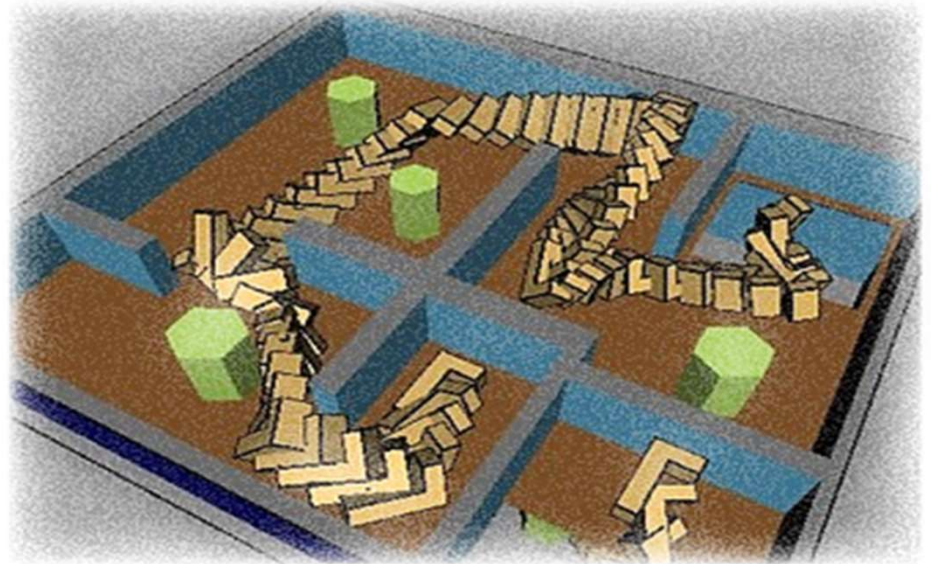
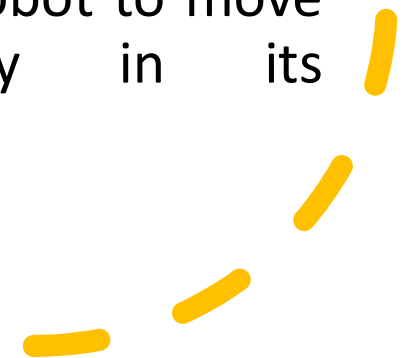


# Module 7: Robot Navigation



# Robot Navigation Definition

Robot navigation refers to the autonomous or semi-autonomous ability of a robot to plan, control, and execute movements within its environment with the goal of reaching a specific destination or following a prescribed path while avoiding obstacles. The navigation process involves the integration of sensory information, localization, mapping, and motion planning to enable the robot to move purposefully and adaptively in its surroundings.



# Key Elements

## Perception:

- **Sensors:** Robots are equipped with sensors such as LIDAR, cameras, ultrasonic sensors, and inertial measurement units (IMUs) to perceive their environment and detect obstacles .

## Localization:

- **Odometry:** Robots use odometry, which involves tracking wheel or joint movements, to estimate their current position.
- **Sensor Fusion:** Combining data from multiple sensors helps improve localization accuracy.
- **SLAM (Simultaneous Localization and Mapping):** In unknown environments, SLAM algorithms help a robot build a map while simultaneously determining its location.

## Mapping:

- **Environment Representation:** The map provides a representation of the environment that the robot can use for navigation, ie. Occupancy Grid Maps.

## Motion Planning:

- **Path Planning:** Robots use algorithms to plan a collision-free path from their current position to a target location.
- **Dynamic Path Planning:** Consideration of dynamic obstacles and real-time adjustments to the planned path.
- **Local vs. Global Planning:** Local planners focus on short-term movements, while global planners handle longer-term navigation.

## Control:

- **Actuators:** Robots use actuators, such as motors or servos, to control their movements based on the planned path.
- **Feedback Control:** Continuous adjustment of movements based on sensor feedback ensures accurate execution of the planned path.

# Challenges

## Sensor Noise and Uncertainty:

- Sensor readings may have noise, and uncertainty must be considered in perception and localization.

## Dynamic Environments:

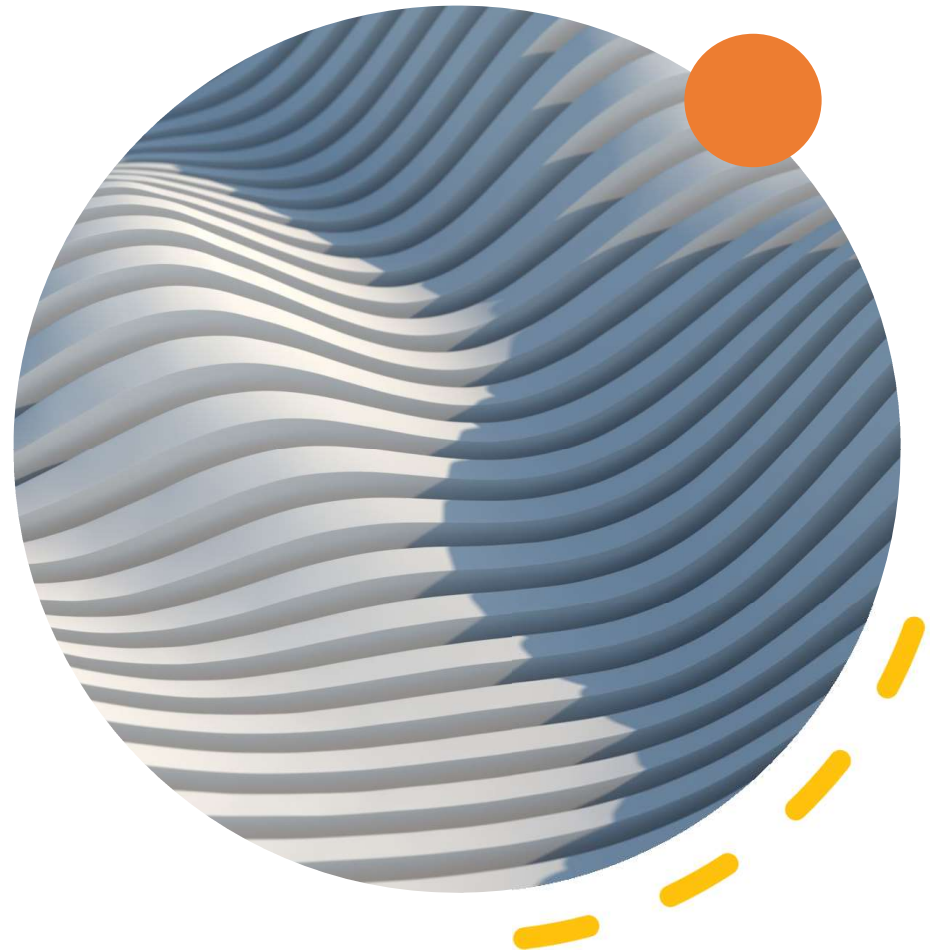
- Moving obstacles or changes in the environment require real-time adaptation of navigation plans.

## Map Accuracy:

- The accuracy of the map affects the precision of navigation.

