Mobile Robot Localization Under intermitted GNSS Service Using Visual Support

Master Thesis

Johanes Patrick S

Aalborg University
Robotics

Robotics
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Mobile Robot Localization Under intermitted GNSS Service Using Visual Support

Theme:
Master Thesis

Project Period:
Spring Semester 2025

Project Group:
XXX

Participant(s):
Johanes Patrick S

Supervisor(s):
Henrik Schiøler

Page Numbers: 78

Date of Completion:
August 14, 2025

Abstract:

Mobile robots, particularly mobile sports field-marking robots, are mostly operated in outdoor environments relying primarily on sensors such as GNSS and cameras for navigation and line detection tasks, the system ensures accurate positioning and environmental awareness. However, its system performance can sometimes be not up to Satisfactory results due to uncertainty variables, especially intermittent signal loss in GNSS and obstruction in cameras caused by weather or lightning conditions. These disturbances cause inaccuracies in estimating robot's pose and environmental perception, that negatively affect its system. As described above, thus this study focuses on mitigating the adverse impact caused by GNSS signal degradation and quality limitations in cameras and proposes a mitigation approach through sensor fusion including IMU and odometry, processed using the EKF algorithm. The results demonstrate that sensor fusion can improve the robustness of the system.

Contents

Nomenclature

Abbreviations

AAVs	Autonomous Aerial Vehicles
GPS	Global Positioning System
GNSS	Global Navigation Satellite System
CNN	Convolutional Neural Network
SLAM	Simultaneous Localization and Mapping
IMU	Inertial Measurement Unit
CMOS	Complementary Metal-Oxide Semiconductor
CCD	Charge-Coupled Device
RANSAC	Random Sample Consensus
3D	Three-Dimensional
2D	Two-Dimensional
ARHS	Attitude and Heading Reference System
GLONASS	Global Navigation Satellite System (Russia)
BDS	BeiDou Navigation Satellite System (China)
RTK	Real-Time Kinematic Positioning
PPP	Precise Point Positioning
DGPS	Differential GPS
WAGPS	Wide-Area GPS

LOS	Line of Sight
EKF	Extended Kalman Filter
ICC	Instantaneous Center of Curvature
VO	Visual Odometry
ROS	Robot Operating System

Chapter 1

Introduction

The usage of robotics has expanded twofolds into a variety of areas, thus, in recent years, many experts have predicted that robotics will displace human workers in various industries. Even in extreme cases, some experts have not ruled out the possibility of robots replacing human workers entirely. This scenario has already happened to occur incrementally, as many human workers have been gradually laid off in favor of robots. For example, an electric car marker, BYD Auto, has employed the army of humanoid robots for assembling, sorting, inspecting, among other tasks. Similarly, EHang, an autonomous aerial vehicle (AAVs) marker, prides itself on being the company that pioneered the electric passenger-grade autonomous aerial vehicle, which allows it to carry people without a driver. These cases are exemplary of how fast robotics has evolved. Previously, robots were limited to industrial automation, which has the following characteristics: a lack of mobility due to the need to be bolted in a specific place; however, they are now capable of performing multiple tasks with high mobility, among other advancements. This is known as mobile robots.

Mobile robots are machines that can navigate on their own without human assistance; they are regarded as a subfield of robotics. Among other benefits related to sensors, they need the use of sensors to gather data about surrounding environments and choose the best course of action to complete a task. For example, the electric car manufacturer, Tesla, has implemented a variety of sensors for lane detection and road safety. It detects lane markings and objects like other cars and pedestrians using data from cameras. In addition to visual sensors, it also utilizes Global Positioning System (GPS) sensors to determine its pose, which enables it to navigate from its current location to the specific location generated by GPS, thereby reaching the desired destination. This illustrates how important sensors are for machines with autonomous function.

As described in the paragraph above, the example of Tesla highlights the use of sensors for perception and localization systems. Thus, the purpose of this study is to implement these systems in the simulation. However, the main purpose is to examine how Global Navigation Satellite System (GNSS) sensors under intermittent signal affects the overall of

systems.

1.1 Perception System

The perception system plays a significant role in autonomous navigation through line detection. Before discussing line detection, this study will first examine line marking, as it is relevant to the process. Line marking is a method used for applying lines in the form of visible markings to various surfaces, such as sports fields, roads, warehouses, and more, by spraying or painting.

