SLAM and Autonomous Navigation

Carmine Tommaso Recchiuto

Simultaneous Localization and Mapping

✓SLAM is the process by which the robot builds a map of the environment (mapping), while, at the same time, it uses the built map to estimate its position in the environment (localization). In SLAM, both the trajectory of the platform and the location of all landmarks are estimated online without the need for any a priori knowledge of location

- Localization: inferring location given a map
- Mapping: inferring a map given a location
- SLAM: learning a map and locating the robot simultaneously

Simultaneous Localization and Mapping

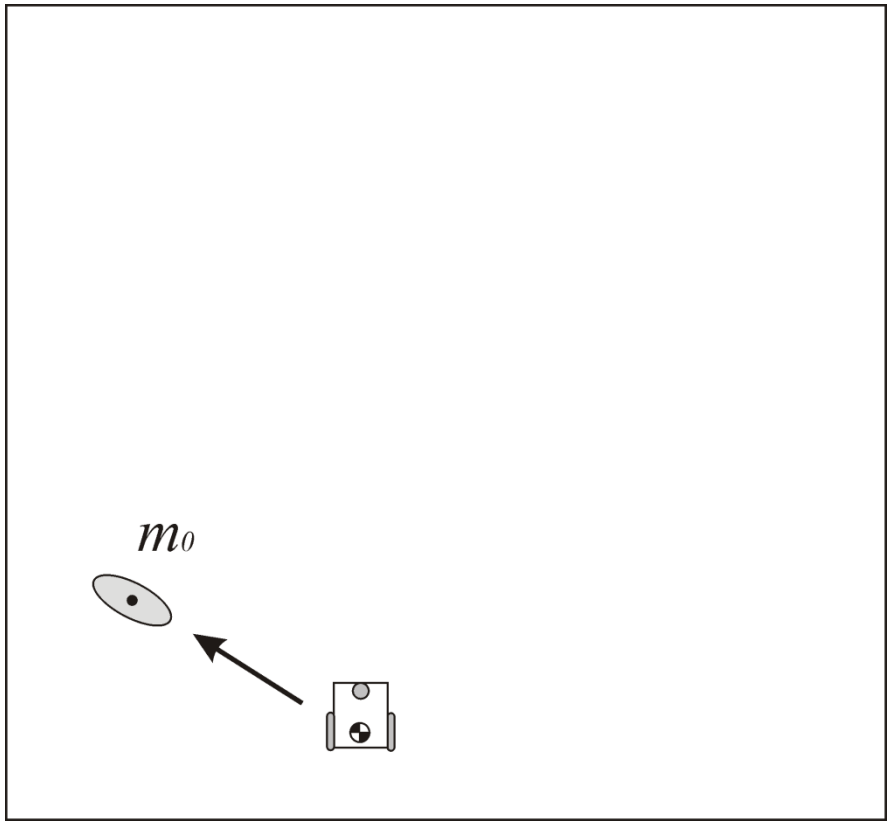
✓This directly explains why SLAM is considered a difficult (and interesting) problem to be solved in mobile robotics:

- ✓A Map is needed for localizing a robot!
- ✓A pose estimate is needed to build a map!

✓A variety of approaches have been proposed in the last 30-40 years, all of them are based on probabilistic methods

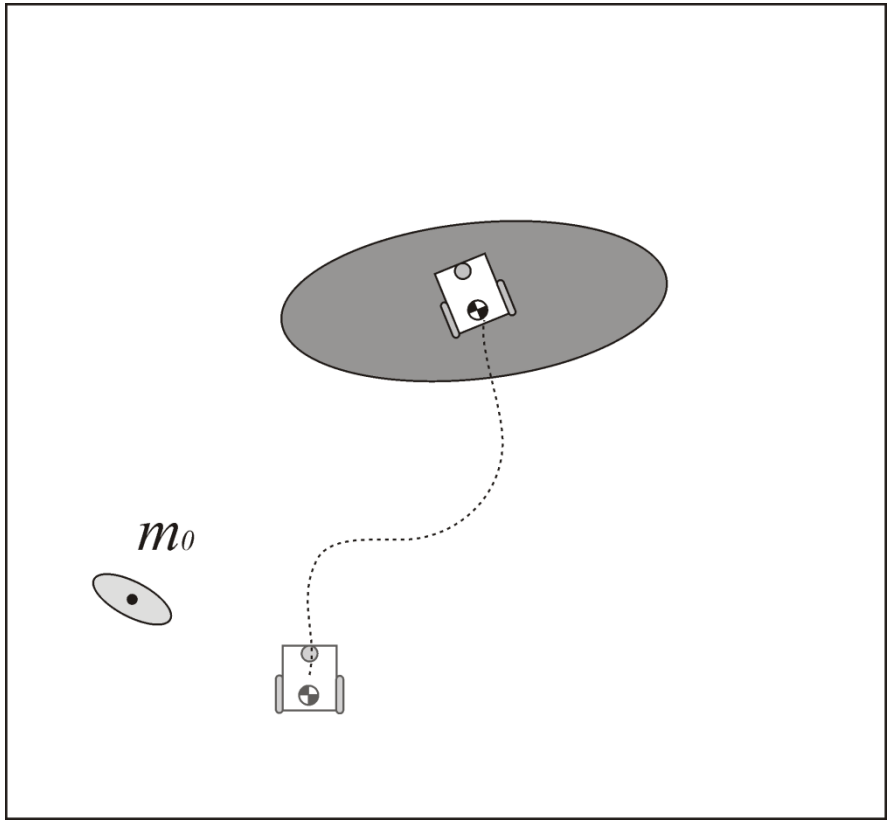
SLAM: Overview

- Let us assume that the robot uncertainty at its initial location is zero.
- From this position, the robot observes a feature which is mapped with an uncertainty related to the sensor error model



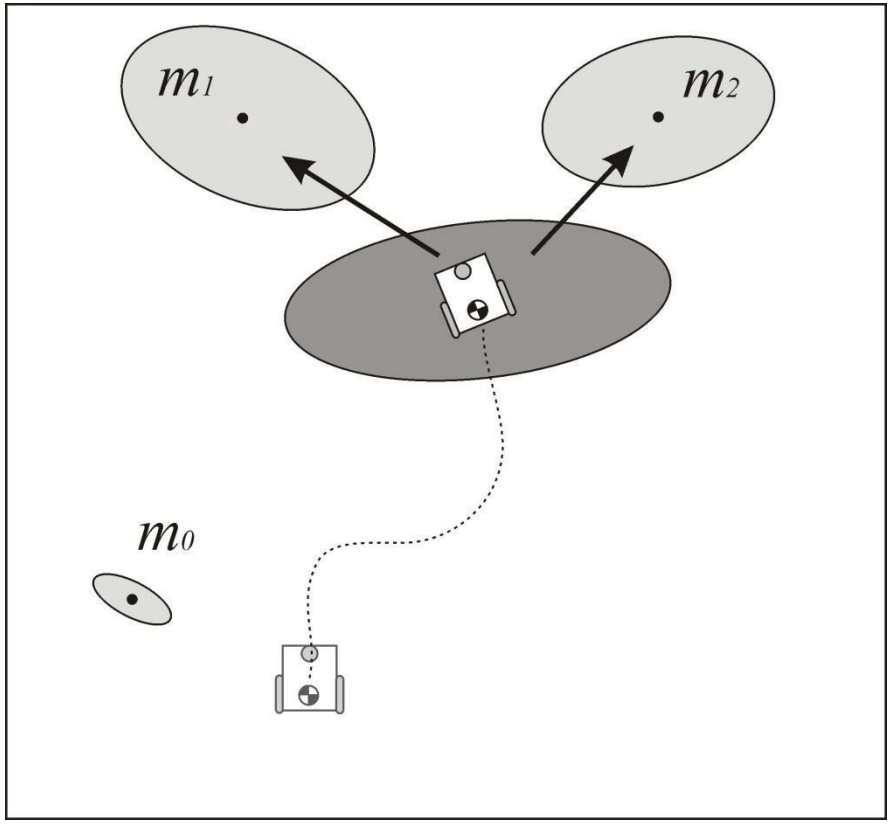
SLAM: Overview

- As the robot moves, its pose uncertainty increases under the effect of the errors introduced by the odometry



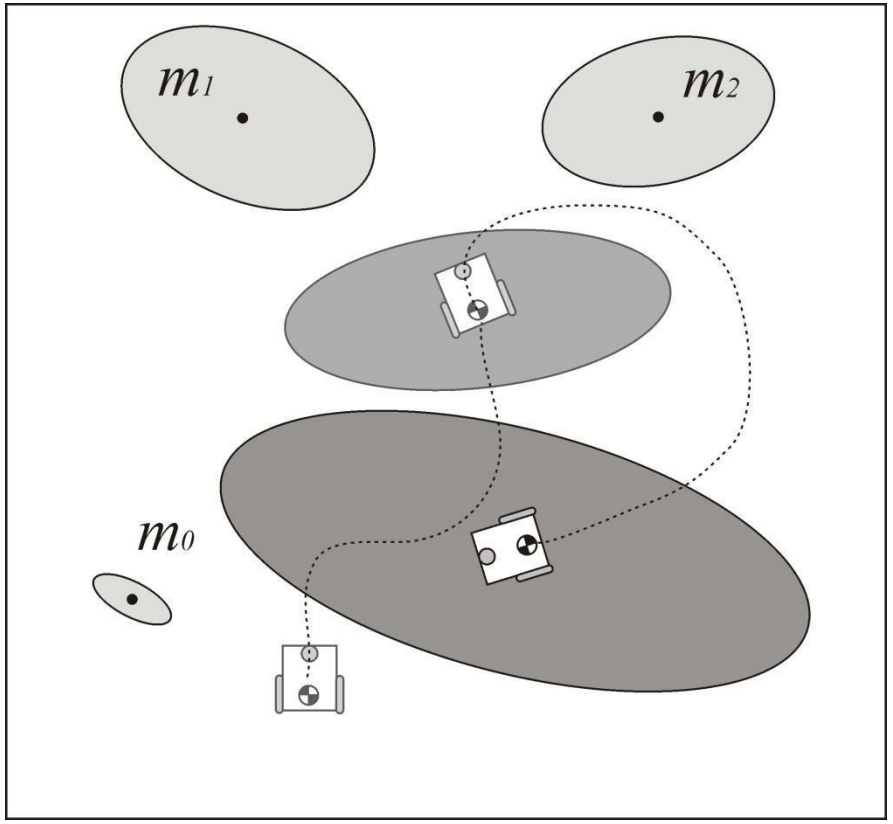
SLAM: Overview

- At this point, the robot observes two features and maps them with an uncertainty which results from the combination of the measurement error with the robot pose uncertainty
- From this, we can notice that the map becomes correlated with the robot position estimate. Similarly, if the robot updates its position based on an observation of an imprecisely known feature in the map, the resulting position estimate becomes correlated with the feature location estimate.



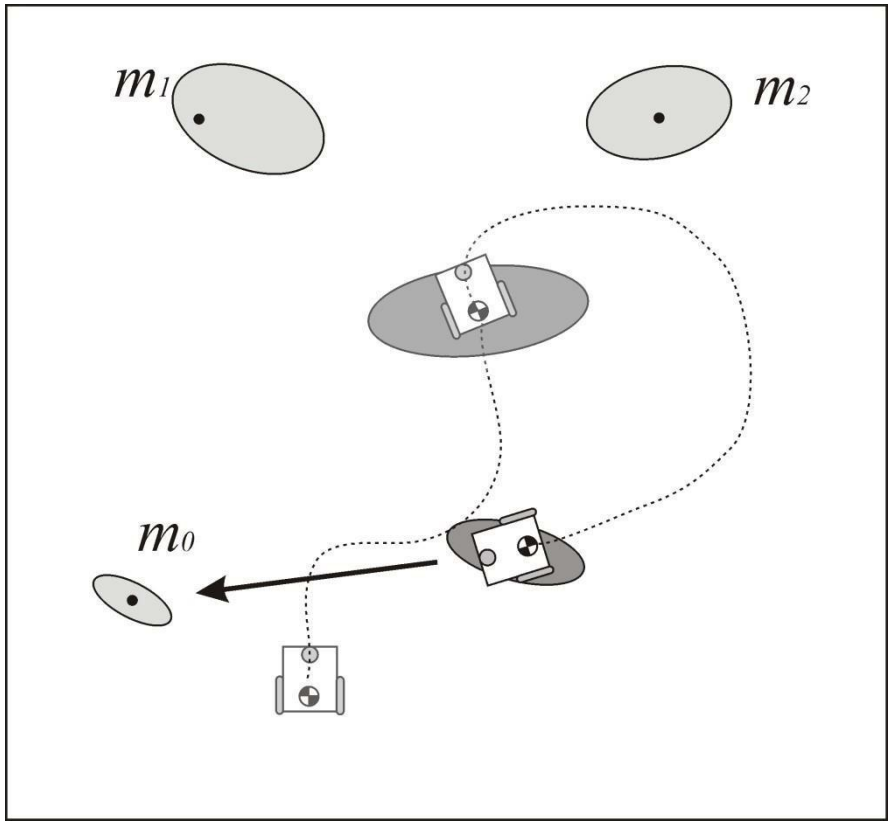
SLAM: Overview

- The robot moves again and its uncertainty increases under the effect of the errors introduced by the odometry
- However, in order to reduce its uncertainty, the robot may observe features whose location is relatively well known. These features can for instance be landmarks that the robot has already observed before



SLAM: Overview

- In this case, the observation is called *loop closure detection.*
- When a loop closure is detected, the robot pose uncertainty shrinks.
- At the same time, the map is updated and the uncertainty of other observed features and all previous robot poses also reduce



SLAM: Classification

✓SLAM approaches may be classified into the following:

- Filter-based approaches, which is the classical approach that performs prediction and update steps recursively. It maintains the information about the environment and the states of the robot as a probability density function.
- Global optimization approaches, which is based on saving some keyframes in the environment and using graph optimization techniques. This is currently a popular approach for vision- based SLAM, but they may be used also in other contexts.
- Artificial Intelligence – based SLAM, which proved in some situations to have a superior performance.