## Computational Efficiency:

- Real-time planning and execution of navigation plans demand efficient algorithms.



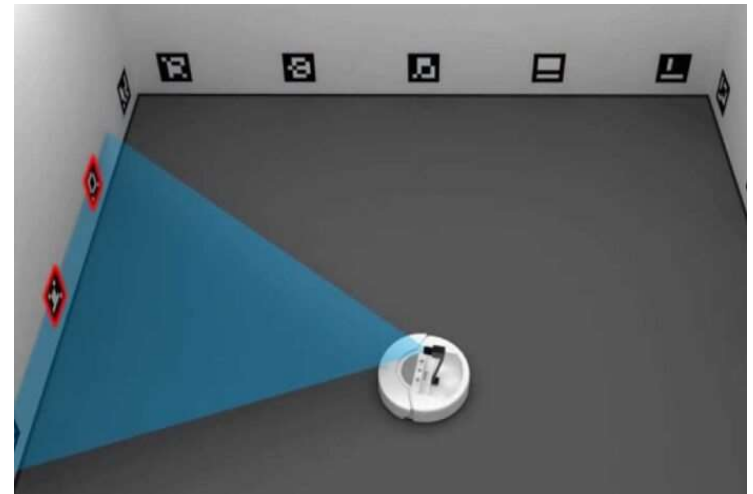
# Localization

---



# Localization

Robot localization refers to the process by which a robot determines and continually updates its own position and orientation within its environment. The goal of localization is to enable the robot to be aware of its spatial coordinates, allowing it to navigate, plan paths, and interact with the surroundings effectively. Localization is a fundamental aspect of robotics, contributing to the autonomy and adaptability of robots in various applications.





# Sensors

Sensors play a pivotal role in the process of robot localization, providing the necessary input for estimating the robot's position and orientation within its environment. Different types of sensors capture various aspects of the surroundings, and the integration of their data is crucial for accurate and reliable localization.

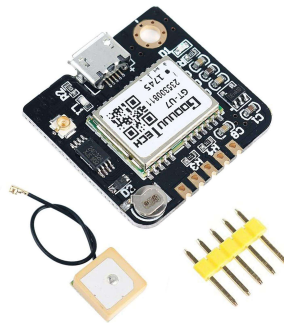


# Sensor Types

---



Linear Encoder



GPS



Lidar



Sonar



Beacon/RFID



Rotary Encoder



IMU



Camera



Magnetometer



# Sensor fusion techniques

## Kalman Filtering:

- **Description:** Kalman filters are widely used for sensor fusion in robotics. They estimate the state of a system by iteratively predicting and updating the state using noisy sensor measurements.
- **Application:** Kalman filters are effective for fusing data from multiple sensors, such as odometry, GPS, and IMU, to achieve accurate localization.

## Extended Kalman Filtering (EKF):

- **Description:** EKF is an extension of the Kalman filter that can handle non-linear system models. It linearizes the system model at each time step to perform predictions and updates.
- **Application:** EKF is commonly applied in sensor fusion scenarios where the relationship between the state variables and sensor measurements is non-linear.

# Sensor fusion techniques

## Particle Filtering:

- **Description:** Particle filters, or Monte Carlo localization, use a set of particles to represent the possible states of a robot. These particles are updated based on sensor measurements and motion models.
- **Application:** Particle filters are effective in scenarios with non-Gaussian uncertainties and can handle complex, non-linear relationships between robot states and sensor measurements.

## Factor Graphs:

- **Description:** Factor graphs are graphical models that represent the dependencies between variables in a system. In the context of robot localization, factor graphs can model relationships between robot poses, landmarks, and sensor measurements.
- **Application:** Factor graphs are versatile and can be used for both online and offline sensor fusion. They are particularly useful in SLAM (Simultaneous Localization and Mapping) problems.

# Sensor fusion techniques

## Complementary Filtering:

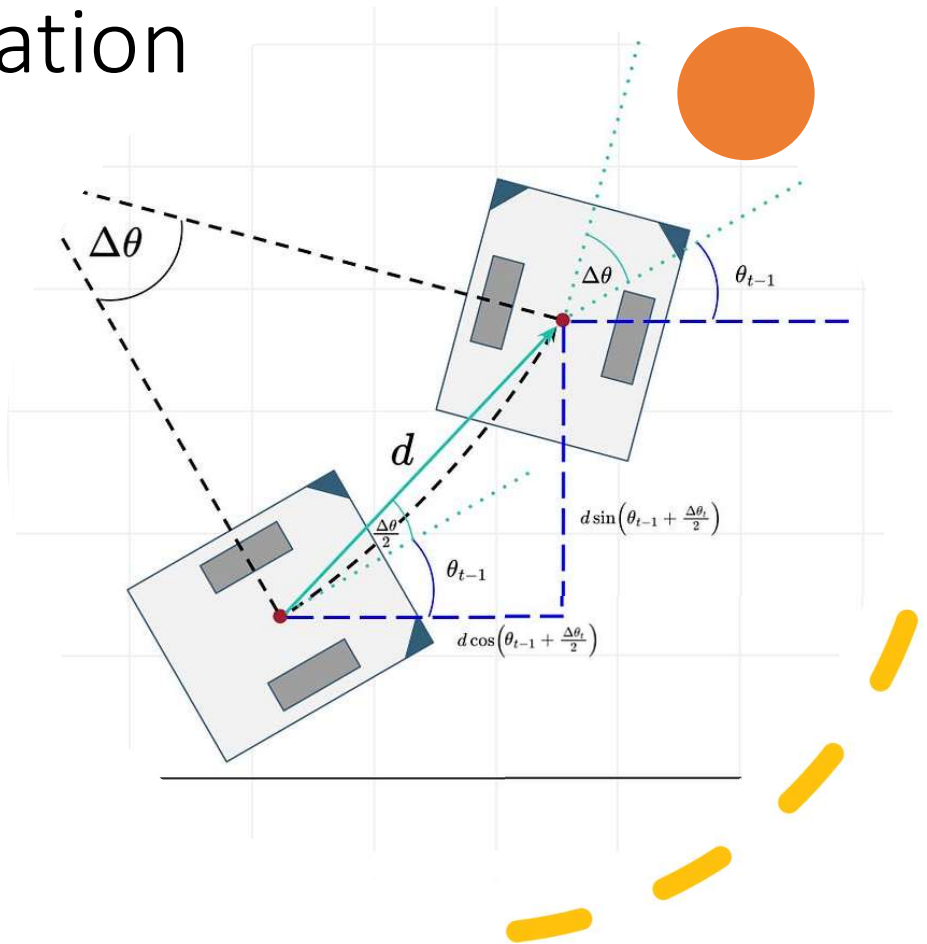
- **Description:** Complementary filters combine high-pass and low-pass filters to fuse data from different sensors. High-pass filters are used for fast-changing signals, while low-pass filters are used for slow-changing signals.
- **Application:** Complementary filtering is often applied in situations where multiple sensors provide complementary information, such as combining accelerometer and gyroscope data for orientation estimation.