The purpose of line markings is to ensure road safety, to communicate information that ensures the structure, fairness, and clarity of sports, to ensure efficient use of spaces while enhancing safety in warehouses, among other purposes. Today, there are two kind types of line markings: traditional and modern line markings. Traditionally, line marking relied on human labor, making it a time-consuming task prone to errors. However, with advancements in robotics, have transformed traditional line markings into modern line marking approaches. This enables painting lines with high accuracy, precision, and consistency.

Both methods have distinct advantages and disadvantages. Traditional line markings are cost-effective (especially in developing countries), highly customizable, easy to implement, and more. Developing countries may not have the financial resources to invest in modern line markings technology. Additionally, the potential profits may not be sufficient to justify such an investment, as labor costs in developing countries are relatively low. However, for developed countries, the situation is different mainly because labor costs are relatively high, making the investment in modern line markings technology more justifiable. As evidence, the majority of robots are predominantly used in developed countries, while developing countries, with the exception of China, are not in a hurry to adopt the usage of robots.

Modern line markings often utilize line detection. Line detection is a classic algorithm in computer vision that acts as an intermediate step in identifying lines; it can also be used to identify vanishing points [1] or lane lines [2]. In general, line detection commonly employs the Hough transform and its variation [3, 4] in traditional approaches and the convolutional neural network (CNN)-based methods and their variations [5] in modern approaches. Both of these approaches have distinct advantages and disadvantages. However, this study will not examine them in detail, as its primary focus relates to intermittent GNSS. Therefore, for the perception system for line detection, this study will employ traditional approaches, as they are relatively simple to implement.

Line detection can serve as an information-related feature, such as previously painted line markings behind the mobile robot; thus, it can serve as a reference for localization tasks. The perception system identifies and extracts meaningful line segments from the scene behind the mobile robot and then matches these extracted lines with a known map (e.g., a map generated using Simultaneous Localization and Mapping (SLAM) methods)

to estimate the robot's pose relative to the line markings.

1.2 Localization System

Localization is an important aspect of robotics, as it addresses the question: Where is the robot? Thus, localization refers to a robot's ability to estimate its pose relative to the coordinate frame in which the map is defined. Once the robot's pose is known, it can act accordingly by performing tasks such as navigating to a designated location, planning paths, among other related activities. [6] Localization problems can be roughly divided into the following type: *pose tracking*, *global localization*, and *map acquisition*.

1.2.1 Pose Tracking

This kind of situation assumes that position tracking localizes the robot's position correctly, with its initial position known. However, it must take into account the noise that affects the robot's motion. In this context, the noise is assumed to be small and confined to a small area, thus the position-tracking problem is known as *local localization*.

1.2.2 Global Localization

In this scenario, the prior knowledge of the whereabouts of the robot is unknown with the assumption that probability distribution cannot be made. Generally, global localization has more problems than position tracking.

1.2.3 Map Acquisition

A map of the surrounding environment is required for both pose tracking and global localization. There are two ways to obtain a map: using a map blueprint or using the robot to generate the map representation through its sensors. In the case of a map generated by the robot, the robot's position as a reference must be known when generating the map.

1.3 Problem Statement

Accurate mapping and localization are essential for mobile robots, as they enable them to navigate autonomously. As for mapping, there are two types of localization-related mapping problems: static and dynamic environments. Static environments occur when all objects and features remain in a fixed position permanently. This is known as immutable environments. Dynamic environments arise when objects undergo persistent changes over time. On the other hand, only the robot's configuration changes over time in static environments, while in dynamic environments, both the objects (e.g., people, movable furniture, doors, and others) and the robot's configuration change over time. This study will assume static environments for mapping, as its primary focus is more closely related to global and local localization problems. The reason for using static environments is that both global and local localization require a map for the mobile robot to localize itself with respect to local and global reference frames in order to estimate the robot's pose accurately.

Once, a mobile robot has information about the map, which represents the surrounding environment. The primary challenge it faces in navigating autonomously is accurately estimating its pose relative to that map. For such tasks, the mobile robot requires the integration of sensors into its system. A common problem within sensor systems is that if one sensor fails, it can lead to serious consequences for the mobile robot. However, this problem can be mitigated by integrating multiple sensors, where each sensor complements the other sensors. In this way, if one sensor fails, then its function can be compensated for by the remaining sensors. For instance, line detection can be achieved solely using cameras. However, what happens if the cameras suffer a failure within their system? Therefore, the best approach is to use multiple sensors; for example, lidar sensors can also perform such tasks. Another benefit of using multiple sensors is that their combined use is often more accurate than relying on a single sensor.

