# Robotics
*Theory*

Christian Rossi

Academic Year 2023-2024

## Abstract

The course is composed by a set of lectures on autonomous robotics, ranging from the main architectural patterns in mobile robots and autonomous vehicles to the description of sensing and planning algorithms for autonomous navigation. The course outline is:

- Mobile robots' kinematics.
- Sensors and perception.
- Robot localization and map building.
- Simultaneous Localization and Mapping (SLAM).
- Path planning and collision avoidance.
- Robot development via ROS.

# Contents

**1 Introduction** **1**

1.1 History . . . 1

1.2 ISO definitions . . . 2

1.3 Robot architecture . . . 3

**2 Sensors and actuators** **4**

2.1 Sensors . . . 4

2.1.1 Encoder . . . 4

2.1.2 Time of flight telemeter . . . 6

2.1.3 Reflective optosensors . . . 7

2.1.4 Light detection and ranging (LIDAR) . . . 8

2.1.5 Position sensor . . . 8

2.2 Inertial sensor . . . 9

2.2.1 Tactile sensor . . . 9

2.2.2 Proximity sensor . . . 9

2.3 Actuators . . . 10

2.3.1 Direct current motor . . . 10

2.3.2 Stepper motor . . . 11

2.3.3 Servo motor . . . 12

**3 Robot Odometry** **13**

3.1 Introduction . . . 13

3.2 Direct kinematic . . . 14

3.3 Differential drive . . . 16

3.3.1 Odometry . . . 17

3.3.2 Sensors . . . 19

3.4 Synchronous drive . . . 19

3.4.1 Odometry . . . 20

3.5 Omnidirectional drive . . . 20

3.6 Tricycle drive . . . 21

3.6.1 Odometry . . . 22

3.7 Ackerman steering drive . . . 22

3.8 Skid steering drive . . . 23

3.9 Inverse kinematics . . . 24

3.9.1 Differential drive . . . 24

3.9.2 Synchro drive . . . 24

**4 Robot localization** **25**
4.1 Introduction . . . 25
4.2 Beam sensor model . . . 25
4.2.1 Calibration . . . 27
4.2.2 Model likelihood . . . 28
4.3 Scan sensor model . . . 28
4.4 Landmark model . . . 29
4.4.1 Landmark detection . . . 29
4.5 Bayesian filter . . . 30
4.5.1 Bayes filter . . . 31
4.5.2 Bayes filter algorithm . . . 31
4.5.3 Discrete Bayes filter algorithm . . . 32
4.6 Kalman filter . . . 33
4.6.1 Discrete time Kalman filter . . . 34
4.6.2 Extended Kalman filter . . . 36
4.7 Particle filter . . . 37

**5 Simultaneous Localization and Mapping** **39**
5.1 Introduction . . . 39
5.1.1 Occupancy from sonar return . . . 39
5.1.2 Two-dimensional occupancy grid . . . 39
5.2 Mapping . . . 40
5.3 Scan matching . . . 40
5.4 Simultaneous Localization and Mapping . . . 41
5.4.1 Online SLAM . . . 41
5.4.2 Bayes filter algorithm for EKF SLAM . . . 42
5.4.3 Kalman filter algorithm for EKF SLAM . . . 42
5.4.4 Summary . . . 43
5.5 Full SLAM . . . 43
5.5.1 FastSLAM . . . 44
5.6 PoseGraph SLAM . . . 44
5.6.1 Computational complexity . . . 46

**6 Robot motion control and planning** **48**
6.1 Introduction . . . 48
6.1.1 Open loop control . . . 48
6.1.2 Feedback control . . . 48
6.2 Local path planning . . . 49
6.2.1 Vector Field Histograms . . . 49
6.2.2 Curvature Velocity Methods . . . 50
6.2.3 Vector Field Histogram Plus . . . 50
6.2.4 Dynamic Window Approach . . . 51
6.3 Global path planning . . . 52
6.3.1 Motion planning . . . 53
6.4 Search-based methods . . . 54
6.4.1 Summary . . . 57
6.4.2 State-lattice planning . . . 58
6.5 Sampling-based methods . . . 58
6.5.1 Probabilistic roadmap algorithm . . . 58

6.5.2 Rapidly Exploring Dense Trees . . . . . 59
6.5.3 Rapidly Exploring Random Trees . . . . . 60

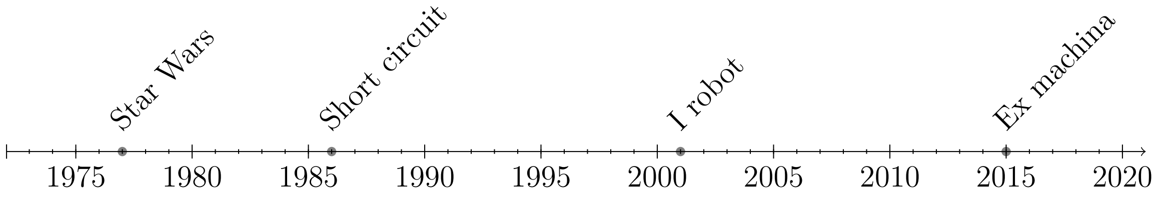
**7 Robot Operating System 62**
7.1 Introduction . . . . . 62
7.1.1 Robot Operating System . . . . . 63
7.2 Computational graph . . . . . 64
7.2.1 Nodes . . . . . 65
7.2.2 Master . . . . . 65
7.2.3 Topics . . . . . 65
7.2.4 Messages . . . . . 66
7.2.5 Services . . . . . 66
7.2.6 Parameter server . . . . . 66
7.2.7 Bags . . . . . 66
7.2.8 ROScore . . . . . 66
7.3 ROS file system . . . . . 66
7.3.1 Packages . . . . . 67
7.3.2 Metapackages . . . . . 67

CHAPTER 1

# Introduction

## 1.1 History

**Filmography** In the play "Rossum Universal Robots" from 1920, the term "robota" was introduced to refer to the first automatic robots. Several years later, Isaac Asimov penned the renowned science fiction series "I, Robot". Additionally, notable instances of robots in film include:



**Robots evolution** The mechanical era commenced in 1700 with the advent of the first automata, initially devised as specialized dolls for specific purposes. Transitioning from this era, the dawn of the 1920s saw a resurgence of interest in universal-purpose robots within the realm of fiction.

By 1940, the cybernetics era took root with the creation of the first turtles and telerobots. Grey Walters pioneered a significant development in this era by crafting a robotic tortoise that exhibited mechanical animal tropism (movement directed by stimuli).

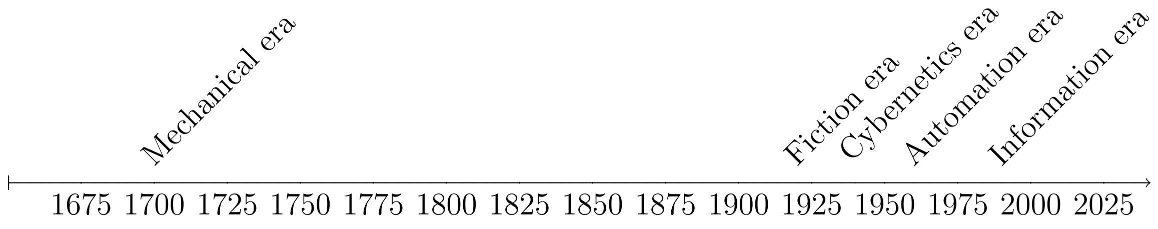
Two decades later, the automation era commenced with the inception of the first industrial robots, marking a shift towards mechanized processes. In 1961, UNIMATE, the inaugural industrial robot, initiated operations at General Motors, executing programmed tasks with precision and efficiency. In 1968, Marvin Minsky introduced the Tentacle Arm, a groundbreaking innovation resembling the movements of an octopus. This hydraulic-powered arm, controlled by a PDP-6 computer, featured twelve flexible joints facilitating maneuverability around obstacles.

In 1972, Shakey pioneered mobility in robotics with the creation of the Stanford cart, heralding advancements in mobile robotics.

The year 1980 witnessed the establishment of the first comprehensive definition of a robot as a reprogrammable, multifunctional manipulator designed for diverse tasks involving material, parts, tools, or specialized devices.

The onset of the information era in 1990 saw robots evolving to possess autonomy, cooperation, and intelligence, marking a significant leap in their capabilities.

Finally, in 2012, the International Organization for Standardization (ISO) established the standard definition for robots, consolidating their diverse functionalities and characteristics into a unified framework.



## 1.2 ISO definitions

**Definition** (*Robot*)**.** A robot is an actuated mechanism programmable in two or more axes with a degree of autonomy, moving within its environment, to perform intended tasks. Autonomy in this context means the ability to perform intended tasks based on current state and sensing, without human intervention.

**Definition** (*Service robot*)**.** A service robot is a robot that performs useful tasks for humans or equipment excluding industrial automation application.

**Definition** (*Personal service robot*)**.** A personal service robot or a service robot for personal use is a service robot used for a noncommercial task, usually by lay persons.

Examples of personal service robots include domestic servant robots, automated wheelchairs, personal mobility assist robots, and pet exercising robots.

**Definition** (*Professional service robot*)**.** A professional service robot or a service robot for professional use is a service robot used for a commercial task, usually operated by a properly trained operator.

Examples of professional service robots encompass cleaning robots for public spaces, delivery robots in offices or hospitals, fire-fighting robots, rehabilitation robots, and surgical robots in hospitals. In this context, an operator is an individual designated to initiate, oversee, and terminate the intended operation of a robot or robot system.

**Notes** A robot system is defined as a system comprising robots, end-effectors, and any machinery, equipment, or sensors that support the robot in performing its tasks.

According to this definition, service robots require a degree of autonomy, which can range from partial autonomy involving human-robot interaction to full autonomy without active human intervention. Human-robot interaction involves information and action exchanges between humans and robots via a user interface to accomplish tasks.

Industrial robots, whether fixed or mobile, can also be considered service robots if they are utilized in non-manufacturing operations. Service robots may or may not feature an arm structure, which is common in industrial robots. Additionally, service robots are often mobile, but this is not always the case.

Some service robots consist of a mobile platform with one or several arms attached, controlled similarly to industrial robot arms. Unlike their industrial counterparts, service robots do not necessarily need to be fully automatic or autonomous. Many of these machines may assist a human user or operate via teleoperation.

## 1.3 Robot architecture

A machine gathers information from a set of sensors and utilizes this data to autonomously execute tasks by controlling its body parts.

One commonly employed model in robotics is the sense plan act paradigm, which forms the foundation of cognitive robotics. In this model, the sensing phase involves collecting data from sensors, the planning phase utilizes algorithms to process this data, and the action phase involves executing commands through actuators. This architecture is illustrated in the following diagram.

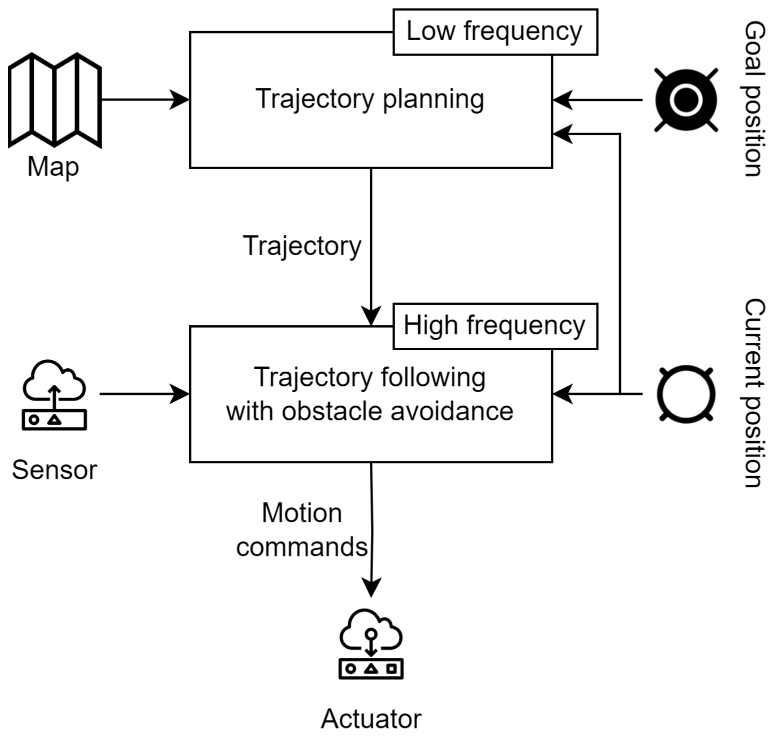


Figure 1.1: Sense plan act architecture

# CHAPTER 2

# Sensors and actuators

## 2.1 Sensors

Sensors serve to detect both the internal condition of the robot (proprioceptive sensors) and the external state of the environment (exteroceptive sensors). Another classification for sensors can be based on whether they are passive, which measure physical properties, or active, which involve an emitter and a detector.

### 2.1.1 Encoder

An encoder translates rotary motion or position of a motor/joint into electronic pulses. Encoders come in two primary types:

- *Linear encoder*: comprising a lengthy linear read track and a compact read head, linear encoders are designed for linear motion measurement.
- *Rotary encoder*: suitable for both rotary and linear motion, rotary encoders convert rotary motion into electrical signals. They are further categorized as incremental or absolute.

Their operation proceeds as follows:

1. A light-emitting diode (LED) projects a light beam onto a tape striped with red and black segments.
2. The reflected light is captured by a photodetector.
3. The photodetector generates a periodic wave whose frequency varies based on the speed of the tape.

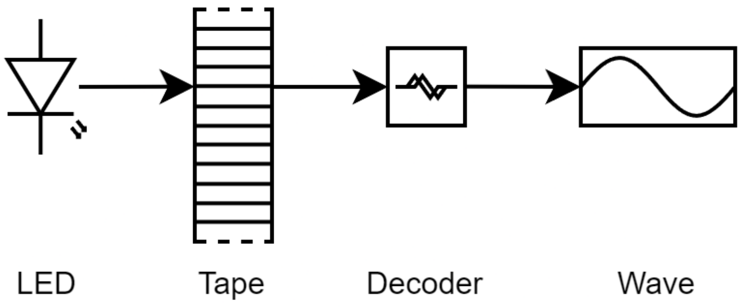


Figure 2.1: Linear encoder structure

**Incremental rotary encoders** Incremental rotary encoders operate based on the photoelectric principle, employing a disk with alternating transparent and opaque zones containing two traces or sensors. These traces facilitate the identification of rotation direction and enhance resolution through quadrature.

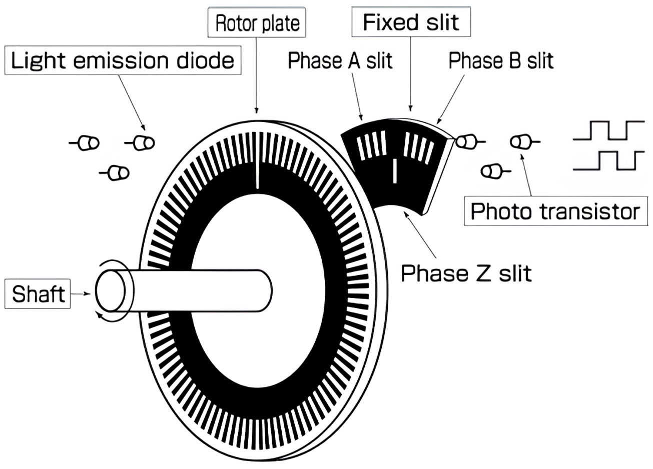


Figure 2.2: Incremental rotary encoder

To determine speed and direction, the quadrature technique is employed, where the two signals are shifted by $\frac{1}{4}$ step. Denoting $N$ as the number of steps of light/dark zones per turn, the resolution is given by:

$$\text{resolution} = \frac{360°}{4N}$$

Using this technique:

- If a transition from 11 to 10 occurs, it indicates a counterclockwise rotation.
- Conversely, if a transition from 11 to 01 occurs, it indicates a clockwise rotation.

The encoders' limitation lies in their inability to determine the actual position relative to the starting point. The only feasible solution involves resetting to the starting position and then incrementally counting until reaching the desired position.

**Absolute rotary encoder** The absolute rotary encoder addresses the limitation of determining absolute position by encoding it directly on the disk.

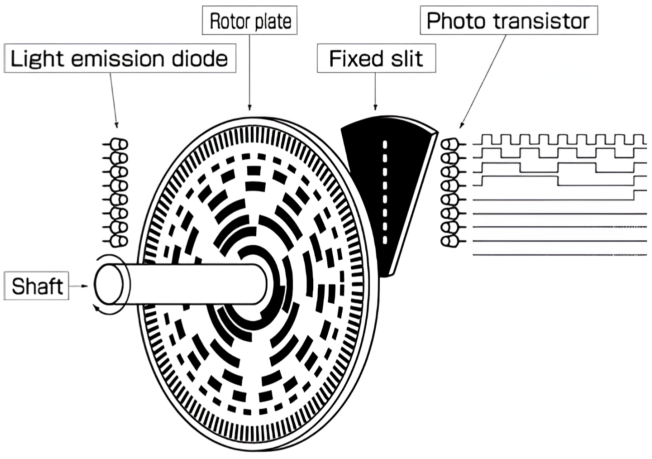


Figure 2.3: Absolute rotary encoder

The disk features transparent and opaque areas arranged in concentric rings. Each bit of position data is represented by a corresponding ring, offering an absolute resolution of:

$$\text{resolution} = \frac{360^\circ}{2^N}$$

In robotic applications, a minimum of 12 rings are typically employed. To prevent ambiguities, binary codes with single variations, such as Gray code, are utilized.

### 2.1.2 Time of flight telemeter

The time-of-flight telemeter records the duration between when the emitter generates a signal and when the detector detects its reflection. The signal travels a distance of $2d$, and the time of flight is given by:

$$\Delta t = \frac{2d}{c}$$

The initial type of sensor utilizing this principle in robotics was the sonar, which relies on sound waves. Sound waves, with their slower speed of approximately 340 $m/s$, and their relatively broad directionality ranging from 20° to 40°, offer an advantage in measuring shorter distances.

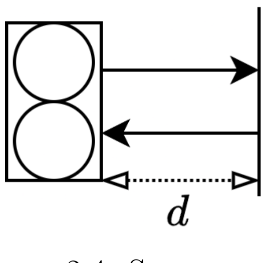


Figure 2.4: Sonar sensor

**Sonar**

| | |
|---|---|
| *Range (m)* | 0.3 up to 10 |
| *Accuracy (m)* | 0.025 |
| *Cone opening (°)* | 30 |
| *Frequency (Hz)* | 50000 |

The primary limitation is the susceptibility of the signal-to-noise, particularly from significant reflections. Selecting an appropriate range is crucial depending on the specific application. However, these sensors may not function optimally in all scenarios, due to factors such as:

- Balancing sampling frequency.
- Dealing with reflections off walls.
- Detecting small or soft objects.

Additionally, it's worth noting that room dimensions may appear distorted, especially around corners.

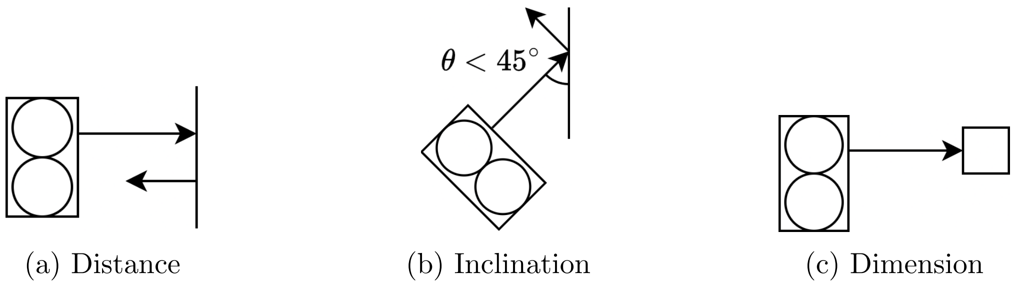


Figure 2.5: Possible problems for sonar sensors

### 2.1.3 Reflective optosensors

Reflective optosensors are active sensors where the emitter is a source of light and the detector is a light detector. This type of sensors uses triangulation to compute distance:

1. Emitter casts a beam of light on the surface.
2. The detector measures the angle corresponding to the maximum intensity of returned light.
3. Being $s$ the distance between the emitter and the detector we have:

$$d = s \cdot \tan \alpha$$

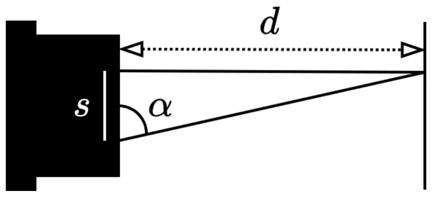


Figure 2.6: Reflective optosensor

Infrared sensors are relatively inexpensive and sturdy, but they have their drawbacks. They exhibit nonlinear characteristics that require calibration. Additionally, there can be ambiguity when used at short ranges, necessitating precise placement within the robot. Their fixed ranges and opening angles mean that careful selection is needed for optimal performance in various applications. Moreover, they may encounter issues with reflections under certain conditions.