## Sensor Fusion with Bayesian Methods:

- **Description:** Bayesian methods, such as Bayesian filtering, use probability theory to model uncertainties in sensor measurements and robot state estimates. Bayesian filters include Bayes' rule to update beliefs based on new evidence.
- **Application:** Bayesian methods are foundational in sensor fusion, providing a probabilistic framework for combining information from multiple sensors in a coherent and mathematically rigorous way.

# Odometry based Localization

Odometry, refers to the method of estimating a robot's position and orientation by tracking the changes in its position over time based on the movement of its wheels or joints. This technique is fundamental for robot navigation, providing continuous feedback about the robot's displacement.



# Key Elements



## Wheel Encoders:

**Function:** Wheel encoders are sensors attached to the robot's wheels or joints. They measure the rotation of the wheels, providing information about the distance traveled.

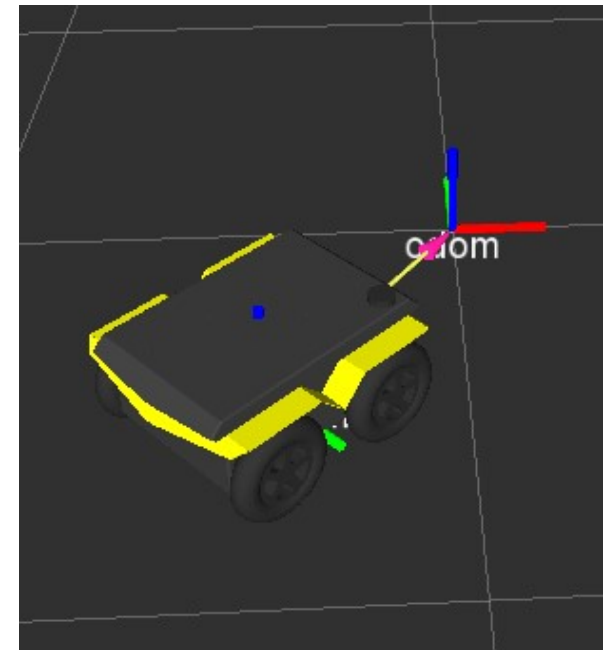
**Calculation:** The number of encoder ticks is used to calculate the angular displacement of the wheels.



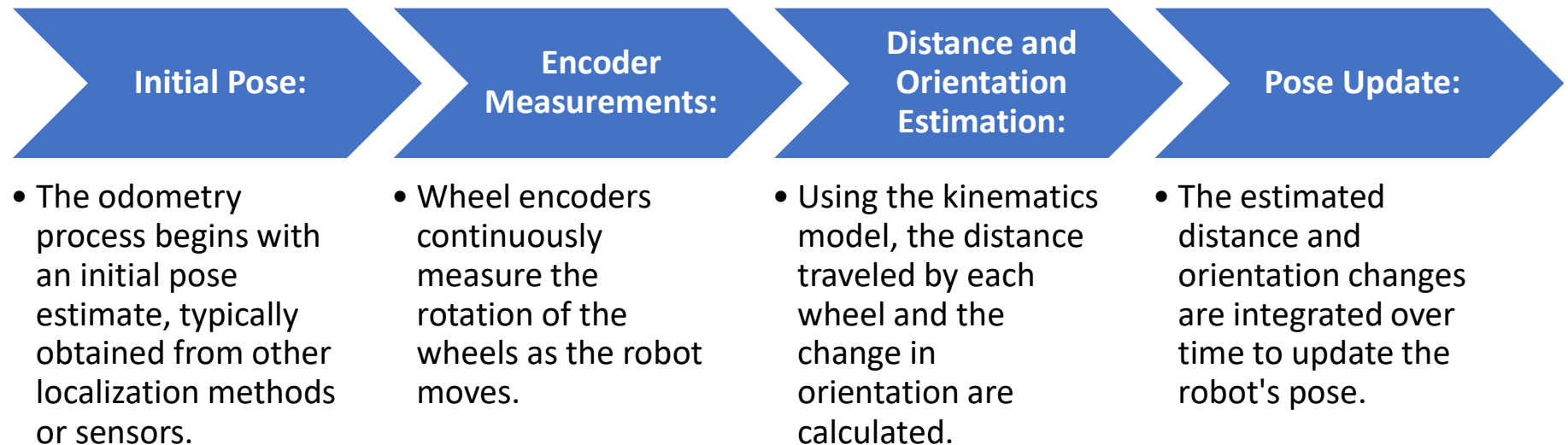
## Kinematics Model:

**Function:** A kinematics model describes how the robot's wheels or joints move relative to its pose (position and orientation).

**Calculation:** Combining encoder data with the kinematics model helps estimate the robot's displacement and change in orientation.



# Steps



# Accuracy



## **Wheel Slip:**

Slip between the wheels and the ground can lead to inaccuracies in distance estimation.



## **Encoders Precision:**

The precision of wheel encoders affects the overall accuracy of odometry. Low-resolution encoders may result in less accurate estimates.



## **Nonlinearities:**

Nonlinearities in wheel movements, such as skidding or variations in terrain, can introduce errors.



## **Integration of Angular and Linear Motion**

Any errors in either Angular or Linear measurement can propagate and result in inaccuracies in the estimated pose.



## **Environmental Changes:**

Changes in the environment, such as variations in friction, can impact the accuracy of odometry



# Accurate localization requirements

- **Sensor Fusion:** Combining data from multiple sensors improves accuracy and robustness. Techniques like Kalman filtering or particle filtering are commonly used for sensor fusion.
- **Redundancy:** Having redundant sensors can enhance reliability, as one sensor may compensate for the limitations or failures of another.
- **Calibration:** Proper calibration of sensors is essential to ensure accurate measurements and correct sensor readings.
- **Computational Efficiency:** Real-time processing of sensor data requires efficient algorithms and computing resources.
- **Dynamic Environments:** Sensors must handle dynamic changes in the environment, such as moving obstacles or changes in lighting conditions.