This study focuses on the localization problem in both global and local contexts during autonomous navigation using visual sensors. As for global localization, a mobile robot uses GNSS sensors to estimate its pose relative to a map. In contrast, for local localization, a mobile robot is equipped with sensors (e.g., wheel encoders and IMU sensors) to track its own motion and estimate its location relative to its starting position. This is known as *odometry*.

Unfortunately, all sensors are vulnerable to uncertainty, noise, and other sources of error. For example, in global localization, GNSS sensors often suffer from signal degradation caused by the surrounding environments where the line of sight to satellites is obstructed. Environmental factors that cause signal degradation include trees, stadium structures, tall buildings, natural interference like fog and rain, and materials like metals, among other obstructions that actively block signals. These challenges prevent the system from pinpointing the location accurately and precisely. Another possible solution to minimize this issue is to use high-accuracy GNSS sensors, such as those used by Turf Tank, can be used to address this issue, but unfortunately, they are very expensive and are not always reliable

due to several challenges including signal shadowing or obstruction, thus, making them economically impractical.

Similarly, in local localization, odometry suffers from cumulative errors in estimating and measuring the motion model, thus making it unreliable for a mobile robot navigating in surrounding environments. These limitations in both global and local localization suggest that a practical solution to tackle this kind of situation can involve the integration of multiple sensors to improve reliability and accuracy.

As outlined above, these challenges underline a key aspect of the research problem: **How can mobile robots maintain accurate localization under uncertainty environments in real-world conditions where both GNSS and visual sensors may suffer from intermittent errors or errors related to visual perception**. Therefore, this study aims to address the problem by simulation of the mobile robot under intermittent GNSS signal conditions and evaluate how traditional visual line detection methods can complement the localization system. Additionally, explore the effectiveness of the integration of sensor fusion other than GNSS and camera sensors in improving system robustness and accuracy in uncertainty environments.

1.4 Objectives

As explained, the main goal of this study is to develop a framework for both global and local localization as a solution to the challenges associated with each, while improving the reliability and accuracy of localization during periods of intermittent GNSS signal loss. In addition to complementing the localization system, the study also aims to enable a mobile robot to detect lines accurately by using visual perception systems as a complementary source of pose information, thereby facilitating autonomous navigation. To achieve this, this study defines the following specific objectives that must be met:

1. Design a mobile robot model that emulates the functional behavior of the Turf Tank mobile robot. For detailed specifications of the model robot will be discussed in the following section 1.5.

2. Simulate a mobile robot model in a structured environment, where both global and local localizations are performed using both GNSS and camera sensors.

3. Implement traditional line detection methods, particularly Hough Transform, that enable the robot to detect and extract line features as a source of pose information.

4. Apply sensor fusion through the EKF method for integrating GNSS, IMU, odometry, and camera sensors to estimate the robot's pose.

5. Evaluate the robot's localization performance under varying sensor configuration as follows:

- Pure Odometry
- Pure GNSS
- Odometry + Camera
- Odometry + Camera + GPS

6. Assess whether the performance of sensors, which is based on a single sensor or multiple sensors as described above, is sufficient to improve the robustness, accuracy, and reliability of localization systems.

Note: The simulation environment is built using ROS2 for communication protocols, Gazebo for physics-based simulation, and RViz2 for visualization.

1.5 The Robot Model Specifications

The mobile robot model in this study is intended to emulate the core functionalities and behaviors of Turf Tank mobile robots, which are specifically developed for sports field line markings. This robot model does not replicate the exact physical design, but instead includes key aspects of sensor systems that are essential for autonomous navigation and localization tasks. The robot model for this study is equipped with the following sensors:

1. **GNSS Sensor**: provides global position data in the simulated environment. It is modeled to realistically mirror real-world scenarios, including intermittent signal loss caused by uncertainty environments such as signal obstructions.

2. **Camera Sensor**: enables the detection of line markings on surfaces and serves as the primary sensor for the perception system. The detected lines provide an additional source of pose information alongside the GNSS sensors. In the simulation, the camera is positioned to face backward, replicating the configuration used in Turf Tank mobile robot.

3. **IMU sensor**: Provides acceleration and angular velocity data, which serve as the foundation for estimating local position data through mathematical integration.

4. **Wheel Encoders**: enables to estimate local position data by converting encoder counts into a distance traveled, using the geometry of the wheel and the resolution of the encoder.