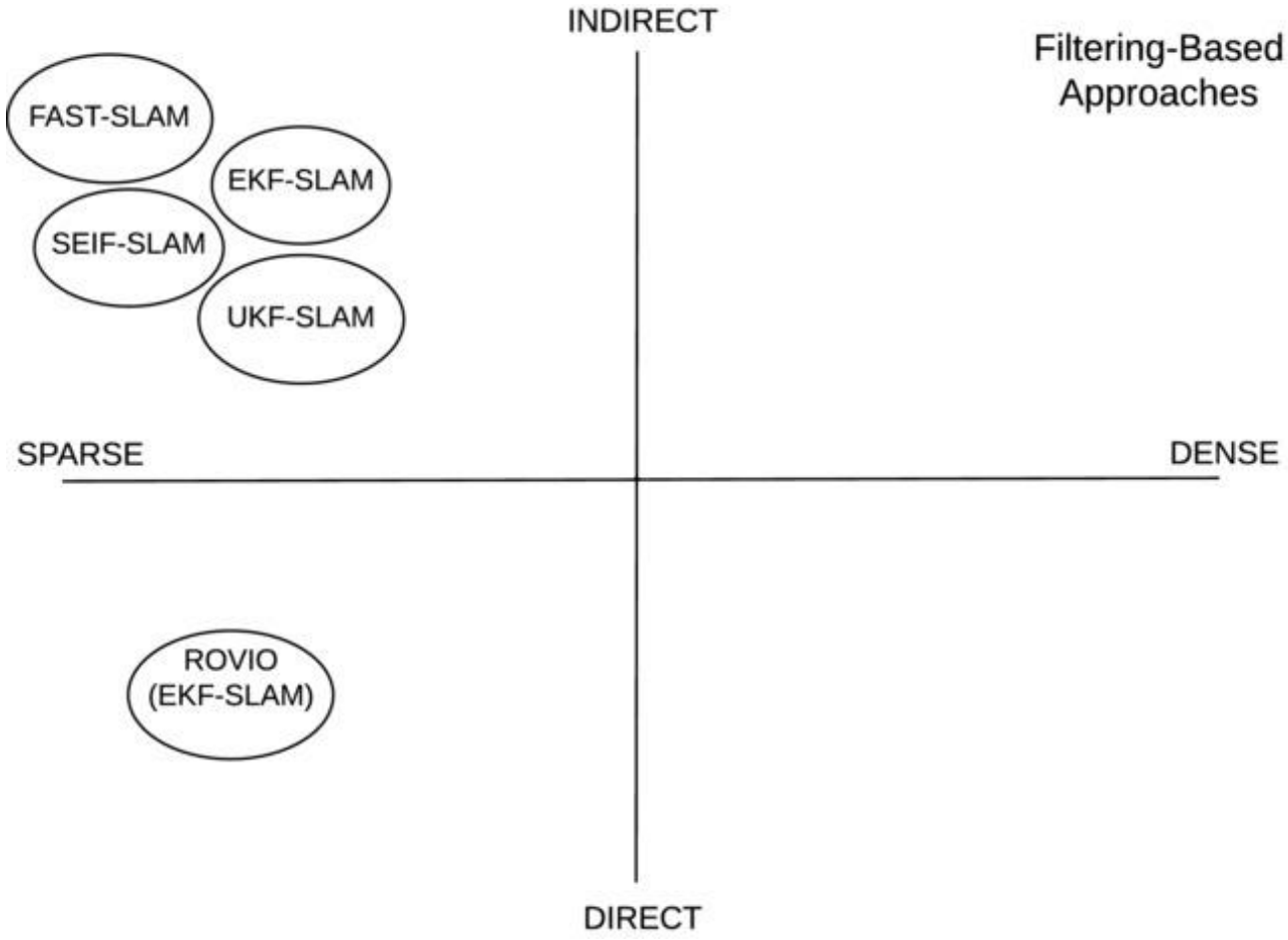
SLAM: Classification

✓Another classification is possible in relation to visual-SLAM:

- Feature-based approaches: SLAM methods in this category process the acquired images trying to find features (edges, corners,…) to compare. Images are abstracted to feature observations; thus, these typologies of methods eliminate all data that cannot be used (nonfeature points) and they are thus relatively fast. On the other side, performances are related to the environment under observation. For example, the feature extraction does not work well in human environments, as many edges would be discarded, or if features are spread all over the images, since features cannot overlap.
- Direct (or featureless) approaches: these methods, on the contrary, compare entire images to each other, creating dense or semidense 3D maps in real time. The optimization of the geometry is based on a comparison of image intensities, which enables using all information included in the image. Obviously, this provides more information about the geometry of the environment, which can be very valuable for robotics or augmented reality applications. However, these approaches are slower than feature-based ones, and do not handle outliers very well, as they will always try to process them and implement them into the final map.

SLAM: Filtering-based approaches

Filtering-SLAM approaches are usually feature-based, but there are some examples of direct approaches: ROVIO: Robust Visual Inertial Odometry - YouTube



SLAM: Optimization-based approaches

✓**GraphSLAM** is an approach in which the entire trajectory and map are estimated.

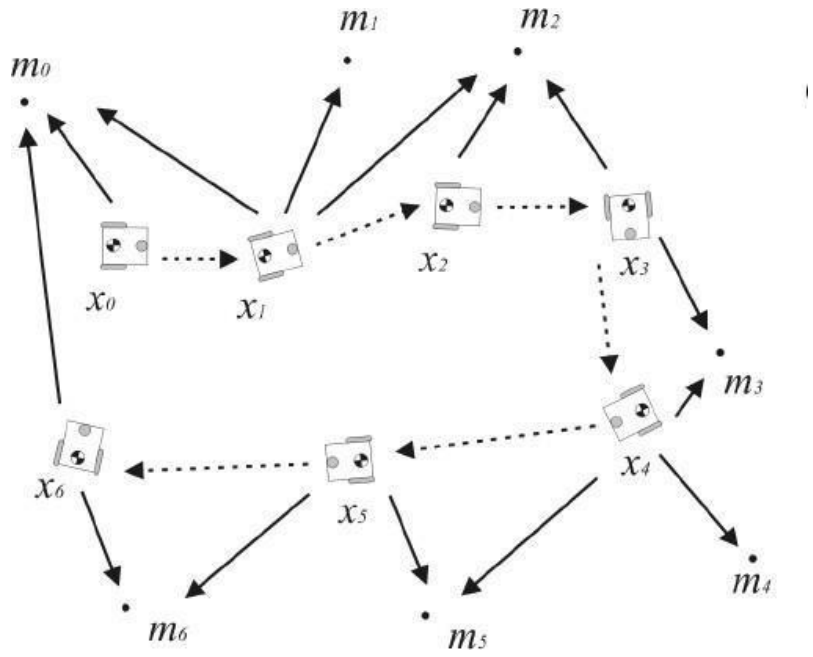
✓A graph-based SLAM approach constructs a simplified estimation problem by abstracting the raw sensor measurements. The raw measurements are indeed replaced by the edges in the graph, which can then be seen as *virtual measurements*

✓The poses of the robot are then represented as nodes in the graph. The aim of GraphSLAM is to build a graph finding a node configuration that minimizes the error introduced by the spatial constraints (the edge of the graph). This method can be also computationally expensive in relation to the solution adopted to calculate the spatial distribution of the edges and the nodes and their uncertainties.

SLAM: Graph-SLAM

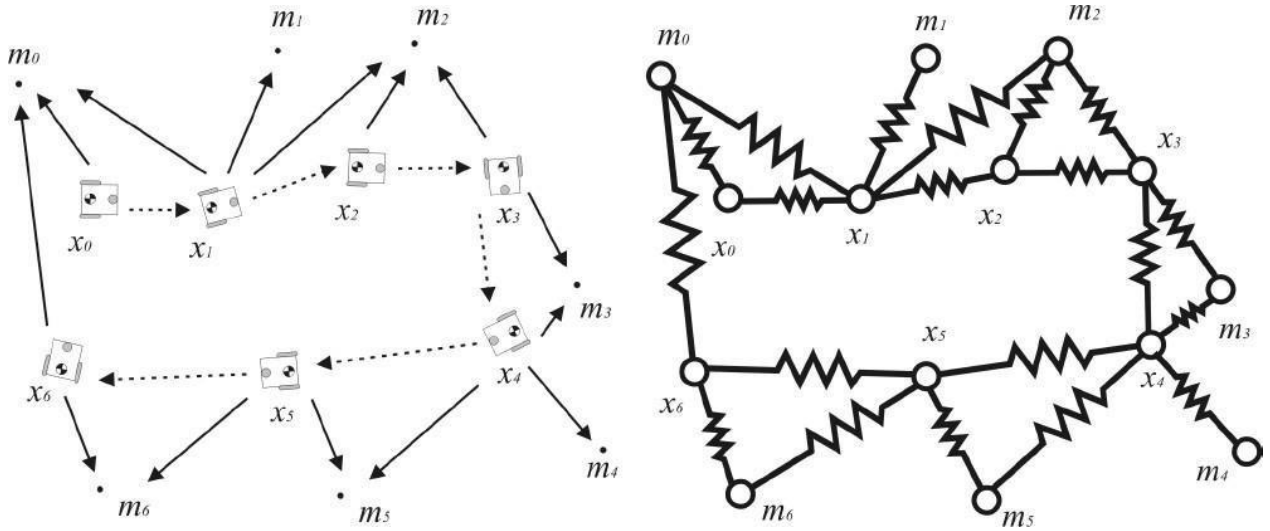
Graph-based SLAM is born from the intuition that the SLAM problem can be interpreted as a sparse graph of nodes and constraints between nodes.

- The nodes of the graph are the robot locations and the features in the map.
- The constraints are the relative position between consecutive robot poses , (given by the odometry input u) and the relative position between the robot locations and the features observed from those locations.



SLAM: Graph-SLAM

- The key property to remember about graph-based SLAM is that the constraints are not to be thought as rigid constraints but as soft constraints.
- It is by relaxing these constraints that we can compute the solution to the full SLAM problem, that is, the best estimate of the robot path and the environment map. By saying this in other words, graph-based SLAM represents robot locations and features as the nodes of an elastic net. The SLAM solution can then be found by computing the state of minimal energy of this net

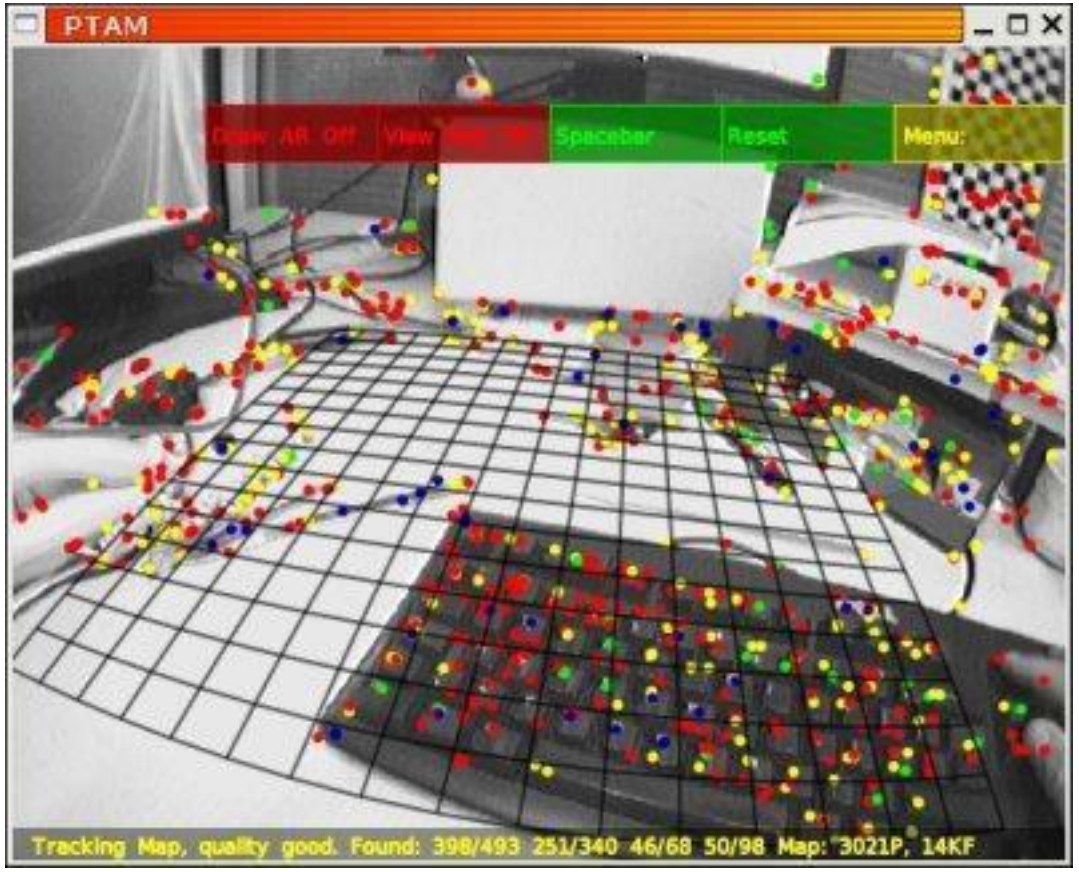


SLAM: Optimization-based approaches

✓**Keyframe-based approaches**

- In these methods, the map is estimated using selected camera frames, allowing to perform accurate bundle adjustment optimizations.
- Examples:
  - **PTAM** (2007), Parallel Tracking and Mapping.
    - ❑The tracking and the mapping processes are split into two different parallel threads: the *tracking* thread processes the video input frames and tracks the pose of the camera relative to the map, assuming that the map is fixed and certain, while the *mapping* thread produces a 3D map using a small subset of these frames, which are called keyframes.
    - ❑This is done with the following steps: first, a simple motion model is applied to predict the new pose of the camera; then the stored points are projected into the camera frame, and corresponding features are searched.