# Localization algorithm examples

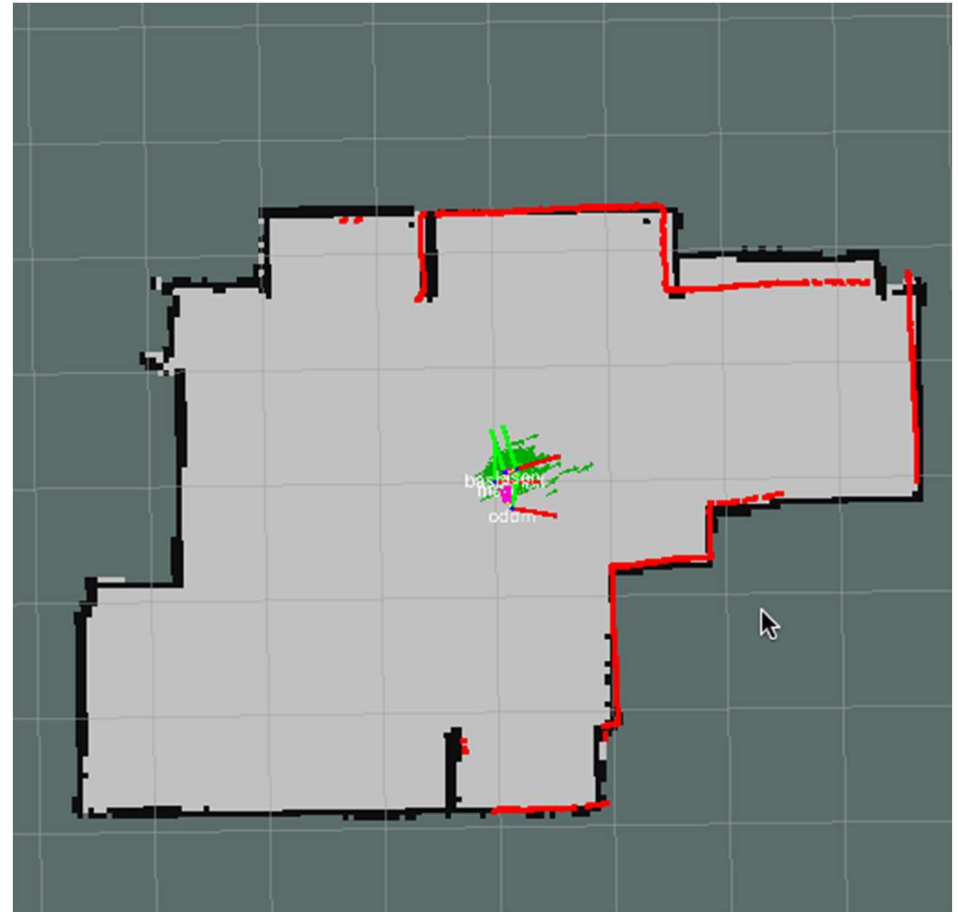
A large orange rounded rectangle with a white wavy line across its center.

AMCL

# AMCL Definition

---

AMCL is a probabilistic localization algorithm that falls under the category of Monte Carlo Localization (MCL). It is widely used in robotics, particularly in the context of mobile robots and autonomous systems.



# Probabilistic-based localization

**Particle Filter:** AMCL employs a particle filter, which is a probabilistic algorithm. The robot's pose is represented by a set of particles, each with an associated weight that represents the probability of that particle being the true pose.

**Probabilistic Sensor Model:** AMCL uses a probabilistic sensor model to evaluate how well each particle aligns with sensor measurements. This helps in assigning appropriate weights during the update process.

# Key Elements

## Monte Carlo Localization (MCL):

- MCL is a probabilistic method for estimating the pose (position and orientation) of a robot within an environment. It uses a particle filter to represent and propagate the belief distribution of the robot's pose.

## Particle Filter:

- In AMCL, the robot's pose is represented by a set of particles (samples) in the configuration space. Each particle has a weight that reflects its likelihood of being the true pose.

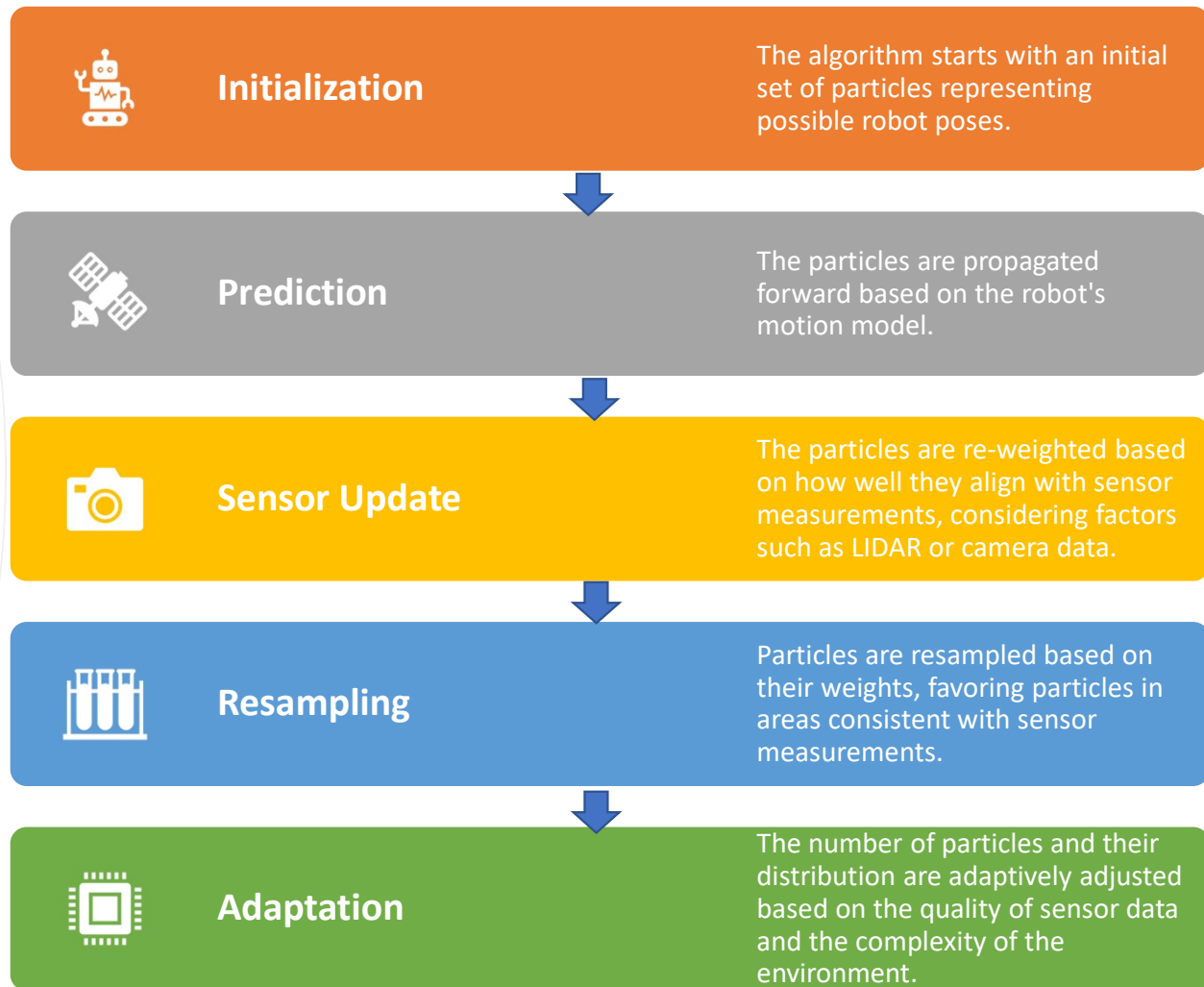
## Adaptation and Resampling:

- AMCL adapts the number and distribution of particles based on the robot's sensor measurements. Particles in areas consistent with sensor data receive higher weights, and particles in less likely areas are replaced during resampling.

## Sensor Model:

- AMCL uses a sensor model to evaluate how well each particle aligns with sensor measurements. This helps in assigning appropriate weights to particles during the update process.

# Flow





# AMCL

## Applications:

AMCL is commonly used in various robotic applications, including mobile robot navigation, localization in unknown environments, and scenarios where a robot needs to determine its position accurately based on sensor feedback.

