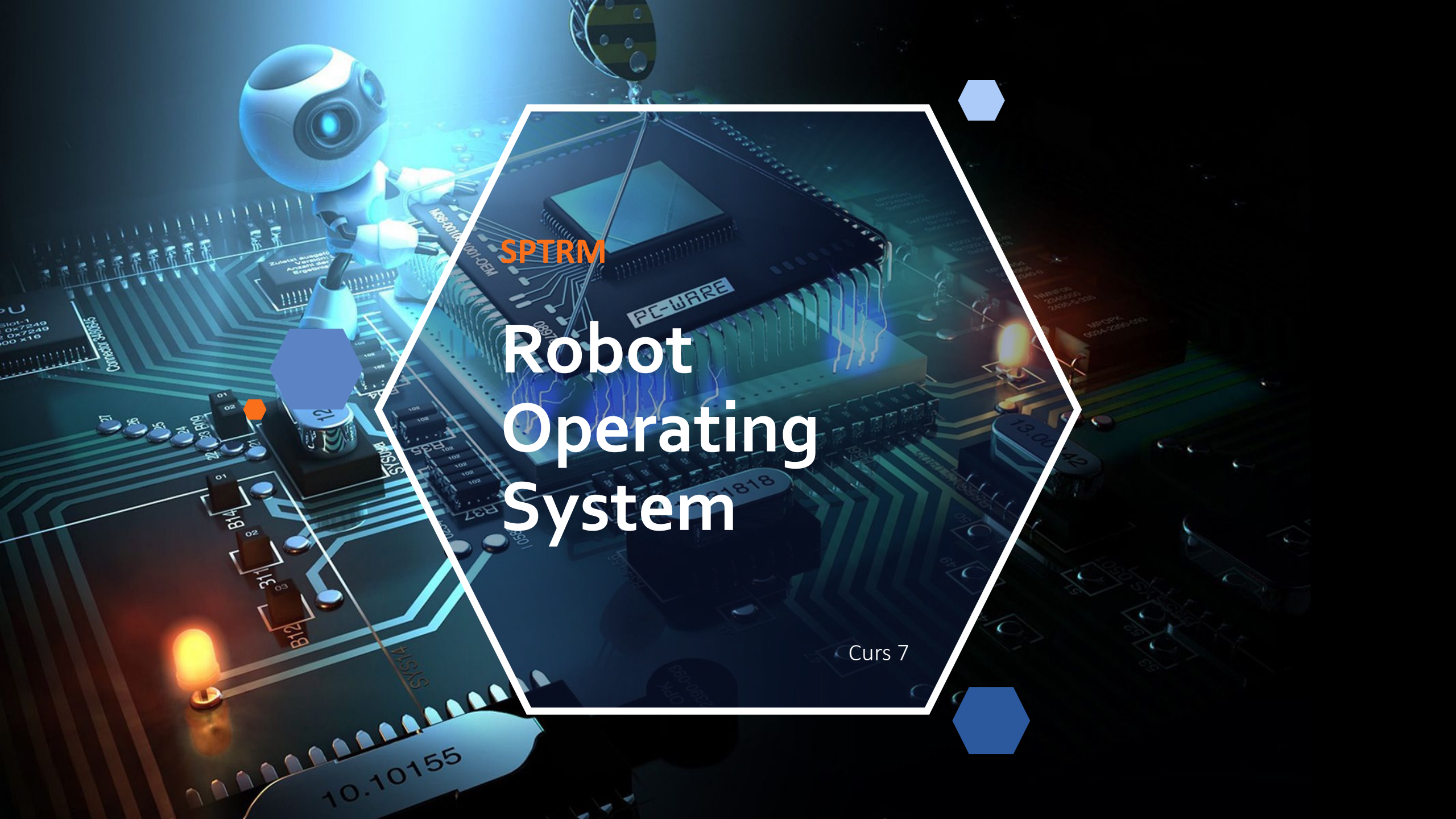


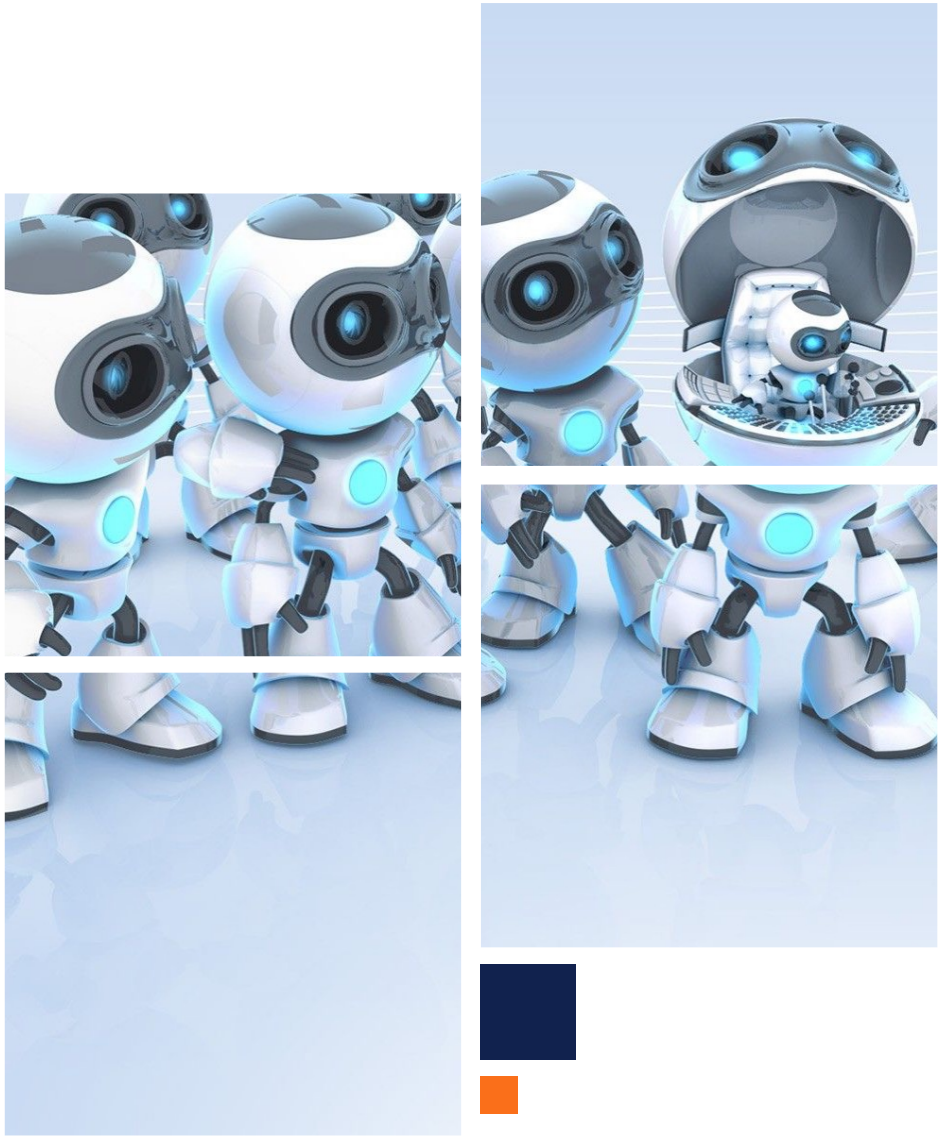
SPTRM

Robot Operating System

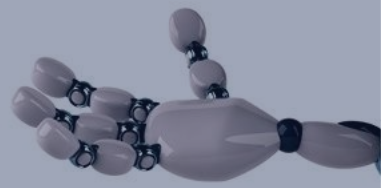
Curs 7

Agenda

- Recap
- SLAM
  - Types
  - Algorithms
- ROS Navigation Stack
  - Map generation
  - Localization
  - Trajectory planning