SLAM: PTAM



[Parallel Tracking and Mapping for Small AR Workspaces (PTAM) - extra - YouTube]

SLAM: Optimization-based approaches

✓**Keyframe-based approaches**

- Examples:
  - A “recent” further improvement of the Klein and Murray approach is the ***ORB-SLAM algorithm (2015)***
  - It is based on ORB features (i.e. keypoints detected by comparing brightness in image), and a loop closure approach, which searches for large loops when the new keyframe is available.
  - It may work with monocular, stereo and RGB-D cameras, and does not require an odometry estimation (only visual odometry is performed).

ORB-SLAM2: an Open-Source SLAM for Monocular, Stereo and RGB-D Cameras - YouTube

SLAM: Other keyframe-based approach

✓**Direct Methods**

These approaches work with the raw pixel information and exploit all the information in the image, even from areas where gradients are small; thus, they can outperform feature-based methods in scenes with poor texture. These methods provide more information about the environment, as well as giving a more meaningful representation to the human eye, performing a dense (all pixels in the image) or a semi-dense (only high gradient areas) reconstruction

E.g. autonomous robot in a disaster scenario:

- a valid SLAM approach should guarantee not only the autonomous navigation of the UAV, but also a preliminary damage assessment of the area.

Drawback: usually direct methods cannot handle outliers very well, as they will always try to process them and implement them into the final map, and they require high computing power (GPUs) for real-time performance, being slower than feature-based variants.

SLAM: Other keyframe-based approach

✓**Direct Methods**

✓Dense approaches usually require the usage of RGB-D sensors: an example is *KinectFusion*.

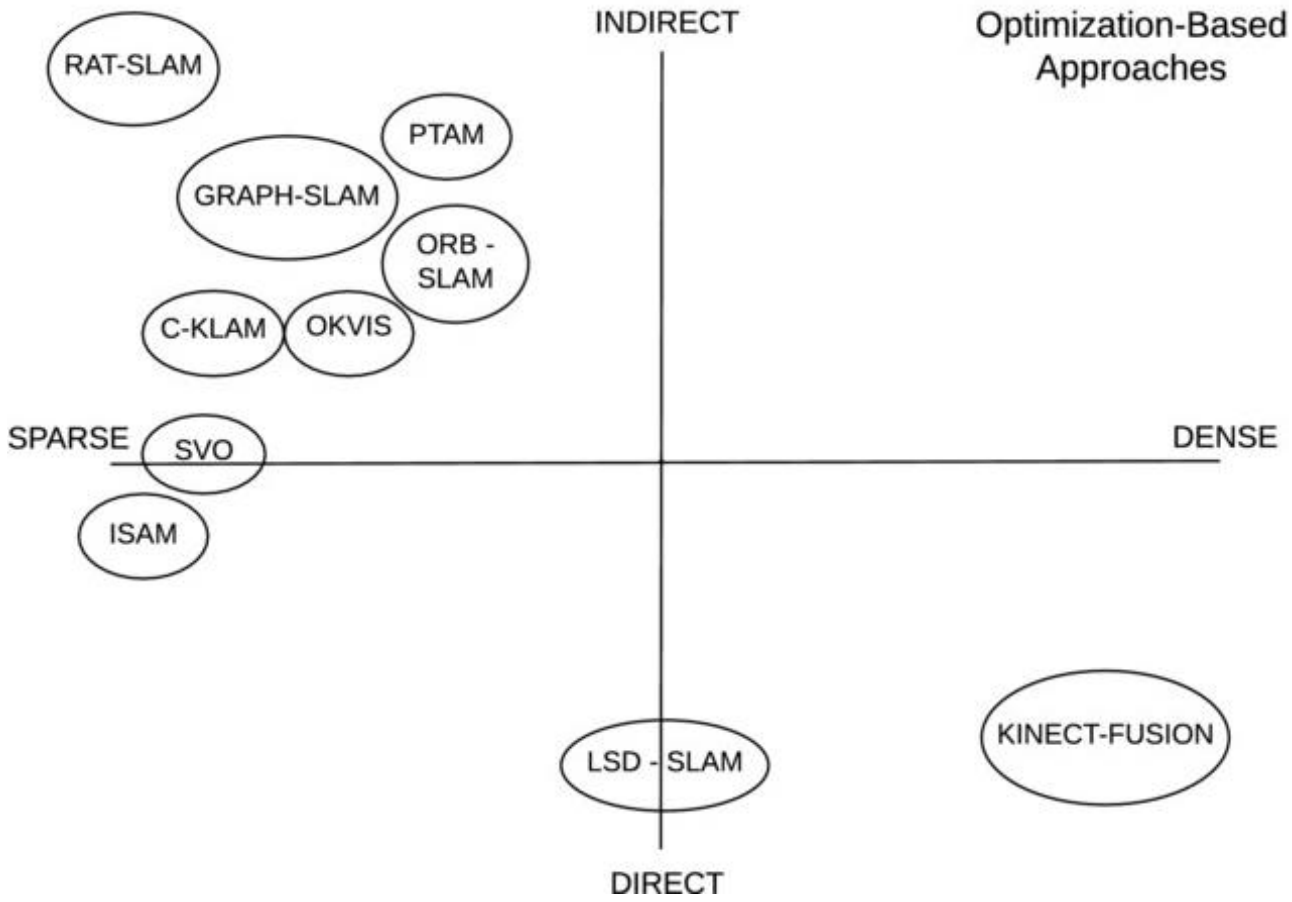
✓Between *semi-dense* approaches, ***LSD-SLAM*** is one of the most impressive methods. The approach represents the environment as point clouds, maintaining and tracking a global map of the environment by aligning different keyframes implementing a direct method.

✓LSD-SLAM: Large-Scale Direct Monocular SLAM (ECCV '14) - YouTube

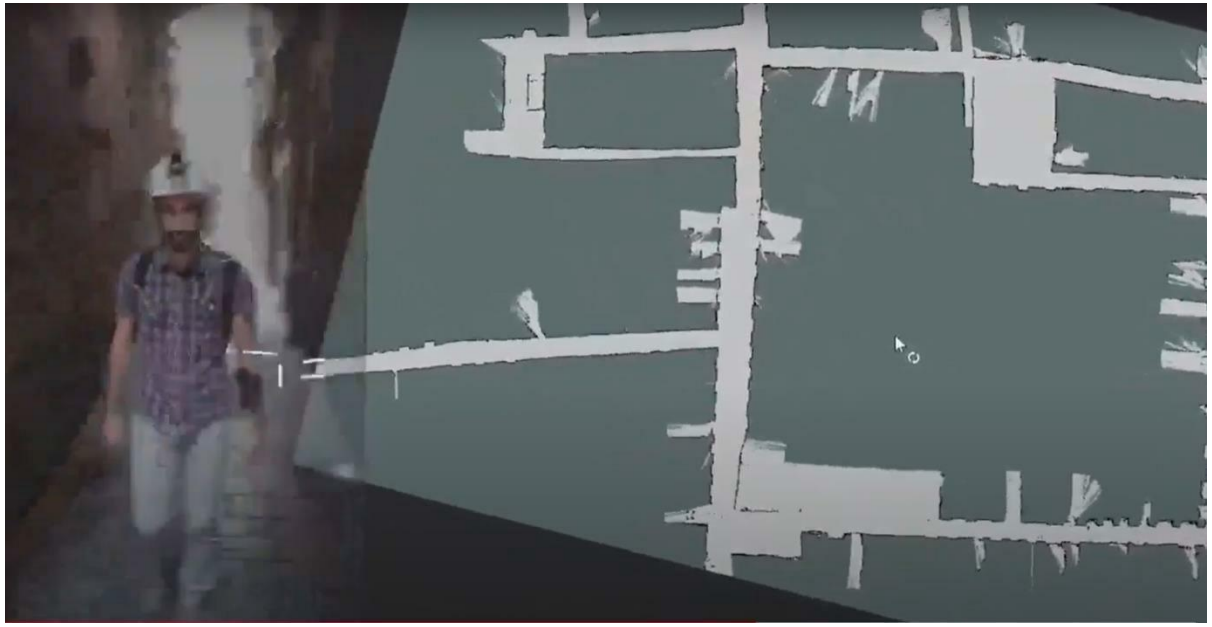
✓We have also *semi-direct* approaches, that use a combination of indirect methods to establish feature correspondences and direct methods to refine the camera pose estimates. Among these, I would mention the fast ***semidirect monocular visual odometry (SVO)*** approach, which uses FAST features in the environment (indirect) and pixel intensities (direct)

✓SVO 2.0: Semi-Direct Visual Odometry for Monocular and Multi-Camera Systems - YouTube

SLAM: Optimization-based approaches



SLAM: not only with robots



[Autonomous mapping of urban areas using wearable sensors - DIONISO - YouTube]

ROS IMPLEMENTATION

LIDAR

LIDAR (an acronym for “Light Detection and Ranging” or “Laser Imaging, Detection, and Ranging”) is a sensing technology that uses laser light to measure distances. LIDAR sensor typically emits pulses of laser light in a scanning pattern (2D or 3D) and measures how long it takes for the light to return after hitting nearby objects. From this, the system computes distances to obstacles or surfaces in the environment.

By continuously scanning the surroundings, the LIDAR provides a 2D or 3D map of distances to any objects around the robot. Lidars are simple and important sensors of almost every mobile robot applications, it's widely used in Simultaneous Localization and Mapping (SLAM) algorithms which use LIDAR scans to build a map of the environment in real time while also estimating the robot’s pose (position and orientation) within that map.

LIDAR

As usual we need to modify: - URDF

```
<joint type="fixed" name="scan_joint">
  <origin xyz="0 0 0.15" rpy="0 0 0"/>
  <child link="scan_link"/>
  <parent link="base_link"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
</joint>


<link name='scan_link'>
  <inertial>
    <mass value="1e-5"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia
        ixx="1e-6" ixy="0" ixz="0"
        iyy="1e-6" iyz="0"
        izz="1e-6"
    />
  </inertial>
  <collision name='collision'>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size=".1 .1 .1"/>
    </geometry>
  </collision>


  <visual name='scan_link_visual'>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
       <mesh                    filename
"package://bme_gazebo_sensors/meshes/lidar.dae"/>
    </geometry>
  </visual>
</link>
```

Gazebo file

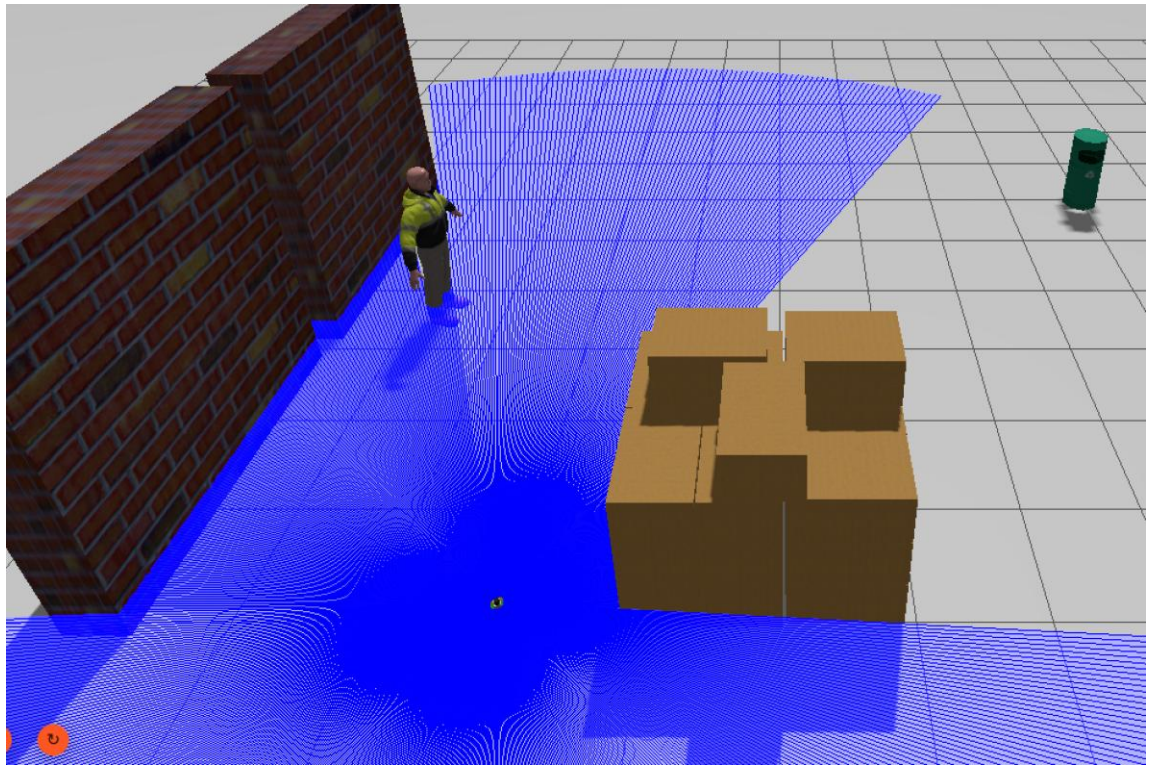
```
<gazebo reference="scan_link">
  <sensor name="gpu_lidar" type="gpu_lidar">
    <update_rate>10</update_rate>
    <topic>scan</topic>
    <gz_frame_id>scan_link</gz_frame_id>
    <lidar>
      <scan>
        <horizontal>
          <samples>720</samples>
          <!--(max_angle-min_angle)/samples * resolution -
->
          <resolution>1</resolution>
          <min_angle>-3.14156</min_angle>
          <max_angle>3.14156</max_angle>
        </horizontal>
        <!-- Dirty hack for fake lidar detections with
ogre 1 rendering in VM -->
        <!-- <vertical>
            <samples>3</samples>
            <min_angle>-0.001</min_angle>
            <max_angle>0.001</max_angle>
        </vertical> -->
      </scan>
      <range>
        <min>0.05</min>
        <max>10.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
          <type>gaussian</type>
          <mean>0.0</mean>
          <stddev>0.01</stddev>
      </noise>
      <frame_id>scan_link</frame_id>
    </lidar>
    <always_on>1</always_on>
    <visualize>true</visualize>
  </sensor>
</gazebo>
```

LIDAR

…and add the topic in the parameter bridge:

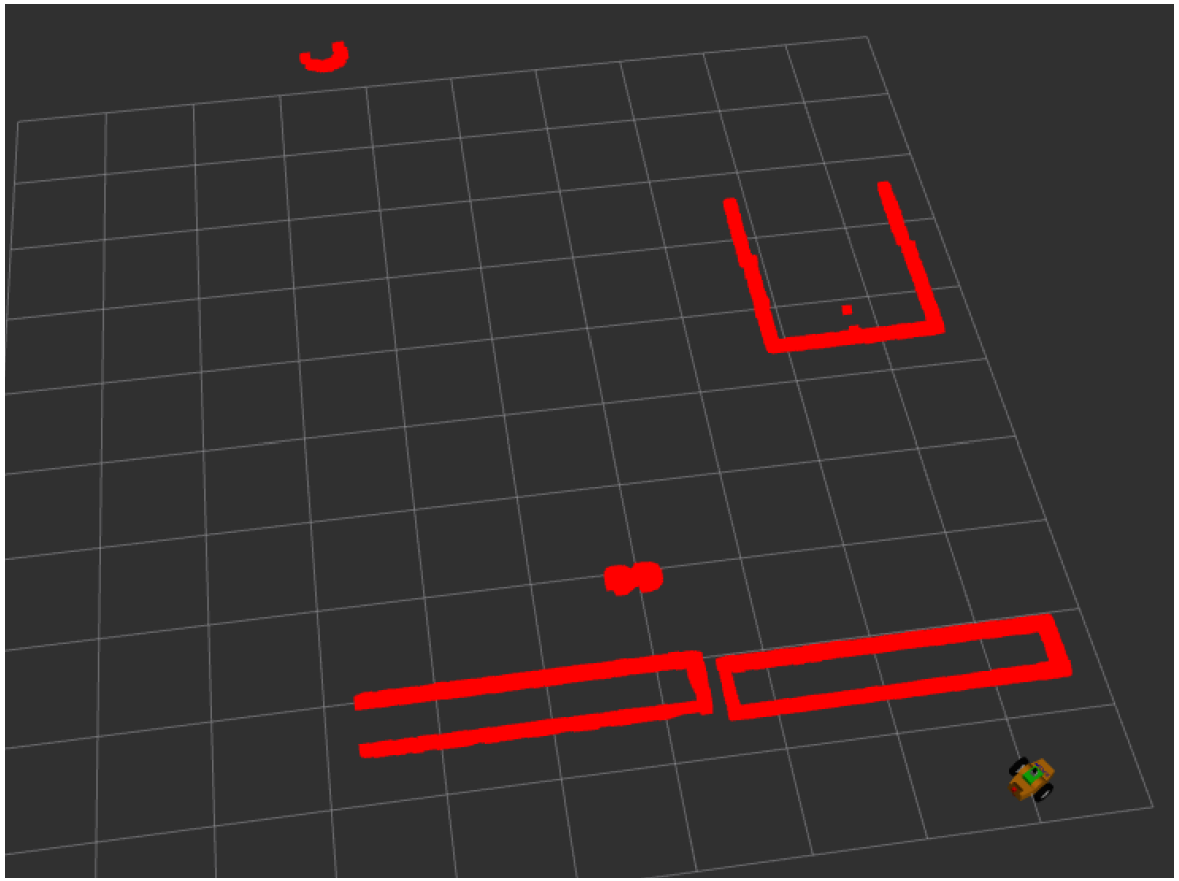
```
"/scan@sensor_msgs/msg/LaserScan@gz.msgs.LaserScan",
```

We can also verify the rendering of lidars in Gazebo with the Visualize Lidar tool



LIDAR

If we increase decay time of the visualization of lidar scans and we drive around the robot we can do a very simple "mapping" of the environment. Although in the course we will see that mapping algorithms are more complicated, usually this a good quick and dirty test on real robots if odometry, lidar scan and the other components are working well together



ROS Implementation – Mapping Algorithm

But, if you want a proper implementation of a mapping algorithm, that could be also reliable with respect to errors in odometry, increasing the decay time is not enough.

In ROS we could use slam-toolbox.

SLAM-Toolbox is a 2D SLAM system built on top of the classic Karto SLAM algorithms (ROS1), extending them with modern improvements, ROS2 support, and practical tools for real-world deployment.

It is is part of the standard Nav2 (Navigation2) ecosystem

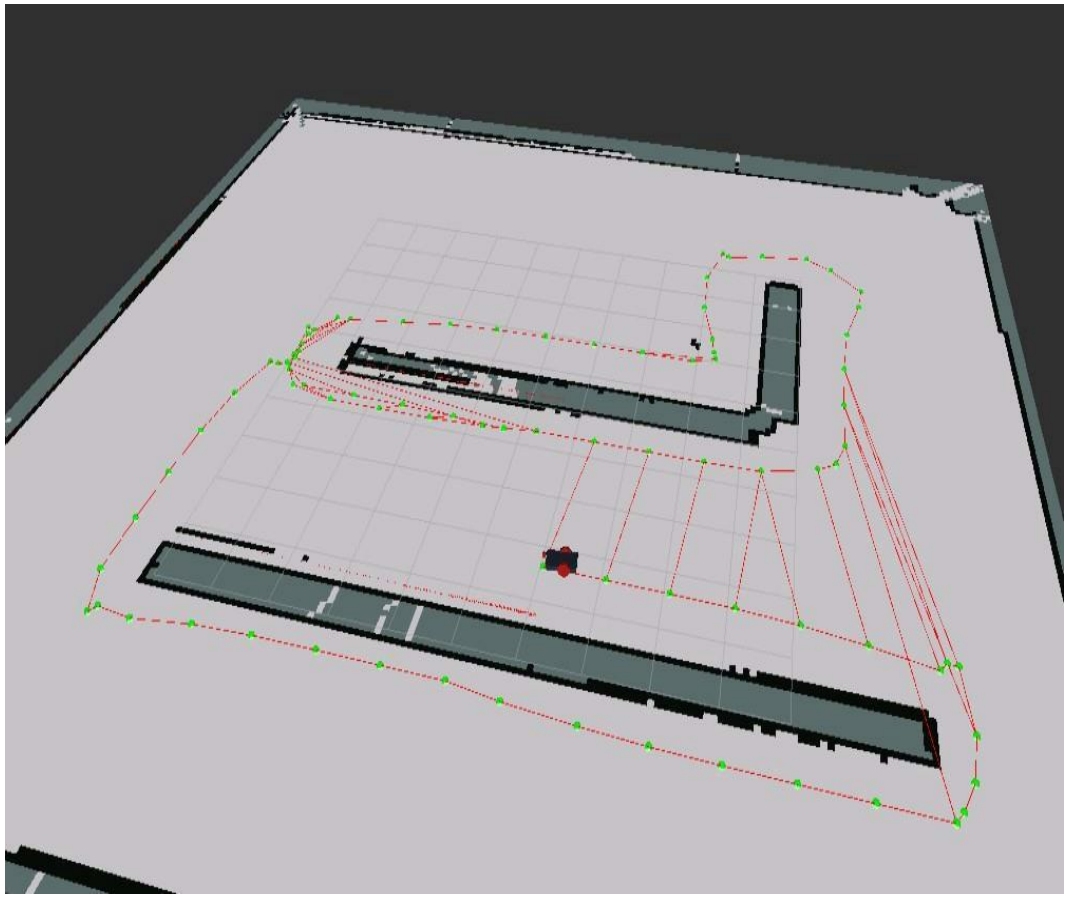
sudo apt-get install ros-jazzy-slam-toolbox

KartoSLAM – ROS Implementation

Optimization-based approach (graphSLAM)

KartoSLAM is a graph-based SLAM approach developed by SRI International's Karto Robotics, which has been extended for ROS.

Being a graph-based approach, the algorithm manages the different poses and measurements of the robot as a graph (see figure below).



ROS Implementation – Mapping Algorithm

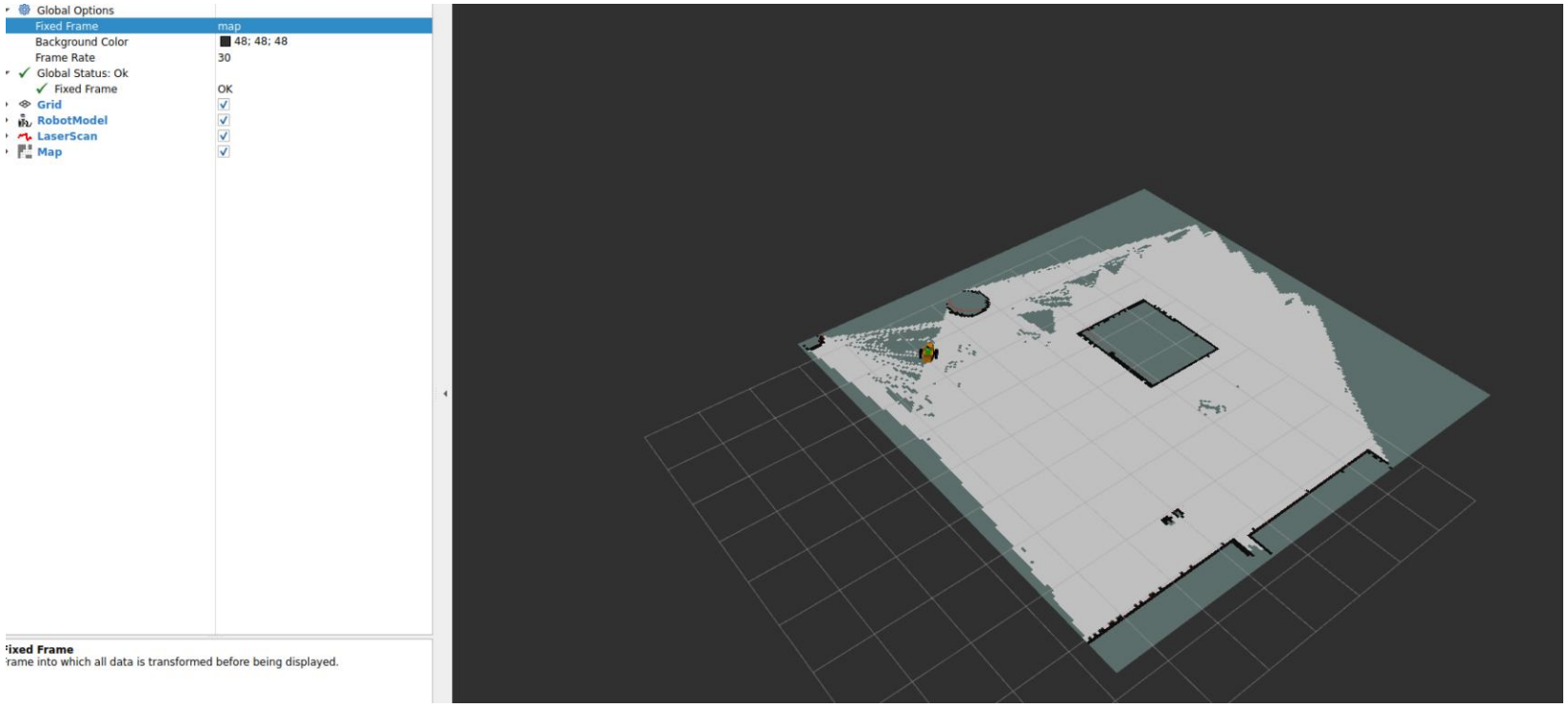
In order to use slam-toolbox, we need to 1) prepare a config file with takes into account the characteristics of our robot, and 2) use the standard slam-toolbox launch file, by passing the config file we want to use.

To this purpose, you could create a dedicated package (e.g., ros2_navigation), with the launch file mapping.launch.py (https://github.com/CarmineD8/ros2_navigation) and the slam_toolbox_mapping.yaml file: https://github.com/CarmineD8/ros2_navigation/blob/main/config/slam_toolbox_mapping.yaml

```
# ROS Parameters
  odom_frame: odom
  map_frame: map
  base_frame: base_footprint
  scan_topic: /scan
  use_map_saver: true
  mode: mapping #localization
```

ROS Implementation – Mapping Algorithm

You will now notice the presence of the map topic. The map frame can be also be used as a fixed reference frame in Rviz. You can also visualize the transform between the frames map and odom. The difference between the odom and map frames show the accumulated drift of our odometry over time.



ROS Implementation – Mapping Algorithm

Once you have created a map, you can save it:

```
ros2 run nav2_map_server map_saver_cli -f my_map
```

[you need to install nav2_map_server for this]

Once you run the command, you will have a map, that can be used for further uses…

… for example for localizing your robot given an existing map!