## Challenges:


- AMCL may face challenges in situations with significant environmental changes or dynamic obstacles, as its performance depends on the consistency of sensor measurements.
- **Implementation:** AMCL is often implemented using robotic frameworks such as ROS (Robot Operating System), where it can be integrated with various sensors and control systems to enable accurate and adaptive localization.

A large orange circle with a thin white border. The word "SLAM" is written in white, uppercase letters in the center. Below the text is a thin, slightly wavy white horizontal line.

SLAM



# SLAM Definition



SLAM stands for Simultaneous Localization and Mapping. It's a technique used in robotics and computer vision to create a map of an environment while simultaneously keeping track of the robot's location within that environment. The goal of SLAM is for a robot or an autonomous system to navigate and understand an unknown environment in real-time.

# Usage



---

SLAM is crucial for various applications, including autonomous vehicles, drones, and robotic systems that need to operate in unknown or dynamic environments. It enables these systems to navigate, avoid obstacles, and make informed decisions based on their understanding of the surrounding space. Implementing SLAM involves complex algorithms and sensor fusion techniques to ensure accurate localization and mapping in real-world scenarios.

# SLAM



**LOCALIZATION**



**MAPPING**

# Probabilistic-based localization

## **Extended Kalman Filters (EKF) or Particle Filters:**

SLAM often utilizes probabilistic methods like Extended Kalman Filters or particle filters to manage the uncertainties in the robot's pose and the environment map.



**Probabilistic Motion Model:** The motion model in SLAM incorporates probabilistic elements to predict the likely distribution of the robot's pose based on its previous pose and motion commands.

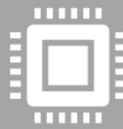


**Probabilistic Sensor Model:** Similar to AMCL, SLAM uses a probabilistic sensor model to evaluate the likelihood of observing specific sensor measurements given the robot's pose.

# Localization



The process of determining the robot's position and orientation within an environment.



This is usually done using sensors like cameras, lidar, or odometry (measuring wheel rotations).



Common localization methods include particle filters, extended Kalman filters, or graph-based approaches.



# Robotic Mapping

Robot mapping is a fundamental aspect of robotics that involves creating a representation of a robot's environment, allowing it to navigate and interact effectively. The goal is to generate a map that the robot can use for localization, path planning, and decision-making. There are various mapping techniques and technologies used in robotics, and they can be broadly categorized into two main types: **Static Mapping** and **Simultaneous Localization and Mapping (SLAM)**.

---

# Static Mapping



## Purpose

**Static mapping** is used when the environment is known and static. In these scenarios, the goal is to create a map of the environment beforehand, and the map is assumed to remain unchanged during the robot's operation.



## Techniques

**Manual Mapping:** Humans manually create a map of the environment using measurements or blueprints.

**Offline Lidar/Depth Sensor Scanning:** Pre-recorded data from Lidar sensors or depth cameras is used to generate a map.



## Applications

Commonly used in environments with known layouts, such as buildings, warehouses, or structured indoor spaces.

# Mapping

---



The creation of a map of the environment that the robot is navigating.



Information from sensors, such as lidar scans or camera images, is used to build a representation of the surroundings.



The map can be 2D or 3D and may include details about obstacles, landmarks, or other relevant features.



# Mapping methods

---



## **Occupancy Grid Mapping:**

Represents the environment as a grid where each cell is labeled as occupied, free, or unknown.

Suitable for 2D mapping.



## **Feature-Based Mapping:**

Identifies and maps distinct features in the environment, such as corners or keypoints.

Often used in both 2D and 3D mapping.



## **EKF-SLAM (Extended Kalman Filter SLAM):**

Utilizes the Extended Kalman Filter for both localization and mapping.

Maintains a probabilistic representation of the robot's pose and the environment.



## **Graph-Based SLAM:**

Represents the environment as a graph where nodes correspond to robot poses and edges represent constraints between poses.

Popular variants include Pose Graph SLAM and Bundle Adjustment.

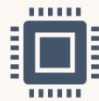
# Mapping methods



## **FastSLAM (Fast Simultaneous Localization and Mapping):**

Applies particle filters for the estimation of the robot's pose and a separate map for the environment.

Particularly efficient for non-linear and high-dimensional problems.



## **Direct SLAM:**

Models the mapping process directly from raw sensor measurements (e.g., pixel intensities) without feature extraction.

Commonly used in visual SLAM.



## **Topological Mapping:**

Represents the environment in terms of topological features and relationships.

Focuses on connectivity and spatial relationships rather than geometric details.



## **Semantic Mapping:**

Integrates semantic information into the map, identifying and labeling objects or regions.

Adds a semantic layer to the traditional geometric map.



## **RGB-D SLAM:**

Utilizes RGB-D cameras (providing both color and depth information) for mapping.

Enables the creation of 3D maps with richer information.

# Common SLAM Algorithm

## EKF-SLAM (Extended Kalman Filter SLAM):

- Utilizes the Extended Kalman Filter for both localization and mapping.
- Assumes Gaussian noise and is suitable for systems with relatively small uncertainties.

## FastSLAM:

- Represents a family of algorithms that use particle filters for SLAM.
- Efficiently handles non-linearities and can scale well for large-scale environments.

## GraphSLAM:

- Represents the environment as a graph, where nodes correspond to robot poses and edges represent constraints.
- Popular variants include Pose Graph SLAM and Bundle Adjustment.

## GMapping:

- Grid-based FastSLAM algorithm.
- Uses particle filters and occupancy grid mapping for mapping in 2D environments.

## ORB-SLAM:

- Visual SLAM algorithm that uses feature points extracted from camera images.
- Well-suited for monocular or stereo camera setups.

## LSD-SLAM (Large-Scale Direct SLAM):

- Direct SLAM approach that works with depth maps obtained from RGB-D cameras.
- Focuses on real-time performance and large-scale mapping.

## LIO-SAM (Lidar-Inertial Odometry and Mapping):

- Integrates lidar and inertial measurements for SLAM.
- Particularly useful for mapping in dynamic environments.

## Hector SLAM:

- Designed for mapping with 2D laser scanners (LIDAR).
- Employs scan matching techniques for real-time mapping.

## RTAB-Map (Real-Time Appearance-Based Mapping):

- Visual SLAM algorithm that considers appearance-based features.
- Suitable for environments with loop closures and dynamic objects.

## Cartographer:

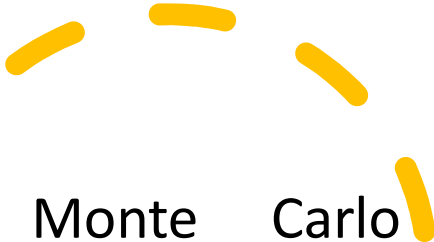
- Developed by Google, Cartographer is a 2D and 3D SLAM system.
- Supports various sensors, including LIDAR and IMU, for mapping and localization.

## Karto SLAM

- Laser-based SLAM
- Real-time Mapping
- Loop Closure Detection



## AMCL vs SLAM



Both AMCL (Adaptive Monte Carlo Localization) and SLAM (Simultaneous Localization and Mapping) are probabilistic-based localization algorithms. They leverage probabilistic methods to represent and manage uncertainty in the estimation of a robot's pose (position and orientation) within its environment.