Recap

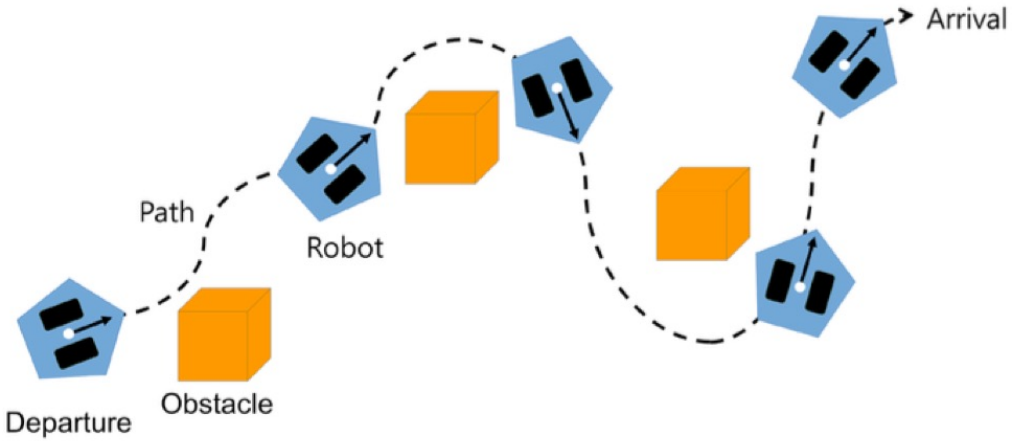


From the last episode...

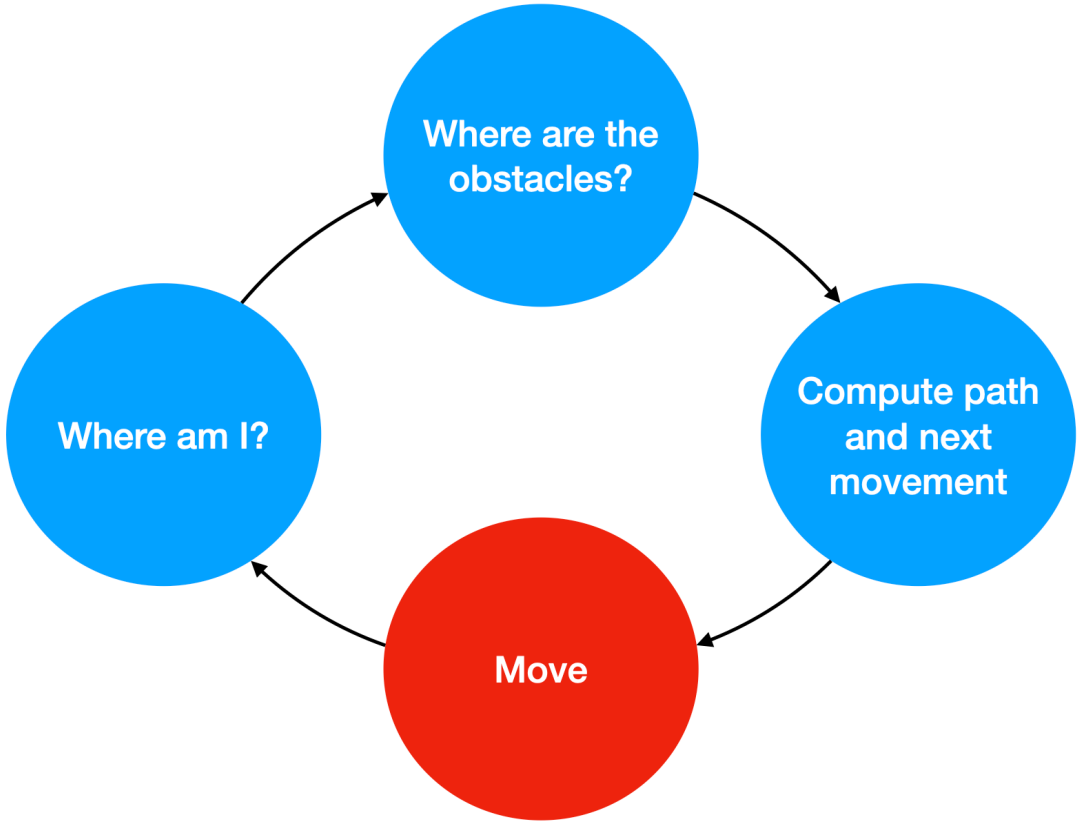
Motion planning. Navigation

- **Motion planning** is a technique that finds the optimum path that moves the robot gradually from initial pose to goal pose without collisions with the world or itself
- Inputs
  - Initial pose of the robot
  - Goal pose of the robot
  - Geometrical description of the world

- **Navigation** is the movement of the robot from the current position to a defined destination, preferably using an optimized route



Continuous problem



MoveIt!

- **MoveIt!** - set of packages and tools for doing mobile manipulation in ROS
- Supports multiple robots
- Supports multiple actions and scenarios:
  - Pick and place
  - Grasping
  - Motion planning using invers kinematics

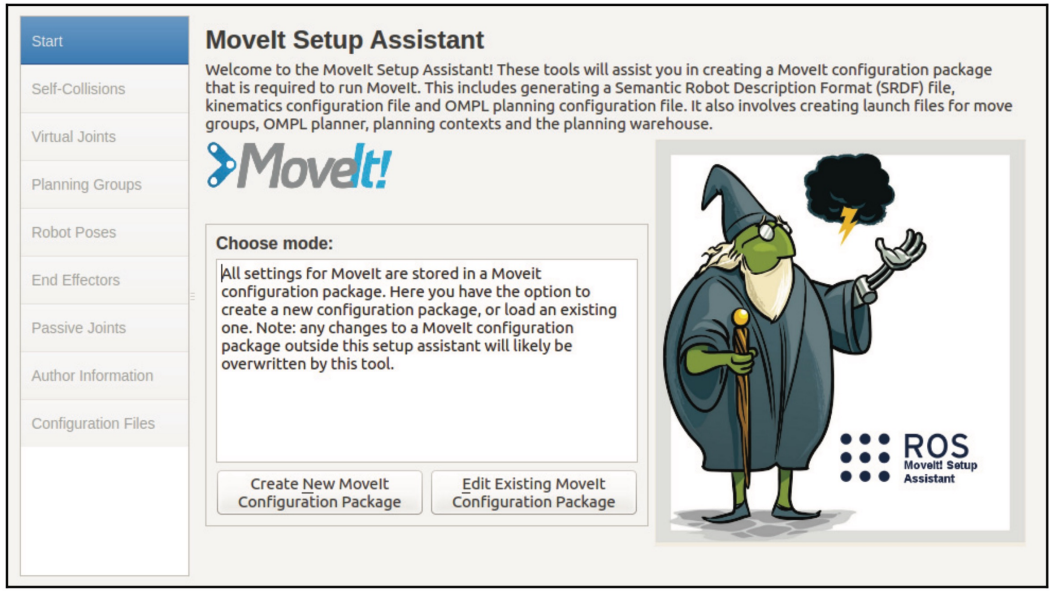
- http://moveit.ros.org

- State-of-the-art software for
  - Motion planning
  - Manipulation
  - 3D perception
  - Kinematics
  - collision checking
  - control
  - Navigation
- Vizualization
  - Dedicated GUI
  - RViz plugin