AMCL (Adaptive Monte Carlo Localization) is a particle filter–based 2D localization algorithm. The robot's possible poses (position + orientation in 2D) are represented by a set of particles. It adaptively samples the robot's possible poses according to sensor readings and motion updates, converging on an accurate estimate of where the robot is within a known map

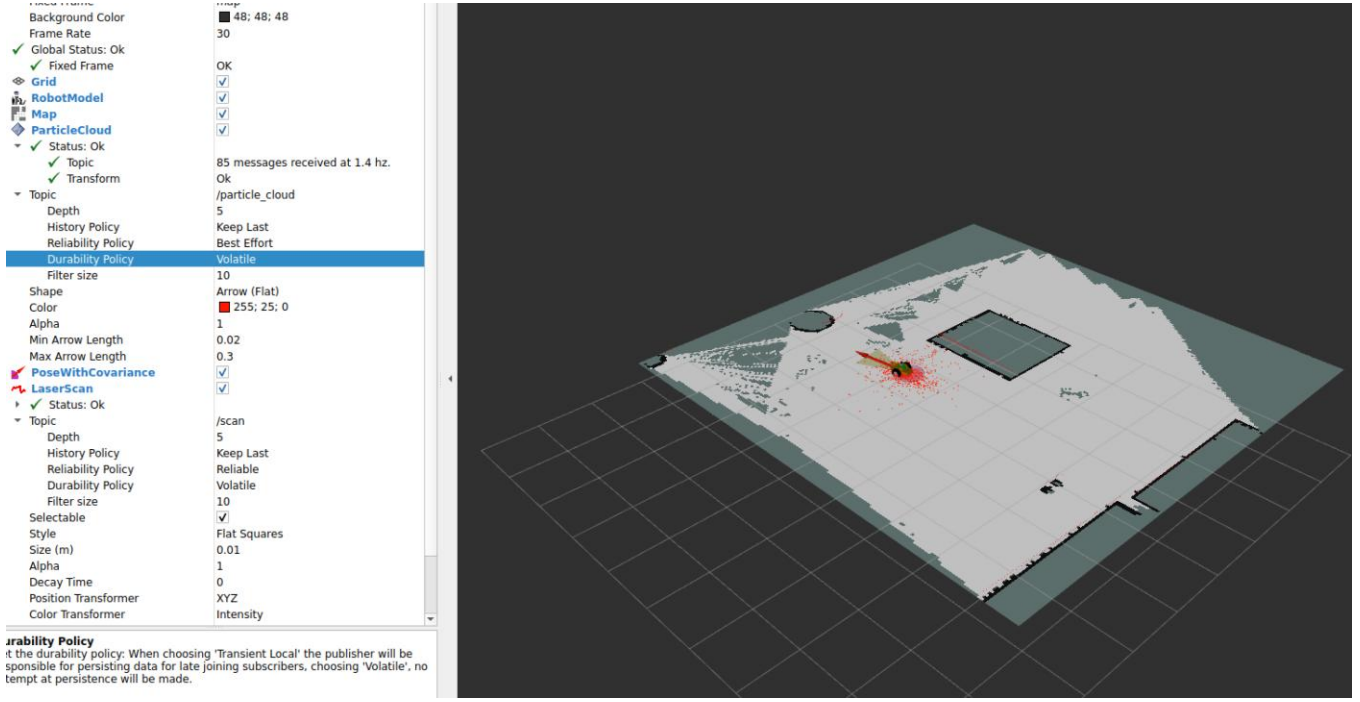
ROS Implementation – AMCL

sudo apt install ros-jazzy-nav2-bringup
sudo apt install ros-jazzy-nav2-amcl

Also in this case, we need to create a new launch file, and a new configuration file.

In the configuration file, we need to specify the map we want to use, that will be loaded by the amcl package.

Once loaded, it will be used as a reference for the laser scanner, to resample the particle filter



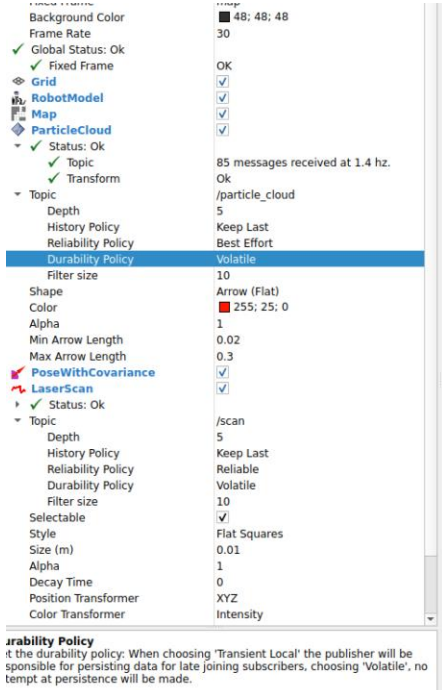
ROS Implementation – AMCL

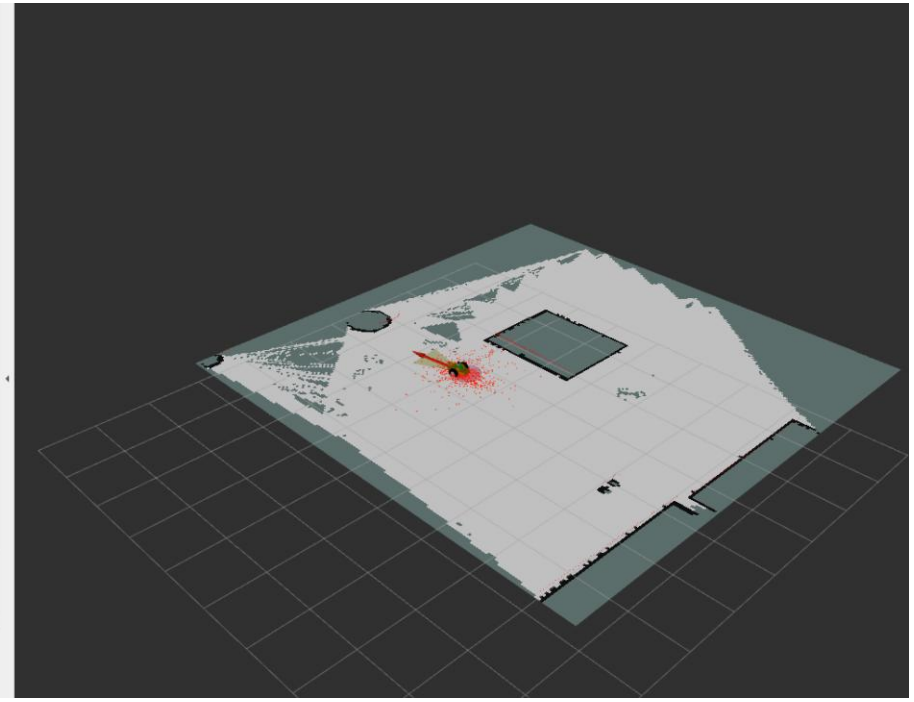
```
max_particles: 2000
min_particles: 500
odom_frame_id: "odom"
```

In Rviz, we could add the particle cloud topic, but to see the particles we need to change the Reliability and Durability policy

To start the localization process, we need to set an initial estimate (we could do this with RViz)

The estimate can be even different from the “real” position of the robot: AMCL will adjust the localization of the robot during its motion.





ROS Implementation – Localization

Please notice that you can perform localization also with SLAM Toolbox. In this case you require the serialized version of the map (it can be saved with the nav2_map_server node as well).

In this localization mode, the map keeps updating unlike with AMCL. Indeed, the localization mode does continue to update the pose-graph with new constraints and nodes, but the updated map expires over some time. SLAM toolbox describes it as "elastic", which means it holds the updated graph for some amount of time, but does not add it to the permanent graph.

3D - SLAM

- We will not see their practical implementation, but you can also build a 3d map of the environment in ROS2. There are different strategies that could be used.

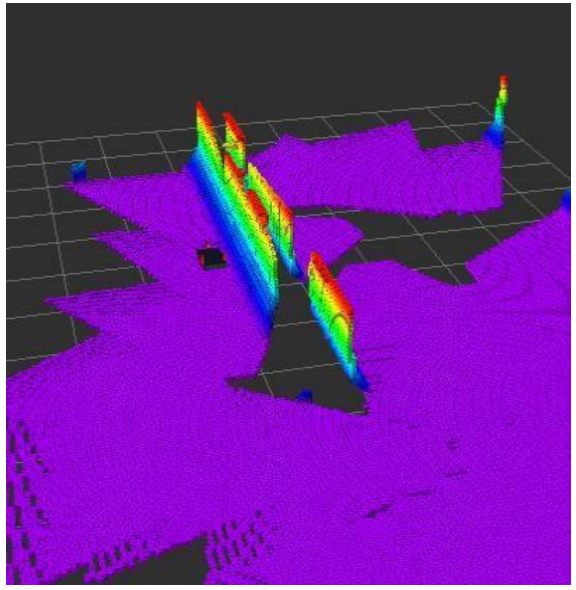
Octomap

Please notice that Octomap is not really a SLAM algorithm, since it does not implement any attempts of improving the robot localization or the knowledge of the surrounding environment by implementing methods seen in the previous class.

On the other side, it offers a set of tools and a way of modelling

3d spaces which may be useful for implementing 3d

SLAM algorithms.

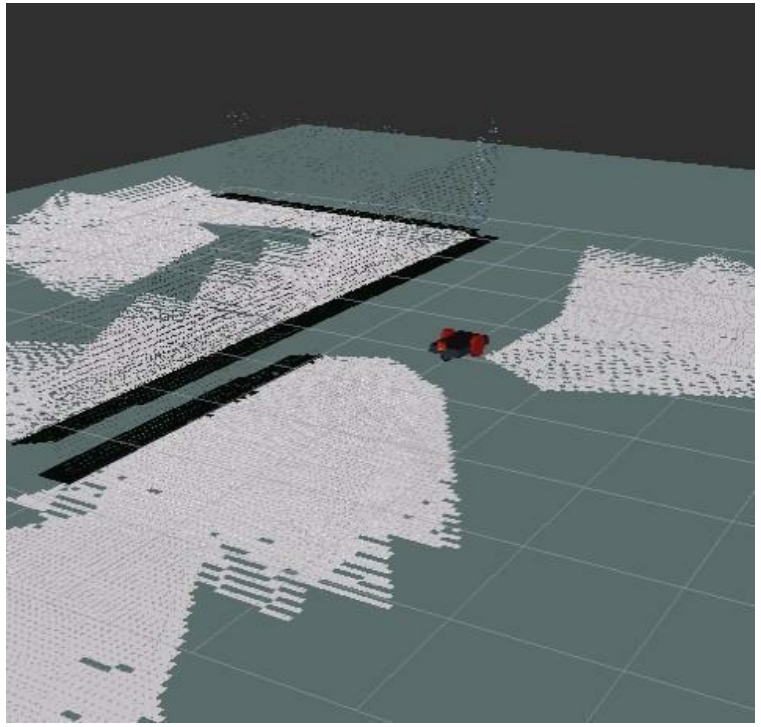


RTABMAP

RTABMap is aa RGB-D, Stereo and Lidar Graph-Based SLAM approach based on an incremental appearance-based loop closure detector (graphSLAM)

When a loop closure hypothesis is accepted, a new constraint is added to the map's graph, then a graph optimizer minimizes the errors in the map. A memory management approach is used to limit the number of locations used for loop closure detection and graph optimization, so that real-time constraints on large-scale environnements are always respected.

RTAB-Map can be used alone with a handheld Kinect, a stereo camera or a 3D lidar for 6DoF mapping, or on a robot equipped with a laser rangefinder for 3DoF mapping.



RTABMAP

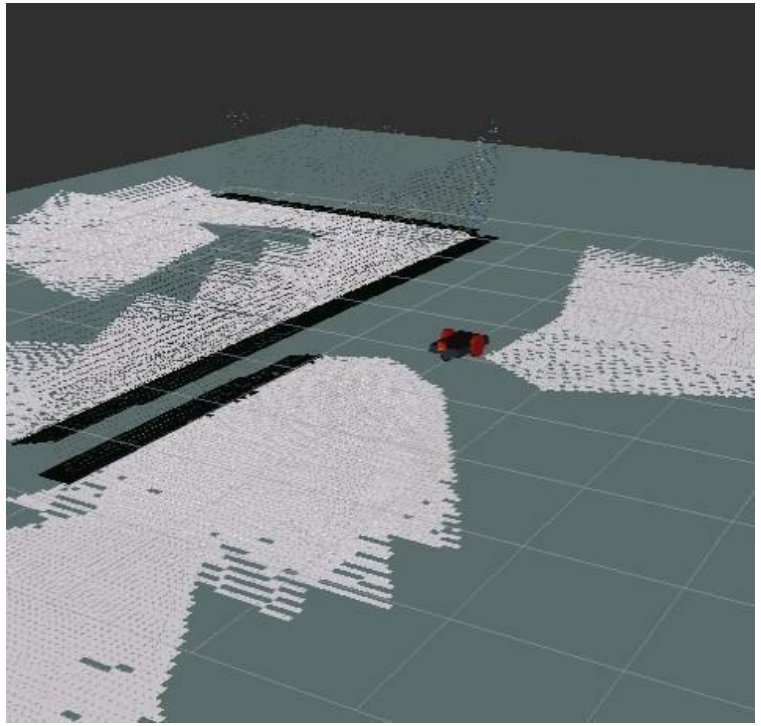
You may also need to increase the parameter queue_size
in rtabmap.launch.

Now we launch our simulation with the robot endowed with an RGBD sensor

- roslaunch gmapping display.launch

And we start the RTABMAP algorithm:

- roslaunch rtabmap_ros mapping.launch



Autonomous Navigation

Carmine Tommaso Recchiuto

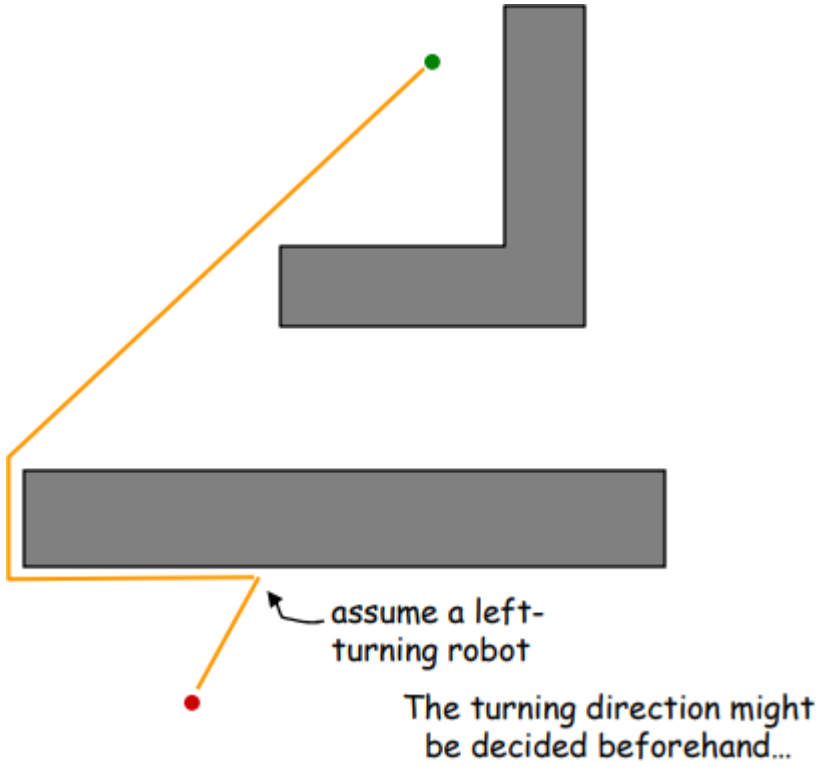
Autonomous Navigation

- The problem of autonomous navigation of robots involves a number of distinct research areas, intertwined and partially overlapped: control, path (or trajectories) planning and following, obstacle avoidance.
- Here, we will focus on some specific aspects***: global path planning, path following,*** *and* ***reactive avoidance***.