An instance of such technology is the Kinect, an input device designed by Microsoft (originally by Primesense) for Xbox 360. This device functions as a three-dimensional scanner and is equipped with an infrared projector, an infrared camera, and an RGB camera.

**Kinect**

| | |
|---|---|
| *Range (m)* | 0.7 up to 6 |
| *Horizontal cone opening (°)* | 57 |
| *Vertical cone opening (°)* | 43 |
| *Infrared camera* | 11-bit 640×480 |
| *RGB camera* | 30 $Hz$ 8-bit 640×480 |

In this device, the distance from the camera $Z_k$ is calculated as:

$$Z_k = \frac{Z_0}{1 + \frac{d}{fb} Z_0}$$

**Time of flight camera** Three-dimensional time-of-flight (TOF) cameras illuminate the scene using a modulated light source and capture the reflected light. The phase shift between illumination and reflection is then translated into distance information.

These sensors encounter challenges such as utilizing illumination from a solid-state laser or a near-infrared ($\sim$ 850 $nm$) LED, where an imaging sensor converts captured light into electrical current. Additionally, distance information is encoded within the reflected component. Consequently, a high ambient component diminishes the signal-to-noise ratio (SNR).

### 2.1.4 Light detection and ranging (LIDAR)

Laser sensors offer superior accuracy with the following capabilities:

- Providing 180 ranges across a 180° field of view (expandable to 360°).
- Scanning 1 to 64 planes.
- Delivering scan rates of 10-75 scans per second.
- Achieving range resolutions of less than 1 $cm$.
- Offering a maximum range of up to 50-80 meters.
- Facing challenges only with mirrors, glass, and matte black surfaces.

### 2.1.5 Position sensor

Positioning outdoors can be determined using a Global Navigation Satellite System (GNSS), with multiple constellations available including GPS, GLONASS, Beidou, Galileo, and more.

The Global Positioning System (GPS) comprises 24 satellites circling the Earth twice daily. These satellites emit synchronized signals containing location and time data. Receivers compare the transmitted and received signal times to determine position. At least four satellite signals are needed for accurate positioning. The typical accuracy of GPS is approximately 2.5 $m$ at a 2 $Hz$ refresh rate, with the potential for even greater precision of around 20 $cm$ with Differential GPS (DGPS).

There is also the RTKGPS that improves the time resolution with respect to the DGPS. There are several limitations associated with GPS:

- It does not function indoors, underwater, or in urban canyons.
- Line of sight reception is required for optimal performance.
- GPS signals are susceptible to multiple paths and reflections, which can affect accuracy.

## 2.2 Inertial sensor

The inertial sensor can be divided into the following categories:

- *Gyroscopes*: measure angular velocities.
- *Accelerometers*: gauge linear accelerations with reference to the gravitational vector.
- *Magnetometers/compasses*: determine orientation based on the earth's magnetic field vector.

An Inertial Measurement Unit (IMU) integrates gyroscopes, accelerometers, and magnetometers to offer a complete six degrees of freedom pose estimate. However, integrating inertial measurements, such as for position computation, accumulates errors and drifts notably over time, particularly when using inexpensive MEMS (Micro-Electro-Mechanical Systems) technology.

### 2.2.1 Tactile sensor

Tactile sensors serve manipulation purposes and fall into two main categories:

- *Binary*: utilize switches placed on the fingers of a manipulator. Can be arranged in arrays (bumpers) on the external side to detect and avoid obstacles.
- *Analogical* (real valued): consist of soft devices producing a signal proportional to the local force. Utilize mechanisms like a spring coupled with a shaft or soft conductive material that changes resistance with compression. Capable of measuring movements tangential to the sensor surface.

### 2.2.2 Proximity sensor

Proximity sensors detect the presence of objects within a defined distance range, employed for grasping items and navigating around obstacles. Various technologies are utilized for this purpose:

- *Ultrasonic*: low cost.
- *Inductive*: detects ferromagnetic materials within a millimeters distance.
- *Hall effect*: detects ferromagnetic materials, small, robust, and inexpensive.
- *Capacitive*: detects any object, binary output, high accuracy when calibrated for a specific object.
- *Optical*: utilizes infrared light, offering binary or real-valued output.

## 2.3 Actuators

Effectors are responsible for altering the state of the environment, with actuators facilitating the actions of effectors. In robotics, we employ various types of actuators:

- *Electric motors*: these devices convert electrical energy into mechanical energy by leveraging the principles of electromagnetism. They produce rotational motion through the interaction between magnetic fields and electric currents.
- *Hydraulics*: this technology utilizes fluids to transmit force, employing the principles of fluid mechanics to generate, control, and transfer power via pressurized liquids.
- *Pneumatics*: a branch of engineering that employs compressed air or gas to transmit and regulate power, akin to hydraulics but using air or gas instead of liquids.
- *Photo-reactive materials*: these substances undergo a chemical change upon exposure to light.
- *Chemically reactive materials*: substances in this category undergo chemical reactions with other materials or their surroundings.
- *Thermally reactive materials*: these substances undergo changes in properties or behavior when subjected to variations in temperature.
- *Piezoelectric materials*: materials that generate electric charges in response to mechanical stress or pressure, while also displaying mechanical deformation under an electric field.

Originally, early robots were equipped with hydraulic and pneumatic actuators. Hydraulic actuators were costly, heavy, and required significant maintenance, making them suitable mainly for larger robots. Pneumatic actuators found use in stop-to-stop applications like pick-and-place tasks due to their swift actuation.

In modern times, electrical motors have become the prevalent choice for actuators. Typically, each joint incorporates its dedicated motor along with a controller. High-speed motors are often paired with elastic gearing to moderate their speed. These motors necessitate internal sensors for precise control. Stepper motors, on the other hand, don't require internal sensors; however, in case of an error, their exact position becomes unknown.

### 2.3.1 Direct current motor

Direct Current (DC) motors transform electrical energy into mechanical energy. They are compact, cost-effective, reasonably efficient, and straightforward to operate.

Electric current flows through coils of wire arranged on a rotating shaft. These wire loops create a magnetic field that interacts with the magnetic fields of permanent magnets positioned nearby. The resulting interaction between these magnetic fields causes them to repel each other, resulting in the rotation of the armature.

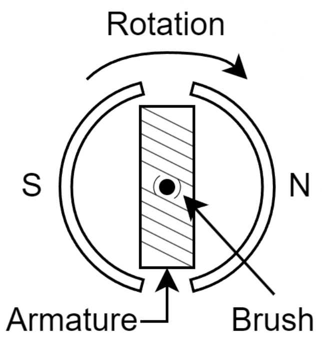


Figure 2.7: Brushed motor structure

Continuously adjusting the current causes the armature to keep rotating and generating motion. This current modification is facilitated by two connectors positioned at the center of the armature, known as brushes. It's worth noting that in lower-cost electrical motors, the external magnets remain stationary. However, these budget-friendly versions encounter several issues related to their brushes:

- Brushes gradually wear out over time.
- Brushes generate noise during operation.
- They impose a maximum speed limit.
- Cooling them proves to be challenging.
- They restrict the number of poles that can be utilized.

To circumvent this issue, one can opt for brushless motors, where external magnets are substituted with copper coils and a magnet is positioned at the center. This configuration yields a motor wherein brushes are replaced by electronics, permanent magnets reside on the rotor, and electromagnets are situated on the stator. While these motors offer superior performance, they also come at a higher cost compared to their brushed counterparts.

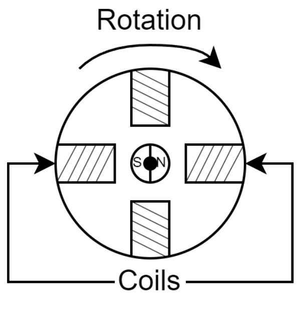


Figure 2.8: Brushless motor structure

### 2.3.2 Stepper motor

The stepper motor, a type of synchronous electric motor lacking brushes, transforms digital pulses into mechanical shaft rotations.

A stepper motor offers several advantages: it provides a direct correlation between input pulse and rotation angle, maintains full torque even at standstill when windings are energized, enables precise positioning and repeatability, responds promptly to starting, stopping, and reversing commands, boasts high reliability due to the absence of contact brushes, facilitates open-loop control which simplifies and reduces costs, supports very low-speed synchronous rotation with directly coupled loads, and offers a wide range of rotational speeds. However, there are also disadvantages: it necessitates a specialized control circuit, consumes more current compared to DC motors, experiences a reduction in torque at higher speeds, risks resonances if not adequately managed, and finds it challenging to operate at extremely high speeds.

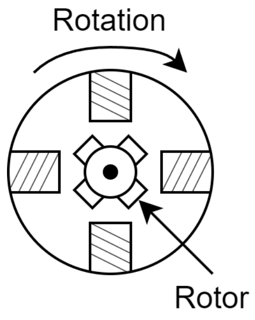


Figure 2.9: Stepper motor structure

The step angle, denoted by $\varphi$, can be determined using the following formula:

$$\varphi = \left( \frac{N_s - N_r}{N_s \cdot N_r} \right) \times 360^\circ$$

In this equation, $N_s$ represents the number of teeth on the stator, and $N_r$ represents the number of teeth on the rotor.

### 2.3.3 Servo motor

A servo is a type of specialized motor designed to precisely move its shaft to a specific position. These motors find common use in hobby radio control applications. They possess the capability to measure their own position and adjust for external loads in accordance with a control signal.

Servo motors are typically constructed from direct current motors with additional components including gear reduction, a position sensor, and control electronics. The travel range of the shaft is usually limited to 180 degrees, which is adequate for the majority of applications.

CHAPTER 3

# Robot Odometry

## 3.1 Introduction

For autonomous robots and unmanned vehicles to execute their tasks effectively, they require: accurate self-location information, and detailed maps of the environment. However, these requirements aren't always feasible or dependable due to the following reasons:

- Global Navigation Satellite Systems (GNSS) may not always be reliable or available.
- Not all areas have been accurately mapped.
- Environmental conditions can change dynamically.
- Maps need regular updates to remain current and reliable.

The robot's position can be regarded as a random variable due to the uncertainty inherent in our estimation of its true position. The full SLAM (Simultaneous Localization and Mapping) problem involves determining the distribution of both the robot's poses and the positions of landmarks, considering the robot's actions and sensor measurements:

$$\Pr\left(\Gamma_{1:t}, l_1, \ldots, l_N | Z_{1:t}, U_{1:t}\right)$$

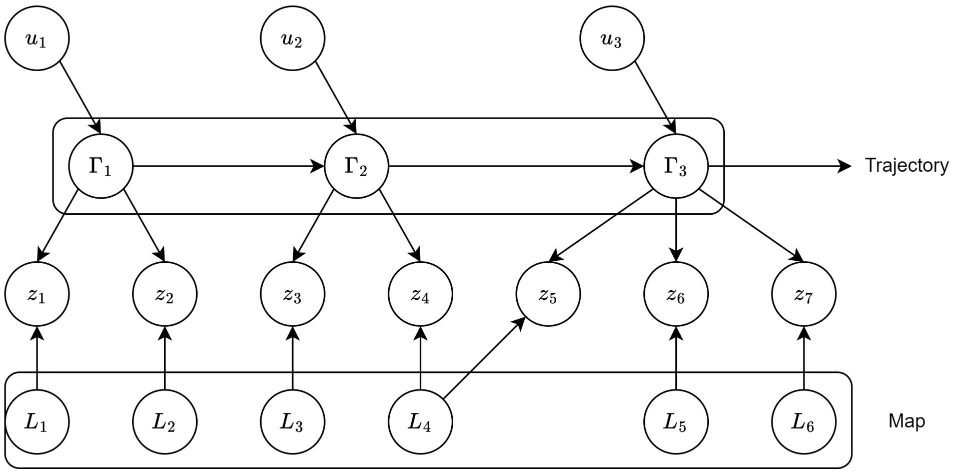


Figure 3.1: Simultaneous Localization and Mapping

If a complete trajectory isn't necessary, a simplified version known as online SLAM can be used. This method provides the entire map and calculates the probability of only the most recent pose based on all measurements and actions:

$$\Pr(\Gamma_t, l_1, \dots, l_N | Z_{1:t}, U_{1:t}) = \int \int \int_1^{t-1} \Pr(\Gamma_{1:t}, l_1, \dots, l_N | Z_{1:t}, U_{1:t})$$

It's important to note that the term pose encompasses not only the position but also the orientation of the robot relative to the environment.

The motion model incorporates all actions $u_1, \dots, u_N$ and their resulting poses $\Gamma_1, \dots, \Gamma_N$ describing how the robot's pose changes through the actuators.

On the other hand, the sensor model involves all poses $\Gamma_1, \dots, \Gamma_N$, all position probabilities $z_1, \dots, z_N$, and the map with landmarks $L_1, \dots, L_N$. It defines the probability distribution of a specific measurement given the robot's pose and the positions of the landmarks.

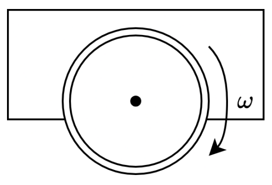
## 3.2 Direct kinematic

The robot kinematic is based on the motion model.

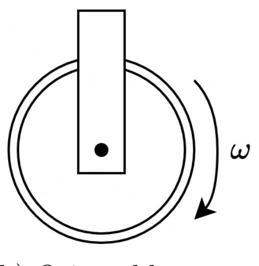
**Definition** (*Wheeled mobile robots*)**.** A robot capable of locomotion on a surface solely through the actuation of wheel assemblies mounted on the robot and in contact with the surface. A wheel assembly is a device which provides or allows motion between its mount and surface on which it is intended to have a single point of rolling contact.

Various kinematic configurations are feasible:

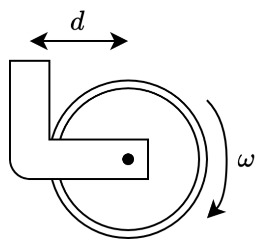
- *Differential drive* (two wheels): basic design, prone to disturbances from uneven terrain, and lacks lateral translation capability.
- *Tracks*: ideal for outdoor surfaces, movement precision compromised, especially during rotations, intricate structure and behavior, and lateral translation not achievable.
- *Omnidirectional* (synchro drive): utilizes all three degrees of freedom, sophisticated design and functionality, and intricate structural composition.



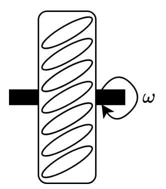
(a) Fixed



(b) Orientable centered



(c) Caster omnidirectional



(d) Swedish or meccanum

Figure 3.2: Wheel classification

**Definition** (*Locomotion*)**.** Locomotion involves initiating movement in an autonomous robot:

Motion is achieved by applying forces to the vehicle.

**Definition** (*Dynamics*)**.** Dynamics encompasses the analysis of motion through the modeling of forces, as well as the associated energies and velocities involved in these movements.

**Definition** (*Kinematics*)**.** Kinematics is the examination of motion devoid of considerations regarding influencing forces.

It focuses on the geometric relationships dictating the system's behavior and the correlation between control parameters and the system's behavior in state space.

**Definition** (*Direct kinematics*)**.** Direct kinematics involves determining the pose $(x, y, \theta)$ that a robot achieves given specific control parameters and a time of movement $t$.

**Definition** (*Inverse kinematics*)**.** Inverse kinematics pertains to finding the control parameters necessary to reach a specified final pose $(x, y, \theta)$ within a given time $t$.

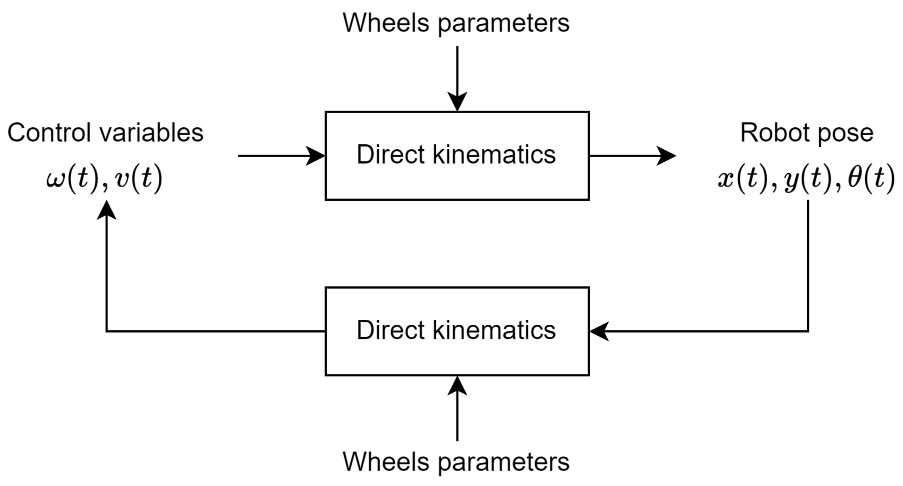


Figure 3.3: Direct and inverse kinematics

**Wheels assumptions** Several assumptions must be established concerning the wheels:

1. The robot consists solely of rigid components.
2. Each wheel possesses a single steering link.
3. Steering axes are perpendicular to the ground.
4. The wheel undergoes pure rolling about its axis ($x$-axis).
5. There is no translational movement of the wheel.

The critical parameters defining the wheels include their radius $r$, linear velocity $v$, and angular velocity $\omega$.

For a robot to maneuver in the plane without slipping, the axes of the wheels must intersect at a specific point known as the Instantaneous Center of Curvature (ICC) or Instantaneous Center of Rotation (ICR). Failure of the wheel axes to intersect at a single point renders the robot immobile.

**Cartesian representation** The global reference system is external to the robot and serves as the frame of reference. The robot is characterized by its coordinates $(x_b, y_b)$ relative to this reference frame. The angle $\theta$ represents the orientation of the robot's frontal face with respect to the $x$-axis.

Furthermore, a robot-centric reference frame can be established, centered within the robot itself.

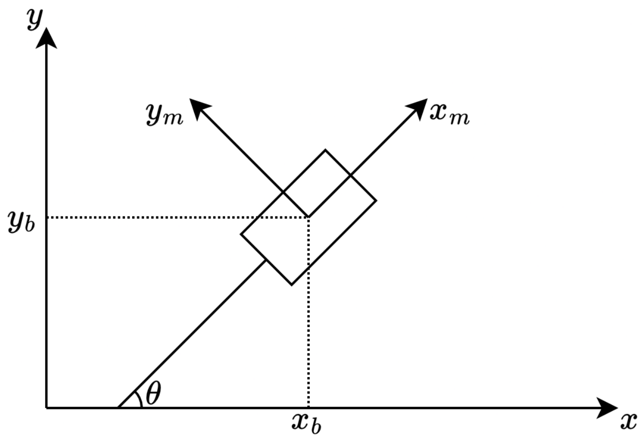


Figure 3.4: Robot reference Cartesian planes

The pose of the robot is determined by its position and orientation relative to the global reference system:

$$P(x_b, y_b, \theta) = (x, y, \theta)$$

## 3.3 Differential drive

The differential drive robot consists of two wheels positioned on the same axis, each with its own independent motor, along with a third passive supporting wheel.

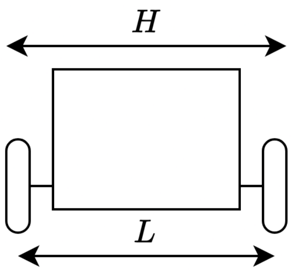


Figure 3.5: Differential drive robot

The baseline $L$ represents the distance between the contact points of the wheels.

The variables under independent control are the velocities of the right wheel, $v_R$, and the left wheel, $v_L$. The robot's pose is represented in the base reference frame as $P(x, y, \theta)$.

Control inputs are the linear velocity of the robot, $v$, and its angular velocity, $\omega$, which are linearly related to $v_R$ and $v_L$. Both right and left wheels follow circular paths with an angular velocity of $\omega$ and different curvature radii $R$:

$$\begin{cases} \omega\left(R+\frac{L}{2}\right) = v_R \\ \omega\left(R+\frac{L}{2}\right) = v_L \end{cases}$$

Given $v_R$ and $v_L$, $\omega$ can be found by solving for $R$ and equating:

$$\omega = \frac{v_R - v_L}{L}$$

Similarly, $R$ can be found by solving for $\omega$ and equating:

$$R = \frac{L}{2}\frac{v_R + v_L}{v_R - v_L}$$

Note that rotation in place is achieved by setting $R = 0$ and $v_R = -v_L$, while linear movement is accomplished by setting $T = \infty$ and $v_R = v_L$.