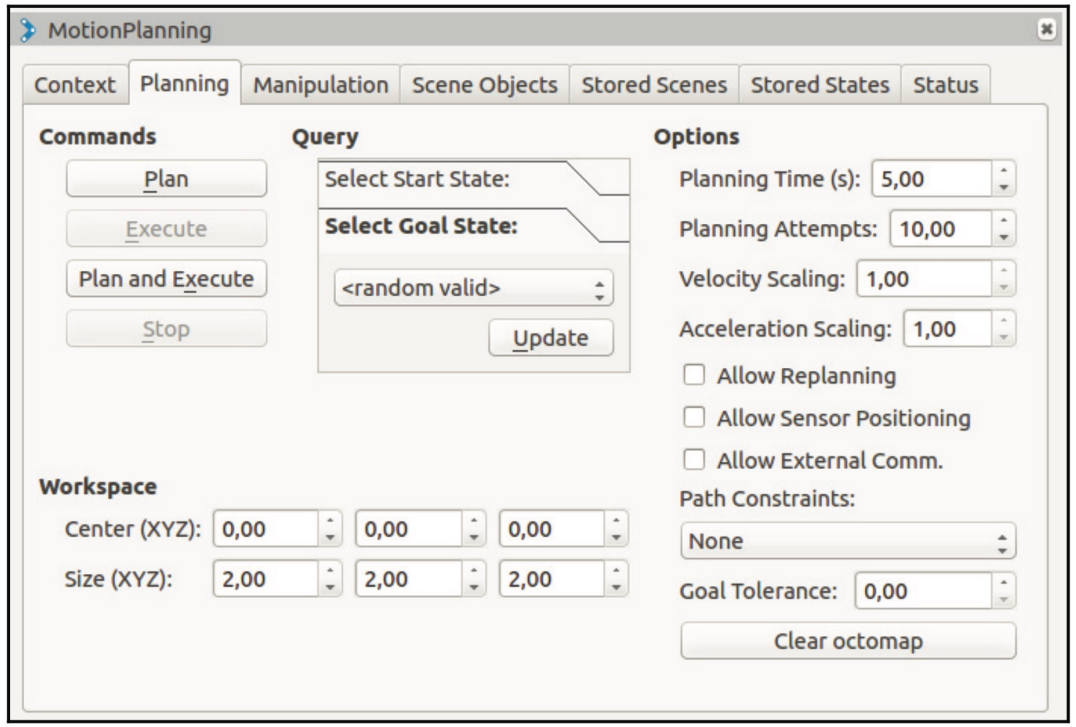
MoveIt! Setup Assistant tool

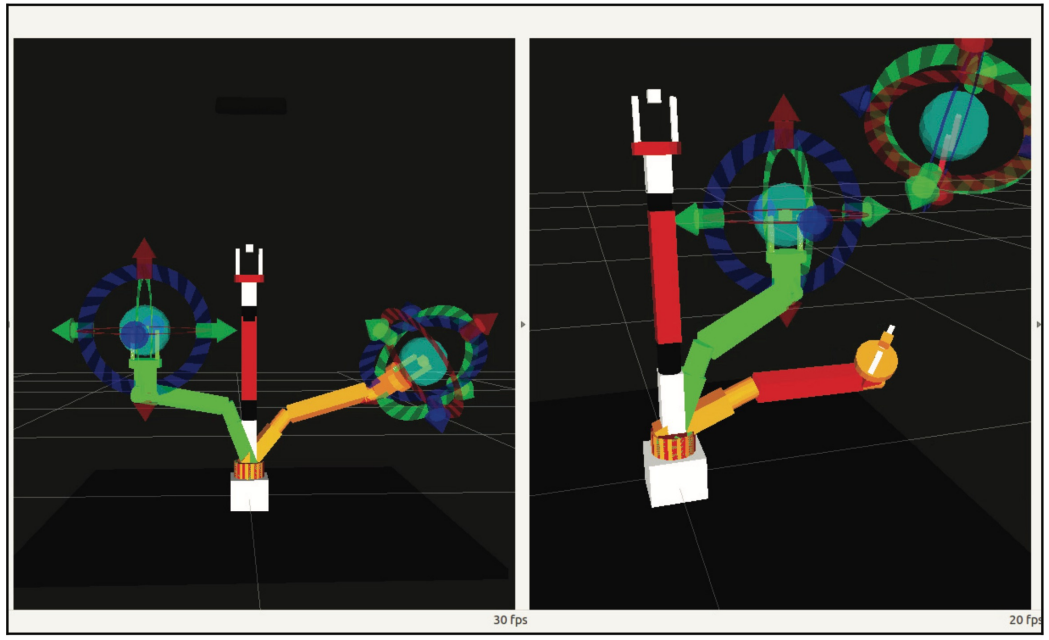
- GUI for configuring any robot in MoveIt!
- Generates the SRDF, configuration files, launch files and scripts from the URDF
- Encapsulated all information into configuration packages that can be edited afterwards

```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

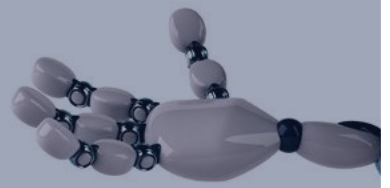


RViz Motion planning





Localization and Mapping

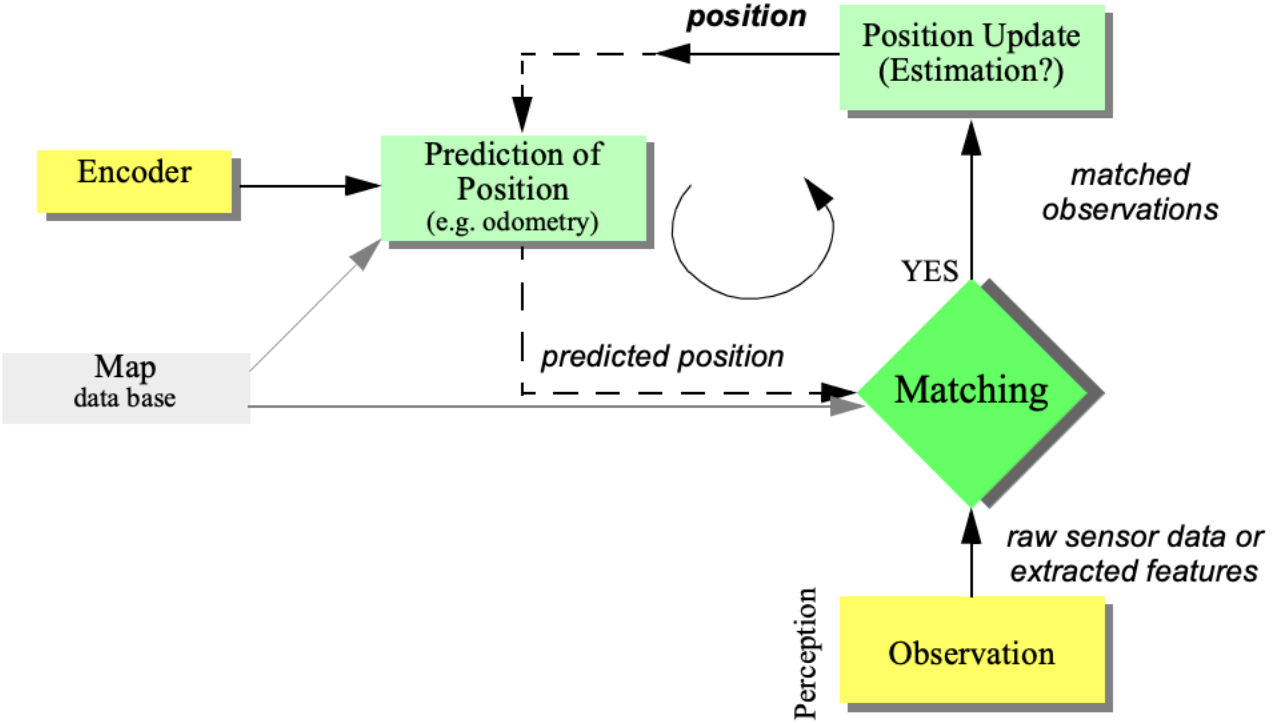


SLAM

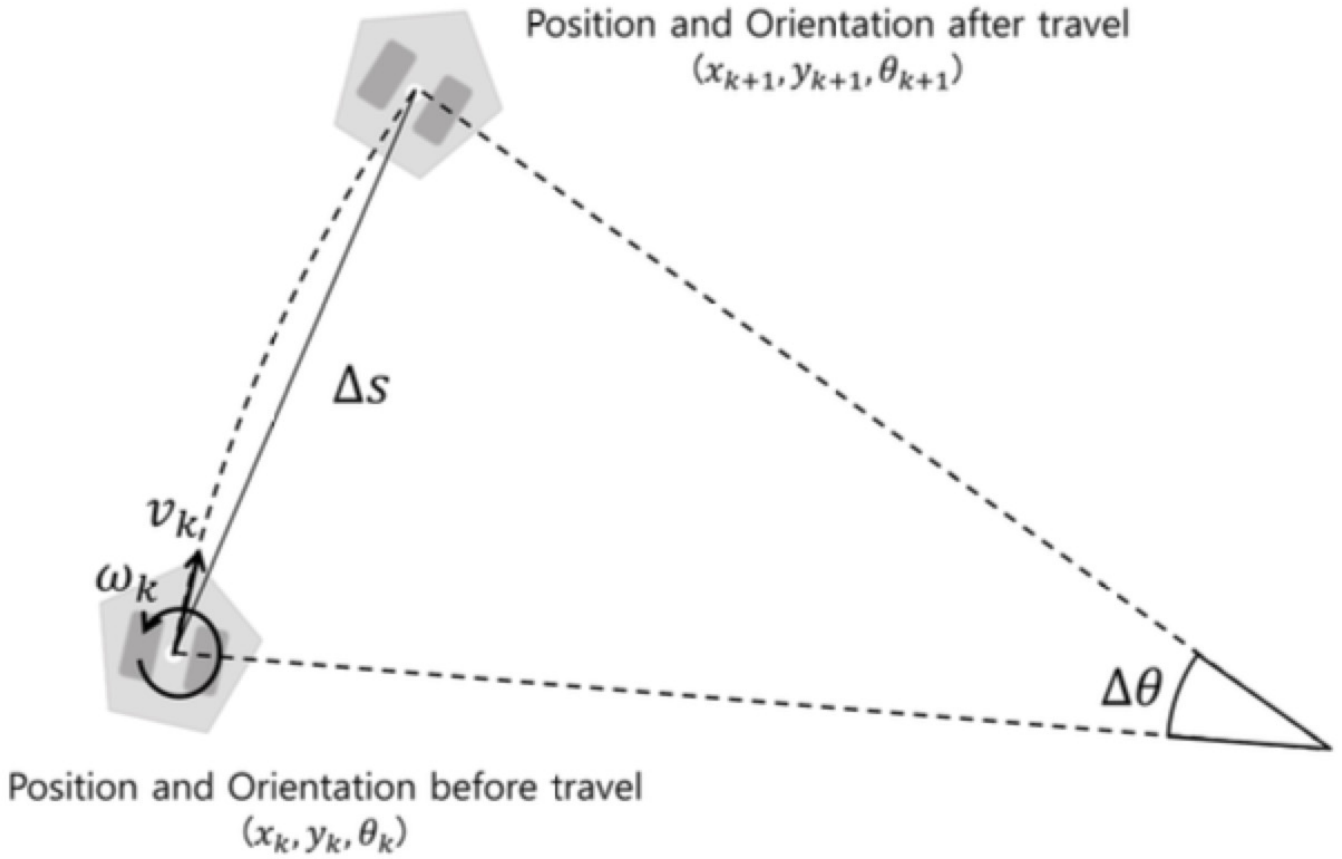
Robot pose

- pose = position (x, y, z) + orientation (w)
- outdoor: GPS, DGPS (Differential GPS)
- indoor: dead reckoning (DR) method
  - starts from a fixed known point
  - calculates the new position based on data recorded by the robot (direction, speed, etc)
  - increased accuracy by using inertial information

Robot Localization



Dead Reckoning method



SLAM

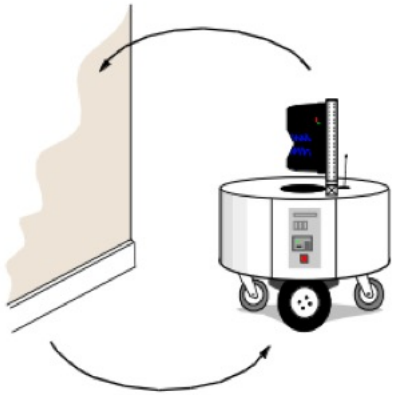
- **Simultaneous localization and mapping (SLAM)** is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it

- Fundamental problem for robots to become truly autonomous

- Inputs:
  - control data
  - observations (sensor data)

- Outputs:
  - agent location
  - map of the environment

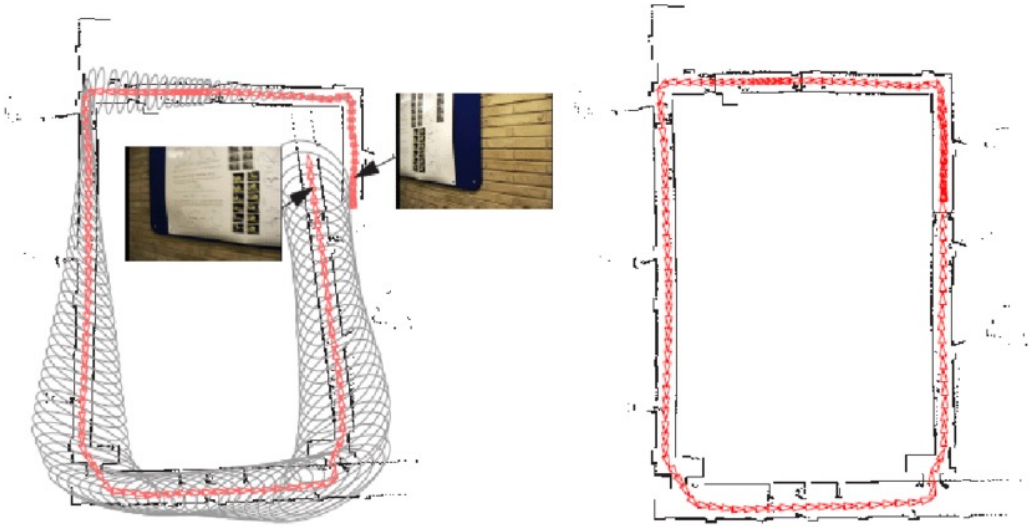
Difficulty



- SLAM problem is a **chicken-or-egg** problem:
  - A map is needed for localization
  - A good pose estimate is needed for mapping

SLAM

- Received data needs to be related to the previous data
- **Correspondence problem** - data needs to be **aligned**
- Uses feature detection and matching
- Identify visited places (loop closure)