  Indeed, in many scenarios, robots should be essentially capable of planning a path between a starting point and a goal, following the planned path (or either a predetermined path), while avoiding collisions with dynamic obstacles.
- Today we will analyze these three aspects.

Bug Algorithms

- The simplest approach for navigating a robot in space is represented by the bug algorithms:



Bug0

1) head toward goal

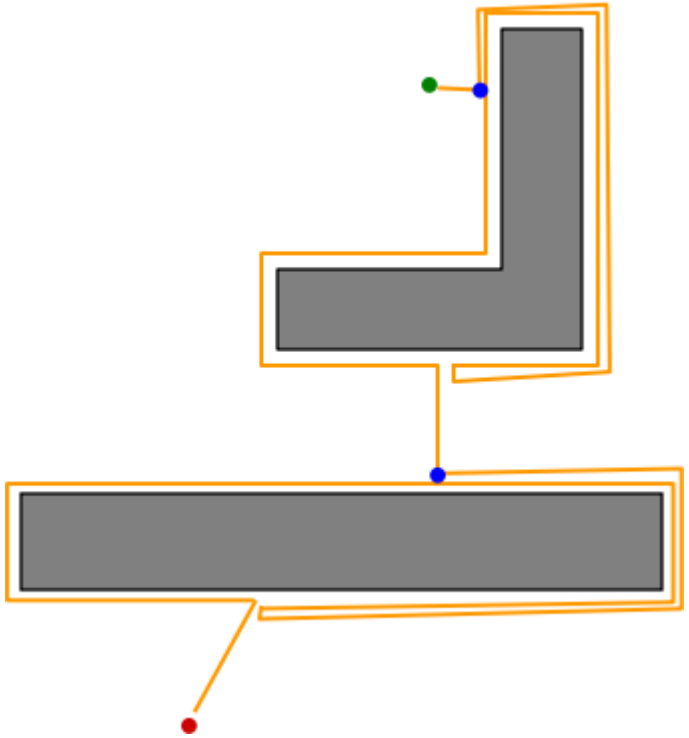
2) follow obstacles until you can head toward the goal again

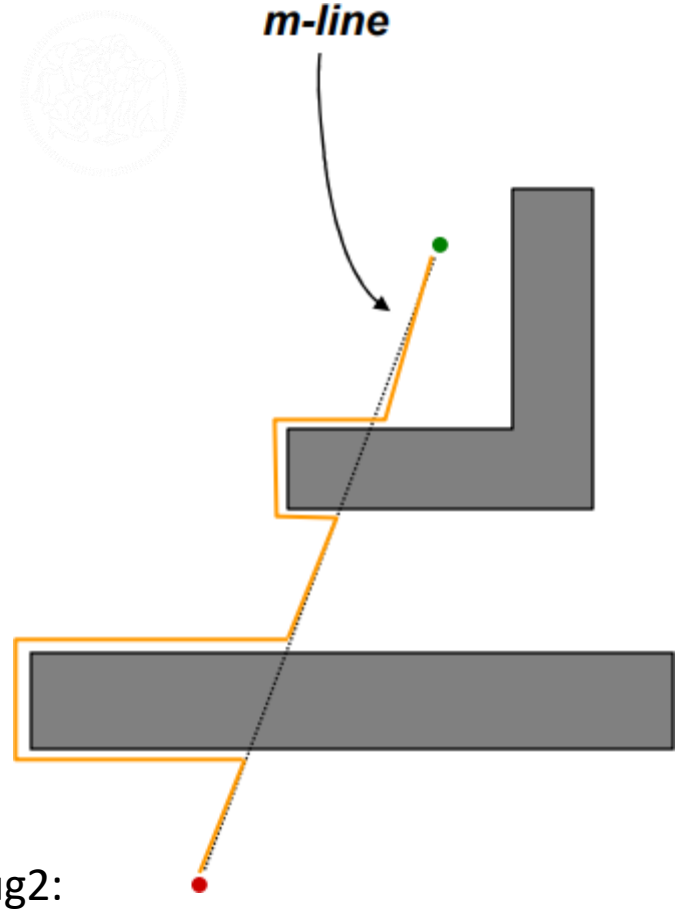
3) continue

Bug Algorithms

- Bug1:

1) head toward goal 2) if an obstacle is encountered, circumnavigate it and remember how close you get to the goal 3) return to that closest point (by wall-following) and continue





- Bug2:

1) head toward goal on the m-line

2) if an obstacle is in the way, follow it until you encounter the m-line again.

Path Planning

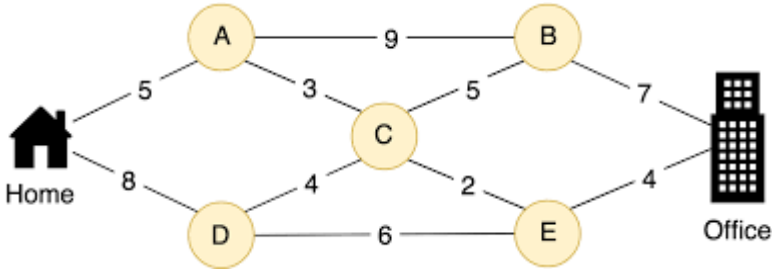
✓A fundamental robotic task is to plan a collision-free path from a start to a goal position, possibly passing through a number of waypoints, taking into account the dynamics of the robot and avoiding static obstacles. As for the SLAM algorithms, we will analyze here the most relevant classes of algorithms used for path planning, which are:

- *Graph-based search algorithms:* as the name says, they represent the world as a grid and they try to find a path between the start and the goal
- *Sampling-based algorithms:* the key idea here is to compute in advance whether a single configuration is collision-free.
- *Potential fields*: the robot is assimilated to a particle moving inside a potential field created according to the target (attractive potential) and the perceived obstacles (repulsive potential)
- *Optimization-based algorithms*: this class of algorithm handles path planning by using constraints or cost functions, usually applying nonlinear optimization
- *Learning-based algorithms:* they use a learning process to improve the performance of a system based on its past experiences.

Graph-based search algorithms

✓ Graph-based search algorithms have been the most popular way to perform path planning, being extensively studied in many different domains. In these approaches, the environment is represented as a grid, where cells can be occupied or not. Then, algorithms for graph exploration are used for finding a path between the start and the goal. Graph-based search algorithms give the advantages of having fast searching abilities and possible online implementation, but they generate non-smooth trajectories and they may not be suitable for large areas. These methods have found wide applications in the robotic domain:

- Dijkstra's algorithm: Given a connected graph, the algorithm finds the path with the lowest cost (i.e., the shortest path) between the starting vertex and the target vertex. The algorithm also creates a set of visited vertices, to avoid endless loops while traversing the graph, and a vertex visiting queue, with priorities. The shorter the vertex distance to the starting vertex the higher its priority.



✓Each edge in the graph is associated with a weight (i.e. the distance between the two nodes).

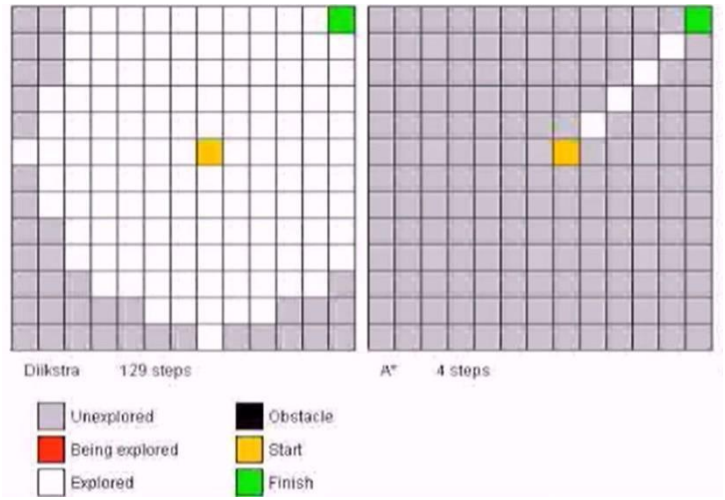
✓In the example, the algorithm will find the path: *Home-A-C-E-Office*

Graph-based search algorithms

✓The Djikstra algorithm does not use any heuristic* to optimize the process. In the example of the previous page, I may have other nodes which brings me in the opposite direction with respect of the office: the algorithm will still consider that.

✓The A* algorithm improves the Dijkstra algorithm by using a heuristic, the distance to the target, combining the two criteria (distance to the source node and to the target)

Djikstra vs A*



✓Another widespread approach in this class is the D*

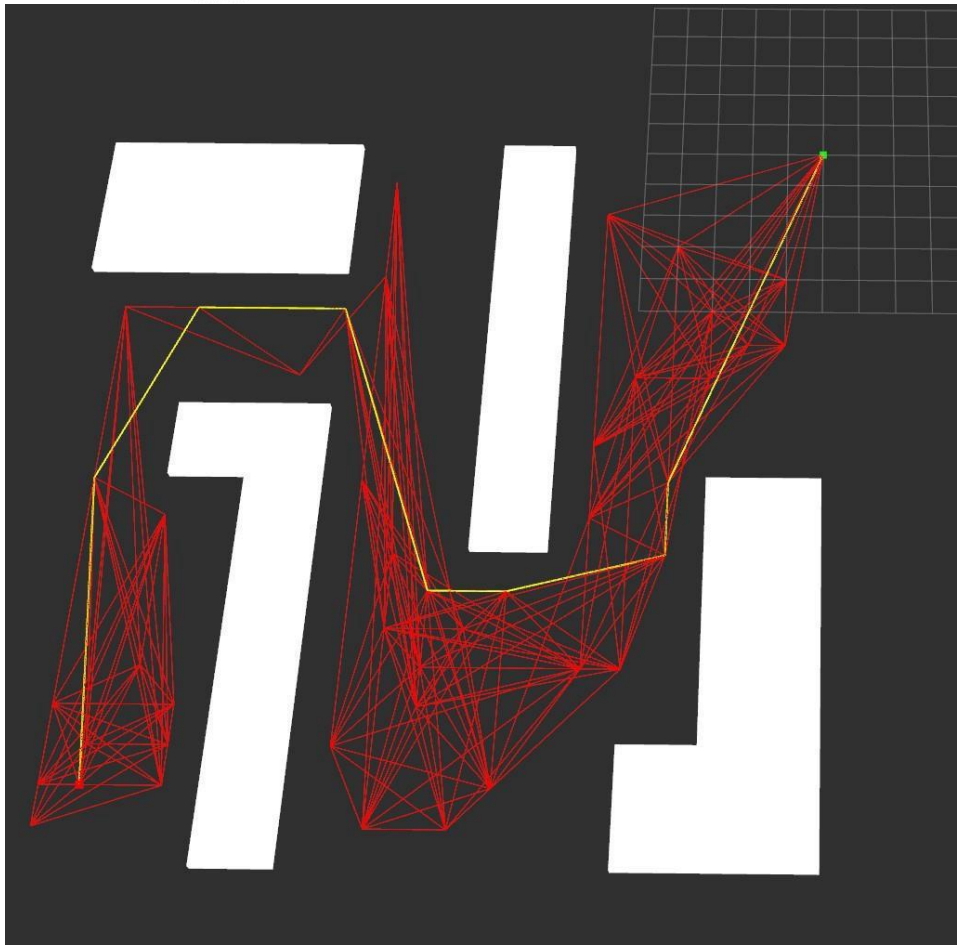
D* begins by searching backwards from the goal node. It differs from A* in particular in relation to how nodes of the graph are handled when an obstacle is encountered.

* A heuristic is a function that informs the algorithm to enable a more directed search

Sampling-based algorithms

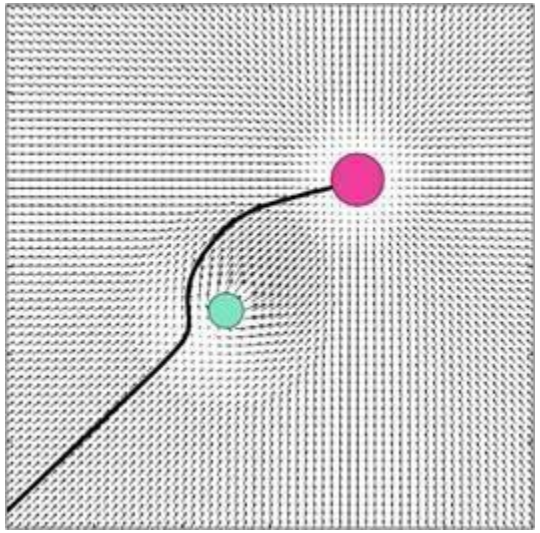
✓In sampling-based algorithms, samples of the free space that connect the start to the goal are generated, by means of a graph or a tree data structure. Given the probabilistic approach, these kinds of algorithms do not offer an optimal solution; however, they are appropriate for a complex environment and present fast searching abilities.

✓Among these methods, the most commonly used are probabilistic roadmap (PRM) and rapidly exploring random trees (RRT)



Potential Fields

✓The artificial potential fields (APF) approach, the original formulation of which was proposed in 1986 is considered one of the fastest methods for path planning. Indeed, in this kind of approach, the robot is assimilated to a particle moving inside a potential field created according to the target (attractive potential) and the perceived obstacles (repulsive potential). The method, in its original formulation, is simple and ensures fast convergence, but it has also the major drawback of local minimum risks, i.e., for some obstacles the potential field can have a number of local minima in which the robot will be trapped, even if some techniques have been proposed to overcome this issue (mostly based on randomness).