1.6 Significance of this study

This study is expected to contribute in the field of robotics, especially in addressing localization challenges commonly faced by mobile robots in outdoor environments. Such environments often involve uncertainties such as tall buildings, weather conditions, and

other obstructions, which can lead to the intermittent availability of the GNSS signal. The significance of this study is as follows:

1. **For Researchers and Academics**: It can serve as a reference and provide a deeper understanding of sensor fusion techniques.
2. **For Related Industries**: It provides as a foundation for developing more reliable localization system for products such as mobile robots or related products, particularly those operating in environments with high uncertainty such as urban canyons.
3. **For Future Research**: It serves as a fundamental and comparative reference for future studies aiming to advance localization system using more complex methods such as deep learning.

1.7 Thesis structure

This thesis is methodically divided into several main chapters as follows:

1. **Introduction**
   Presents the background of the study, including perception and localization systems. This chapter also outlines the problem statement, research objectives, detailed specifications of the mobile robot model, the significance and contributions of the study, and the overall structure of the thesis.
2. **Background**
   Discusses the theoretical foundation and related work concerning various sensors used in mobile robotics, including camera, wheel encoders, IMU, and GNSS sensors.
3. **Robot Localization**
   Explains the concepts robot localization including environment representations and pose estimation. This section also covers the Kalman filter and the EKF.
4. **Perception System**
   Describes the camera-based perception system, including how the stereo vision system works. Mathematical models are provided to explain it.
5. **Methods**
   Describes the methodology for implementing the perception processes. Additionally, this section also cover Hough Transform for line and the Hough Transform process, odometry, and visual odometry.
6. **Simulation Tools**
   Provides a detailed description of the tools used for simulation such as Gazebo, ROS2, RViz2 and other relevant platforms or libraries.

7. **Implementation**
   Presents the implementation of the mobile robot model resembling the Turf Tank system. Includes system architecture, sensor configuration, and software integration.

8. **Results**
   Analyzes and compares the localization accuracy for each configuration: GNSS only, camera only, GNSS + camera, and GNSS + camera + IMU + odometry. Discusses how each sensor combination performs under normal and degraded GNSS conditions.

9. **Conclusion**
   Summarizes the research findings, discusses limitations, and provides recommendations for future work to improve robustness in mobile robot localization systems.

Chapter 2

Background

2.1 Sensor Systems

This section explores various sensors in this study. These sensors facilitate interaction between the robot and its surrounding environment by allowing it to acquire meaningful information, which is essential for localization and perception systems. Sensors can generally be classified into two types of categories: **proprioceptive vs. exteroceptive** and **passive vs. active** sensors. [7]

1. **Proprioceptive sensors** are used to measure the internal variables related to the robot's own state, such as motor speed, robot arm joint angles, wheel load, and other internal dynamics.

2. **Exteroceptive sensors** can be used to acquire information from the robot's external environment, including measurements like distance measurement, light intensity, surface textures, and others.

Another classification concerns to how sensors interact with the environments:

1. **Active sensors** emit energy into the environment and then measure environmental reactions. For example, the GPS inside the robot emits signals to measure the absolute position of the robot relative to the environment.

2. **Passive sensors** measure the energy that enters the sensors. For example, a tactile sensor is a passive sensor that measures based on force without emitting energy.

2.1.1 Camera Sensors

Camera sensors enable the extraction of visual information from the surrounding environment. They provide a rich source of detailed data for estimation and sensing tasks.

Various configurations exist in camera sensors. Presently, the vast majority of camera sensors in the consumer market use Complementary Metal-Oxide Semiconductor (CMOS) image sensors, while others utilize Charge-Coupled Device (CCD) image sensors. Despite offering lower image quality primarily due to higher noise levels compared to CCD image sensors, CMOS image sensors offer advantages in cost, power efficiency, and faster readouts, making them a popular choice in mainstream applications. However, when high-quality images are paramount, such as in medical imaging, video production, or professional photography, then CCD image sensors remain a preferred option. [8]

Another example that utilizes either CMOS or CCD image sensors is the stereo camera. Stereo vision is a well-established technology that has been extensively studied. It operates by using two cameras—typically designated as left and right—positioned at a fixed distance apart, enabling them to capture a scene from slightly different perspectives. The core principle of this method involves matching pixels from one image to their corresponding counterparts in the other image in order to calculate depth information. [9]

M. Bertozzi et al. [10] demonstrated that a stereo vision system could be used for real-time obstacle detection and lane tracking. Additionally, they presented algorithms specifically designed for efficient stereo vision processing, despite the limited computational resources available at the time. Their work significantly contributed to the development of autonomous vehicle technology and, subsequently, to broader applications such as mobile robotics. However, the success of numerous studies on line detection would not have been possible without the algorithm method introduced by [11], which serves as a foundation for robot navigation and computer vision. Subsequently, numerous advanced algorithms based on line detection such as Hough transform [12], Random Sample Consensus (RANSAC) [13], Deep Learning-Based Line Detection [14], and more were introduced, further expanding the possibility for advancement in the field.