SLAM applications

- indoors: vacuum cleaner
- ground: lawn mower
- air: unmanned air vehicles
- underwater: reef monitoring
- underground: exploration of mines
- space: unmanned terrain mapping

Feature-Based SLAM

- Given:
  - The robot's controls:

$$\boldsymbol{U}_{1:k} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k\}$$

  - Relative observations

$$\boldsymbol{Z}_{1:k} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_k\}$$

- Wanted:
  - Map of features

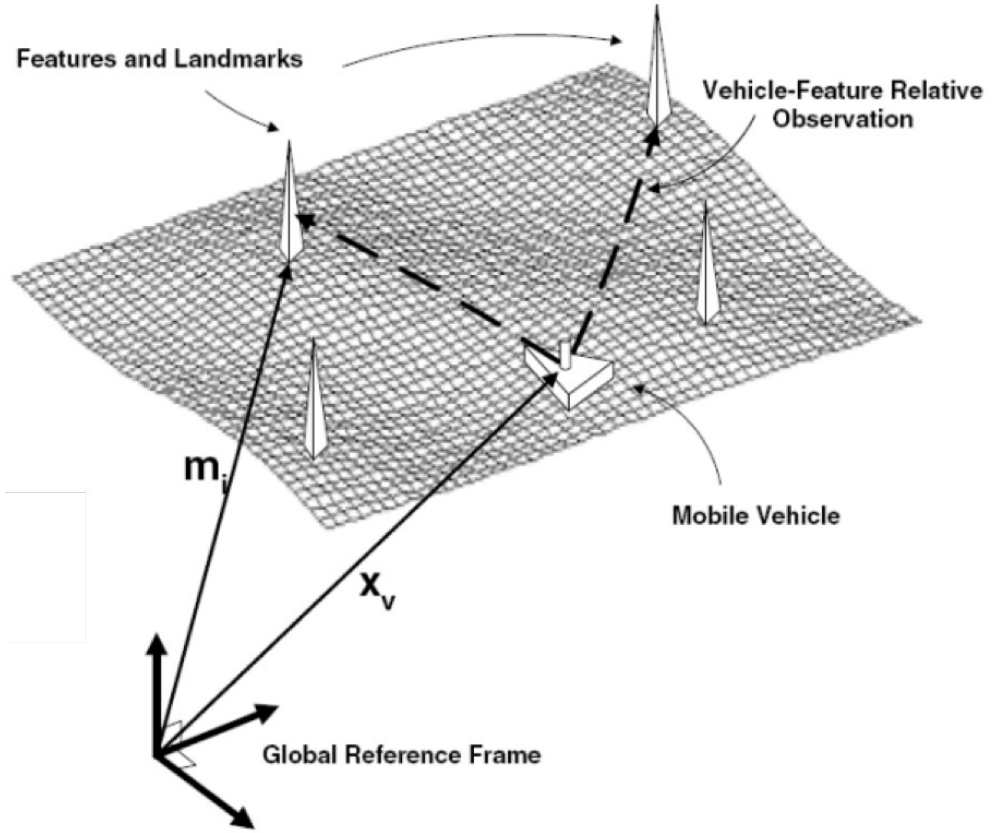
$$\boldsymbol{m} = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_n\}$$

  - Path of the robot

$$\boldsymbol{X}_{1:k} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k\}$$

Feature-Based SLAM

- **Absolute** robot pose
- **Absolute** landmark positions
- **Relative** measurements of landmarks



Types of SLAM

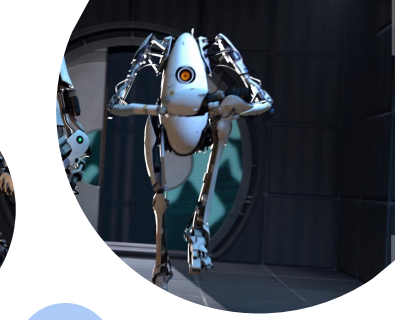
- Full SLAM - estimate the entire path and map

$$p(x_{0:t}, m \mid z_{1:t}, u_{1:t})$$

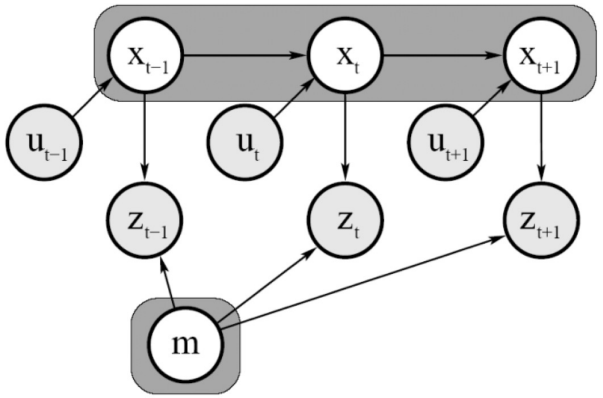
- Online SLAM - estimate most recent pose and map

$$p(x_t, m \mid z_{1:t}, u_{1:t}) = \int \int \dots \int p(x_{1:t}, m \mid z_{1:t}, u_{1:t})\, dx_1 dx_2 .. dx_{t-1}$$

Types of SLAM

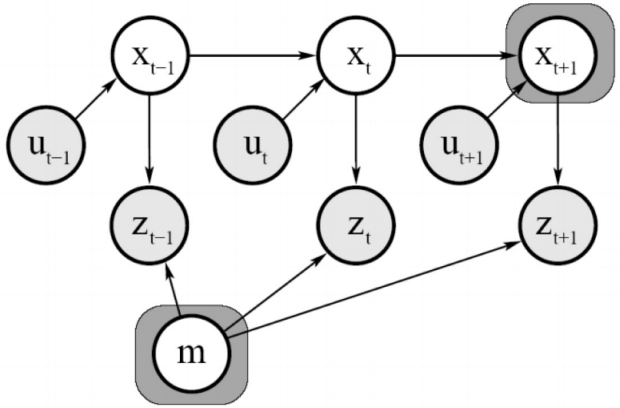


Full SLAM



$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t})$$

Online SLAM



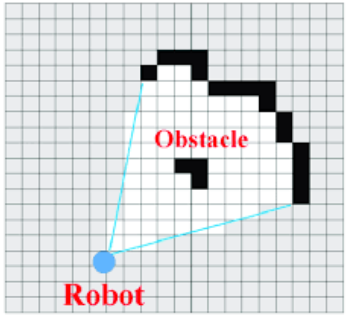
$$p(x_t, m \mid z_{1:t}, u_{1:t}) = \iint \dots \int p(x_{1:t}, m \mid z_{1:t}, u_{1:t})\, dx_1\, dx_2 \dots dx_{t-1}$$

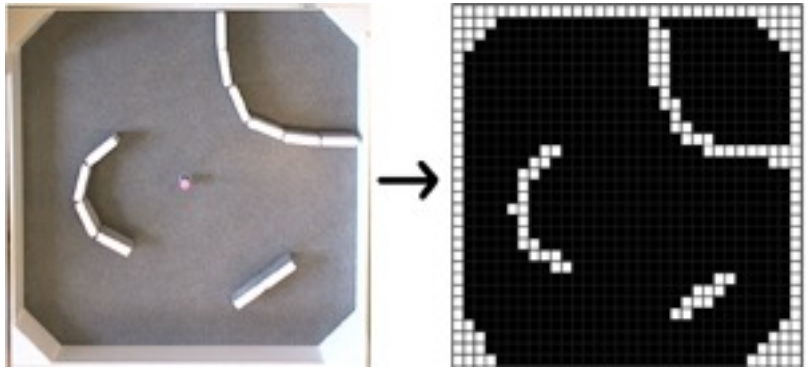
SLAM Algorithms

- **statistical techniques**: Kalman filters or Monte Carlo methods