The wheels move around an Instantaneous Center of Curvature (ICC) on a circumference with an instantaneous radius $R$ and angular velocity $\omega$:

$$\text{ICC} = \left\{x + R\cos\left(\theta + \frac{\pi}{2}\right), y + R\sin\left(\theta + \frac{\pi}{2}\right)\right\} = \{x - R\sin(\theta), y + R\cos(\theta)\}$$

The overall change in position over time is given by the equation:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\cdot\delta t) & -\sin(\omega\cdot\delta t) & 0 \\ \sin(\omega\cdot\delta t) & \cos(\omega\cdot\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - \text{ICC}_x \\ y - \text{ICC}_y \\ \theta \end{bmatrix} + \begin{bmatrix} \text{ICC}_x \\ \text{ICC}_y \\ \omega\cdot\delta t \end{bmatrix}$$

Finally, the control inputs are defined by the following formulas:

$$v = \frac{v_R + v_L}{2} \qquad \omega = \frac{v_R - v_L}{L}$$

Where a positive angular velocity indicates the robot is turning left, otherwise it turns right. Additionally, when velocities are equal, the robot exhibits lower angular velocity if $L$ is large, and higher otherwise.

### 3.3.1 Odometry

Given the known angular speed $\omega$, the radius of the Instantaneous Center of Curvature (ICC) $R$, and the linear velocity $v$ at each time instant, we can reconstruct the path and the final trajectory. This reconstruction of the path via integration is termed odometry.

The odometry of this robot involves computing the velocity in the base frame:

$$\begin{cases} v_x = v(t)\cos(\theta(t)) \\ v_y = v(t)\sin(\theta(t)) \end{cases}$$

Integrating the position in the base frame yields:

$$\begin{cases} x(t) = \int v(t)\cos(\theta(t)) \\ y(t) = \int v(t)\sin(\theta(t)) \\ \theta(t) = \int \omega(t)dt \end{cases}$$

Considering the velocities of both wheels $v_R$ and $v_L$ at a discrete time $t'$, we have:

$$\begin{cases} x(t) = \frac{1}{2}\int_0^t \left(v_R(t') + v_L(t')\right)\cos\left(\theta(t')\right) \\ y(t) = \frac{1}{2}\int_0^t \left(v_R(t') + v_L(t')\right)\sin\left(\theta(t')\right) \\ \theta(t) = \frac{1}{L}\int_0^t \left(v_R(t') - v_L(t')\right) dt' \end{cases}$$

Since computing these integrals is computationally intensive, we may resort to using approximations at discrete time instants.

**Euler's integration** When assuming constant linear velocity $v_k$ and angular velocity $\omega_k$ within the time interval $[t_k, t_{k+1}]$, Euler integration can be employed to compute the robot odometry:

$$\begin{cases} x_{k+1} = x_k + v_k T_S \cos\theta_k \\ y_{k+1} = y_k + v_k T_S \sin\theta_k \\ \theta_{k+1} = \theta_k + \omega_k T_S \\ T_S = t_{k+1} - t_k \end{cases}$$

Here, the position $\{x_{k+1}, y_{k+1}\}$ is approximated, while the angle $\theta_{k+1}$ remains exact.

**Runge-Kutta's integration** Assuming constant linear velocity $v_k$ and angular velocity $\omega_k$ within the time interval $[t_k, t_{k+1}]$, second-order Runge-Kutta integration can be used for computing robot odometry:

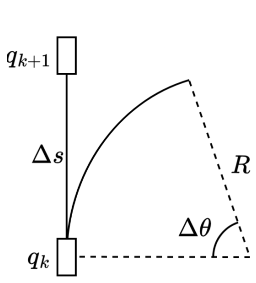
$$\begin{cases} x_{k+1} = x_k + v_k T_S \cos\left(\theta_k + \dfrac{\omega_k T_S}{2}\right) \\ y_{k+1} = y_k + v_k T_S \sin\left(\theta_k + \dfrac{\omega_k T_S}{2}\right) \\ \theta_{k+1} = \theta_k + \omega_k T_S \\ T_S = t_{k+1} - t_k \end{cases}$$

Although this method provides a better approximation, the orientation is not exact.

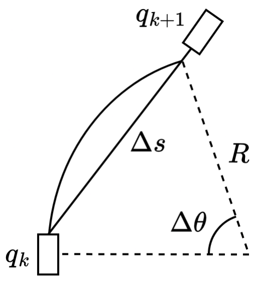
**Exact integration** When assuming constant linear velocity $v_k$ and angular velocity $\omega_k$ within the time interval $[t_k, t_{k+1}]$, exact integration can be utilized to compute the robot odometry, resulting in:

$$\begin{cases} x_{k+1} = x_k + \dfrac{v_k}{\omega_k}\left(\sin\theta_{k+1} - \sin\theta_k\right) \\ y_{k+1} = y_k - \dfrac{v_k}{\omega_k}\left(\cos\theta_{k+1} - \cos\theta_k\right) \\ \theta_{k+1} = \theta_k + \omega_k T_S \\ T_S = t_{k+1} - t_k \end{cases}$$

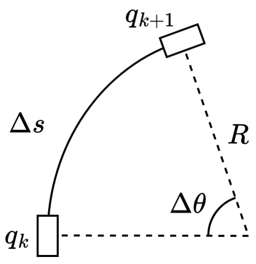
In this case, all measurements are exact, but special attention must be given when the robot travels straight ($\omega \sim 0$). In such cases, Runge-Kutta integration should be used.



(a) Euler



(b) Runge-Kutta



(c) Exact

Figure 3.6: Integration techniques

These integration techniques serve different purposes depending on the system's frequency and the frequency at which parameters are checked. Euler and Runge-Kutta methods are suitable for high-frequency systems, while the exact approximation is necessary for low-frequency systems.

### 3.3.2 Sensors

Proprioceptive measurements are utilized to calculate linear velocity $v_k$ and angular velocity $\omega_k$:

$$\begin{cases} v_k T_S = \Delta s \\ \omega_k T_S = \Delta\theta \end{cases} \implies \frac{\Delta s}{\Delta\theta} = \frac{v_k}{\omega_k}$$

Here, $\Delta s$ represents the distance traveled and $\Delta\theta$ denotes the change in orientation.

In a differential drive system, these quantities become:

$$\Delta s = \frac{r}{2}\left(\Delta\phi_R + \Delta\phi_L\right) \qquad \Delta\theta = \frac{r}{L}\left(\Delta\phi_R - \Delta\phi_L\right)$$

Here, $\Delta\phi_R$ and $\Delta\phi_L$ correspond to the total rotations measured by wheel encoders.

It's important to note that this formula is applicable primarily for small-time frames, as larger ones introduce increased error due to various factors.

## 3.4 Synchronous drive

A synchronous drive robot is a sophisticated mechanical robot design incorporates three wheels for both propulsion and steering. It employs two motors: one for driving the wheels and another for steering them. All wheels are oriented in the same direction to ensure smooth movement. Additionally, the robot may include an extra actuator for precise control of angular velocity $\omega$.

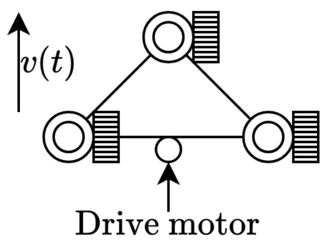


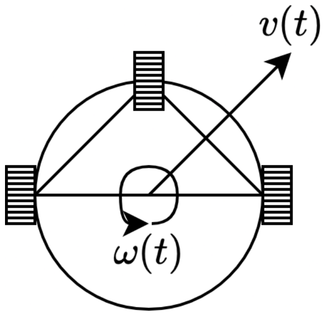
Figure 3.7: Synchronous drive robot

The robot's control variables consist of the linear velocity $v(t)$ and the angular velocity $\omega(t)$.

Its Instantaneous Center of Curvature remains at infinity, indicating non-holonomic behavior where the robot can only translate freely.

For the synchronous drive of the robot:

- Both linear velocity $v(t)$ and angular velocity $\omega(t)$ are directly controlled.
- Steering adjusts the direction of the Instantaneous Center of Curvature (ICC).



In this configuration, when $v(t) = 0$ and $\omega(t) = \omega$ at a specific time instant, the robot rotates in place. Conversely, when $v(t) = v$ and $\omega(t) = 0$ at a particular time instant, the robot moves linearly.

### 3.4.1 Odometry

To compute the velocity in the base frame, we use the following equations:

$$\begin{cases} v_x = v(t)\cos(\omega(t)) \\ v_y = v(t)\sin(\omega(t)) \end{cases}$$

Integrating the position in the base frame, we obtain the robot odometry:

$$\begin{cases} x(t) = \int_0^t \cos\left(\theta(t')\right) dt' \\ y(t) = \int_0^t \sin\left(\theta(t')\right) dt' \\ \theta(t) = \int_0^t \omega(t') dt' \end{cases}$$

## 3.5 Omnidirectional drive

The omnidirectional drive robot embodies a straightforward mechanical design featuring a minimum of three Swedish wheels, each powered by an independent motor. These wheels are strategically oriented in different directions, facilitating direct control over movements in $x$, $y$, and $\theta$.

The robot's control parameters consist of the linear velocity $v(t)$ for each axis and the overall angular velocity $\omega(t)$.

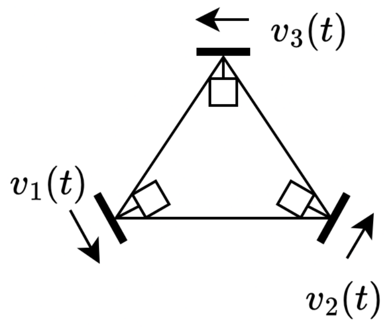


Figure 3.8: Omnidirectional drive robot

To manipulate the robot's position, the following matrix governs its motion:

$$\begin{bmatrix} v_x \\ v_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{\sqrt{3}}r & \frac{1}{\sqrt{3}}r \\ -\frac{2}{3}r & \frac{1}{3}r & \frac{1}{3}r \\ \frac{r}{3L} & \frac{r}{3L} & \frac{r}{3L} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

## 3.6 Tricycle drive

The Tricycle setup defines the kinematics of AGV as follows: it features a single actuated and steerable wheel along with two additional passive wheels. Independent control of $\theta$ is not possible unless $\alpha(t)$ can achieve up to 90 degrees. Additionally, the Instantaneous Center of Curvature (ICC) must align with the line passing through the fixed wheels.

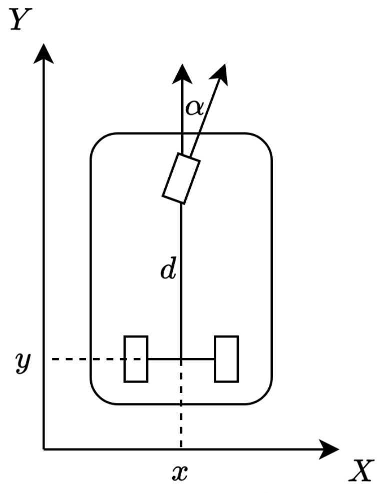


Figure 3.9: Tricycle drive robot

The robot's control parameters consist of the steering direction $\alpha(t)$ the angular velocity of the steering wheel $\omega_s(t)$.

In this configuration, when $\alpha(t) = 0$ and $\omega_s(t) = \omega$ at a specific time instant, the robot moves linearly. Conversely, when $\alpha(t) = 90°$ and $\omega(t) = \omega$ at a particular time instant, the robot rotates in place.

### 3.6.1 Odometry

The direct kinematics can be derived as follows:

$$\begin{cases} r = \text{steering wheel radius} \\ V_S(t) = \omega_S(t) r \\ R(t) = d \tan\left(\frac{\pi}{2} - \alpha(t)\right) \\ \omega(t) = \frac{\omega_s(t) r}{\sqrt{d^2 + R(t)^2}} = \frac{V_S(t)}{d} \sin\alpha \end{cases}$$

In the base frame:

$$\begin{cases} \dot{x}(t) = V_S(t) \cos\alpha(t) + \cos\theta(t) = V(t)\cos\theta(t) \\ \dot{y}(t) = V_S(t) \cos\alpha(t) + \cos\theta(t) = V(t)\sin\theta(t) \\ \dot{\theta} = \frac{V_S(t)}{d} \sin\alpha(t) = \omega(t) \end{cases}$$

## 3.7 Ackerman steering drive

The most common form of kinematics worldwide involves vehicles equipped with four wheels in motion. These wheels are capable of turning but within certain limitations in their angles of rotation. Notably, this kinematic system does not incorporate in-place rotation.

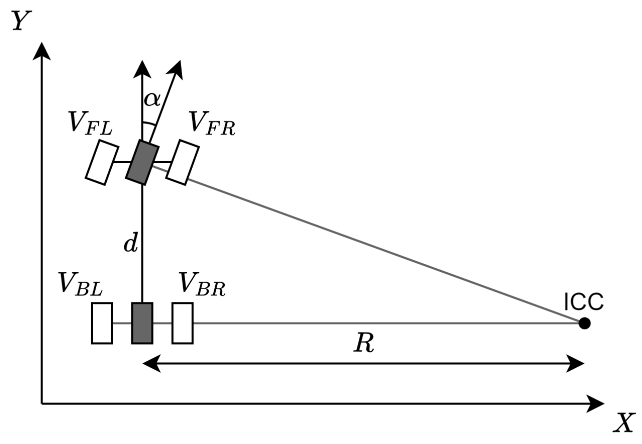


Figure 3.10: Ackerman steering drive robot

We simplify the model to resemble a bicycle:

$$\begin{cases} R = \frac{d}{\tan\alpha} \\ \frac{\omega d}{\sin\alpha} \end{cases}$$

This is with reference to the center of actual wheels:

$$\omega R = V \rightarrow \omega = V\frac{\tan\alpha}{d}$$

## 3.8 Skid steering drive

Vehicles equipped with tracks operate on a kinematic principle akin to the differential drive system. Each track's speed is individually controlled. The height of the track serves as the equivalent of wheel diameter. This configuration is commonly referred to as Skid Steering. However, it requires meticulous calibration and accurate modeling of slippage.

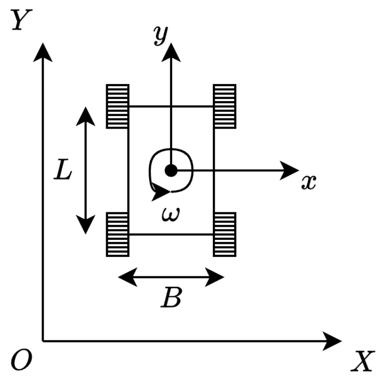


Figure 3.11: Skid steering drive robot

Let's make the following assumptions:

- The vehicle's mass is centralized.
- All wheels maintain contact with the ground.
- Wheels on the same side move at the same speed.

While in motion, we encounter multiple Instantaneous Centers of Rotation (ICRs), all sharing the same angular velocity $\omega_z$:

$$\begin{cases} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = j_\omega \begin{bmatrix} \omega_l r \\ \omega_r r \end{bmatrix} \\ J_\omega = \dfrac{1}{y_l - y_r} \begin{bmatrix} -y_r & y_l \\ x_G & -x_G \\ -1 & - \end{bmatrix} \end{cases}$$

Assuming symmetry in the robot ($y_0 = y_l = -y_r$), we find:

$$\begin{cases} v_x = \dfrac{v_l + v_r}{2} \\ v_y = 0 \\ \omega_z = \dfrac{-v_l + v_r}{2y_0} \end{cases}$$

We can determine the instantaneous radius of curvature:

$$\begin{cases} R = \dfrac{v_G}{\omega_z} = \dfrac{v_l + v_r}{-v_l + v_r} y_0 \\ \lambda = \dfrac{v_l + v_r}{-v_l + v_r} \\ X = \dfrac{2y_0}{B} \end{cases}$$

## 3.9 Inverse kinematics

When faced with a desired position or velocity, several strategies can be employed to achieve it. While it's relatively straightforward to find some solution, determining the best solution can pose significant challenges. This best solution could be defined by various criteria such as:

- Shortest time to reach the goal.
- Most energy-efficient path or trajectory.
- Smoothest velocity profiles for comfortable operation.

Furthermore, if we encounter non-holonomic constraints and are limited to just two control variables, it becomes impossible to directly reach any of the three degrees of freedom final positions.

### 3.9.1 Differential drive

To tackle the problem effectively, we can decompose it and focus on controlling only a few degrees of freedom at a time:

1. Begin by turning the robot so that the wheels align parallel to the line between its original and final positions:
$$-v_L(t) = v_R(t) = v_{max}$$
2. Proceed to drive straight until the robot's origin coincides with the destination:
$$v_L(t) = v_R(t) = v_{max}$$
3. Finally, rotate again to achieve the desired final orientation:
$$-v_L(t) = v_R(t) = v_{max}$$

### 3.9.2 Synchro drive

To address the challenge systematically, we can break it down and manage only a select few degrees of freedom at each stage:

1. Initiate a turn to align the wheels parallel to the line connecting the robot's original and final positions:
$$\omega(t) = \omega_{max}$$
2. Proceed to drive straight until the robot's origin reaches the destination:
$$v(t) = v_{max}$$
3. Rotate once more to achieve the desired final orientation:
$$\omega(t) = \omega_{max}$$

CHAPTER 4

# Robot localization

## 4.1 Introduction

Given a map, our objective is to determine the precise location of a robot within that map. This problem can be addressed by leveraging a sensor model, which characterizes $\Pr(z|x)$, representing the probability of obtaining a measurement $z$ when the robot is positioned at $x$. Specifically, a scan $z$ comprises $K$ measurements:

$$z = \{z_1, z_2, \dots, z_K\}$$

Each individual measurement is statistically independent, conditioned on the robot's position and the surrounding map:

$$\Pr(z|x, m) = \prod_{k=1}^{K} \Pr(z_k|x, m)$$

Measurements may originate from various sources:

- Beams reflected by obstacles.
- Beams reflected by individuals or influenced by crosstalk.
- Random measurements.
- Maximum range measurements.