## AMCL vs SLAM

Aspect	AMCL (Adaptive Monte Carlo Localization)	SLAM (Simultaneous Localization and Mapping)
<b>Purpose</b>	Localizing a robot within a known map.	Simultaneously creating a map of an unknown environment and localizing the robot within it.
<b>Input</b>	Requires a pre-existing map of the environment.	Starts with no prior knowledge of the environment and constructs a map on-the-fly.
<b>Workflow</b>	Utilizes a particle filter to adaptively adjust the number and distribution of particles based on sensor data.	Combines sensor measurements with the robot's motion to incrementally build a map and estimate the robot's pose. Often uses probabilistic methods like Extended Kalman Filters or particle filters.
<b>Applications</b>	Suitable for navigation in environments with a predefined map.	Ideal for exploration and mapping missions in unknown or partially known environments. Commonly used in scenarios where the environment is dynamic or changes over time.

# GMapping

GMapping is a popular algorithm for Simultaneous Localization and Mapping (SLAM) in robotics. It falls under the category of FastSLAM, a family of algorithms designed to efficiently solve the SLAM problem. GMapping specifically focuses on creating a grid-based map of the environment while simultaneously localizing a robot within that map. This allows a robot to navigate and explore unknown environments autonomously.

# Key Components



## **Grid-based Mapping:**

GMapping uses an occupancy grid map to represent the environment. The map is discretized into cells, and each cell is labeled as occupied, free, or unknown based on sensor measurements.



## **Particle Filter Localization:**

GMapping employs a particle filter for localization. This involves maintaining a set of particles, each representing a potential pose of the robot. The particles are updated based on sensor measurements, helping to estimate the robot's position.



## **Scan Matching:**

GMapping uses scan matching techniques to align the robot's sensor readings with the map. This helps in refining the estimate of the robot's pose and updating the occupancy grid map.



## **Probabilistic Mapping:**

The algorithm uses probabilistic models to handle uncertainty in the environment and robot pose. This is crucial for creating reliable maps in the presence of sensor noise and dynamic elements.

# Workflow



## Initialization:

GMapping starts with an initial estimate of the robot's pose and an empty occupancy grid map.



## Sensor Readings:

The robot takes sensor readings, typically from a 2D laser range finder (LIDAR), capturing information about the surrounding environment.



## Particle Filter Update:

The particle filter is updated based on the sensor readings. Particles with poses consistent with the sensor data are assigned higher weights.



## Scan Matching:

Scan matching aligns the sensor data with the map, improving the accuracy of the robot's pose estimate.



## Map Update:

The occupancy grid map is updated based on the aligned sensor readings, marking areas as occupied, free, or unknown.



## Loop Closure:

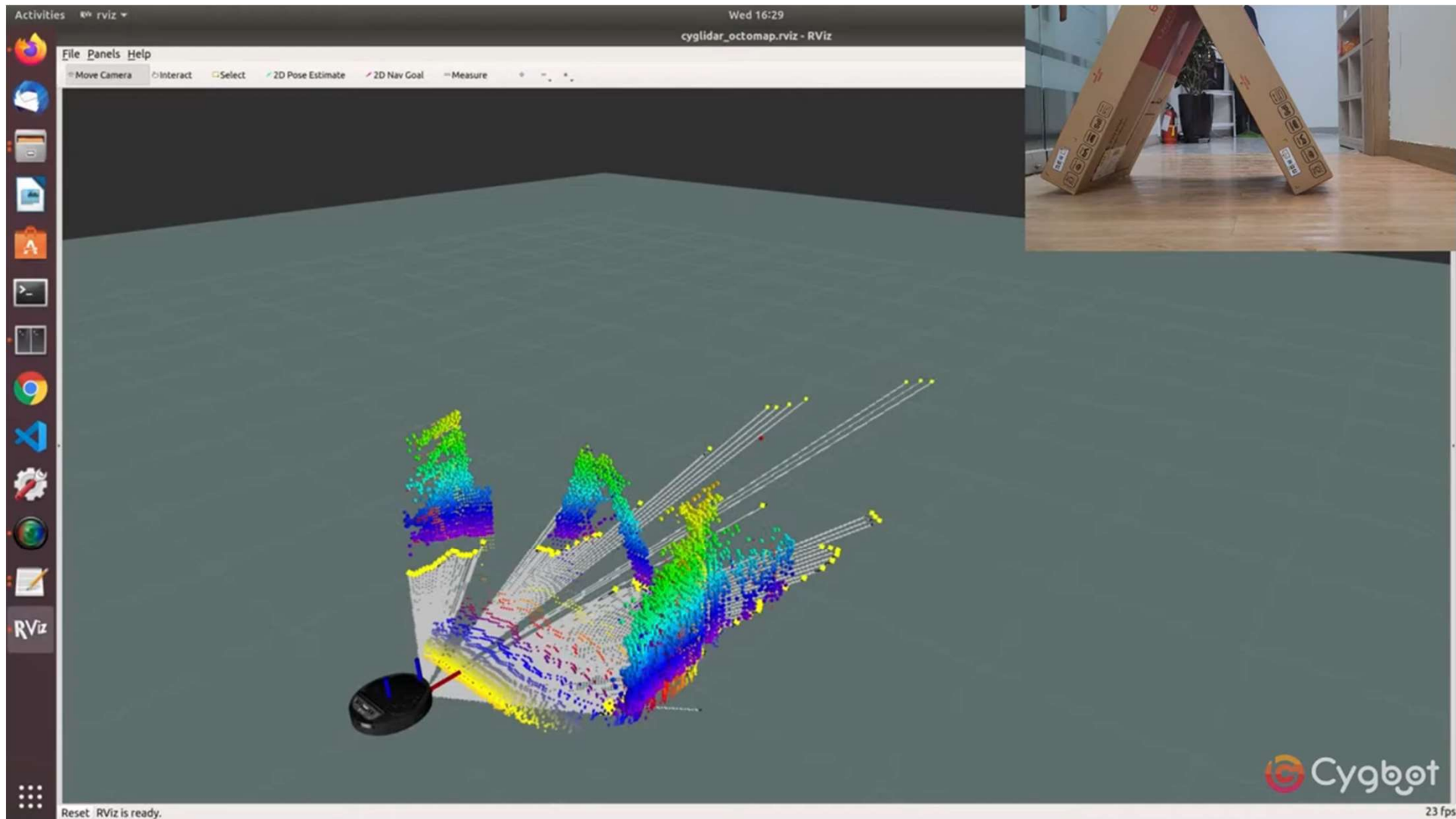
GMapping handles loop closures, situations where the robot revisits a previously visited location. This helps correct accumulated errors in the map.



# Usage

---

- GMapping is widely used in robotics applications for mapping environments. It is particularly suitable for scenarios where a robot equipped with LIDAR explores and maps unknown environments, such as in robotic exploration, autonomous navigation, and mobile robotics.
- However:
- GMapping's performance may degrade in dynamic environments with moving obstacles.
- Proper tuning of parameters is crucial for balancing accuracy and computational efficiency.