- **mapping driven algorithms** - use highly detailed maps collected in advance
- **sensing driven algorithms** - make use of multiple types of sensors
- **kinematics driven algorithms** - rely on the action commands given to the robot in order to model the next pose
- **collaborative algorithms** - rely on data provided by multiple robots

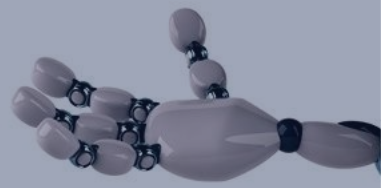
OGM

- Occupancy Grid Map (OGM)
  - generate maps from noisy and uncertain sensor measurement data, with the assumption that the robot pose is known
  - represent a map of the environment as an evenly spaced field each representing the presence of an obstacle at that location in the environment.



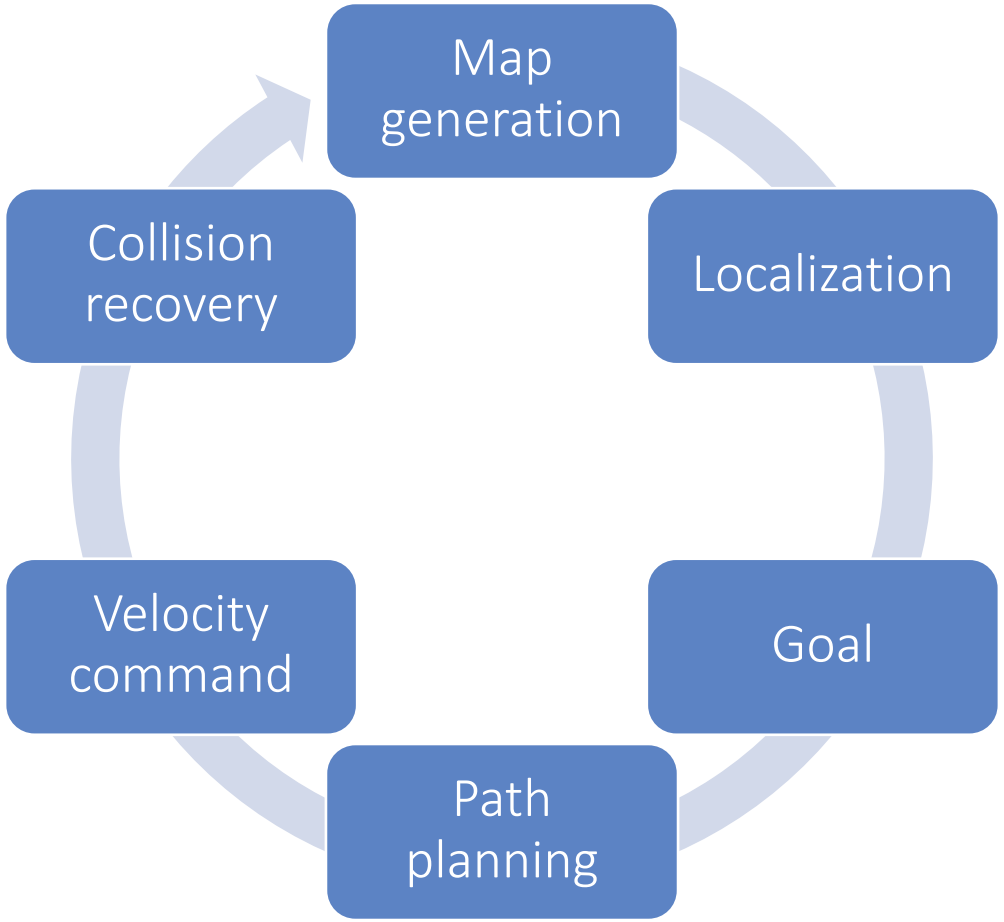


Navigation



ROS Navigation Stack

Navigation


Map generation
Localization
Goal
Path planning
Velocity command
Collision recovery

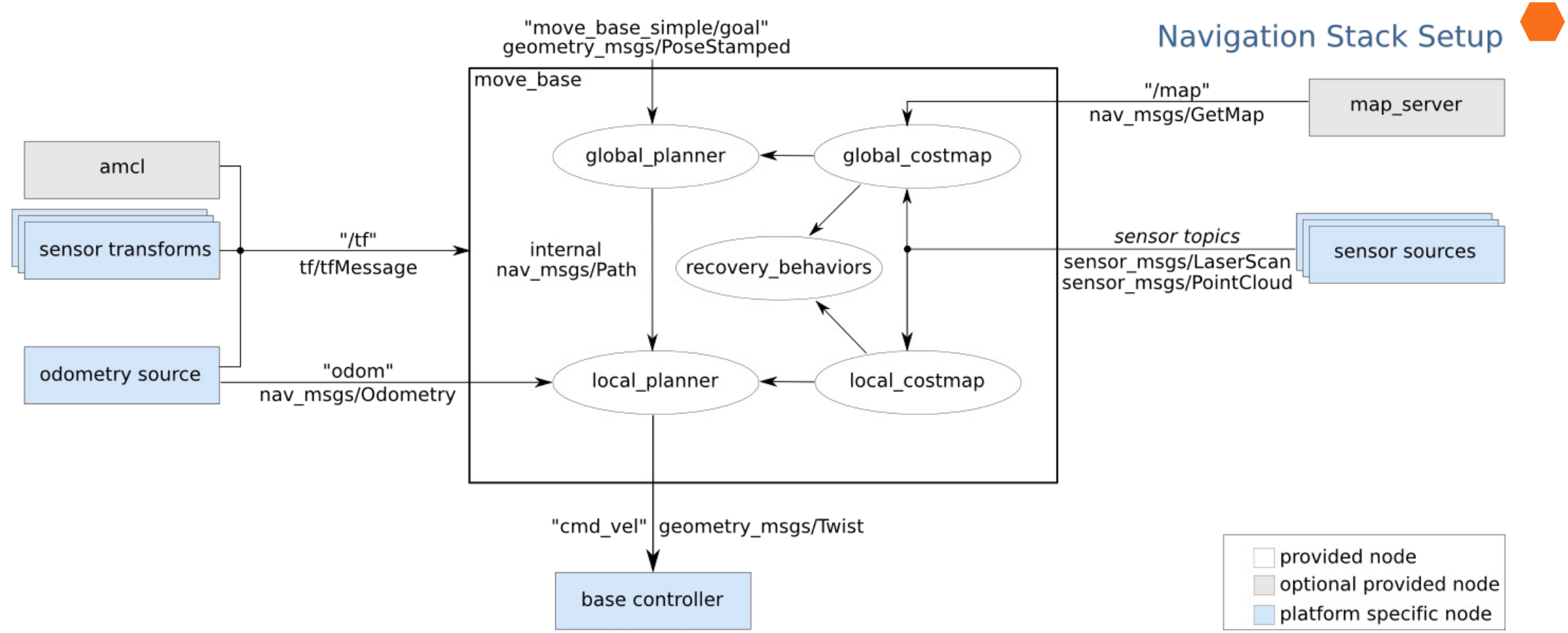
Navigation stack

- set of algorithms that uses the sensors and the odometry information from a robot and provides the interface to control the robot using a standard message
- can move the robot autonomously in the environment, without crashing, getting stuck in a location, or getting lost elsewhere
- integrate this stack with any mobile robot, but tuning a few of the configuration parameters is necessary, and this depends upon the robot's specification
- some of the interface nodes need to be developed so that we can use the stack efficiently
- only support a differential drive and holonomic-wheeled robot
- the shape of the robot must either be a square or a rectangle
- the robot must provide information about positions of all the joints and sensors and the relationship between their coordinates frames