## 4.2 Beam sensor model

If laser measurements are utilized to determine distances between the robot and obstacles, the beam sensor model can be applied. This model treats each beam independently and integrates various sources of error:

1. *Measurement Noise*: each measurement is subject to noise, which is represented using a Gaussian distribution:

$$\Pr_{hit}(z|x, m) = \eta \frac{1}{\sqrt{2\pi b}} e^{-\frac{1}{2}\frac{(z - z_{\text{exp}})^2}{b}}$$

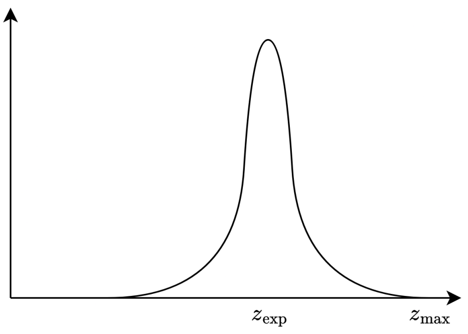


Figure 4.1: Measurement noise

2. *Unexpected obstacles*: measurements may deviate from the actual values due to temporary obstacles obstructing the beam's path. This probability is expressed as:

$$\Pr_{\text{unexp}}(z|x,m) = \begin{cases} \eta\lambda e^{-\lambda z} & z > z_{\exp} \\ 0 & \text{otherwise} \end{cases}$$

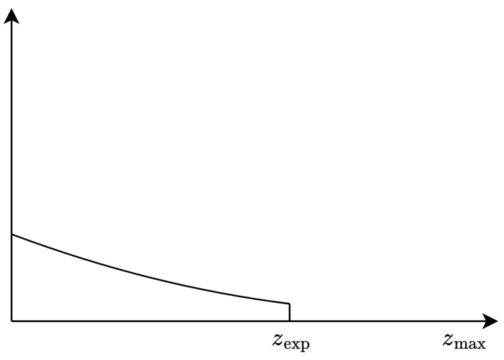


Figure 4.2: Unexpected obstacles

3. *Random measurement*: occasionally, a completely erroneous measurement may occur with a certain probability:

$$\Pr_{\text{rand}}(z|x,m) = \eta\frac{1}{z_{\max}}$$

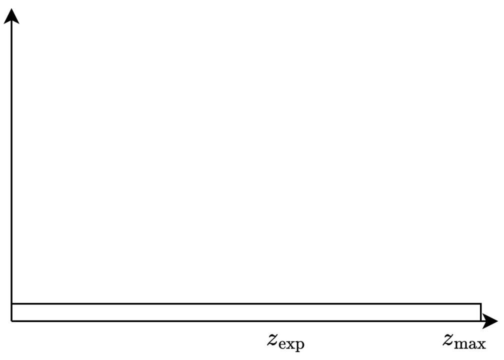


Figure 4.3: Random measurement

4. *Maximum range*: uncertainty arises from distances beyond the sensor's reach, which is modeled as:

$$\Pr_{\max}(z|x,m) = \eta\frac{1}{z_{\text{small}}}$$

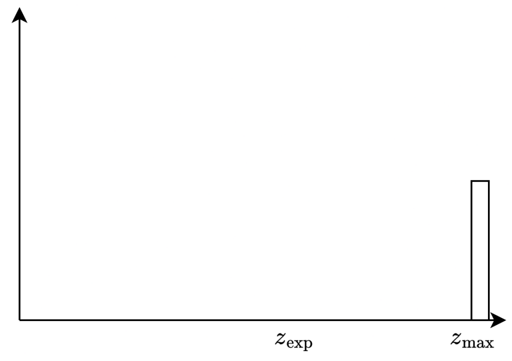


Figure 4.4: Random measurement

The total probability can be computed by combining these individual probabilities:

$$\Pr(z|x,m) = \begin{bmatrix} \alpha_{\text{hit}} \\ \alpha_{\text{unexp}} \\ \alpha_{\max} \\ \alpha_{\text{rand}} \end{bmatrix}^T \cdot \begin{bmatrix} \Pr_{\text{hit}}(z|x,m) \\ \Pr_{\text{unexp}}(z|x,m) \\ \Pr_{\max}(z|x,m) \\ \Pr_{\text{rand}}(z|x,m) \end{bmatrix}$$

The overall structure of this probability distribution is depicted as follows:

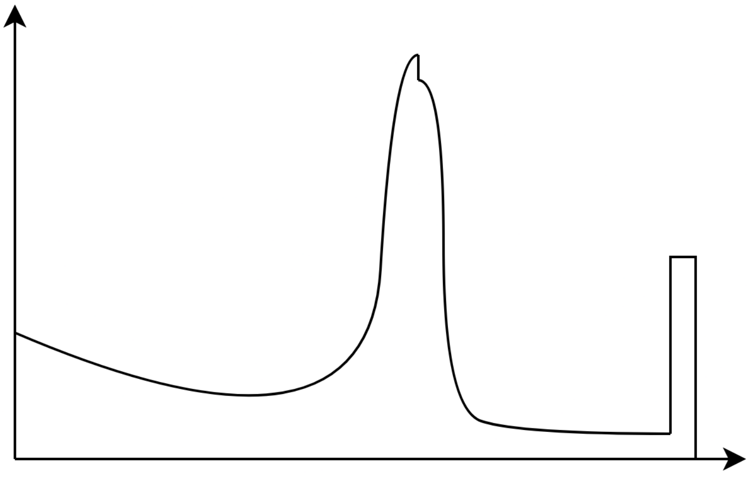
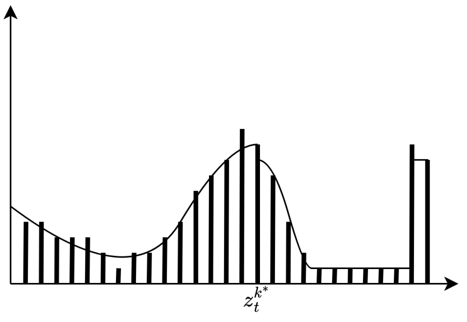


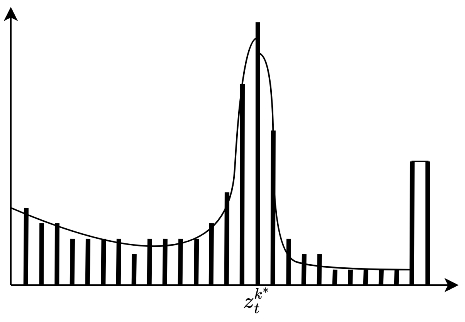
Figure 4.5: Probability distribution of beam sensor model

### 4.2.1 Calibration

To calibrate the sensor, we can collect data at specific distances, such as 300 cm and 400 cm, and then estimate the model parameters using maximum likelihood $\Pr(z|z_{\text{exp}})$. Since it's impractical to calibrate the sensor for every possible distance, we typically select a few representative distances and interpolate or use a mean for the missing data.



(a) Sonar



(b) Laser

Figure 4.6: Calibration at three hundreds centimeters

By acquiring data at these distances, we can estimate the parameters of the sensor model to improve its accuracy across a range of distances.

### 4.2.2 Model likelihood

In this model, our objective is not to find the most likely position given the measurements, but rather to determine the position that maximizes the likelihood of the sensor readings. Therefore, in this context, we are searching for a position $x$ such that the measurements from the sensors are maximized. This approach focuses on identifying the position that aligns best with the observed sensor data, rather than estimating the robot's actual position.

## 4.3 Scan sensor model

The beam sensor model, while assuming independence between beams and the physical causes of measurements, exhibits several issues:

- It tends to be overconfident due to its independence assumptions.
- Parameters need to be learned from data, adding complexity.
- A distinct model is required for different angles relative to obstacles, leading to increased complexity.
- It's inefficient as it relies on ray tracing for calculations.

To address these challenges, the scan sensor model simplifies the beam sensor model by:

- Utilizing a Gaussian distribution with the mean set at the distance to the closest obstacle.
- Employing a uniform distribution for random measurements.
- Incorporating a small uniform distribution for maximum range measurements.

In this model, we calculate the likelihood of encountering an obstacle along the trajectory of the ray.

From the occupancy grid map, that is the real map with the real obstacles, we can compute the likelihood field.

The likelihood field enables matching of various scans (except for sonars). It operates highly efficiently, relying solely on 2D tables. It maintains smoothness concerning minor shifts in robot position, facilitating gradient descent pose optimization. However, it disregards the physical attributes of beams.

**Summary** In highly dynamic environments, opting for the beam sensor model is crucial as it accounts for temporary obstacles, unlike the scan model. Conversely, in static environments, this approach proves faster due to precomputed maps.

## 4.4 Landmark model

Landmark sensors provide information on distance, bearing, or both. These measurements can be obtained through active beacons such as radio or GPS, or passive methods like visual or retro-reflective techniques. The standard approach for utilizing this data is triangulation.

Explicitly incorporating uncertainty into sensing processes is crucial for ensuring robustness:

1. Establish a parametric model for noise-free measurements.
2. Analyze sources of noise, such as distance and angle.
3. Introduce appropriate noise to parameters, possibly incorporating densities for noise distribution.
4. Learn and verify parameters by fitting the model to empirical data.

The likelihood of a measurement is determined by probabilistically comparing actual measurements with expected ones.

### 4.4.1 Landmark detection

The measurement $z = (i, d, \alpha)$ for a robot positioned at $(x, y, \theta)$, relative to landmark $i$ on map $m$ (denoted as $m_i$), is expressed as follows:

$$\hat{d} = \sqrt{(m_x(i) - x)^2 + (m_y(i) - y)^2}$$

$$\hat{\alpha} = \arctan\left[2(m_y(i) - y, m_x(i) - x) - \theta\right]$$

The detection probability of a particular landmark may rely on either the distance or the bearing:

$$\Pr_{\text{det}} = \Pr(\hat{d} - d, \varepsilon_d) \Pr(\hat{\alpha} - \alpha, \varepsilon_\alpha)$$

Additionally, consideration for false positives is necessary:

$$z_{\text{det}} \Pr_{\text{det}} + z_{\text{fp}} \Pr_{\text{uniform}} (z|x, m)$$

## 4.5 Bayesian filter

As the robot navigates from an initial position to a target destination along a specific path, it's essential to integrate both its motion and sensor models effectively. Bayesian filtering provides a method to achieve this integration seamlessly.

The motion model encompasses the robot's pose $\Gamma$ and the actions it takes $u$. Meanwhile, the sensor model incorporates the robot's pose $\Gamma$, observations $z$, and the map of the environment $L$.

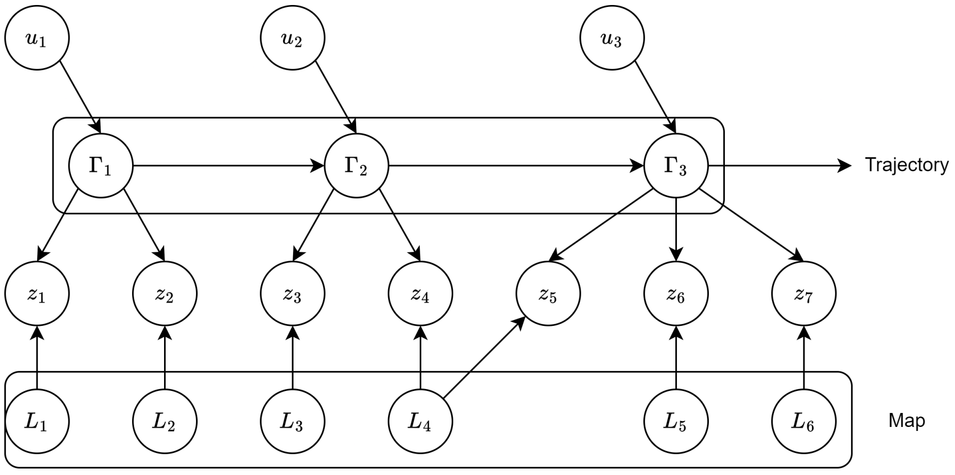


Figure 4.7: Localization framework

The filtering process aims to estimate the posterior probability distribution of the robot's state $x(t)$ given the stream of information about movement and sensors $d_t = \{u_1, z_1, \ldots, u_t, z_t\}$ and the map of the environment $m$. We have the motion model which provides the probability distribution $\Pr(\Gamma_t|\Gamma_{t:1}, l_1, \ldots, l_n)$ representing the robot's pose at time $t$ given its past poses and the map. Our goal is to compute the belief $\text{Bel}(x_t) = P(x_t|u_1, z_1, \ldots, u_t, z_t, m)$ representing the probability distribution of the robot's state $x_t$ given all the available information. To compute $\text{Bel}(x_t)$, we typically use Bayes' rule, incorporating both the motion and sensor models. The motion model predicts the next state given the current state and action, while the sensor model updates this prediction based on the observations. This recursive process allows us to continuously refine our estimate of the robot's state as new information becomes available.

**Assumptions** We make the following assumptions, all of which are known:

- The prior probability of the system state $\Pr(x_0)$.
- The motion model $\Pr(x'|x, u)$, which describes the probability of transitioning from state $x$ to state $x'$ given action $u$.
- The sensor model $\Pr(z|x, m)$, which describes the probability of obtaining observation $z$ given the system state $x$ and the map $m$.

**Markov assumption** Additionally, we require the Markov assumption:

**Property 4.5.1.** A stochastic process possesses the Markov property if the conditional probability distribution of future states, conditioned on both past and present values, depends solely on the present state.

In other words, given the present state, the future states are independent of the past. Furthermore, we assume:

- Perfect model: the models accurately represent the system and environment.
- No approximation errors: the computations are exact.
- Static and stationary world: the environment does not change over time.
- Independent noise: the noise in sensor readings and system dynamics is independent and identically distributed.

### 4.5.1 Bayes filter

With the given assumptions, we can derive the belief on the robot's position using Bayes' theorem:

$$\begin{aligned}\mathrm{Bel}(x_t) &= \Pr(x_t, z_1, \ldots, u_t, z_t, m) \\ &= \eta \Pr(z_t | x_t, u_1, z_1, \ldots, u_t, m) \Pr(x_t | u_1, z_1, \ldots, u_t, m)\end{aligned}$$

Using the Markov property and Bayes' rule, we simplify further:

$$\mathrm{Bel}(x_t) = \eta \Pr(z_t | x_t, m) \Pr(x_t | u_1, z_1, \ldots, u_t, m)$$

Expanding the second term using the recursive Bayes filter equation:

$$\mathrm{Bel}(x_t) = \eta \Pr(z_t | x_t, m) \int \Pr(x_t | u_1, z_1, \ldots, u_t, x_{t-1}, m) \Pr(x_{t-1} | u_1, z_1, \ldots, u_{t-1}, m) \, dx_{t-1}$$

Applying the Markov property again and simplifying:

$$\mathrm{Bel}(x_t) = \eta \Pr(z_t | x_t, m) \int \Pr(x_t | u_t, x_{t-1}) \Pr(x_{t-1} | u_1, z_1, \ldots, u_{t-1}, m) \, dx_{t-1}$$

Now, recognizing that the belief at $x_{t-1}$ is the same as $\mathrm{Bel}(x_{t-1})$, we can rewrite the integral:

$$\mathrm{Bel}(x_t) = \eta \Pr(z_t | x_t, m) \int \Pr(x_t | u_t, x_{t-1}) \mathrm{Bel}(x_{t-1}) \, dx_{t-1}$$

In this expression, $z$ represents an observation, $u$ represents an action, $x$ represents a state, and $m$ represents a map. This recursive formulation allows us to iteratively update our belief about the robot's position based on new sensor readings and actions.

### 4.5.2 Bayes filter algorithm

The Bayes filter algorithm computes the belief of a certain position $x$ given perceptual data $z$ or action data $u$. If the map is given, the belief at $x_t$ is updated using the formula:

$$\mathrm{Bel}(x_t | m) = \eta \Pr(z_t | x_t, m) \int \Pr(x_t | u_t, x_{t-1}, m) \mathrm{Bel}(x_{t-1} | m) \, dx_{t-1}$$

Here's the Bayes filter algorithm:

**Algorithm 1** Bayes filter algorithm

```
 1: if $d$ is a perceptual data item $z$ then
 2:     for all $x$ do
 3:         $\mathrm{Bel}'(x) = \Pr(z|x)\mathrm{Bel}(x)$
 4:     end for
 5:     normalize $\mathrm{Bel}'(x)$
 6: else if $d$ is an action data item $u$ then
 7:     for all $x$ do
 8:         $\mathrm{Bel}'(x) = \int \Pr(x|u, x')\mathrm{Bel}(x')\, dx'$
 9:     end for
10: end if
11: return $\mathrm{Bel}'(x)$
```

This algorithm updates the belief state based on either a perceptual data item $z$ or an action data item $u$. If $d$ is a perceptual data item, it computes the product of the probability of observation given each state and the current belief state, then normalizes the result. If $d$ is an action data item, it integrates over the motion model to update the belief state.

Based on this representation, various filtering algorithms can be implemented, including discrete filters, Kalman filters, sigma-point filters, and particle filters. These algorithms differ in how they represent and update the belief state and handle uncertainties in the system and sensor measurements.

### 4.5.3 Discrete Bayes filter algorithm

The discrete Bayes filter algorithm provides a method for updating the belief state when the map is discretized into regions. Here's the algorithm:

**Algorithm 2** Discrete Bayes filter algorithm

```
 1: $h = 0$
 2: if $d$ is a perceptual data item $z$ then
 3:     for all $x$ do
 4:         $\mathrm{Bel}'(x) = \Pr(z|x)\mathrm{Bel}(x)$
 5:         $\eta = \eta + \mathrm{Bel}'(x)$
 6:     end for
 7:     for all $x$ do
 8:         $\mathrm{Bel}'(x) = \eta^{-1}\mathrm{Bel}'(x)$
 9:     end for
10:     normalize $\mathrm{Bel}'(x)$
11: else if $d$ is an action data item $u$ then
12:     for all $x$ do
13:         $\mathrm{Bel}'(x) = \sum_{x'} \Pr(x|u, x')\mathrm{Bel}(x')$
14:     end for
15: end if
16: return $\mathrm{Bel}'(x)$
```

This algorithm updates the belief state based on either a perceptual data item $z$ or an action data item $u$:

- When the data item $d$ is a perceptual input $z$, the algorithm computes the product of the probability of observation given each state and the current belief state. Then, it normalizes the resulting belief to ensure it sums to one.
- When $d$ represents an action $u$, the algorithm updates the belief state by summing over all possible previous states $x'$ weighted by the transition probabilities $\Pr(x|u, x')$.

The algorithm facilitates belief update upon sensory input and normalization, iterating over all cells. For efficiency, especially when the belief is peaked (e.g., during position tracking), it's advisable to avoid updating irrelevant parts. Instead, focus on updating relevant components based on the likelihood of observations given the active components.

For updating the belief upon robot motion, the algorithm assumes a bounded Gaussian model to reduce the computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. This involves shifting the data in the grid according to the measured motion and then convolving the grid using a Gaussian kernel.

## 4.6 Kalman filter

The belief can be computed as:

$$\text{Bel}(x_t|m) = \eta \Pr(z_t|x_t, m) \int \Pr(x_t|u_t, x_{t-1}, m)\text{Bel}(x_{t-1}|m)\, dx_{t-1}$$

It can be partially computed using the following two formulas:

$$\begin{cases} \bar{\text{Bel}}(x_t|m) = \int \Pr(x_t|u_t, x_{t-1}, m)\text{Bel}(x_{t-1}|m)\, dx_{t-1} \\ \text{Bel}(x_t|m) = \eta \Pr(z_t|x_t, m)\bar{\text{Bel}}(x_t|m) \end{cases}$$

Note that the variable $\eta$ is also part of an integral. While it's impossible to compute these integrals in closed form for continuous distributions, it's feasible to do so at least for Gaussian distributions.

**Univariate Gaussian distribution** The univariate Gaussian distribution, denoted as $X \sim \mathcal{N}(\mu, \sigma^2)$, is described as follows:

$$\Pr(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

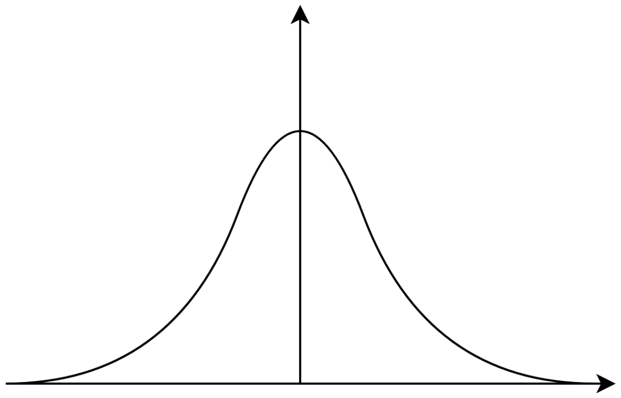


Figure 4.8: Univariate Gaussian distribution

Given an univariate Gaussian distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, the distribution $Y = aX + b$ is still Gaussian and is defined as:

$$Y \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$$

Given two univariate Gaussian distributions $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, the product of these distributions is:

$$\Pr(X_1) \cdot \Pr(X_2) \sim \mathcal{N}\left(\frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}\mu_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}\mu_2, \frac{1}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}\right)$$

**Multivariate Gaussian distribution** The multivariate Gaussian distribution, denoted as $X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, is described as follows:

$$\Pr(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d \left|\boldsymbol{\Sigma}\right|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})^T}$$

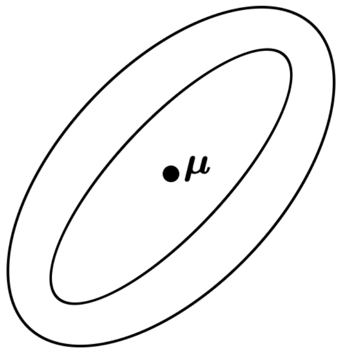


Figure 4.9: Multivariate Gaussian distribution

Given a multivariate Gaussian distribution $X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, the distribution $Y = \mathbf{A}X + \mathbf{B}$ is still Gaussian and is defined as:

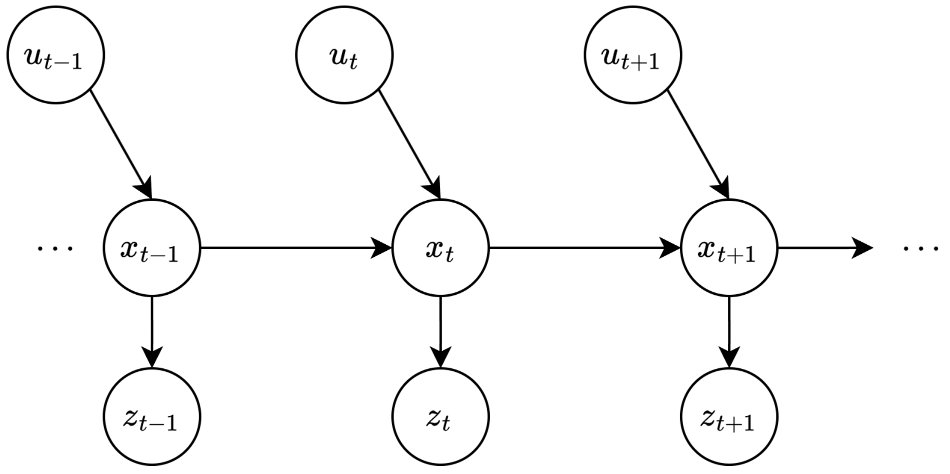
$$Y \sim \mathcal{N}(\mathbf{A}\mu + \mathbf{B}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T)$$

Given two multivariate Gaussian distributions $X_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $X_2 \sim \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, the product between these distributions is:

$$\Pr(X_1) \cdot \Pr(X_2) \sim \mathcal{N}\left(\frac{\boldsymbol{\Sigma}_2}{\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2}\boldsymbol{\mu}_1 + \frac{\boldsymbol{\Sigma}_1}{\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2}\boldsymbol{\mu}_2, \frac{1}{\frac{1}{\boldsymbol{\Sigma}_1} + \frac{1}{\boldsymbol{\Sigma}_2}}\right)$$

### 4.6.1 Discrete time Kalman filter

Consider the scenario depicted below, where $z$ represents observations, $x$ denotes position, and $u$ signifies the action performed:



Let's define the observation as a linear function of the position:

$$z_t = \mathbf{C}_t x_t + \delta_t$$

Here, $C_t$ is a $k \times n$ matrix mapping the state $x_t$ to observation $z_t$, and $\delta_t$ is a random variable representing independent and normally distributed process and measurement noise, with covariance $Q_t$

Similarly, we can model the position of the robot (motion model) as:

$$x_t = \mathbf{A}_t x_{t-1} + \mathbf{B}_t u_t + \varepsilon_t$$

Here, $A_t$ is an $n \times n$ matrix describing state evolution from time $t-1$ to $t$ without controls or noise, $B_t$ is an $n \times l$ matrix illustrating how control $u_t$ affects the state from time $t-1$ to $t$, and $\varepsilon_t$ is a random variable representing independent and normally distributed process and measurement noise, with covariance $R_t$.