# Cartographer Mapping

- Cartographer is an open-source, real-time, and high-performance SLAM (Simultaneous Localization and Mapping) library developed by Google. It is designed to work with a variety of sensor configurations, including 2D and 3D LIDAR, IMU (Inertial Measurement Unit), and odometry sensors. Cartographer is often used in robotics applications for mapping environments and localizing robots within those maps.

# Key features

## Multi-Sensor Support:

- Cartographer supports multiple sensors, including LIDAR, IMU, and odometry. The fusion of data from these sensors helps improve the accuracy of the generated maps.

## Flexible Configurations:

- Users can configure Cartographer for different hardware setups and environmental conditions. This flexibility makes it suitable for a wide range of robotic platforms.

## Real-Time Operation:

- Cartographer is designed to operate in real-time, making it suitable for applications where quick mapping updates are essential for navigation.

## Loop Closure Detection:

- The algorithm includes mechanisms for loop closure detection, which helps correct accumulated errors and improves the overall accuracy of the map.

## ROS Integration:

- Cartographer is integrated with the Robot Operating System (ROS), a popular middleware in robotics. This integration simplifies the process of incorporating Cartographer into robotic systems that use ROS for communication and control.

# Key features

## 2D and 3D Mapping:

- Cartographer can be used for both 2D and 3D mapping. This flexibility allows users to choose the appropriate mapping dimension based on their specific requirements.

## Applications:

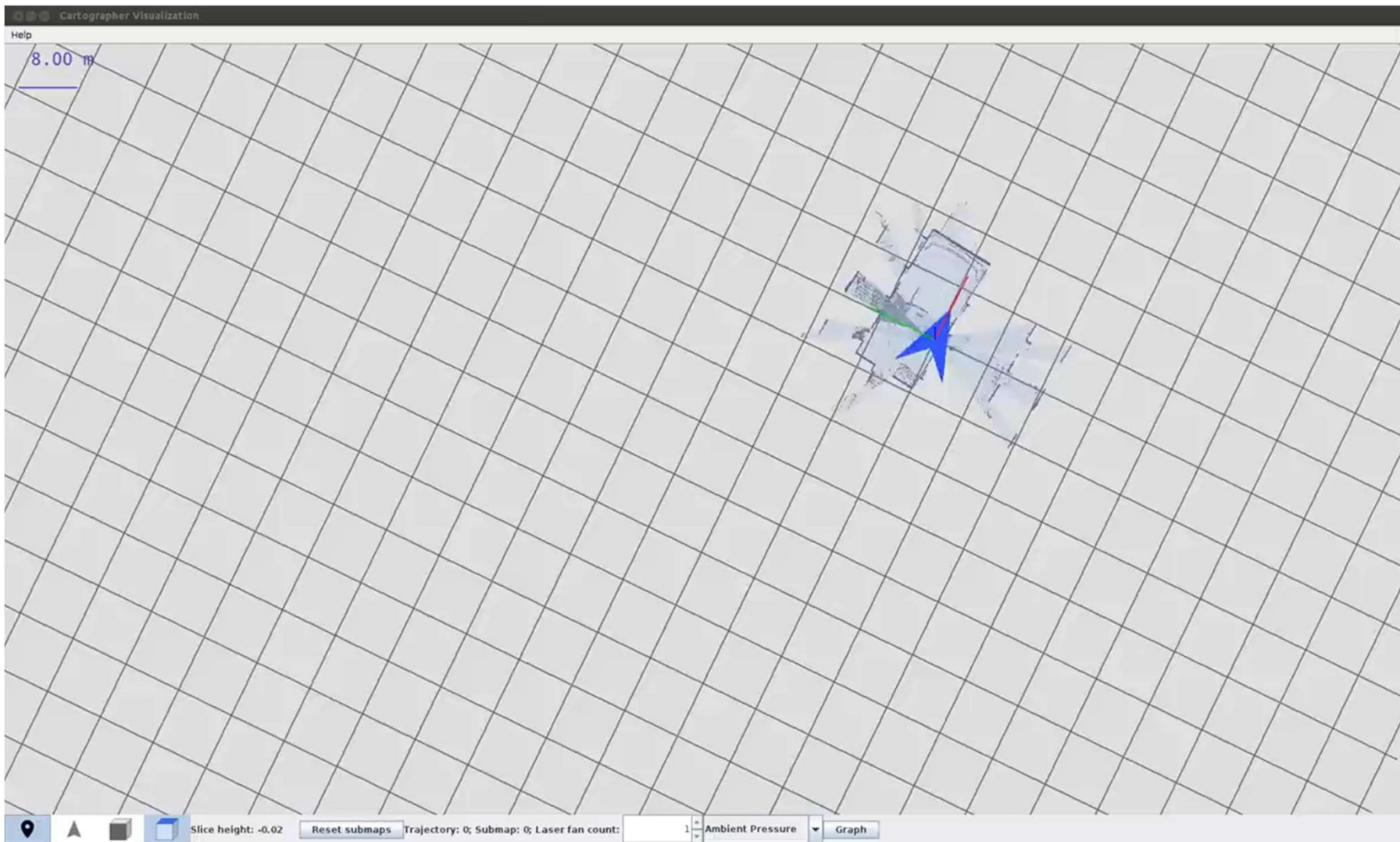
- Cartographer is used in a variety of robotic applications, including autonomous vehicles, drones, and mobile robots navigating in dynamic environments.

## Global and Local SLAM:

- Cartographer supports both global SLAM, which builds maps of the entire environment, and local SLAM, which focuses on mapping the immediate surroundings of the robot.

## Offline Processing:

- While Cartographer operates in real-time, it also supports offline processing of recorded sensor data for map generation and analysis.



An orange circle with a thin white border. The text "OMPL" is centered in white, with a white horizontal line underneath it.

OMPL

# OMPL Definition

---

- OMPL typically refers to the Open Motion Planning Library. OMPL is an open-source software package that provides a collection of sampling-based motion planning algorithms. Motion planning involves the computation of a path for a robot or other autonomous system to navigate from an initial state to a goal state while avoiding obstacles.
- OMPL is widely used in robotics research and applications. It supports a variety of motion planning problems, including single-query and multi-query problems, as well as problems with differential constraints. The library is designed to be easily extensible and customizable, making it a valuable tool for researchers and developers working on motion planning for robots and autonomous systems.





# Motion Planners

- The Open Motion Planning Library (OMPL) provides various motion planning algorithms or planners.
- These planners offer a variety of approaches to address different types of motion planning problems. When selecting a planner, it's important to consider the specific characteristics of the problem, such as dimensionality, constraints, and the desired level of optimality.