- the robot must, at the very least, have a range sensor or similar such as a planar laser, a sonar or depth sensors

Robot Locomotion

- relationship between controllable and total degrees of freedom of a robot
- Holonomic Drive
  - the controllable degree of freedom is equal to total degrees of freedom
  - **E.g.** A robot built on castor wheels or Omni-wheels can freely move in any direction and the controllable degrees of freedom is equal to total degrees of freedom
- Non-Holonomic Drive
  - the controllable degree of freedom are less than the total degrees of freedom, then the robot is said to be Holonomic
  - **E.g.** A car has three degrees of freedom; i.e. its position in two axes and its orientation. However, there are only two controllable degrees of freedom which are acceleration (or braking) and turning angle of steering wheel. This makes it difficult for the driver to turn the car in any direction (unless the car skids or slides)
- Redundant Drive
  - the controllable degree of freedom are more than the total degrees of freedom, then the robot is said to be Holonomic
  - **E.g.** A robot arm or even a human arm has only six degrees of freedom, but seven controllable degrees of freedom.

Navigation stack architecture



The move_base node

- **move_base** package
- Main function is to move a robot from its current position to a goal position with the help of other navigation node
- links the **global-planner** and the **local-planner** for the path planning, connecting to the **rotate-recovery** package if the robot is stuck in some obstacle and connecting **global costmap** and **local costmap** for getting the map