The initial belief is normally distributed:

$$\text{Bel}(x_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$$

We assume that observations are linear functions of the state, with additive noise. Thus, the probability of having a certain observation becomes:

$$\Pr(z_t | x_t) = \mathcal{N}(z_t; \mathbf{C}_t x_t, \mathbf{Q}_t)$$

Similarly, we assume that states are linear functions of the previous state and control, with additive noise. Consequently, the probability of being in a certain state becomes:

$$\Pr(x_t | u_t, x_{t-1}) = \mathcal{N}(x_t; \mathbf{A}_t x_{t-1} + \mathbf{B}_t u_t, \mathbf{R}_t)$$

**Closed form prediction** With these assumptions, we can predict the position as follows:

$$\begin{aligned}
\bar{\text{Bel}}(x_t) &= \int \Pr(x_t | u_t, x_{t-1}) \cdot \text{Bel}(x_{t-1})\, dx_{t-1} \\
&= \int \mathcal{N}(x_t; \mathbf{A}_t x_{t-1} + \mathbf{B}_t u_t, \mathbf{R}_t) \cdot \mathcal{N}(z_t; \mathbf{C}_t x_t, \mathbf{Q}_t)\, dx_{t-1} \\
&= \eta \int e^{-\frac{1}{2}(x_t - \mathbf{A}_t x_{t-1} - \mathbf{B}_t u_t)\mathbf{R}_t^{-1}(x_t - \mathbf{A}_t x_{t-1} - \mathbf{B}_t u_t)^T} \cdot e^{-\frac{1}{2}(x_{t-1} - \boldsymbol{\mu}_{t-1})\boldsymbol{\Sigma}_{t-1}^{-1}(x_{t-1} - \boldsymbol{\mu}_{t-1})^T}\, dx_{t-1}
\end{aligned}$$

In the end, the closed-form prediction step is equal to:

$$\bar{\text{Bel}}(x_t) = \begin{cases} \bar{\boldsymbol{\mu}}_t = \mathbf{A}_t \boldsymbol{\mu}_{t-1} + \mathbf{B}_t u_t \\ \bar{\boldsymbol{\Sigma}}_t = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^T + \mathbf{R}_t \end{cases}$$

**Closed form correction** With these assumptions, we can correct the position as follows:

$$\begin{aligned}\text{Bel}(x_t) &= \eta \Pr(z_t|x_t) \cdot \bar{\text{Bel}}(x_t) \\ &= \eta \mathcal{N}(z_t; \mathbf{C}_t x_t, \mathbf{Q}_t) \mathcal{N}(x_t; \bar{\boldsymbol{\mu}}_t, \bar{\boldsymbol{\Sigma}}_t) \\ &= \eta \int e^{-\frac{1}{2}(z_t - \mathbf{C}_t x_t)\mathbf{Q}_t^{-1}(z_t - \mathbf{C}_t x_t)^T} \cdot e^{-\frac{1}{2}(x_t - \bar{\boldsymbol{\mu}}_t)\bar{\boldsymbol{\Sigma}}_t^{-1}(x_t - \bar{\boldsymbol{\mu}}_t)^T} \, dx_{t-1}\end{aligned}$$

In the end, the closed-form update step is equal to:

$$\text{Bel}(x_t) = \begin{cases} \mathbf{K}_t = \bar{\boldsymbol{\Sigma}}_t \mathbf{C}_t^T \left(\mathbf{C}_t \bar{\boldsymbol{\Sigma}}_t \mathbf{C}_t^T + \mathbf{Q}_t\right)^{-1} \\ \boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + \mathbf{K}_t(z_t - \mathbf{C}_t \bar{\boldsymbol{\mu}}_t) \\ \boldsymbol{\Sigma}_t = (I - \mathbf{K}_t \mathbf{C}_t)\bar{\boldsymbol{\Sigma}}_t \end{cases}$$

**Algorithm** With these steps, we can finally construct the Kalman algorithm:

**Algorithm 3** Kalman filter algorithm

```
1: $\bar{\boldsymbol{\mu}}_t = \mathbf{A}_t \boldsymbol{\mu}_{t-1} + \mathbf{B}_t u_t$                                    ▷ Prediction step
2: $\bar{\boldsymbol{\Sigma}}_t = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^T + \mathbf{R}_t$
3: $\mathbf{K}_t = \bar{\boldsymbol{\Sigma}}_t \mathbf{C}_t^T \left(\mathbf{C}_t \bar{\boldsymbol{\Sigma}}_t \mathbf{C}_t^T + \mathbf{Q}_t\right)^{-1}$     ▷ Correction step
4: $\boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + \mathbf{K}_t(z_t - \mathbf{C}_t \bar{\boldsymbol{\mu}}_t)$
5: $\boldsymbol{\Sigma}_t = (I - \mathbf{K}_t \mathbf{C}_t)\bar{\boldsymbol{\Sigma}}_t$
6: return $\boldsymbol{\mu}_t$,$\boldsymbol{\Sigma}_t$
```

The complexity is polynomial in measurement dimensionality $k$ and state dimensionality $n$: $\mathcal{O}(k^{2.376} + n^2)$. It is optimal for linear Gaussian systems.

However, most robotics systems are nonlinear, and the Kalman filter represents unimodal distributions, limiting its applicability in such cases.

### 4.6.2 Extended Kalman filter

In linear systems, Gaussian noise is assumed to follow:

$$\begin{cases} x_t = \mathbf{A}_t x_{t-1} + \mathbf{B}_t u(t) + \varepsilon_t \\ z_t = \mathbf{C}_t x_t + \delta_t \end{cases}$$

However, in non-linear systems, Gaussian noise is distributed as:

$$\begin{cases} x_t = g(u_t, x_{t-1}) \\ z_t = h(x_t) \end{cases}$$

Here, $g(\cdot)$ and $h(\cdot)$ are nonlinear functions.

**Prediction step** In the case of nonlinear functions, the prediction is computed as:

$$\begin{aligned} g(u_t, x_{t-1}) &\approx g(u_t, \boldsymbol{\mu}_{t-1}) + \frac{\partial g(u_t, \boldsymbol{\mu}_{t-1})}{\partial x_{t-1}}(x_{t-1} - \boldsymbol{\mu}_{t-1}) \\ &\approx g(u_t, \boldsymbol{\mu}_{t-1}) + \mathbf{G}_t(x_{t-1} - \boldsymbol{\mu}_{t-1}) \end{aligned}$$

Here, $\mathbf{G}_t$ represents the Jacobian matrix of $g(\cdot)$ evaluated at $\big(\boldsymbol{\mu}_{t-1}, u(t)\big)$.

**Correction step** In case of nonlinear functions we have that the correction is computed as:

$$\begin{aligned} h(x_t) &\approx h(\bar{\boldsymbol{\mu}}_t) + \frac{\partial h(\bar{\boldsymbol{\mu}}_t)}{\partial x_t}(x_t - \bar{\boldsymbol{\mu}}_t) \\ &\approx h(\bar{\boldsymbol{\mu}}_t) + \mathbf{H}_t(x_t - \bar{\boldsymbol{\mu}}_t) \end{aligned}$$

Here, $\mathbf{H}_t$ represents the Jacobian matrix of $h(\cdot)$ evaluated at $\bar{\boldsymbol{\mu}}_t$.

**Algorithm** With these steps, we can finally construct the Extended Kalman Filter algorithm:

**Algorithm 4** Extended Kalman filter algorithm

```
1: $\bar{\boldsymbol{\mu}}_t = g(u_t, \boldsymbol{\mu}_{t-1})$                                        ▷ Prediction step
2: $\bar{\boldsymbol{\Sigma}}_t = \mathbf{G}_t \boldsymbol{\Sigma}_{t-1} \mathbf{G}_t^T + \mathbf{R}_t$
3: $\mathbf{K}_t = \bar{\boldsymbol{\Sigma}}_t \mathbf{H}_t^T \left(\mathbf{H}_t \bar{\boldsymbol{\Sigma}}_t \mathbf{H}_t^T + \mathbf{Q}_t\right)^{-1}$          ▷ Correction step
4: $\boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + \mathbf{K}_t(z_t - h(\bar{\boldsymbol{\mu}}_t))$
5: $\boldsymbol{\Sigma}_t = (I - \mathbf{K}_t \mathbf{H}_t)\bar{\boldsymbol{\Sigma}}_t$
6: return $\boldsymbol{\mu}_t$,$\boldsymbol{\Sigma}_t$
```

Here, $\mathbf{G}_t = \frac{\partial g(u_t, \boldsymbol{\mu}_{t-1})}{\partial x_{t-1}}$, and $\mathbf{H}_t = \frac{\partial h(\bar{\boldsymbol{\mu}}_t)}{\partial x_t}$. Extended Kalman Filter indeed exhibits polynomial complexity in measurement dimension $k$ and state dimension $n$, typically $\mathcal{O}(k^{2.376} + n^2)$. However, it's not optimal and can diverge if nonlinearities are significant. Despite its shortcomings, the Extended Kalman Filter often performs well even when some assumptions are violated.

As alternatives, methods like the Unscented Kalman Transform are employed, offering more robustness in handling nonlinearity and uncertainties.

## 4.7 Particle filter

Particle Filters are a methodology utilized for estimating non-Gaussian and nonlinear processes by representing belief through random samples. This technique, also known as Monte Carlo filter, embodies the concept of survival of the fittest, where the most informative particles prevail. Through iterations, the process of condensation refines the particle distribution, enhancing accuracy. Another term often used interchangeably is the Bootstrap filter, highlighting the resampling technique at its core.

The computation of belief can be expressed as:

$$\text{Bel}(x_t) = \eta \Pr(z_t|x_t) \int \Pr(x_t|x_{t-1}, u_{t-1}) \cdot \text{Bel}(x_{t-1})\, dx_{t-1}$$

For the particle filter we select:

1. Select a sample $x_{t-1}^i$ from $\text{Bel}(x_{t-1})$.

2. Choose a sample $x_t^i$ from $\Pr(x_t|x_{t-1}^i, u_{t-1})$.

3. Compute the importance factor $w_i^t$ for the sample $x_t^i$ as:

$$w_t^i = \frac{\text{target distribution}}{\text{proposal distribution}} = \frac{\eta \Pr(z_t|x_t) \Pr(x_t|x_{t-1}, u_{t-1}) \text{Bel}(x_{t-1})}{\Pr(x_t|x_{t-1}, u_{t-1}) \text{Bel}(x_{t-1})} \propto \Pr(z_t|x_t)$$

**Algorithm** The particle filter algorithm can be executed in the following manner:

**Algorithm 5** Particle filter algorithm

```
1: $S_t = \varnothing$
2: $\eta = 0$
3: for $i = 1$ to $n$ do                                   ▷ Generate new samples
4:     Sample index $j(i)$ from the discrete distribution given by $w_{t-1}$
5:     Sample $x_t^i$ from $\Pr(x_t|x_{t-1}, u_{t-1})$ using $x_{t-1}^{j(i)}$ and $u_{t-1}$
6:     $w_t^i = \Pr(z_t|x_t^i)$                             ▷ Compute importance weight
7:     $\eta = \eta + w_t^i$                                ▷ Update normalization factor
8:     $S_t = S_t \cup \{\langle x_t^i, w_t^i \rangle\}$    ▷ Insert
9: end for
10: for $i = 1$ to $n$ do
11:     $w_t^i = \frac{w_t^i}{\eta}$                        ▷ Normalize weights
12: end for
```

**Monte Carlo localization** Monte Carlo localization, also known as particle filter localization, is an algorithm utilized by robots to determine their position and orientation within an environment. This method leverages a particle filter to estimate the robot's pose as it moves and gathers sensory information within a mapped environment.

CHAPTER 5

# Simultaneous Localization and Mapping

## 5.1 Introduction

One approach to creating a map involves measuring distances from landmarks, adjusting the position slightly, and then measuring again. However, the challenge lies in the inherent noise in these measurements, making it impossible to achieve a perfectly accurate map. Additionally, this method assumes perfect knowledge of the robot's pose, which is unattainable in practical scenarios.

Alternatively, a grid map can be employed, which discretizes the environment into a grid of cells. Each cell is either activated with a value of 1 if occupied or 0 if not. Another option is to utilize a grid map that incorporates occupancy probability.

### 5.1.1 Occupancy from sonar return

Within the framework of sonar-based occupancy estimation, two key components are employed:

- A 2D Gaussian distribution is utilized to model occupied space.
- Another 2D Gaussian distribution is employed to represent free space.

However, the integration of sonar sensors introduces several challenges. Wide sonar cones contribute to noisy maps, while specular (multi-path) reflections generate unrealistic measurements.

### 5.1.2 Two-dimensional occupancy grid

A straightforward representation for maps in two dimensions involves an occupancy grid, where each cell is treated as independent. The probability of a cell being occupied is typically estimated using Bayes' theorem:

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B|A)\Pr(A) + \Pr(B|\sim A)\Pr(\sim A)}$$

The environment is mapped as an array of cells, with each cell representing a specific area. Typically, the size of each cell ranges from 5 to 50 centimeters. Within this array, each cell contains a probability value representing the likelihood of the cell being occupied. This approach is particularly useful for integrating data from various sensors and sensor modalities to create a comprehensive map.

**Cell occupancy** The occupancy `occ(i,j)` of a cell with coordinates $(i, j)$ in the grid can be computed in several ways:

1. *Probability*: this represents the likelihood of the cell being occupied, ranging between zero and one:
$$\Pr\left(occ\left(i,j\right)\right) \in [0,1]$$
2. *Odds*: This method calculates the ratio of the probability of the cell being occupied to the probability of it not being occupied. It offers an advantage over the probability method since updates can be applied to either the numerator or the denominator independently:
$$o(occ(i,j)) = \frac{\Pr\left(occ\left(i,j\right)\right)}{\Pr\left(\neg occ\left(i,j\right)\right)} \in [0,+\infty]$$
3. *Log odds*: Log odds: a variation of the odds that can take any real number as its value. If it's more probable for a cell to be unoccupied, the value is negative; otherwise, it's positive:
$$\log o(occ(i,j)) \in [-\infty,+\infty]$$

## 5.2 Mapping

Initially, the log odds are set to zero, corresponding to a probability of 50%. Subsequently, the cells are recursively updated using Bayes' theorem:
$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)}$$
Here, $A$ denotes the actual occupancy value of the cell, while $B$ represents the new measurement for the cell.

The challenge with this approach arises from the fact that real-world robots lack precise localization, making it impossible to generate a usable map.

## 5.3 Scan matching

To address the challenge of perfect pose assumption, we can alternate between localization and mapping in the following manner:

1. Correct odometry by maximizing the likelihood of pose $x_t$ based on the estimates of pose and map at time $t-1$:
$$\hat{x}_t = \underset{x_t}{\operatorname{argmax}} \left\{\Pr(z_t|X_t,\hat{m}_{t-1})\Pr(x_t|u_{t-1}\hat{x}_{t-1})\right\}$$
Here, $\hat{m}_{t-1}$ represents the map constructed so far.
2. Compute the map $\hat{m}_t$ based on mapping with known poses, leveraging the new pose and current observations.

However, this method fails to adequately track uncertainty during the odometry correction process. While the final map will likely be superior to those obtained with naive methods, uncertainty could lead to ambiguously defined obstacles and duplications.

## 5.4 Simultaneous Localization and Mapping

In Simultaneous Localization and Mapping (SLAM), our goal is to compute both the trajectory and the map simultaneously.

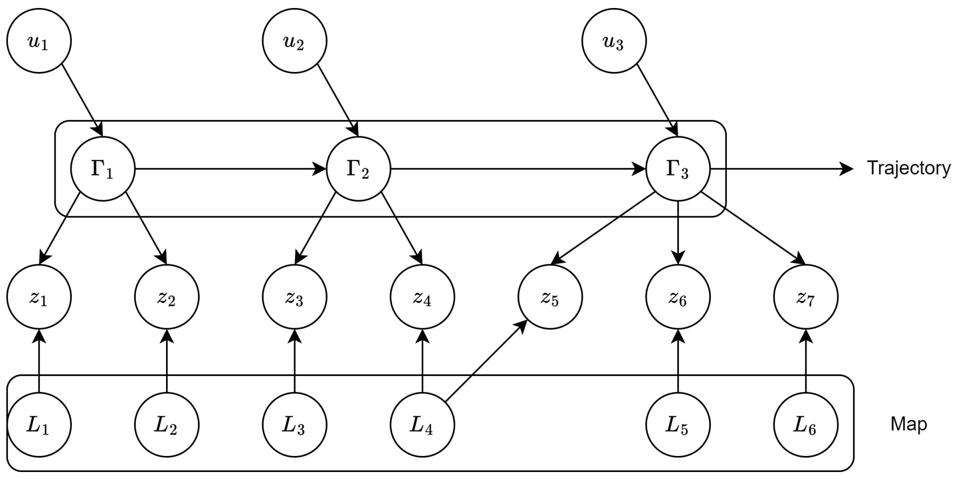


Figure 5.1: Simultaneous Localization and Mapping

**Full SLAM** A full SLAM is achieved when both the map and trajectory are fully computed. In this scenario, the probability

$$\Pr(x_{1:t}, m|z_{1:t}, u_{1:t})$$

is known for each time instant. To tackle this problem, FastSLAM can be employed, leveraging a sampled particle filter distribution model.

**Online SLAM** Online SLAM occurs when we possess the full map but only have knowledge of the current pose. In this scenario, we achieve a simultaneous estimate of the most recent pose and map:

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

The integrals are computed sequentially, but if they become overly complex, then opting for full SLAM is preferable. For this task, Extended Kalman Filter (EKF) SLAM can be employed, utilizing a linearized Gaussian probability distribution.

### 5.4.1 Online SLAM

For Extended Kalman Filter SLAM, we define a Belief matrix. Considering a map with $N$ landmarks, we need to define a $(3+2N)$-dimensional Gaussian with the following Belief matrix:

$$\text{Bel}(x_t, m_t) = \left\langle \begin{bmatrix} x \\ y \\ \theta \\ l_1 \\ l_2 \\ \vdots \\ l_N \end{bmatrix}, \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xl_1} & \sigma_{xl_2} & \cdots & \sigma_{xl_N} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\theta} & \sigma_{yl_1} & \sigma_{yl_2} & \cdots & \sigma_{yl_N} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_\theta^2 & \sigma_{\theta l_1} & \sigma_{\theta l_2} & \cdots & \sigma_{\theta l_N} \\ \sigma_{xl_1} & \sigma_{yl_1} & \sigma_{\theta l_1} & \sigma_{l_1}^2 & \sigma_{l_1 l_2} & \cdots & \sigma_{l_1 l_N} \\ \sigma_{xl_2} & \sigma_{yl_2} & \sigma_{\theta l_2} & \sigma_{l_1 l_2} & \sigma_{l_2}^2 & \cdots & \sigma_{l_2 l_N} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{xl_N} & \sigma_{yl_N} & \sigma_{\theta l_N} & \sigma_{l_1 l_N} & \sigma_{l_2 l_N} & \cdots & \sigma_{l_N}^2 \end{bmatrix} \right\rangle$$

Here, $\begin{bmatrix} x & y & \theta \end{bmatrix}^T$ represents the robot's position, and $\begin{bmatrix} l_1 & \cdots & l_N \end{bmatrix}^T$ represents the positions of landmarks. The top left and bottom right parts represent the covariances of the robot poses and the landmark positions, respectively. The other two sub-matrices represent the covariances between the robot and the landmarks. The covariance between the robot pose and landmark position is not null since the uncertainty lies in the measurements made from the robot itself.