Path Planning

Path-planning approach		Pros	Cons
Graph-based	Djikstra's algorithm A* LazyTheta* D*-Lite	Fast searching abilities and online implementation	Nonsmoothness. Difficult implementation in large areas
Sampling-based	Belief Roadmap (PRM) RRT RRBT Kinodynamic RRT* Informed RRT* Adaptive step-length RRT	Fast searching abilities Suitable for complex environments	The found solution is not optimal
Potential fields	APF	Fast convergence and relatively simple implementation	The robot can be stranded at points of local minima
Optimization-based	Differential Flatness MIQP MISDP MILP BLP	Optimal trajectories that can satisfy constraints on velocities and accelerations	Complex implementation and high number of parameters
Swarm-optimization	PSO Ant Colony Optimization Bee Colony Optimization Memetic Algorithms	Efficient methods, able to deal with multiobjective problems	High time complexity Approaches sensitive to the optimization parameters
Learning-based	Genetic Algorithms Neural Network MPC + Reinforcement Learning MDP Adaptive Informative	Algorithms able to solve high complexity and multiobjective problems	A training process is usually necessary, where full state observations are required

Path Following

While the ability to plan a motion through a safe path in a complex environment is a relevant aspect in the mobile robotics context, the ability to follow the path (or some specific predetermined paths) it is also important as well.

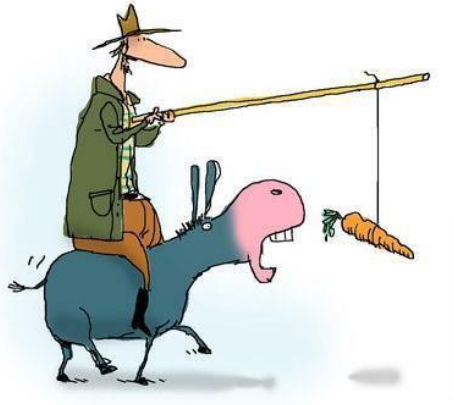
Again, a general classification of the commonly used algorithms for path following applied to robots is given. In the analysis, we focused on path-following approaches, i.e., methods for following a predefined path that does not involve time as a constraint, deliberately omitting trajectory-tracking approaches.

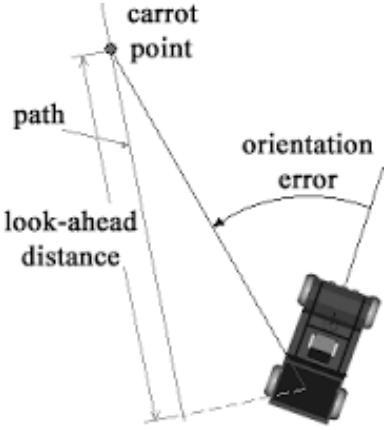
- *Carrot Chasing*
- *Nonlinear control or Lyapunov functions*
- *Pure pursuit*
- *Vector Fields*
- *Linear Quadratic Regulators*
- *Model-based predictive controllers*
- *Surface intersection*

Carrot Chasing

This is probably the simpler approach for path following, consisting of the introduction of a Virtual Target Point moving on the path, and the robot is controlled in order to reach the VTP. In this approach, the VTP is called the carrot and the robot chases the carrot, which explains the name *carrot-chasing* algorithm. The VTP position is updated online in relation to the path to be followed. Thus, the robot moves in consecutive waypoints.

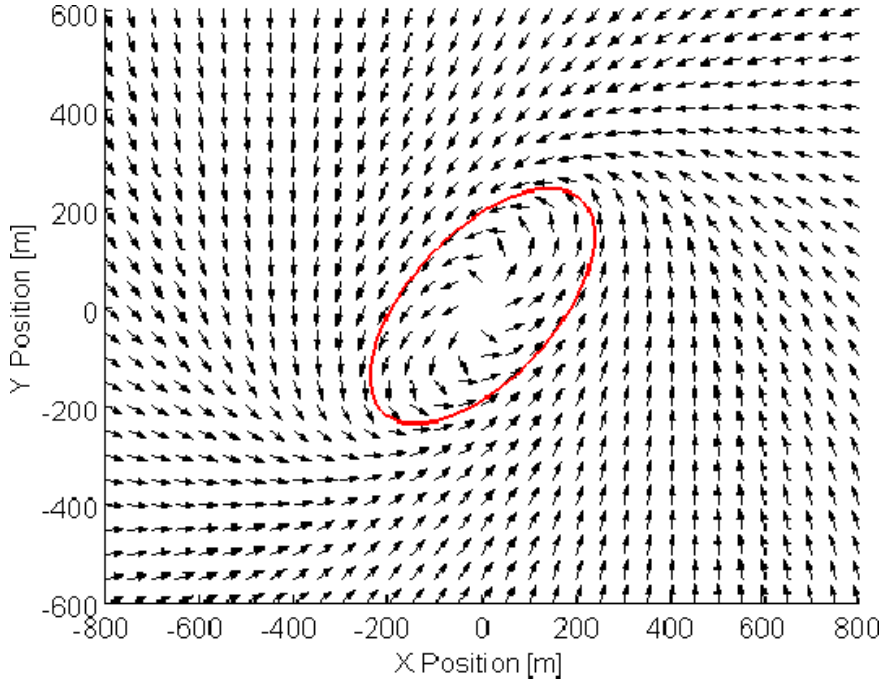
However, if the robot moves directly toward the VTP, it may cause an overshoot and therefore more time required to settle on the path. Thus, usually the carrot-chasing approach chooses a point ahead of the closest point along the path.





Vector Fields

In these methods, the vehicle is driven with the aid of a vector field, which generates, in each point in space, the target direction (and also the speed, in the case of trajectory tracking) for the convergence of the robot to the path.



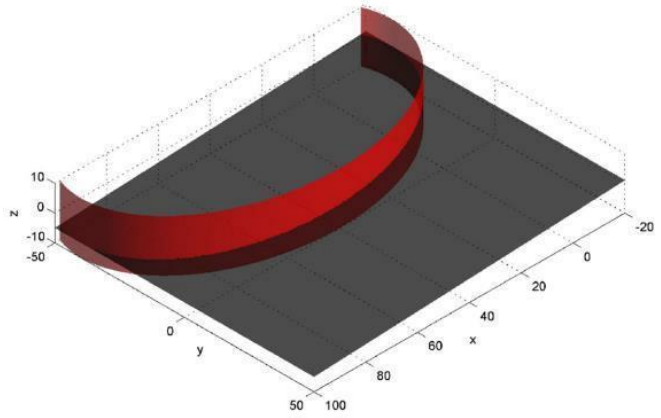
Also in this scenario, appropriate Lyapunov based on the distance between the robot and the path, may be used in a manner to force the vehicle to follow the path.

Surface Intersection

✓This method consists in a feedback control model for path following: differently from the others, the approach does not require to compute a projection of the robot's position on the path, and it does not consider a moving virtual target, but it relies on the general idea that a path can be represented as the intersection of two properly chosen surfaces, f1(x,y)=0 , f2(x,y)=0.

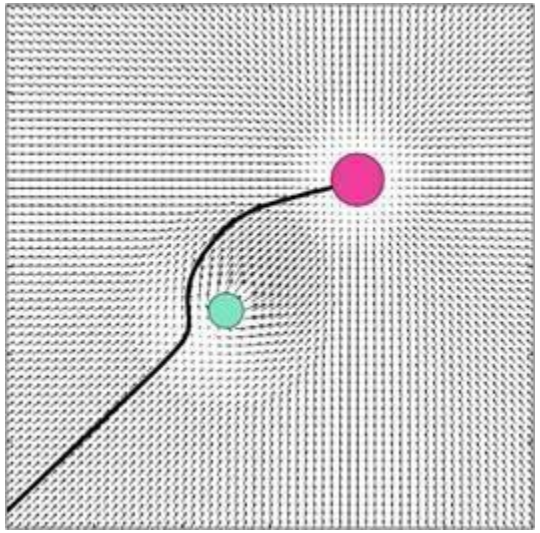
✓Given this mathematical formulation, a velocity vector for tracking the path may be computed considering the gradient vector and the tangential vector to both curves

✓The same approach can be also extended to the 3D scenarios (UAVs or underwater vehicles).



Potential Fields

✓The artificial potential fields (APF) approach, the original formulation of which was proposed in 1986 is considered one of the fastest methods for path planning. Indeed, in this kind of approach, the robot is assimilated to a particle moving inside a potential field created according to the target (attractive potential) and the perceived obstacles (repulsive potential). The method, in its original formulation, is simple and ensures fast convergence, but it has also the major drawback of local minimum risks, i.e., for some obstacles the potential field can have a number of local minima in which the robot will be trapped, even if some techniques have been proposed to overcome this issue (mostly based on randomness).



Path Following

Path-following approach	Pros	Cons
Carrot Chasing	- Simple implementation	- Low tolerance to disturbance
NLGL	- Asymptotic convergence guaranteed for any type of path	- Cross-track error higher than VF
PLOS	- Simple implementation;	- Sensitive to gains
	- Robust in challenging conditions	- Convergence is not always guaranteed
VFs-based	- The control effort is optimized	- Higher error
		- Convergence is not always guaranteed
LQR-based	- Low cross-track error	- Complex implementation
		- Chattering effect
MPC	- Good performance even in challenging environments	- Complex implementation
Surface intersection	- Absence of any bounds in the initial position - Simple implementation	- Paths need to be represented analitically

Collision Avoidance

Finally, during its motion, a robot should be capable of avoiding unplanned obstacle as well as collisions with other robots. Therefore, robots must be endowed with reactive obstacle avoidance strategies. Differently from the Path Planning problem, approaches described here do not deal with the problem of finding a global collision-free path from a start to a goal position, but take into account only local, eventually moving, obstacles that cannot be *a priori* considered while finding the global path.

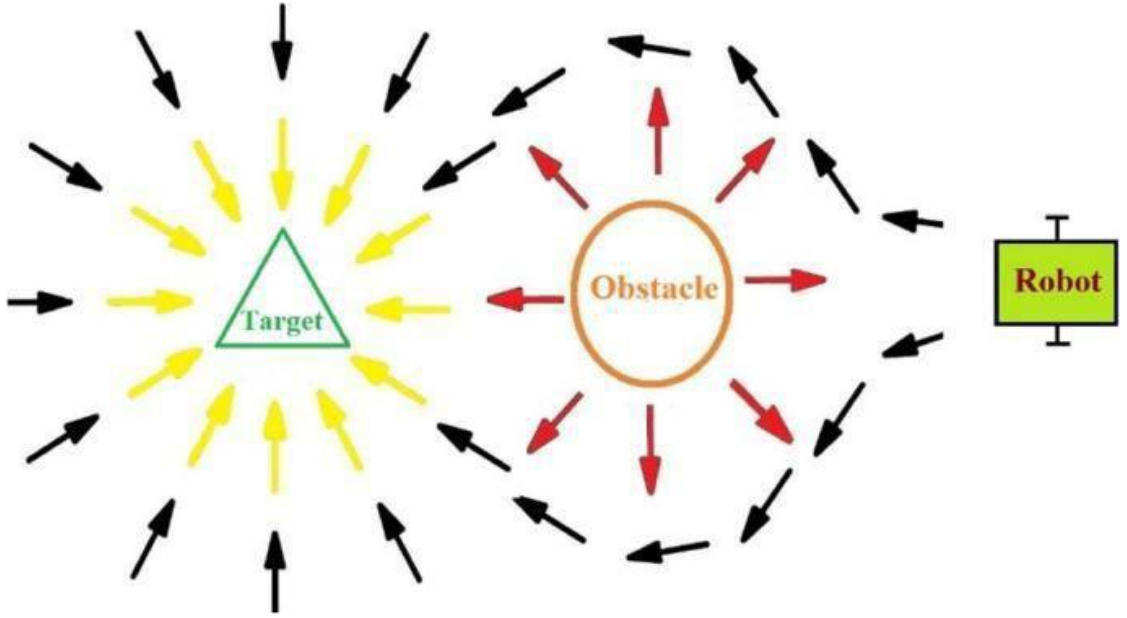
As we will see next class, in practical implementation we usually have a global planner (which implements one of the aforementioned algorithms for finding a global path) and a local planner, which deals with the other two aspects (following the path avoiding obstacles).

As usual, we can define a general classification of all approaches:

- *Artificial Potential Fields*
- *Vector Field Histograms*
- *Dynamic Window Approach*
- *Velocity Obstacles*
- *Path Deformation*
- *Fuzzy Logic*
- *Bio-inspired approach*

Artificial Potential Fields

- As for global path planning, some of the most successful approaches to local obstacle avoidance rely on the concept of computing artificial repulsive forces, which are exerted on the robot by surrounding obstacles, and an attractive force exerted by the goal (the target or the path to be followed).
- However, as seen before, APFs have well known problems when dealing with robot navigation, among which is the presence of local minima in the field

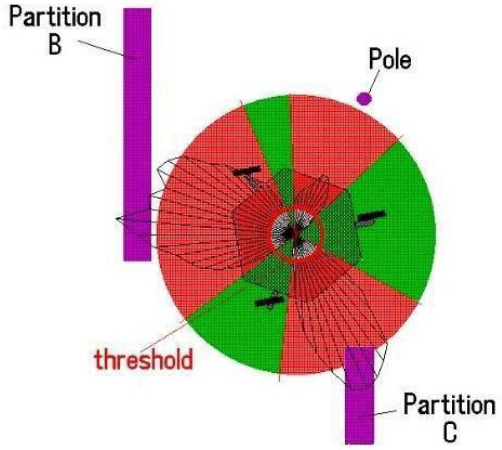


Vector Field Histograms

The VFH approach solves the problem by computing a set of candidate motion directions and then selecting one of them.

Practically, the space is divided into cells of the same size, and every cell has a value that represent the likelihood of an obstacle.