- an implementation of **SimpleActionServer**
- goal pose **geometry_msgs/PoseStamped** on topic **move_base_simple/goal**

Requirements

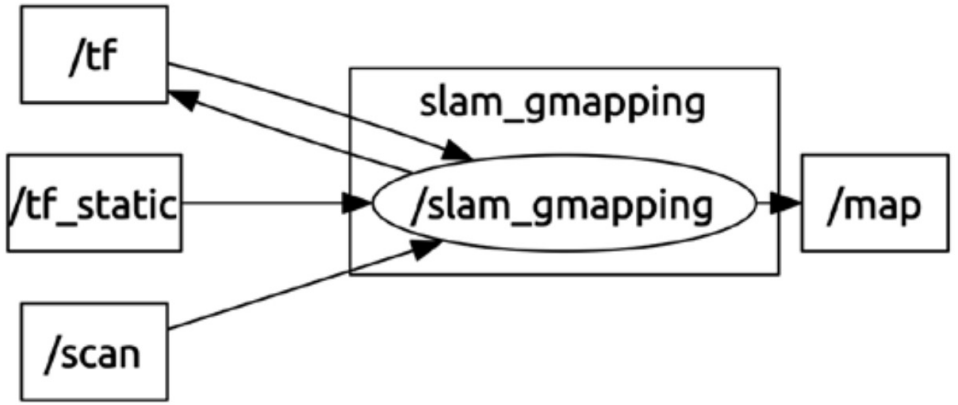
- The navigation stack requires that the robot be publishing information about the relationships between coordinate frames using **tf**
- Sensor source
  - Laser scan data or point cloud data needs to be provided to the Navigation stack for mapping the robot environment
  - Typical sensors are Laser Range Finders or Kinect 3D sensors
  - Message types **sensor_msgs/LaserScan**, **sensor_msgs/PointCloud**
- Odometry source
  - Odometry data of a robot gives the robot position with respect to its starting position
  - The main odometry sources are wheel encoders, IMU, and 2D/3D cameras (visual odometry)
  - Message type of **nav_msgs/Odometry (**holds the position and the velocity of the robot)
  - Sensor data combined with odometry data are generating the global and local cost map of the robot
- Base controller (base_controller)
  - The main function of the base controller is to convert the output of the Navigation stack, which is a twist (**geometry_msgs/Twist**) message, and convert it into corresponding motor velocities of the robot.
- Mapping (map_server)
  - The navigation stack does not require a map to operate

Map generation

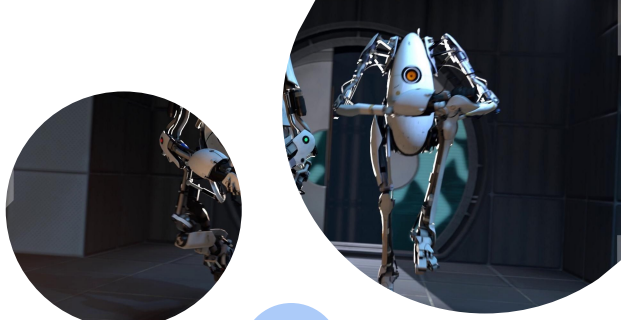
- costmap-2D package
  - Map the robot environment
  - A robot can only plan with respect to the map
  - Using grid maps to represent the environment
- map-server package
  - Save and load maps generated by costmap-2D
- gmapping package
  - Implementation of the Fast SLAM algorithm