Additionally, we assume the robot's position to be a linear function $x_t = \mathbf{A}_t x_{t-1} + \mathrm{B}_t u(t) + \varepsilon_t$. The measurement is also assumed to be a linear function $z_t = \mathbf{C}_t x_t + \delta_t$. Therefore, the belief is updated as follows:

$$\begin{cases} \Pr(x_t|u_t, x_{t-1}) = \mathcal{N}(x_t; \mathbf{A}_t x_{t-1} + \mathrm{B}_t u(t), \mathbf{R}_t) \\ \Pr(z_t|x_t) = \mathcal{N}(z_t; \mathbf{C}_t x_t, \mathbf{Q}_t) \end{cases}$$

### 5.4.2 Bayes filter algorithm for EKF SLAM

The Bayes filter algorithm for Simultaneous Localization and Mapping mirrors the one used for localization.

### 5.4.3 Kalman filter algorithm for EKF SLAM

The Kalman filter algorithm for Simultaneous Localization and Mapping closely resembles the one used for localization. The key difference lies in the handling of the Belief matrix, which contains all the covariances and can be updated dynamically during the algorithm's execution. In terms of complexity, there isn't much deviation from standard EKF, but the state dimension increases.

**Extended Kalman filter algorithm** To address complexity concerns, we can approximate the SLAM posterior with a high-dimensional Gaussian using the Extended Kalman filter algorithm.

**Properties of Kalman Filter SLAM** The linear Kalman filter for SLAM is characterized by the following theorems.

**Theorem 5.4.1.** *The determinant of any sub-matrix of the map covariance matrix decreases monotonically as successive observations are made.*

**Theorem 5.4.2.** *In the limit, the landmark estimates become fully correlated.*

These theorems have several implications:

- Quadratic complexity in the number of landmarks: $\mathcal{O}(n^2)$.
- Convergence outcomes are established for the linear scenario.
- Divergence may occur when dealing with significant nonlinearities.
- Successful applications have been observed in large-scale environments.
- Computational complexity is mitigated through the use of approximations.

Currently, EKF SLAM finds its niche in sparse landmark-based maps, where it exhibits its highest efficiency.

**Monocular SLAM** Monocular SLAM is a technique used in robotics and computer vision to create maps of an environment while simultaneously determining the position and orientation of the observer within that environment using a single camera. This is achieved by correlating visual features in successive frames of video to track movement and construct a map of the surroundings. Initially, monocular SLAM algorithms relied on Extended Kalman Filter (EKF) SLAM to estimate the pose of the camera. This technique necessitates that the images captured contain identifiable landmarks or features.

### 5.4.4 Summary

While EKF-SLAM demonstrates effectiveness, it relies on linearized models of nonlinear motion and observation, thus inheriting various limitations. Computational demands increase significantly, scaling quadratically with the number of landmarks. Several potential solutions have been proposed:

- Employing local sub-maps.
- Utilizing sparse links (correlations).
- Implementing sparse extended information filters.
- Applying Rao-Blackwellisation (FastSLAM), which represents nonlinear processes and non-Gaussian uncertainty, thereby reducing computational burden.

## 5.5 Full SLAM

In the general case, we encounter:

$$\Pr(x_t, m|z_t) \neq \Pr(x_t|z_t)\Pr(m|z_t)$$

This disparity arises due to the correlation between position and pose. However, when considering the entire trajectory $X_t$ rather than just a single pose $x_t$, we find:

$$\Pr(X_t, m|z_t) \neq \Pr(X_t|z_t)\Pr(m|X_t, z_t)$$

Thus, in this scenario, the map becomes independent of the current state. In FastSLAM, the trajectory $X_t$ is represented by particles $X_t^{(i)}$, while the map is represented by a factorization known as the Rao-Blackwellized Filter:

$$\Pr\left(m|X_t^{(i)}, z_t\right) = \prod_j^M \Pr\left(m_j|X_t^{(i)}, z_t\right)$$

Here, $\Pr\left(X_t|z_t\right)$ is computed using particles, and $\Pr\left(m|X_t, z_t\right)$ is estimated using an Extended Kalman Filter.

To decouple the map of features from poses, each particle represents a robot trajectory. Feature measurements are correlated throughout the robot trajectory. If the robot trajectory is known, all features would be uncorrelated. Treat each pose particle as if it is the true trajectory, processing all feature measurements independently.

**SLAM posterior** The factored posterior is given by:

$$\begin{aligned}\Pr(x_{1:t}, l_{1:m}|z_{1:t}, u_{0:t-1}) &= \Pr(x_{1:t}|z_{1:t}, u_{0:t-1}) \Pr(l_{1:m}|x_{1:t}, z_{1:t}) \\ &= \Pr(x_{1:t}|z_{1:t}, u_{0:t-1}) \prod_{i=1}^{M} \Pr(l_i|x_{1:t}, z_{1:t})\end{aligned}$$

Here, $\Pr(x_{1:t}|z_{1:t}, u_{0:t-1})$ represents the robot path posterior (localization problem), and $\Pr(l_i|x_{1:t}, z_{1:t})$ denotes the conditionally independent landmark positions. The dimension of conditionally independent landmark positions is $m$.

### 5.5.1 FastSLAM

In practice, FastSLAM is implemented via Rao-Blackwellized particle filtering based on landmarks. Each particle represents a trajectory, consisting of the last pose and a reference to the previous one. Each landmark is represented by a $2 \times 2$ Extended Kalman Filter (EKF). Therefore, each particle needs to maintain $M$ EKFs.

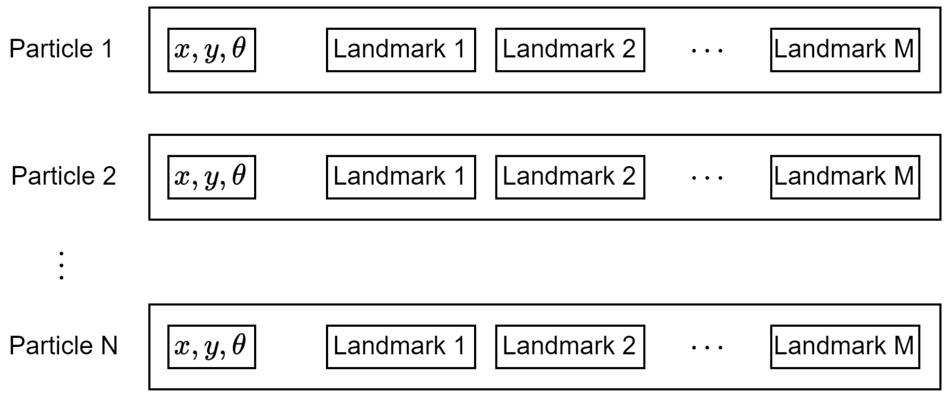


Figure 5.2: Particles in FastSLAM

**Complexity** The update of the robot particles depends on the number of particles $N$, leading to a complexity of $\mathcal{O}(N)$. Incorporating observations into the Kalman filter involves updating a section of the map features $M$, resulting in a complexity of $\mathcal{O}(N \log M)$. Resampling the particle set also requires the same complexity of $\mathcal{O}(N \log M)$. Therefore, the total complexity of the algorithm is:

$$\mathcal{O}(N \log M)$$

Here, $N$ is the number of particles, and $M$ is the number of map features.

**Improvements** Since each particle contains a map of the entire environment, and each grid is independent from the others, we can select the best map at each time by updating all the maps.

## 5.6 PoseGraph SLAM

Consider the problem of full SLAM:

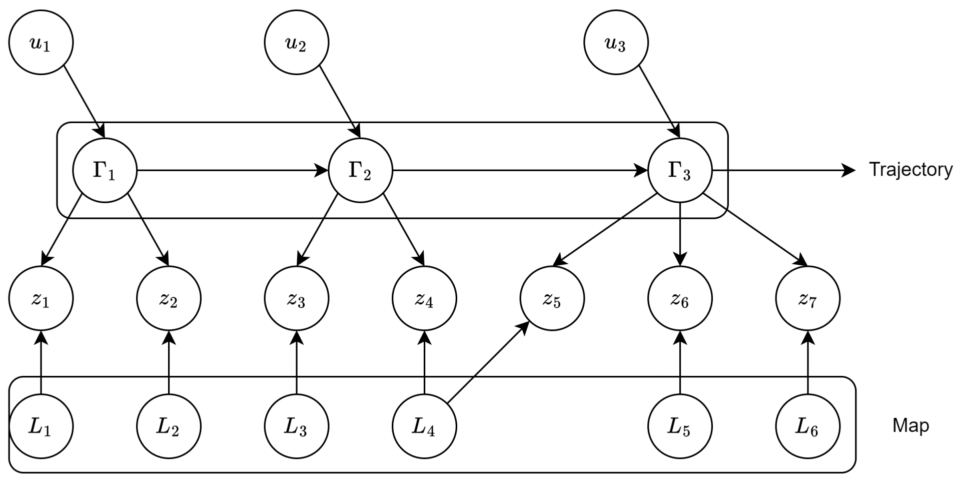


Figure 5.3: Simultaneous Localization and Mapping

In Full SLAM we model:

$$\Pr(X|Z,U) = \Pr(\Gamma_{0:t}, L_1, \dots, L_n | z_{1:t}, u_{1:t})$$

then we look for the most likely solution

$$X^{MAP} = \underset{X}{\operatorname{argmax}} \Pr(X|Z,U)$$

This can be rewritten using the Bayes' rule as

$$X^{MAP} = \underset{X}{\operatorname{argmax}} \Pr(X,Z,U)$$

The maximum of this probability will be located in the same spot of the previous, since the denominator is only a normalizing factor and do not depends on $X$.

The full joint distribution of a Bayesian network is the product of the conditionals:

$$\begin{aligned}\Pr(X,Z,U) = \Pr(\Gamma_{0:3}, L_1, \dots, L_6 Z_{1:7}, u_{1:7}) = \Pr(\Gamma_1|\Gamma_0,u_1)\Pr(\Gamma_2|\Gamma_1,u_1)\Pr(\Gamma_3|\Gamma_2,u_3)\\ \Pr(Z_1|\Gamma_1,L_1)\Pr(L_1)\Pr(Z_2|\Gamma_1,L_2)\Pr(L_2)\Pr(Z_3|\Gamma_2,L_3)\Pr(L_3)\\ \Pr(Z_4|\Gamma_2,L_4)\Pr(L_4)\Pr(Z_5|\Gamma_3,L_4)\Pr(Z_6|\Gamma_3,L_5)\Pr(L_5)\Pr(Z_7|\Gamma_3,L_6)\Pr(L_6)\end{aligned}$$

By using the Bayes' rule we obtain that it is the product of factors:

$$\begin{aligned}\Pr(X,Z,U) = \phi(\Gamma_1,\Gamma_0,u_1)\phi(\Gamma_2,\Gamma_1,u_1)\phi(\Gamma_3,\Gamma_2,u_3)\\ \phi(Z_1,\Gamma_1,L_1)\phi(L_1)\phi(Z_2,\Gamma_1,L_2)\phi(L_2)\phi(Z_3,\Gamma_2,L_3)\phi(L_3)\\ \phi(Z_4,\Gamma_2,L_4)\phi(L_4)\phi(Z_5,\Gamma_3,L_4)\phi(Z_6,\Gamma_3,L_5)\phi(L_5)\phi(Z_7,\Gamma_3,L_6)\phi(L_6)\end{aligned}$$

At this point the Bayesian Network becomes:

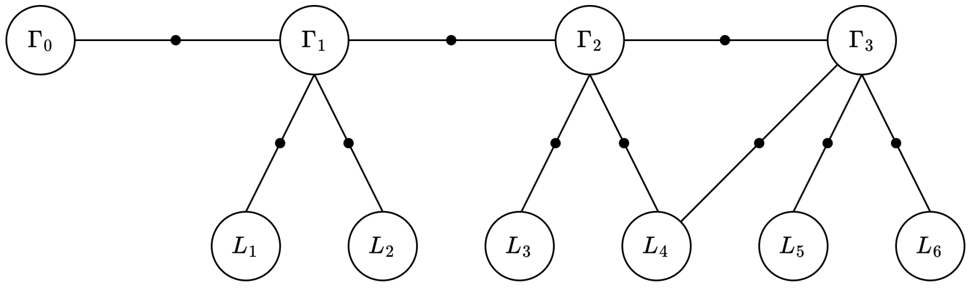


Figure 5.4: Factorized Simultaneous Localization and Mapping

Given the Factor Graph Full Joint Distribution:

$$\Pr(X, Z, U) = \prod_i \phi_i(X_i)$$

The Full SLAM problem is reformulated as

$$X^{MAP} = \underset{X}{\operatorname{argmax}} \Pr(X|Z, U) = \underset{X}{\operatorname{argmax}} \Pr(X, Z, U) = \underset{X}{\operatorname{argmax}} \prod_i \phi_i(X_i)$$

Let's also assume to have Gaussian Factors (not mandatory but convenient):

$$\phi(\Gamma_1, \Gamma_0, u_1) = \mathcal{N}(g(\Gamma_0, u_1), R) = \frac{1}{\sqrt{|2\pi R|}} e^{-\frac{1}{2}(g(\Gamma_0,u_1)-\Gamma_1)^T R^{-1}(g(\Gamma_0,u_1)-\Gamma_1)}$$

$$\phi(Z_1, \Gamma_1, L_1) = \mathcal{N}(h(\Gamma_1, L_1), Q) = \frac{1}{\sqrt{|2\pi Q|}} e^{-\frac{1}{2}(h(\Gamma_1,L_1)-Z_1)^T Q^{-1}(h(\Gamma_1,L_1)-Z_1)}$$

The Odometry factors are proportional to the exponential:

$$\phi_{i=u_i} \propto e^{-\frac{1}{2}(g_i(X_i)-\Gamma_i)^T R^{-1}(g_i(X_i)-\Gamma_i)}$$

The measurement factors are proportional to the exponential:

$$\phi_{i=z_i} \propto e^{-\frac{1}{2}(h_i(X_i)-Z_i)^T Q^{-1}(h_i(X_i)-Z_i)}$$

The (Gaussian) Full SLAM problem becomes:

$$\begin{aligned} X^{MAP} &= \underset{X}{\operatorname{argmax}} = \prod_i \phi_i(X_i) = \underset{X}{\operatorname{argmax}} \\ &= \prod_i e^{-\frac{1}{2}(g_i(X_i)-\Gamma_i)^T R^{-1}(g_i(X_i)-\Gamma_i)} \prod_i e^{-\frac{1}{2}(h_i(X_i)-Z_i)^T Q^{-1}(h_i(X_i)-Z_i)} \end{aligned}$$

If we solve for the logarithm we get a simpler optimization algorithm:

$$\begin{aligned} X^{MAP} &= \underset{X}{\operatorname{argmax}} \prod_i \phi_i(X_i) = \underset{X}{\operatorname{argmax}} \log \prod_i \phi_i(X_i) = \underset{X}{\operatorname{argmax}} \sum_i \log \phi_i(X_i) \\ &= \underset{X}{\operatorname{argmax}} \left[ \sum_{i=u_i} \log \left( e^{-\frac{1}{2}\|g_i(X_i)-\Gamma_i\|_R^2} \right) + \sum_{i=z_i} \log \left( e^{-\frac{1}{2}\|h_i(X_i)-Z_i\|_Q^2} \right) \right] \\ &= \underset{X}{\operatorname{argmax}} \left[ \sum_{i=u_i} \left( -\frac{1}{2} \|g_i(X_i) - \Gamma_i\|_R^2 \right) + \sum_{i=z_i} \left( -\frac{1}{2} \|h_i(X_i) - Z_i\|_Q^2 \right) \right] \\ &= \underset{X}{\operatorname{argmin}} \left[ \sum_{i=u_i} \|g_i(X_i) - \Gamma_i\|_R^2 + \sum_{i=z_i} \|h_i(X_i) - Z_i\|_Q^2 \right] \end{aligned}$$

That is the nonlinear least squares on a graph. Sometimes landmarks get attached to poses in PoseGraph SLAM.

### 5.6.1 Computational complexity

Solving non-linear least squares needs iterative adjustments (gradient descend).

**Measurement factors** Let's focus on measurement factors, then the following extends to all factors

$$h_i(X_i) = h_i(X_i^0) + \Delta_i \approx h_i(X_i^0) + H_i\Delta_i$$

Here, $\Delta_i = X_i - X_i^0$, and $H_i$ is the Hessian computed in $X_i^0$. We look for the single adjustment step which minimizes all measurement factors:

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum_{i=z_i} \|h_i(X_i) - Z_i\|_Q^2 = \underset{\Delta}{\operatorname{argmin}} \sum_{i=z_i} \left\| \underbrace{H_i\Delta_i - \left(Z_i - h_i(X_i^0)\right)}_{e_i} \right\|_Q^2$$

That is equivalent to minimize the error. We can rewrite the Mahalanobis norm as it follows turning it into quadratic:

$$\|e_i\|_Q^2 = e_i^T Q e_i = \left(Q^{-\frac{1}{2}} e_i\right)^T \left(Q^{-\frac{1}{2}} e_i\right) = \left\|Q^{-\frac{1}{2}} e\right\|_2^2$$

We can now rewrite the minimization as:

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum_{i=z_i} \left\| Q_i^{-\frac{1}{2}} H_i\Delta_i - Q_i^{-\frac{1}{2}} \left(Z_i - h_i(X_i^0)\right)\right\|_2^2$$

By imposing $A_i = Q_i^{-\frac{1}{2}} H_i$ and $b_i = Q_i^{-\frac{1}{2}} (Z_i - h_i(X_i^0))$ we get:

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum_{i=z_i} \|A_i\Delta_i - b_i\|_2^2 = \underset{\Delta}{\operatorname{argmin}} \sum_{i=z_i} \|A\Delta - B\|_2^2$$

This is Linear least squares problem. Let's assume Odometry is included too from now on.

Let's solve the least squares problem

$$\|A\Delta - B\|_2^2 = (A\Delta - B)^T(A\Delta - B) = \Delta^T A^T A\Delta - 2\Delta^T A^T B + BB^T$$

Deriving for $Delta$ and imposing the derivative to zero we get:

$$\frac{\partial \|A\Delta - B\|_2^2}{\partial\Delta} = 0 \rightarrow \Delta = A^T B (A^T A)^{-1}$$

Matrix $A$ from Odometry and Measurement Jacobians Factors are constraints between 2 variables Matrix $A$ is sparse and matrix $A^T A$ too We can use sparse methods which are fast .

**COmplexity** Naïve least squares uses pseudo inverse, however $(A^T A)^{-1}$ is $\mathcal{O}(n^3)$. Cholesky decomposition $A^T A = R^T R$ ($R$ upper triangular) is $\mathcal{O}(n^{1.5})$ to $\mathcal{O}(n^2)$. Solve by forward / backward substitution and via $LDL^T$ decomposition is even faster.

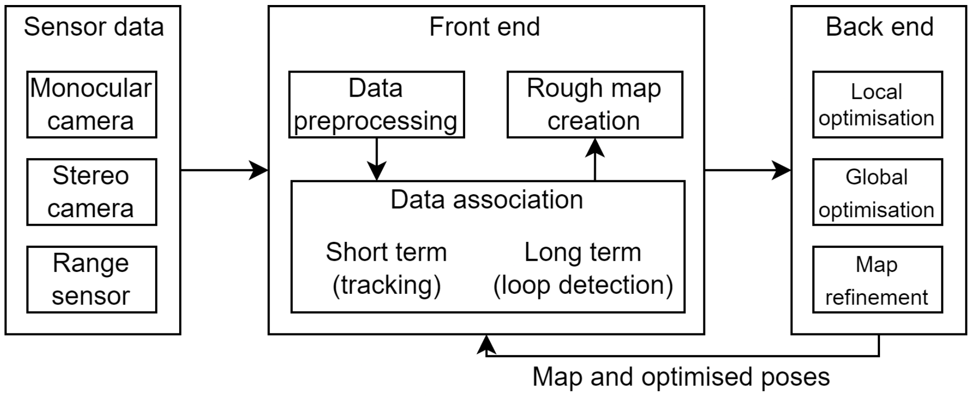


Figure 5.5: Modern SLAM System

CHAPTER 6

# Robot motion control and planning

## 6.1 Introduction

Robot motion control can be implemented in two primary ways: open loop and feedback.

### 6.1.1 Open loop control

In open loop control, the mobile robot follows a predetermined trajectory computed beforehand, typically composed of motion segments from the starting point to the destination. The robot executes this planned trajectory without feedback until it reaches the goal.

The main challenges associated with open loop control include:

- Difficulty in precomputing a feasible trajectory.
- Constraints and limitations on the robot's velocities and accelerations.
- Inability to handle dynamic changes such as obstacles.
- Lack of error recovery mechanisms.

### 6.1.2 Feedback control

In contrast, feedback control involves recomputing or adapting the trajectory online, utilizing a simple control scheme for path following. This approach adjusts the robot's orientation by modulating angular velocity and controls its distance by adjusting linear velocity.