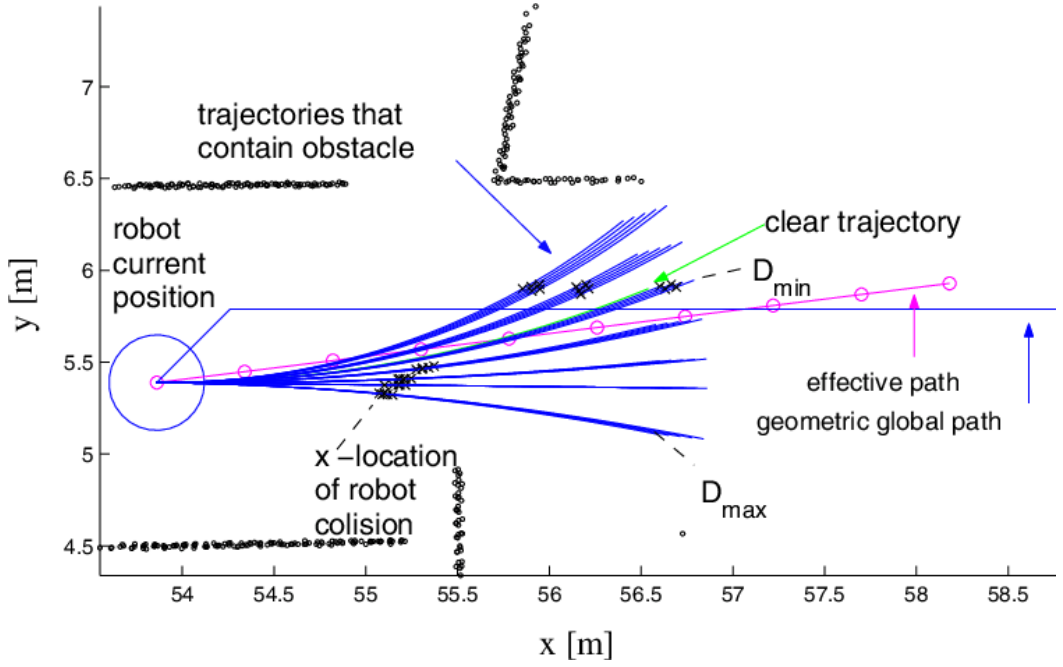
The map is thus translated into a polar histogram that represents the space surrounding the robots and the nearness of the obstacles. The obstacle avoidance behavior is achieved by choosing a direction within the free angular sectors, using a cost function taking into account the orientation of the robot and the goal.



Dynamic Window Approach

✓The approach consists in a real-time collision avoidance strategy for mobile robots that is directly derived from the dynamics of the robot, dealing with the constraints imposed by limited velocities and accelerations.

✓Indeed, the robot's control space is sampled, and for each sample the resulting trajectory is evaluated using metrics that consider proximity to obstacles, to the goal, or to the path. The highest-scoring trajectory is thus used for the robot's motion.

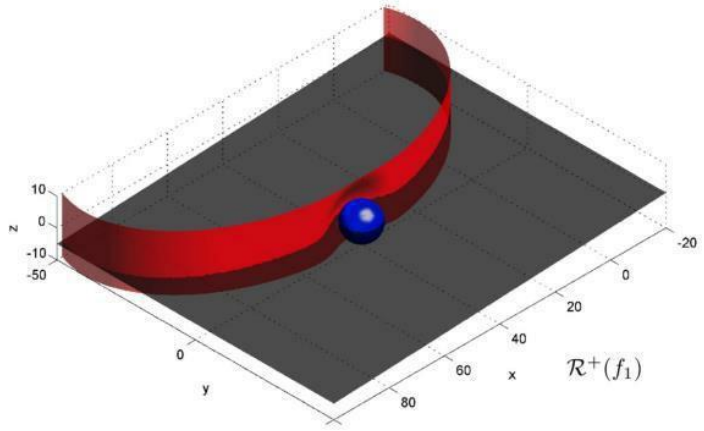


Path Deformation

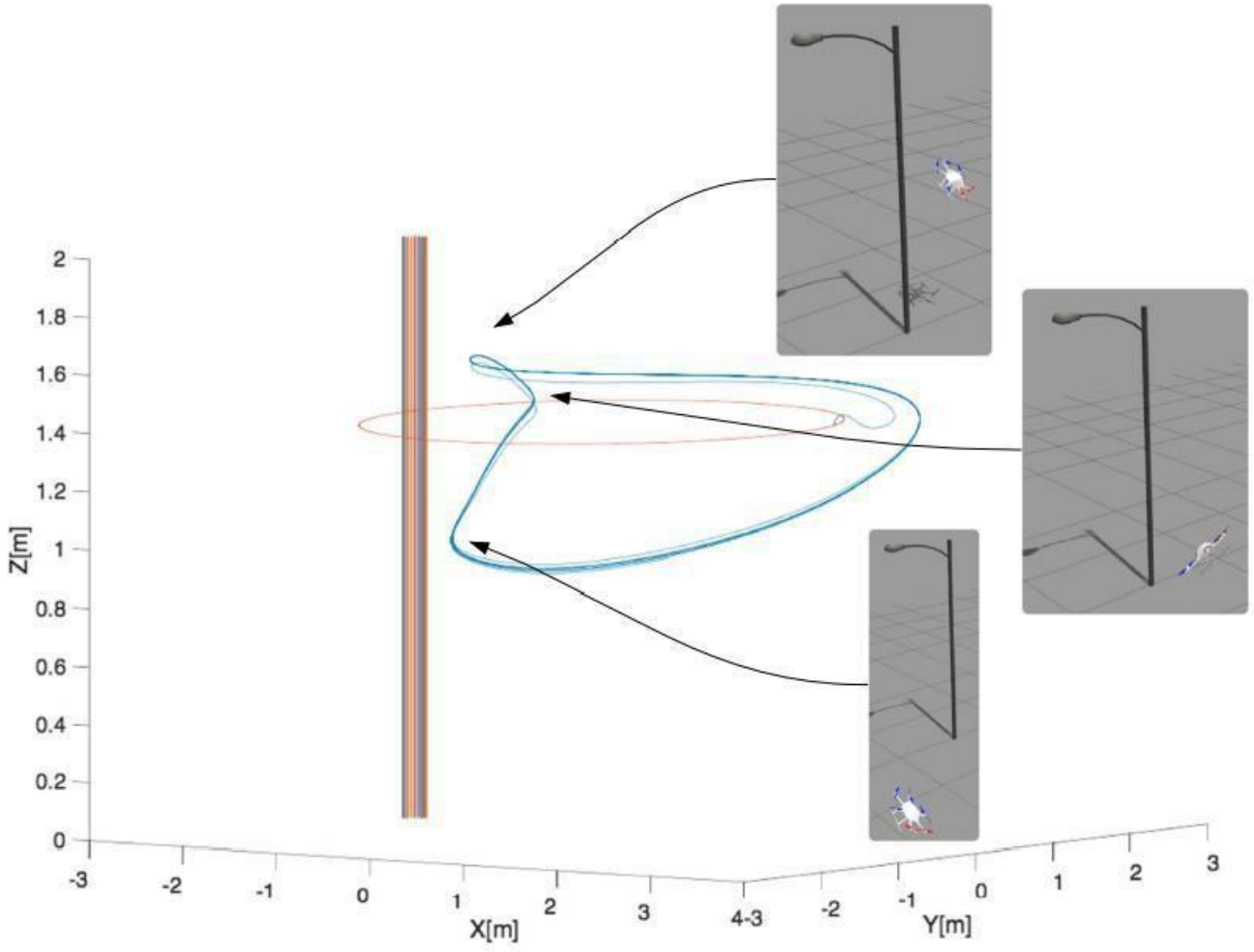
✓This approach shares some similarities with the APFs, but offers some advantages. It goes together with the Surface Intersection path following approach

✓When a robot perceives the presence of any obstacle, the path is deformed by means of obstacle functions added to one of the surface functions. Gaussian functions are used as obstacle functions, and by tuning the amplitude and the width of the function it is possible to guarantee that the deformed path does not collide with obstacles

✓One of the advantages of this approach is that it requires very few computational resources, as a consequence of its extreme simplicity.



Path Deformation



Collision Avoidance

Obstacle avoidance approach	Pros	Cons
Artificial potential fields	- Smooth path from the source to the destination	- Narrow passages can be critical
	- Fast convergence	- The robot can be stranded at points of local minima
Vector fields histograms	- Ensures obstacle avoidance in noisy environments	- Computationally expensive
Dynamic window approach	- Directly derived from the robot's dynamics	- Narrow passages may be critical
	- It may be used at high speed	
Velocity obstacles	- It works well in dynamic environments	- Geometry of the obstacles should be given explicitly
	- It allows robots to move in narrow passages	
Path deformation	- Robust to disturbance in dynamic environments	- The path needs to be represented analitically
	- Simple implementation	
Fuzzy logic	- Fast convergence	- Subject to classical problems such as oscillations and local minima
	- Simple implementation	
Bio-inspired approaches	- Robust behaviours in dynamic and complex environments	- Optical flow: not suited for 3D Neural networks: a training process is necessary, and the size of the network can become large

ROS 2 Implementation

The ROS2 navigation stack implements all the aspects seen before:

- Path planning (Djikstra or A*)
- Path following + Obstacle Avoidance (Dynamic Window approach)

The stack should be used in conjunction with a mapping (or localization) algorithm: a path can be planned if a map is given!

We need a further launch file + config file (navigation.launch.py, navigation.yaml)

```
sudo apt-get install ros-jazzy-nav2*
```

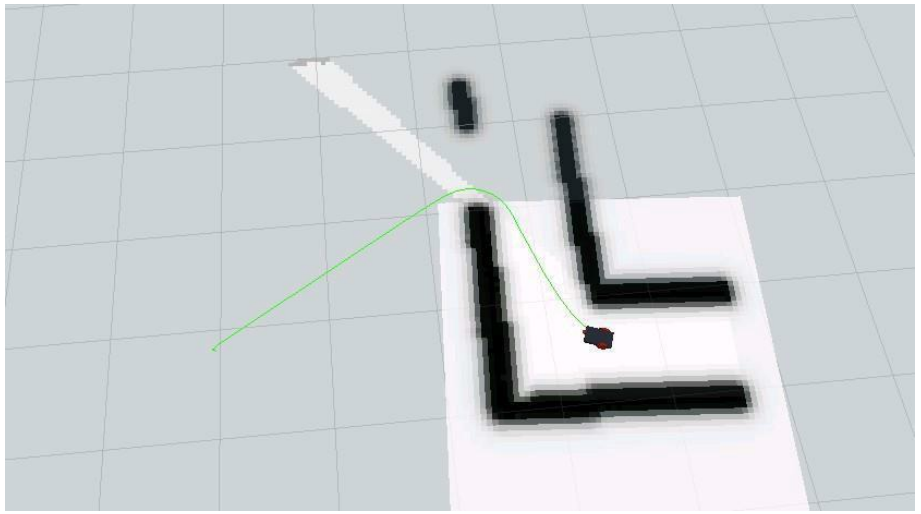
Costmap

Costmap parameters tuning is essential for the success of local planners. In ROS, costmap is composed of *static map layer*, *obstacle map layer* and *inflation layer*.

Static map layer directly interprets the given static SLAM map provided to the navigation stack.

Obstacle map layer includes 2D obstacles and 3D obstacles (voxel layer).
Inflation layer is where obstacles are inflated to calculate cost for each 2D costmap cell.

Besides, there is a global costmap, as well as a local costmap. Global costmap is generated by inflating the obstacles on the map provided to the navigation stack. Local costmap is generated by inflating obstacles detected by the robot's sensors in real time.



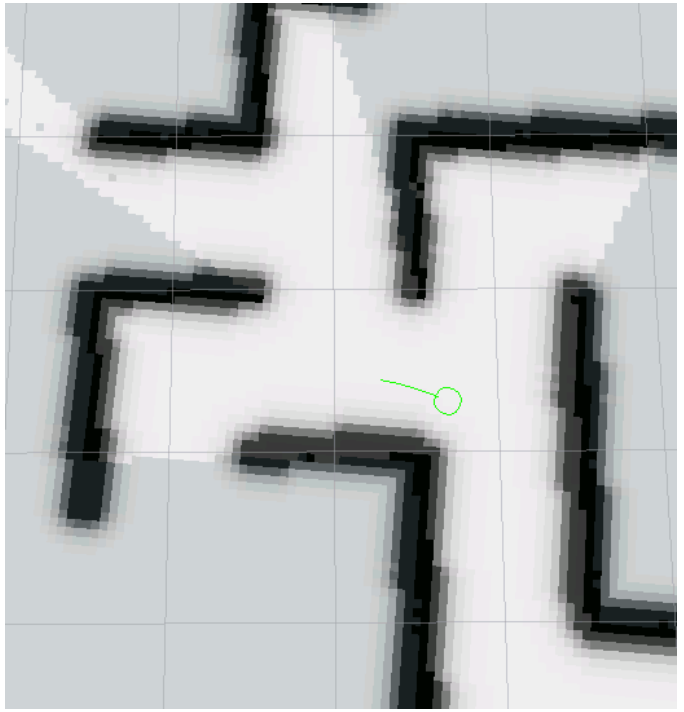
Costmap

Together with the costmap, it is necessary to define the footprint of our robot, i.e., the contour of the mobile base.

In ROS, it is represented by a two dimensional array of the form [x0, y0], [x1, y1], [x2, y2], ...]. This footprint will be used to compute the radius of inscribed circle and circumscribed circle, which are used to inflate obstacles in a way that fits this robot. Usually for safety, we want to have the footprint to be slightly larger than the robot's real contour.

Also, the radius of the circle may be directly given.

The inflation layer is consisted of cells with cost ranging from 0 to 255. Each cell is either occupied, free of obstacles, or unknown. *inflation radius* and *cost scaling factor* are the parameters that determine the inflation. *inflation radius* controls how far away the zero cost point is from the obstacle. *cost scaling factor* is inversely proportional to the cost of a cell. Setting it higher will make the decay curve more steep.



Costmap

A more gentle inflation curve of the costmap has the advantage the robot would prefer to move in the middle of obstacles.

The *costmap resolution* can be set separately for local

costmap and global costmap. They affect computation load and path planning. With low resolution (>= 0.05), in narrow passways, the obstacle region may overlap and thus the local planner will not be able to find a path through. For global costmap resolution, it is enough to keep it the same as the resolution of the map provided to navigation stack.

The width and the height of the global costmap determine the distance of the goals that we can set.