  - Uses laser scan data and odometry to build the 2D occupancy grid map

Example: Map generation with gmapping

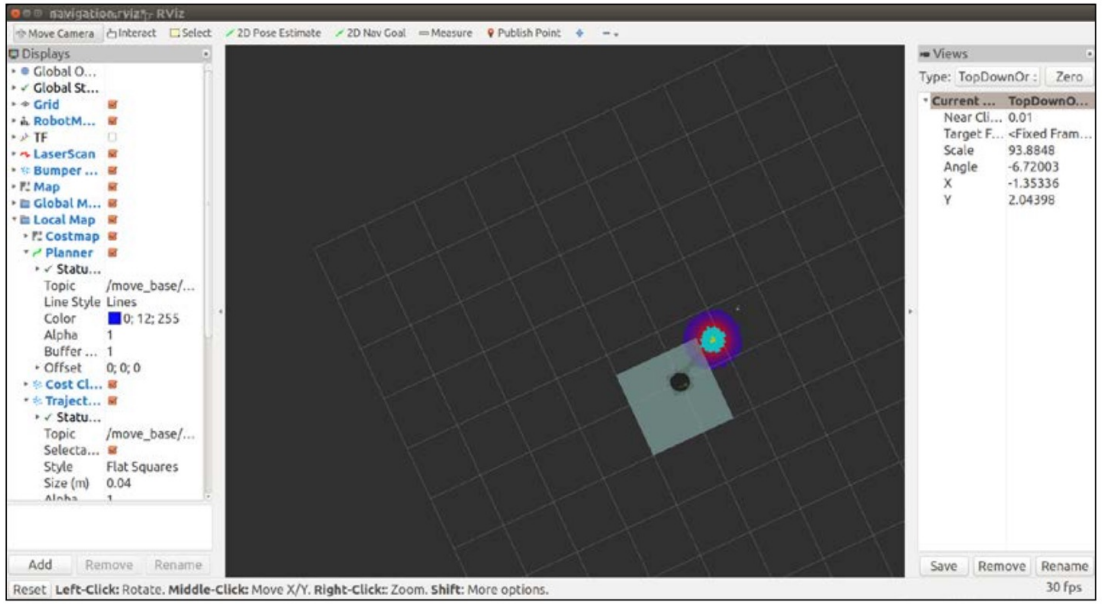
- Based on OpenSlam's Gmapping http://openslam.org/gmapping.html
- Rao-Blackwellized particle filter algorithm based on laser scanning
- Listens for
  - **sensor_msgs/LaserScan** messages from /scan
  - **tf/tfMessage** messages for odometry
- Combines data into /map
- Maps can be saved using the **map_saver** node from **map_server** package.



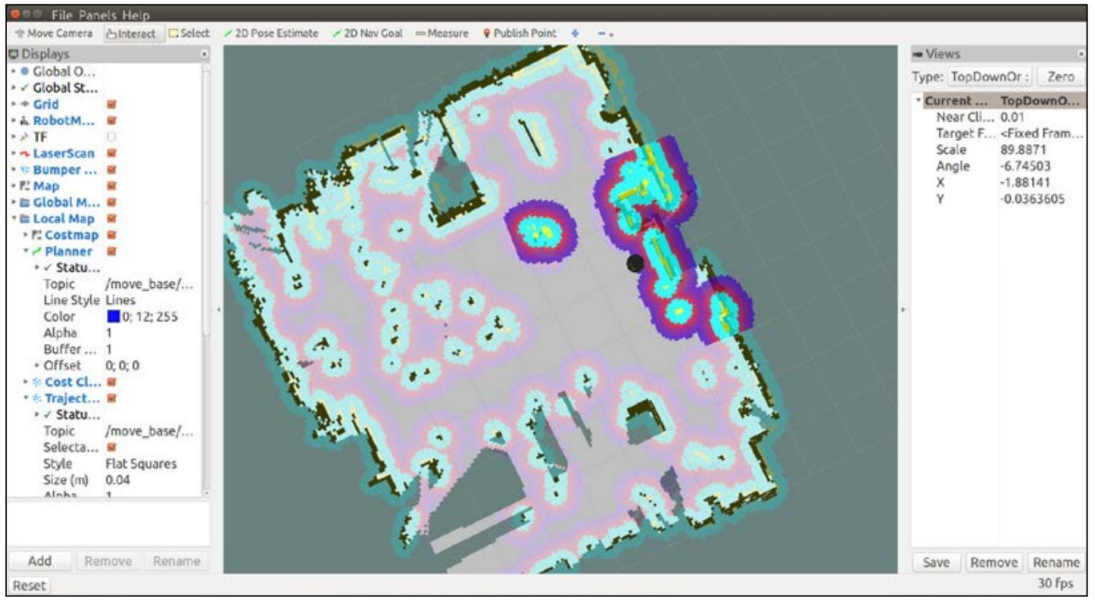
Example: Map generation with gmapping



Initial Map

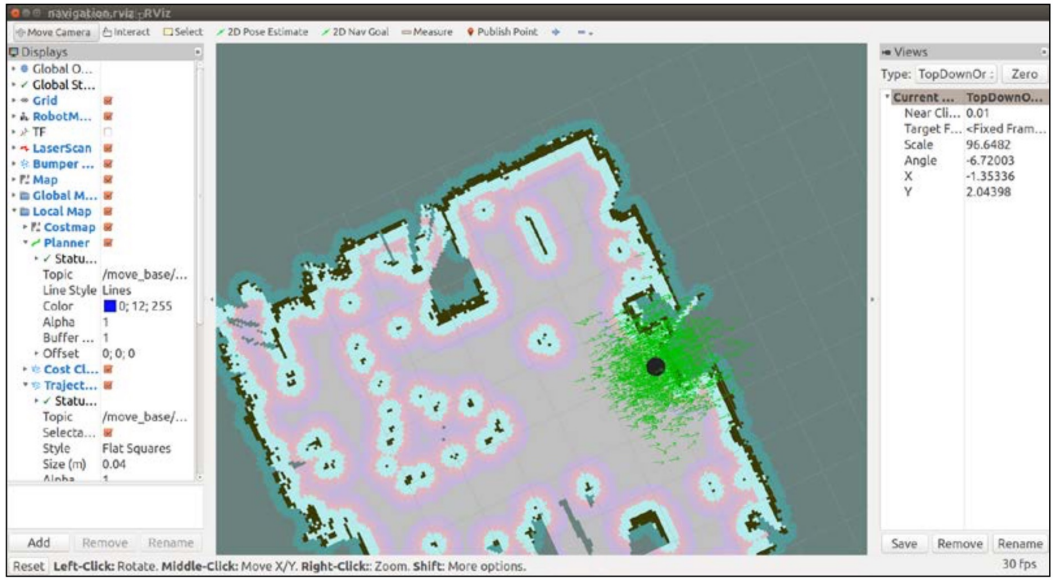


Final Map



Localization

- AMCL package
  - Adaptive Monte Carlo Localization method to localize the robot in a map
  - Uses particle filter to track the pose of the robot with the help of probability theory
  - Accepts **sensor_msgs/LaserScan** messages to generate the map



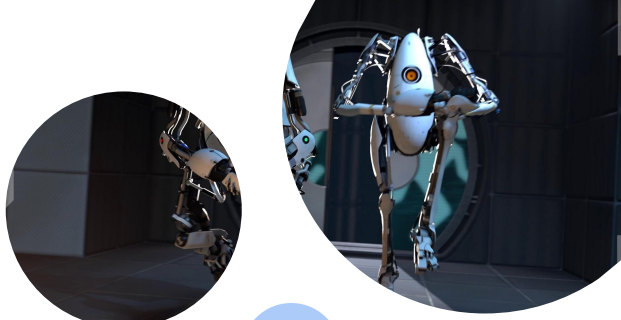
Trajectory planning

- **global-planner** package
  - Libraries and nodes for planning the optimum path from the current position to the goal position
  - Path-finding algorithms: A*, Dijkstra

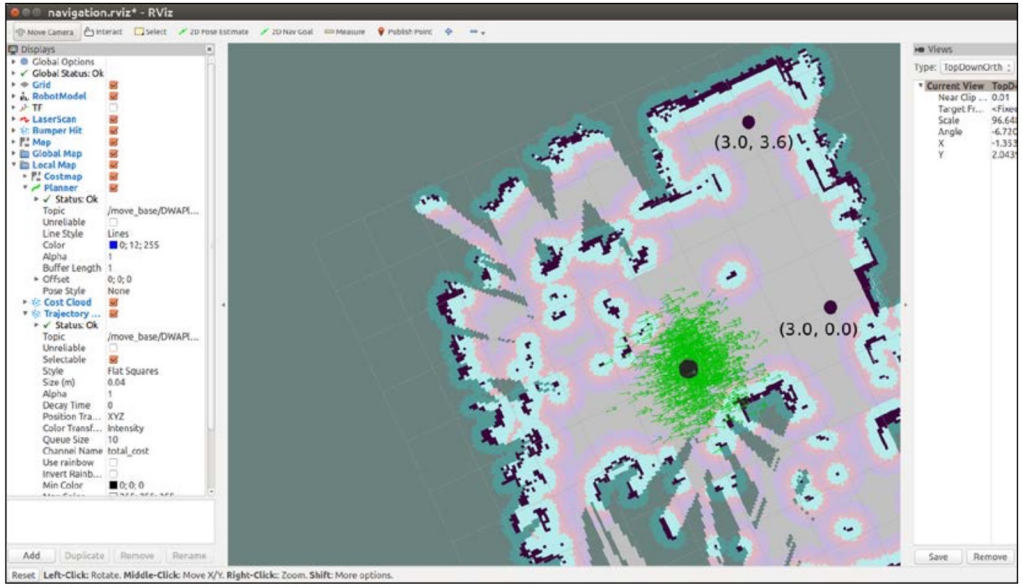
- **local-planner** package
  - Navigate the robot in a section of the global path planned by the global-planner
  - Inputs: odometry and sensor reading
  - Outputs: trajectory and velocity commands for completing the segment

- **rotate-recovery** package
  - Recovering from a local obstacle
  - Allows 360 degree rotation

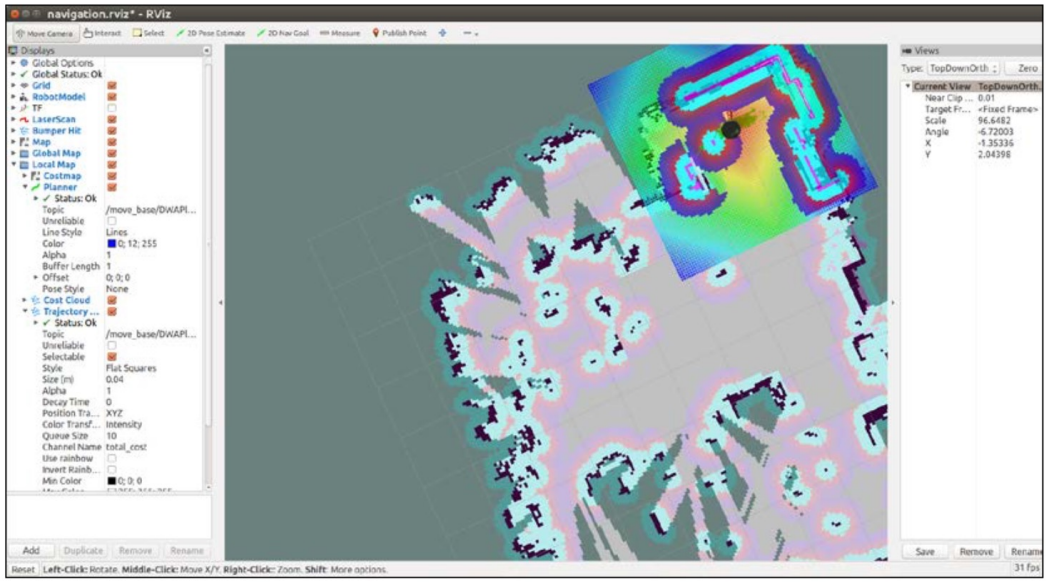
Example: Navigating



Initial Position



Final Position



Thank you

Question Time