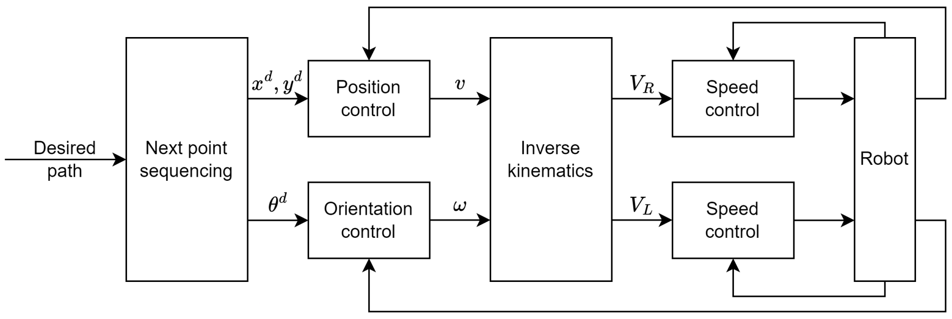


Figure 6.1: Feedback control for a differential drive robot

The primary challenge within this framework arises from the occurrence of temporary obstacles that alter the planned path.

## 6.2 Local path planning

Obstacle avoidance should follow the planned path and simultaneously avoid any unexpected obstacle that is not in the map. There are several proposed methods in the literature, the main ones are:

- Potential field methods.
- Vector field histogram.
- Curvature-Velocity.
- Nearness diagram.
- Dynamic Window Approach.

**Bug-like robots** Bug-like robots operate with limited knowledge: they are aware of the direction to the goal and possess local sensing capabilities for detecting obstacles using encoders. Additionally, their world adheres to certain constraints: obstacles are finite within any finite range, and a line intersects an obstacle a finite number of times. The fundamental concept is to alternate between two primary behaviors:

1. Directing toward the goal.
2. Following obstacles until a clear path toward the goal is available again.

### 6.2.1 Vector Field Histograms

Vector Field Histograms (VFH) utilize a local map of the environment to determine the optimal angle for driving. The environment is typically represented in a grid format (2 degrees of freedom) using local measurements, with all openings for the robot to pass through identified.

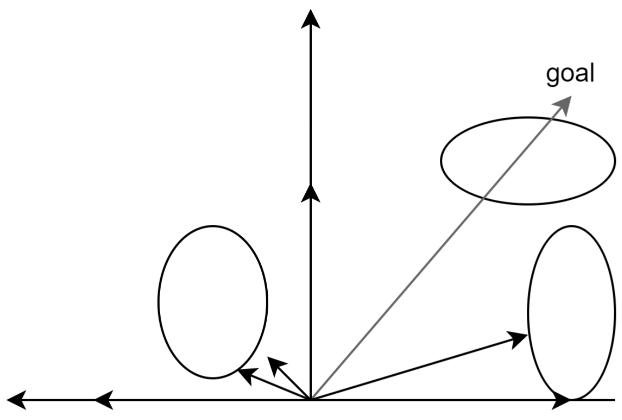


Figure 6.2: Vector Field Histograms

To select the best angle for driving, a cost function needs to be defined and minimized:

$$G = a \cdot \text{target direction} + b \cdot \text{wheel orientation} + c \cdot \text{previous direction}$$

This function consists of three components:

- $a \cdot$ target direction considers the alignment of the robot's path with the goal.
- $b \cdot$ wheel orientation accounts for the difference between the new direction and the current wheel orientation.
- $c \cdot$ previous direction takes into account the difference between the previously selected direction and the new direction.

The function computes the probability of hitting an obstacle and considers only the directions below a fixed threshold value.

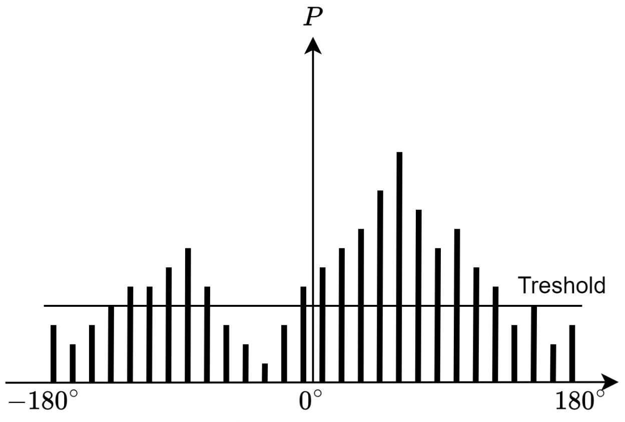


Figure 6.3: Vector Field Histograms graph

This algorithm executes in constant time.

### 6.2.2 Curvature Velocity Methods

Curvature Velocity Methods incorporate physical constraints imposed by both the robot and the environment on the linear and angular velocities $(v, \omega)$. These methods assume that the robot moves along arcs (where curvature $c = \frac{\omega}{v}$) with constraints on acceleration. Obstacles are transformed into velocity space, and an objective function is employed to select the optimal speed.

### 6.2.3 Vector Field Histogram Plus

VFH+ extends the capabilities of traditional VFH by incorporating vehicle kinematics. This includes considerations for a robot moving on arcs or straight lines. In VFH+, obstacles that block a particular direction impede all trajectories. Moreover, obstacles are enlarged to encompass all kinematically blocked trajectories.

**Limitations** However, VFH+ shares some limitations with VFH:

- Challenges arise when navigating through narrow areas like doors.
- Local minima might not be successfully avoided.
- There is no guarantee of reaching the goal.
- The dynamics of the robot are not fully considered.

### 6.2.4 Dynamic Window Approach

In the Dynamic Window Approach (DWA), the robot's kinematics are taken into account through a local search in velocity space. This approach considers only circular trajectories represented by pairs of linear and angular speeds $V_s = (v, \omega)$. A velocity pair $V_a = (v, \omega)$ is deemed admissible if the robot can come to a stop before reaching the closest obstacle. A dynamic window constrains the reachable velocities $V_d$ to those achievable within a short time frame given the robot's limited accelerations:

$$V_d = \begin{cases} v \in [v - a_{tr} \cdot t, v + a_{tr} \cdot t] \\ \omega \in [\omega - a_{rot} \cdot t, \omega + a_{rot} \cdot t] \end{cases}$$

Graphically, the scenario can be illustrated as follows:

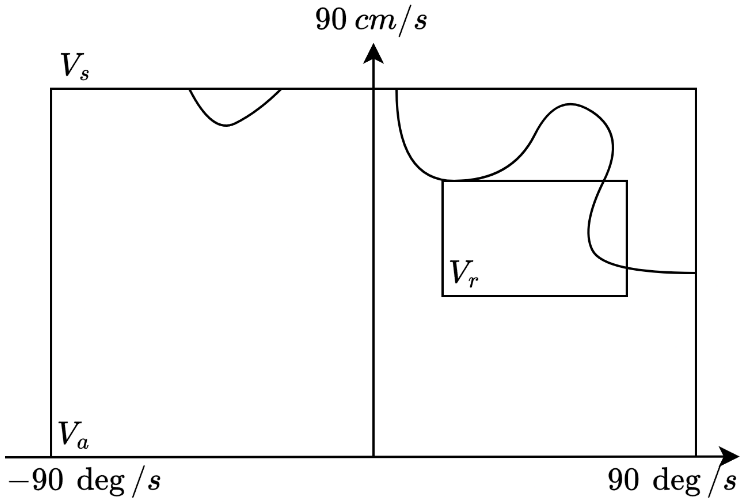


Figure 6.4: Dynamic Window Approach search space

The objective is to find a feasible point within the search space defined as:

$$V_r = V_s \cap V_a \cap V_d$$

**Optimal combination** The selection of the best pair $(v, \omega)$ is determined by maximizing a heuristic navigation function that minimizes travel time by driving quickly in the correct direction. Planning is confined to the $V_r$ space, so the objective function is:

$$G(v, \omega) = \sigma \left(\alpha \cdot \text{heading}(v, \omega) + \beta \cdot \text{distance}(v, \omega) + \gamma \cdot \text{velocity}(v, \omega)\right)$$

Here, $\sigma(\cdot)$ is utilized because the function returns a probability. In this formulation:

- $\alpha \cdot \text{heading}(v, \omega)$: considers alignment with the target direction.
- $\beta \cdot \text{distance}(v, \omega)$: considers the distance to the closest obstacle intersecting with curvature.
- $\gamma \cdot \text{velocity}(v, \omega)$: considers the forward velocity of the robot.

While this function provides local optimality, it may not be precise in selecting the best trajectory. To address this, a global approach using a navigation function in two-dimensional spaces can be employed:

$$\text{NF} = \alpha \cdot \text{velocity} + \beta \cdot \text{nf} + \gamma \cdot \Delta\text{nf} + \delta \cdot \text{goal}$$

Here:

- $\alpha \cdot$ velocity: considers the forward robot velocity.
- $\beta \cdot$ nf: evaluates the cost to reach the goal.
- $\gamma \cdot \Delta$nf: assesses following the global path.
- $\delta \cdot$ goal: takes into account the proximity to the goal.

**Dynamic Window Approach via trajectory rollout** To estimate the trajectory, we employ the fundamental concept of the Dynamic Window Approach but with sampled control inputs. The process unfolds as follows:

1. Discretely sample the robot's control space.
2. For each sampled velocity, conduct a forward simulation to predict the outcome if applied for a short duration.
3. Evaluate each trajectory resulting from the forward simulation.
4. Discard illegal trajectories, namely those that intersect with obstacles, and select the trajectory with the highest score.

This approach is versatile and can handle non-circular trajectories as well. For instance, it can accommodate a clothoid, which is a trajectory characterized by a circular path with a radius that changes linearly.

## 6.3 Global path planning

The objectives for robot motion planning are: to generate trajectories that are free from collisions and to guide the robot to its goal location as swiftly as possible, or by maximizing an optimality criterion.

The overarching objective is to identify a path devoid of collisions between an initial pose and the goal, while adhering to various constraints such as geometric, physical, and temporal limitations.

**Definition** (*Path*)**.** A path represents a series of waypoints in a defined space that the vehicle must traverse.

**Definition** (*Trajectory*)**.** A trajectory encompasses a path with a specified temporal law, detailing factors like acceleration and velocity at each waypoint.

**Definition** (*Maneuver*)**.** A maneuver denotes a sequence of actions or a plan that the vehicle should execute to navigate successfully.

### 6.3.1 Motion planning

In the context provided:

- $A$ represents a single rigid object (the robot).
- $W$ denotes the Euclidean space where $A$ operates.
- $B_1, B_2, \cdot, B_m$ are fixed rigid objects distributed within $W$ (obstacles).

Assumptions:

- The geometry of both $A$ and $B_i$ is known.
- The precise localization of $B_i$ within $W$ is available.
- There are no kinematic constraints on the motion of $A$; it is a free-flying object.

The objective is, given an initial pose and a goal pose of $A$ in $W$, to generate a continuous sequence of poses for $A$ while avoiding contact with $B_i$. This sequence starts at the initial pose and concludes at the goal pose.

**Map representation** Various map representations are available for path planning:

- Paths, such as probabilistic road maps.
- Free space representations, including Voronoi diagrams.
- Obstacle representations, like geometric obstacle maps.
- Composite representations, such as grid maps.

**Planner** Planners are typically classified into two main categories:

- Search-Based Planning Algorithms (A$^*$).

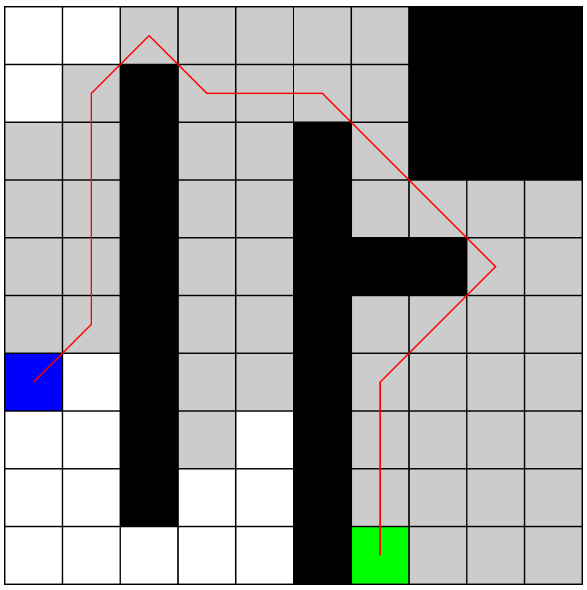


Figure 6.5: Search Based Planning Algorithms

These algorithms reframe the problem as a graph on the map. This planning approach is optimal and offers the advantages of finding the optimal solution, enabling cost assignment, employing heuristics, and determining solution existence (completeness). However, it incurs a high computational cost.

- Random Sampling Algorithms (PRM).

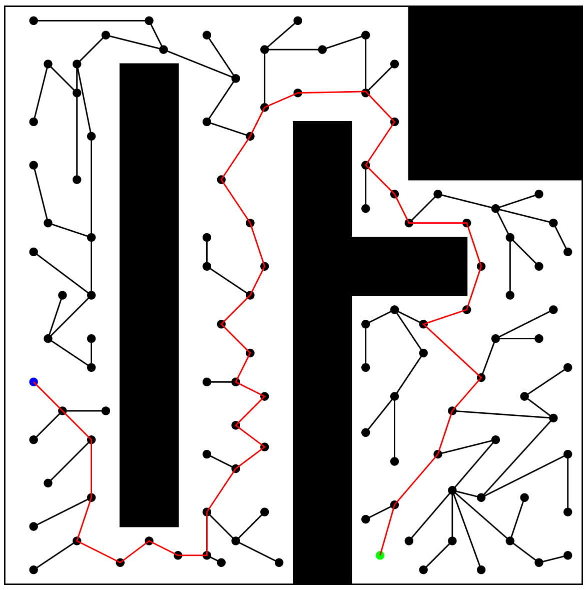


Figure 6.6: Random Sampling Algorithms

Instead of using a graph, these algorithms generate a set of samples in the environment and connect them. They excel in quickly finding feasible solutions. However, they face challenges in cost assignment and may only be probably complete, lacking a definitive way to test for solution existence.

## 6.4 Search-based methods

Search-Based Planning Algorithms are straightforward, versatile, and offer valuable theoretical guarantees when connectivity assumptions are met.

These algorithms are well-suited for composite maps, but also to free space representation. Considering a cell on the grid, we have a total of eight connectivity: four diagonal and four orthogonal

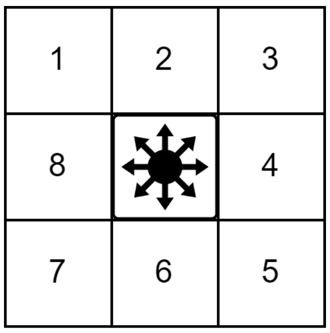


Figure 6.7: Grid map cells connectivity

The general concept involves:

1. Creating a discretized representation of the planning problem.
2. Constructing a graph based on this discretized representation, often using 4 or 8 neighbors connectivity.
3. Searching the graph to find the optimal solution.

Optionally, integrating the construction of the representation with the search process, only generating what is required at each step.

**Robot shape** Modeling a real mobile robot as a point isn't practical; instead, its shape is considered, and obstacles are expanded accordingly. However, this approach introduces challenges and necessitates a trade-off between memory usage and performance.

**Configuration space** To achieve precise collision detection, the configuration space is employed, where a configuration of an object is denoted by a point $q = (q_1, q_2, \cdots, q_n)$. A point $q$ is considered free if the robot positioned at $q$ doesn't collide.

**Definition** (*C-obstacle*)**.** C-obstacle is the union of all points $q$ where the robot collides.

**Definition** (*C-fredd*)**.** C-free is the union of all points $q$ where the robot does not collide.

Hence, we define

$$\text{C-space} = \text{C-free} \cup \text{C-obstacle}$$

Planning tasks can be executed within C-Space.

A robot is capable of both translation and rotation within the plane. To accommodate this movement, obstacles need to be enlarged based on the orientation of the robot.

**Algorithms** Various algorithms are at disposal:

- Some provide the optimal path (e.g., Dijkstra, A$^*$).
- Others yield an $\varepsilon$ sub-optimal path (e.g., weighted A$^*$, ARA$^*$, AD$^*$, R$^*$, D$^*$ Lite).

**Graph-Based path-finding for minimizing costs** The objective is to search a graph to find the path that minimizes costs.

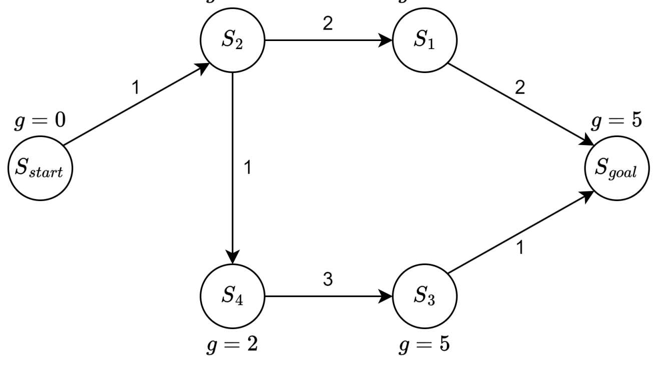


Figure 6.8: Searching graph

Numerous search algorithms calculate optimal g-values for pertinent states. The function $g(s)$ estimates the cost of the least-cost path from the starting state $s_{\text{start}}$ to $s$. The optimal values of this function satisfy:

$$g(s) = \min_{s'' \text{ in } pred(s)} g(s'') + c(s'', s)$$

The least-cost path is determined by backtracking: begin with $s_{\text{goal}}$ and from any state $s$, move to the predecessor state $s'$ such that:

$$s' = \operatorname*{argmin}_{s'' \text{ in } pred(s)} g(s'') + c(s'', s)$$

## A* search algorithm

A* accelerates the search process by computing $g$-values as follows:

- A function $g(s)$ corresponding to the cost of the shortest path found so far from $s_{\text{start}}$ to $s$ found so far.
- A function $h(s)$ estimating the cost of the shortest path from $s$ to $s_{\text{goal}}$.

The heuristic function must meet the following criteria:

- *Admissible*: for every state $s$ we have that $h(s) \leq c^*(s, s_{\text{goal}})$.
- *Consistent*: it satisfies the triangle inequality:
  - $h(s_{\text{goal}}, s_{\text{goal}}) = 0$
  - For every $s \neq s_{\text{goal}}$, $h(s) \leq c(s, succ(s)) + h(succ(s))$

Admissibility follows from consistency, and often vice versa.

**Property 6.4.1.** When the heuristic is both admissible and consistent, A* becomes optimal.

**Algorithm 6** A* algorithm

```
1: $g(s_{\text{start}}) = 0$
2: All other $g$-values are infinite
3: OPEN = $\{s_{\text{start}}\}$
4: while $s_{\text{goal}}$ is not expanded do
5:     Remove $s$ with the smallest $f(s) = g(s) + h(s)$ from OPEN
6:     Insert $s$ into CLOSED
7:     for every successor $s'$ of s such that $s'$ not in CLOSED do
8:         if $g(s') > g(s) + c(s, s')$ then
9:             $g(s') = g(s) + c(s, s')$
10:            insert $s'$ into OPEN
11:        end if
12:    end for
13: end while
```

**Properties** A$^*$ ensures:

- The return of an optimal solution path.
- A minimally required number of state expansions.

Regarding state expansion:

- Dijkstra's algorithm expands states based on the order of $f = g$ values (approximately).
- A$^*$ Search expands states based on the order of $f = g + h$ values.
- Weighted A$^*$ expands states according to $f = g + \varepsilon h$ values, where $\varepsilon > 1$ introduces a bias towards states closer to the goal.

In many domains, Weighted A$^*$ Search has been demonstrated to be significantly faster than A$^*$ by orders of magnitude.

**Variations** We have several variations, such as:

- ARA$^*$ (Anytime Repairing A$^*$):
  - Subsequent queries with decreasing sub-optimality factor $\varepsilon$.
  - Fast initial (suboptimal) solution.
  - Refinement over time.
- D*/D*-Lite: re-use parts of the previous query and only repair solution locally where changes occurred.
- Anytime D* (D* + ARA$^*$): anytime graph-search re-using previous query.

### 6.4.1 Summary

Elements of a Search-based planner:

- Graph construction: determines successor states for a given state. The graph can be constructed using motion primitives, offering several advantages such as a sparse graph, feasible path generation, and the ability to incorporate various constraints. However, a drawback is the potential for incompleteness.
- Cost function: associates a cost with each transition in the graph.
- Heuristic function: provides estimates of the cost-to-goal.
- Graph search algorithm: executes the search process (e.g., A* search).

Additionally, the graph can incorporate robot dynamics or kinematics constraints.