# Motion Planners

<b>Probabilistic Roadmap (PRM):</b>	PRM is a randomized planner that constructs a roadmap of the configuration space. It is suitable for problems with complex spaces and obstacles.
<b>Rapidly Exploring Random Tree (RRT):</b>	RRT is a tree-based planner that efficiently explores the configuration space by growing a tree of feasible paths. It is particularly effective for problems with high-dimensional spaces.
<b><i>RRT (Rapidly Exploring Random Tree Star):*</i></b>	RRT* is an extension of RRT that aims to find optimal paths in the configuration space. It optimizes the paths iteratively for improved efficiency.
<b><i>BIT (Batch Informed Trees):*</i></b>	BIT* is a sampling-based planner that constructs a tree incrementally. It is designed to handle problems with differential constraints and provides asymptotic optimality.
<b>LazyPRM:</b>	LazyPRM is a variant of PRM that defers the collision checking until a path is actually needed. This lazy approach can be more efficient for certain scenarios.
<b>EST (Expansive Space Trees):</b>	EST is a tree-based planner that incrementally expands the tree to explore the configuration space. It is suitable for problems with narrow passages and high-dimensional spaces.
<b>KPIECE1 (Kinematic Planning by Interior-Exterior Cell Exploration):</b>	KPIECE1 is a geometric planner that uses an adaptive grid to explore the configuration space. It is well-suited for problems with multiple narrow passages.
<b>SBL (Single-Query Bi-directional Lazy Planner):</b>	SBL is a single-query, bi-directional planner that explores the configuration space from both the start and goal simultaneously. It is efficient for solving complex problems.
<b>LBKPIECE1 (Lattice-Based KPIECE1):</b>	LBKPIECE1 is an extension of KPIECE1 that uses a lattice-based sampling strategy. It can be effective in problems with discrete configurations.
<b>PDST (Path-Defined Sampling Tree):</b>	PDST is a geometric planner that focuses on path-defined sampling. It incrementally builds a tree to explore the configuration space.

# Selection

Consideration	Example Planners	Remarks
Problem Requirements	Autonomous Delivery Robot in Urban dynamic Environments require balancing speed and optimality, consider planners such as RRT	Align the planner with specific requirements such as obstacle density, dynamic changes, etc.
Nature of the Environment	PRM is effective in static environments, while RRT variants are suitable for dynamic and changing environments.	Select based on whether the environment is static or dynamic.
Completeness and Optimality	PRM is probabilistically complete, providing a solution with high probability, while RRT* aims for optimality but may require more computational resources.	Decide on the trade-off based on the need for probabilistic completeness or optimality.
Computational Resources	RRT*, optimization-based planners consume more resources	Consider the available computational resources and choose a planner accordingly.
Constraint Handling	BIT* may be more suitable for For problems with differential constraints or kinodynamic constraints	Choose based on how well the planner handles specific constraints like differential constraints.
Dimensionality Considerations	High-dimensional spaces may benefit from tree-based planners like RRT or PRM. For problems with discrete configurations, planners like LBKPIECE1 might be more appropriate.	Select based on the dimensionality of the configuration space.
Iterative Approach	Adjust planners and parameters based on observations and feedback.	Adopt an iterative approach to planner selection.

# Motion Planners

## Global Planner:

- Planning a path from the robot's initial position to the goal position in the entire environment.
- Often used for longer-distance navigation.

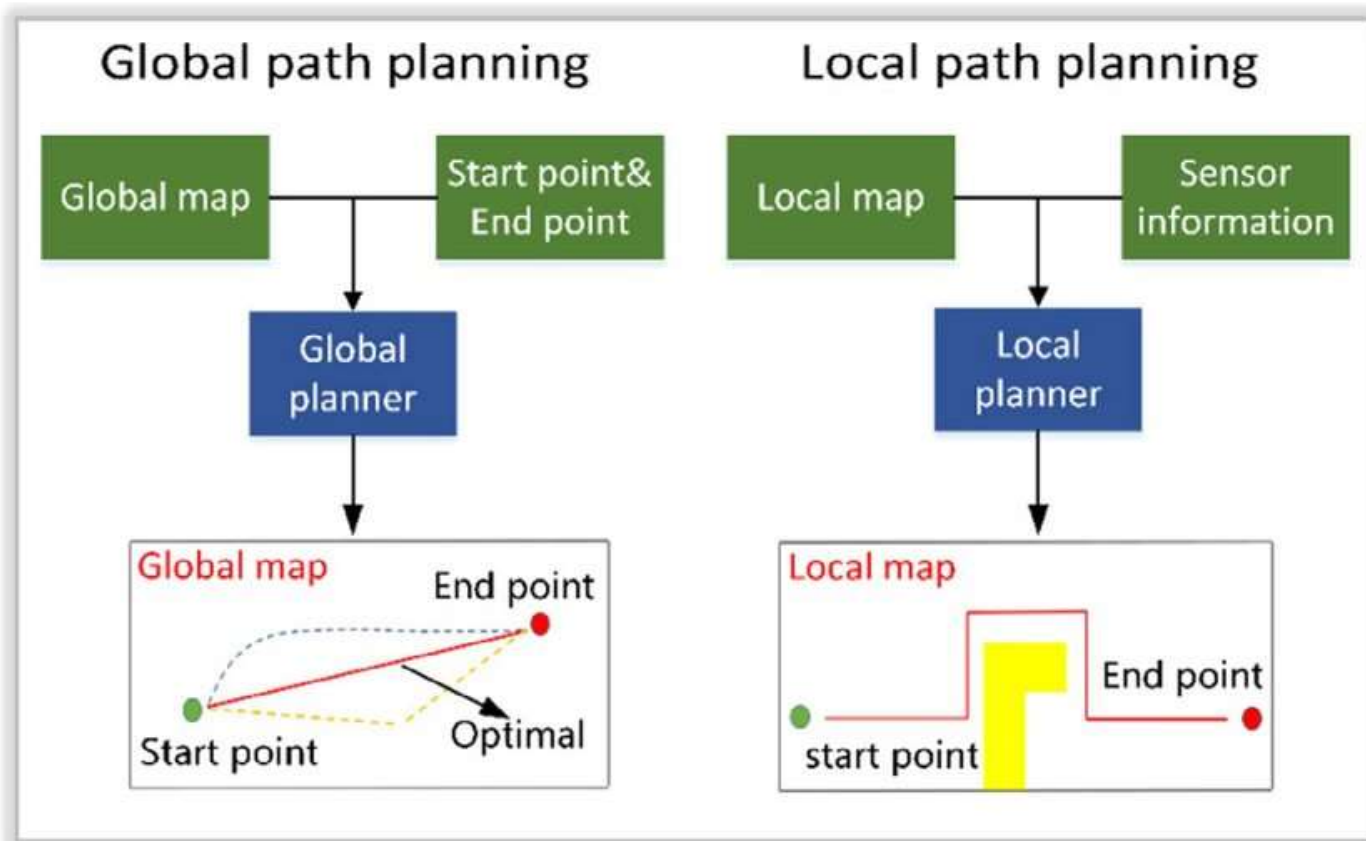
## Local Planner:

- Short-term adjustments to navigate around immediate obstacles.
- Reactive responses to dynamic changes in the environment.

## Dynamic Planner:

- Handling dynamic obstacles and adapting the planned path in real-time.
- Requires continuous sensor monitoring and quick decision-making.

# Global vs Local





# Standalone library

- <https://github.com/ompl>
  - <https://ompl.kavrakilab.org/download.html>
  - If you use ROS, the recommended way to use OMPL is through MoveIt.
-