### 6.4.2 State-lattice planning

State-Lattice planning involves motion planning for constrained platforms by conducting a graph search in state-space. The process typically includes:

1. Discretizing the state-space into a hypergrid (e.g., $x, y, \theta, \kappa$).
2. Determining the neighborhood set by connecting each tuple of states with feasible motions.
3. Defining the cost function or edge weights.
4. Executing any graph-search algorithm to discover the lowest-cost path.

The benefits of State-Lattice planning include achieving resolution completeness and enabling feasible optimal offline computations owing to its regular structure. Drawbacks include the curse of dimensionality, where the number of states exponentially increases with the dimensionality of the state-space. State-lattice construction necessitates solving nontrivial two-state boundary value problems. Regular discretization may encounter challenges in narrow passages not aligned with the hypergrid, and discretization can cause discontinuities in state variables not accounted for in the hypergrid, thereby rendering motion plans non-executable.

To address these challenges, minimal neighborhood sets should be designed, avoiding the insertion of edges that can be decomposed with existing controls. Decomposition should be close in cost-space.

**Hybrid A*** Hybrid A* generates motion primitives by sampling the control space, eliminating the need to solve boundary value problems. Continuous states resulting from this process are then associated with discrete states in the hypergrid. Each grid cell stores a continuous state. However, there is no longer a guarantee of completeness due to changes in the reachable state space and the pruning of continuous-state branches.

## 6.5 Sampling-based methods

Sampling-based motion planning is driven by the impracticality and time-consuming nature of constructing explicit representations of collision-free space. Conversely, checking if a position is in free space is rapid, facilitated by fast collision-checking algorithms capable of testing configurations or short paths for collision-free status in less than 0.001 seconds.

The fundamental approach involves sampling the space of interest, connecting sampled points via simple paths, checking path collision status, and subsequently searching the resulting graph.

### 6.5.1 Probabilistic roadmap algorithm

**Algorithm 7** Probabilistic roadmap construction algorithm

```
1: Pick uniformly at random $s$ configurations in $F$ and create $M$, the set of milestones
2: Construct the graph $R = (M, L)$ where $L$ is every pair of milestones that see each other
3: Call $R$ the roadmap
```

**Algorithm 8** Probabilistic roadmap algorithm

```
for $i = \{b, e\}$ do
    if there is a milestone $m$ that sees $q_i$ then
        $m_i \leq m$
    else
        repeat
            Select $q$ in $F$ at random near $q_i$ until $q$ sees both $q_i$ and a milestone $m$
            if all $t$ trials fail then
                return Failure
            else
                $m_i = m$
            end if
        until $count = t$
    end if
    if $m_b$ and $m_e$ are in the same connected component of the roadmap then
        return Path connecting them
    else
        return No path
    end if
end for
```

Sample-based methods incrementally construct a search tree by progressively enhancing resolution, eliminating the necessity for a roadmap. It's an incremental sampling and searching approach without any parameter tuning. Ultimately, the tree densely covers the space, guided by a dense sequence of samples during construction.

### 6.5.2 Rapidly Exploring Dense Trees

**Algorithm 9** Simple Rapidly Exploring Dense Trees

```
$\mathcal{G}$.init($q_0$)
for $i = 1$ to $k$ do
    $\mathcal{G}$.add_vertex($\alpha(i)$)
    $q_n = nearest(S(\mathcal{G}), \alpha(i))$
    $\mathcal{G}$.add_edge($q_n, \alpha(i)$)
end for
```

Here, $\alpha$ is a dense sequence of samples in $C$, $\alpha(i)$ is the $i$-th sample of the sample sequence, $G(V, E)$ is the topological representation of Rapidly exploring dense trees, $S \subset C_{\text{free}}$ is the set of points reached by $\mathcal{G}$.

Some years later the same algorithm were improved and became:

**Algorithm 10** Rapidly Exploring Dense Trees

```
1: $\mathcal{G}$.init($q_0$)
2: for $i = 1$ to $k$ do
3:     $q_n = nearest(S, \alpha(i))$
4:     $q_s = stopping\_configuration(q_n, \alpha(i))$
5:     if $q_s \neq q_n$ then
6:         $\mathcal{G}$.add_vertex($q_s$)
7:         $\mathcal{G}$.add_edge($q_n, q_s$)
8:     end if
9: end for
```

Some years later the same algorithm were improved and became:

**Algorithm 11** Balanced Bidirectional Rapidly Exploring Dense Trees

```
1: $T_a$.init($q_1$)
2: $T_b$.init($q_G$)
3: for $i = 1$ to $K$ do
4:     $q_n = nearest(S_a, \alpha(i))$
5:     $q_s = stopping\_configuration(q_n, \alpha(i))$
6:     if $q_s \neq q_n$ then
7:         $T_a.add\_vertex(q_s)$
8:         $T_a.add\_edge(q_n, q_s)$
9:         $q'_n = nearest(S_b, q_s)$
10:        $q'_s = stopping\_configuration(q'_n, q_s)$
11:        if $q'_s \neq q'_n$ then
12:            $T_b.add\_vertex(q'_s)$
13:            $T_b.add\_edge(q'_n, q'_s)$
14:        end if
15:        if $q_s = q_n$ and $q'_s = q'_n$ then
16:            return solution
17:        end if
18:    end if
19:    if $|T_b| > |T_a|$ then
20:        $swap(T_a, T_b)$
21:    end if
22: end for
23: return failure
```

### 6.5.3 Rapidly Exploring Random Trees

RRT enhances the basic RDT by:

- Steering the system towards random samples in accordance with kinodynamics.
- Biasing the tree towards unexplored regions through Voronoi bias.

**Algorithm 12** Balanced Bidirectional Rapidly Exploring Dense Trees

```
1: $\tau = InitializeTree()$
2: $\tau = InsertNode(\varnothing, z_{\text{init}}, \tau)$
3: for $i = 1$ to $N$ do
4:     $z_{\text{rand}} = Sample(i)$
5:     $z_{\text{nearest}} = \text{Nearest}(\tau, z_{\text{rand}})$
6:     $(x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) = \text{Steer}(z_{\text{nearest}}, z_{\text{rand}})$
7:     if $ObstacleFree(x_{\text{new}})$ then
8:         $\tau = \text{InsertNode}(z_{\text{new}}, \tau)$
9:     end if
10: end for
11: return $\tau$
```

The quality of RRT exploration depends greatly on the chosen distance metric, especially in the context of non-holonomic systems, which pose challenges in metric selection.

Key advantages include asymptotic completeness, effectiveness in high-dimensional state spaces, elimination of the need for a two-state boundary value solver, simple implementation, and adaptability to constrained platforms.

However, drawbacks include the lack of asymptotic optimality, absence of optimality guarantees, generation of jerky paths within finite time, and limited potential for offline computations.

**Extensions** Several extensions have been suggested for the fundamental Rapidly Exploring Random Trees algorithm:

- *Bidirectional RRT*: this approach involves growing two trees simultaneously from the start and goal states, with frequent attempts to merge them.
- *Goal-biased RRT*: this variant samples the goal state every nth sample, balancing exploration and exploitation.
- *RRT**: introducing a local rewiring step, RRT* aims to achieve asymptotic optimality.

These extensions offer various advantages: they are asymptotically complete and provide an asymptotically optimal guarantee. They perform well in high-dimensional state spaces, do not necessitate a two-state boundary value solver, are straightforward to implement, and can handle constrained platforms with ease.

However, there are drawbacks to consider: they require a two-state boundary value solver for the rewiring process, tend to produce jerky paths within finite time, and have limited potential for offline computations.

CHAPTER 7

# Robot Operating System

## 7.1 Introduction

Middleware was introduced by d'Agapeyeff in 1968, the concept of a wrapper emerged in the 1980s as a bridge between legacy systems and new applications. Today, it is prevalent across various domains, including robotics. Examples outside of robotics include Android, SOAP, and Web Services.

The concept of Middleware is widely recognized in software engineering. It serves as a computational layer, acting as a bridge between applications and low-level details. It's important to note that Middleware is more than just a collection of APIs and libraries.

**Issues** The challenge lies in fostering cooperation between hardware and software components. Robotics systems face architectural disparities that affect their integration. Ensuring software reusability and modularity is also a critical concern in this context.

**Main features** The key attributes of middlewares include:

- *Portability*: offering a unified programming model irrespective of programming language or system architecture.
- *Reliability*: middlewares undergo independent testing, enabling the development of robot controllers without the need to delve into low-level details, while leveraging robust libraries.
- *Manage the complexity*: handling low-level aspects through internal libraries and drivers within the middleware, thereby reducing programming errors and speeding up development time.

**Middlewares** Several middleware have been developed in recent years:

- *OROCOS*: originating in December 2000 as an initiative of the EURON mailing list, OROCOS evolved into a European project with three partners: K.U. Leuven (Belgium), LAAS Toulouse (France), and KTH Stockholm (Sweden). The requirements for OROCOS encompass open source licensing, modularity, flexibility, independence from specific robot industries, compatibility with various devices, software components covering kinematics, dynamics, planning, sensors, and controllers, and a lack of dependency on a singular programming language. The structure includes:

  - Real-Time Toolkit (RTT): offering infrastructure and functionalities tailored for real-time robot systems and component-based applications.
  - Component Library (OCL): supplying ready-to-use components such as drivers, debugging tools, path planners, and task planners.
  - Bayesian Filtering Library (BFL): featuring an application-independent framework encompassing (Extended) Kalman Filter and Particle Filter functionalities.
  - Kinematics and Dynamics Library (KDL): facilitating real-time computations for kinematics and dynamics.

- *ORCA*: the project's objective is to emphasize software reuse in both scientific and industrial applications. Its key properties include enabling software reuse through the definition of commonly-used interfaces, simplifying software reuse via high-level libraries, and encouraging software reuse through regularly updated software repositories. ORCA defines itself as an "unconstrained component-based system".

  The primary distinction between OROCOS and ORCA lies in their communication toolkits. OROCOS utilizes CORBA, whereas ORCA employs ICE. ICE, a contemporary framework created by ZeroC, functions as an open-source commercial communication system. ICE offers two fundamental services: the IceGrid registry (Naming service), which facilitates logical mapping between various components, and the IceStorm service (event service), which forms the foundation for publisher-subscriber architecture.

- *OpenRTM*: RT-Middleware (RTM) serves as a widely adopted platform standard for assembling robot systems by integrating software modules known as robot functional elements (RTCs). These modules include components such as camera, stereo vision, face recognition, microphone, speech recognition, conversational, head and arm, and speech synthesis. OpenRTM-AIST (Advanced Industrial Science and Technology) is built upon CORBA technology to realize the extended specifications of RTC implementation.

- *BRICS*: the objective is to uncover the most effective strategies for developing robotic systems by examining several critical areas: Initially, by thoroughly examining the weaknesses found in current robotic projects. Subsequently, by delving into the integration between hardware and software components within these systems. Thirdly, by advocating for the incorporation of model-driven engineering principles in the development process. Moreover, by creating a tailored Integrated Development Environment (IDE) specifically for robotic projects, known as BRIDE Finally, by defining benchmarks aimed at evaluating the strength and effectiveness of projects based on BRICS principles.

- *ROS*: introduced by Willow Garage in 2009, the Robot Operating System (ROS) serves as a meta-operating system tailored for robotics, boasting a diverse ecosystem replete with tools and programs. ROS has expanded to encompass a vast global community of users. The developer community stands out as one of ROS's most significant features.

### 7.1.1 Robot Operating System

The key features of ROS:

1. *Decentralized framework*: ROS operates on a decentralized architecture, facilitating robust communication and coordination among various components.
2. *Code reusability*: ROS promotes the reuse of code, allowing developers to efficiently leverage existing modules and algorithms for faster development.
3. *Language neutrality*: ROS supports multiple programming languages, enabling developers to work in their preferred language while seamlessly integrating with the ROS ecosystem.
4. *Seamless real robot and simulation testing*: ROS provides an environment for easy testing on both real robots and simulations, ensuring smooth transition from development to deployment.
5. *Scalability*: ROS is designed to scale efficiently, accommodating projects of various sizes and complexities without compromising performance or functionality.

ROS is composed by the following elements:

- *File system utilities*: ROS provides tools for managing files and directories, simplifying data organization and access within the ROS environment.
- *Construction tools*: ROS offers construction tools to streamline the development process, facilitating the creation and configuration of robotic systems and components.
- *Package management*: ROS includes robust package management capabilities, allowing users to easily install, update, and manage software packages and dependencies.
- *Monitoring and graphical user interfaces* (GUIs): ROS features monitoring tools and graphical user interfaces to visualize and analyze system behavior, aiding in debugging, optimization, and user interaction.
- *Data logging*: ROS supports data logging mechanisms for recording and analyzing sensor data, system state, and other relevant information, facilitating research, analysis, and debugging tasks.

## 7.2 Computational graph

The computation graph is the peer-to-peer network of ROS processes that are processing data together.

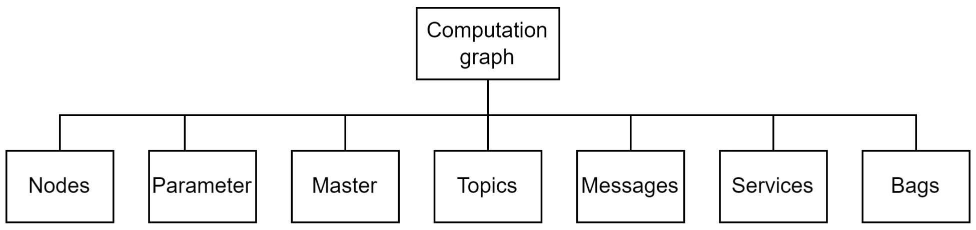


Figure 7.1: ROS computational graph

### 7.2.1 Nodes

ROS executable units include Python scripts and C++ compiled source code. They are processes responsible for computation within the ROS framework. These units, referred to as nodes, exchange information through a graph structure. Nodes are designed to operate at a fine-grained scale and are integral components of robot systems, which are composed of multiple interconnected nodes.

### 7.2.2 Master

The ROS Master serves as the central hub, facilitating naming and registration services crucial for node interactions. In every system, including distributed architectures, there exists a single master. This centralized entity enables ROS nodes to effectively locate each other.

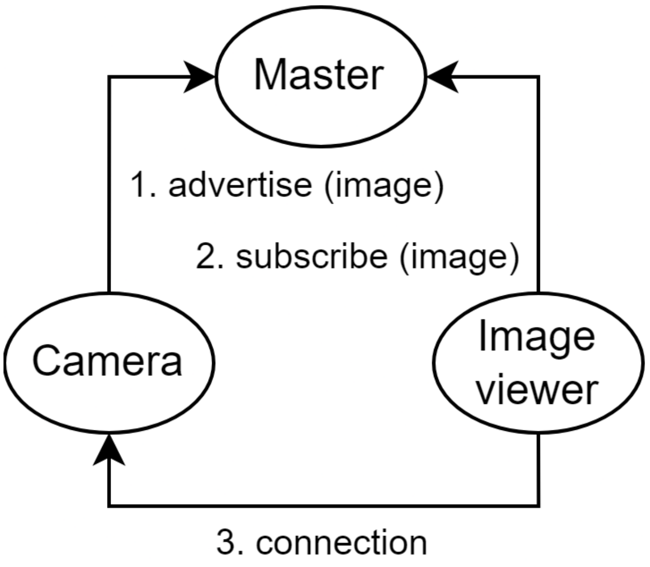


Figure 7.2: ROS master

In this scenario, the camera node advertises the master with an image, making it available on a specific topic. Subsequently, the image viewer node subscribes to the same topic through the master. Once the requested topic is found, the ROS master facilitates communication by providing the node name, allowing the subscriber to establish communication with the node that generated the image. This interaction demonstrates the pivotal role of the ROS master in mediating communication between nodes in the ROS ecosystem.

### 7.2.3 Topics

Topics serve as named channels for communication within the ROS framework, implementing the publish/subscribe paradigm. They facilitate communication between nodes by allowing multiple nodes to publish messages on a topic and multiple nodes to read those messages from the same topic. Topics are associated with specific message types, but they do not guarantee message delivery.

It's advisable to avoid scenarios where multiple talkers are connected to the same topic in ROS. This is because ROS lacks a mechanism to discern the origin of a message in such cases.

### 7.2.4 Messages

Communication on topics involves the exchange of messages, which define the type of information being transmitted. A wide range of predefined message types is available. Additionally, it is possible to create custom messages using a straightforward language. These custom messages can incorporate existing message types along with base types as needed.

### 7.2.5 Services

Services function akin to remote function calls within ROS, adhering to the client/server paradigm. When a service call is made, the code waits for its completion, ensuring a guarantee of execution. Services utilize message structures for their operation.

### 7.2.6 Parameter server

The parameter server is a shared, network-accessible multivariable dictionary utilized by nodes for storing and accessing parameters during runtime. It's not optimized for performance or data exchange. This server is linked to the master, constituting one of the functionalities provided by ROScore. The parameter server can store a range of data types, including 32-bit integer, Boolean, string, double, ISO8601 date, and base64-encoded binary data.

### 7.2.7 Bags

Bags serve as a file format (*.bag) designed for storing and replaying messages within ROS. They constitute the primary mechanism for data logging, capable of recording all exchanges occurring on the ROS graph, including messages, services, parameters, and actions. Bags are essential tools for data analysis, storage, visualization, and algorithm testing within the ROS ecosystem.

### 7.2.8 ROScore

ROScore is a foundational component of ROS-based systems, comprising nodes and programs essential for operation. It must be active to facilitate communication among ROS nodes. Launching ROScore via the command initiates its elements, which include a ROS master, a ROS parameter server, and a ROSout logging node.

## 7.3 ROS file system

The foundation of the ROS file system revolves around packages, depicted in the diagram below.

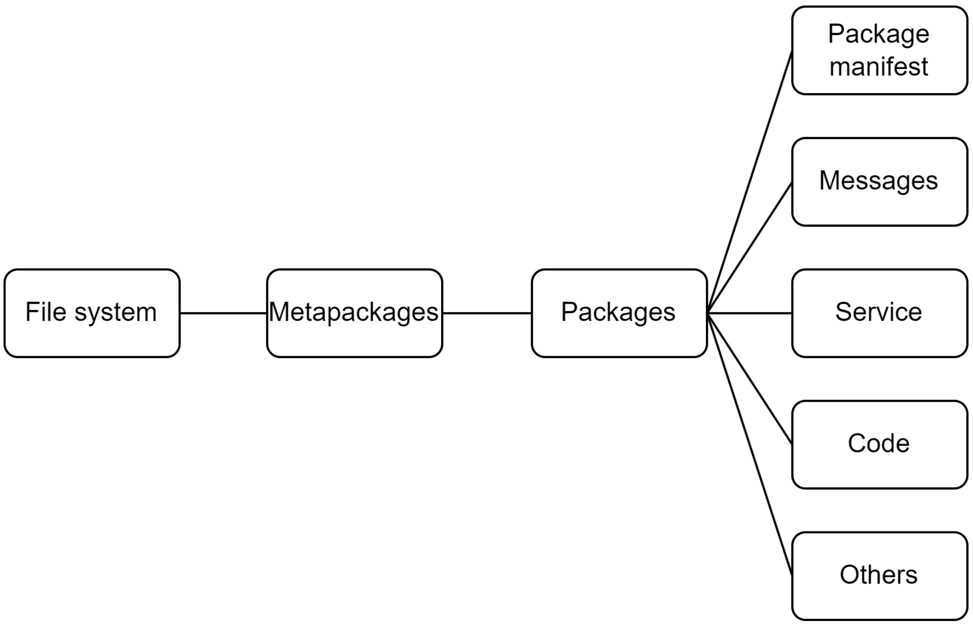


Figure 7.3: ROS file system

Packages serve as the fundamental units within the ROS file system. They are essential references for most ROS commands and encompass nodes, messages, and services. Each package is described by a `package.xml` file and acts as a mandatory container for its contents.

Additionally, there are metapackages, which aggregate logically related elements. Unlike packages, metapackages are not directly utilized when navigating the ROS file system. They contain other packages and are described by a package.xml file as well, but they are not obligatory components.

### 7.3.1 Packages

Packages serve as fundamental units within the ROS file system, acting as the cornerstone for most ROS commands. They encapsulate nodes, messages, and services, providing a structured organization for ROS functionalities. Each package is described by a package.xml file, serving as a mandatory container that delineates the package's properties and dependencies. The general structure of a package includes the following components:

- Folder Structure:
  - /src, /include, /scripts (for coding).
  - /launch (for launch files).
  - /config (for configuration files).
- Required Files:
  - CMakeLists.txt: contains build rules for catkin.
  - package.xml: provides metadata for ROS.

### 7.3.2 Metapackages

Metapackages are collections of logically related elements within ROS, serving as aggregations rather than individual components. Unlike regular packages, they are not typically used when navigating the ROS file system. Metapackages encompass other packages and are described by a package.xml file, yet they are not obligatory structures within ROS.