Perception systems that rely on camera sensors typically face various challenges, including the need for complex data processing and feature extraction. The primary limitations of camera sensors include the following:

1. A camera in the traditional sense is used to capture visual data by projecting a three-dimensional (3D) scene onto a two-dimensional (2D) image plane. This projection inherently result in the loss of depth information, specifically, the camera does not know how far away objects in the z-coordinate are. [15] This limitation can be addressed by incorporating additional sources of information through sensor fusion.

2. Illumination limitations cause inconsistencies in visual information, primarily due to lightning conditions such as low light, bright light, shadows, and reflections. These challenges become noticeable in the outdoor environment, where lightning conditions vary, and numerous, often unknown objects may be present-posing significant challenges in perception systems.

2.1.2 Wheel Encoder Sensors

Wheel encoder sensors in a mobile robot are used to measure the robot's internal state and its dynamic behavior. Various types of wheel encoder sensors are available on the market; however, among them, the optical incremental encoder is the most preferred choice for measuring the angular speed and position in motor drives or at the shaft of a wheel. The working principle of wheel encoder sensors is based on the pulses generated as the wheel rotates, with the number of pulses corresponding to the distance traveled. These sensors can be divided into two main types: incremental and absolute encoders. [16]

Despite their widespread application in mobile robots, wheel encoder sensors are cost-effective, offer excellent resolution, and are relatively straightforward to implement. However, unfortunately, they are prone to the accumulation of errors over time. These errors are primarily caused by factors such as wheel slippage, uneven terrain, and other environmental disturbances. Therefore, to mitigate these issues, mobile robots often integrate wheel encoder sensors with other sensors. [16]

2.1.3 IMU Sensors

An IMU sensor generally consists of a combination of gyroscopes and accelerometers, which are used to measure a robot's relative position, velocity, and acceleration during motion. A single device called an attitude and heading reference unit (ARHS), which maintains a 6-DOF estimate of The robot's pose: position (x, y, z), and orientation (ϕ, θ, ψ) is generally created by integrating the sensing technology of an IMU sensor with an onboard computational unit. [17]

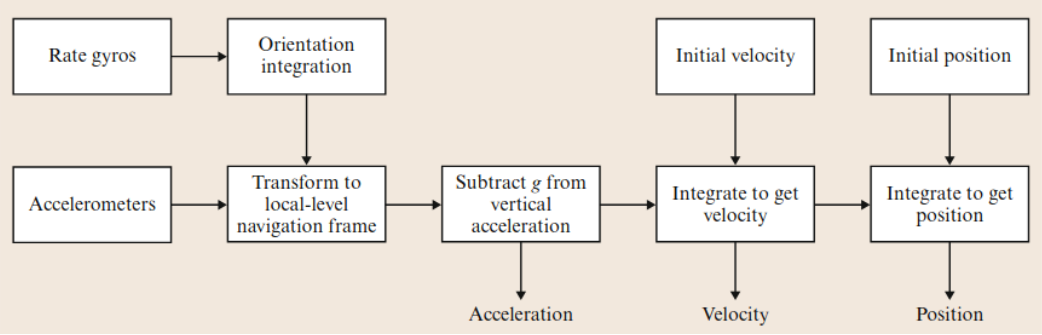


Figure 2.1: IMU's diagram [17]

The working principle of the IMU sensor is illustrated in Figure 2.1. The IMU sensor typically employs three orthogonal gyroscopes and three orthogonal accelerometers. The continuous estimation of the robot's orientation is maintained through the integration of gyroscope data, denoted as ω. These data are subsequently combined with accelerometer measurements, which measure the robot's acceleration, denoted as a, using the current estimate of the robot's orientation with respect to gravity. Through this process, the re-

sulting acceleration is first integrated to obtain the robot's velocity, denoted as v, and the velocity is then integrated again to obtain the robot's position, denoted as r. [17]

However, IMUs are highly sensitive to measurement errors, primarily due to the instability of the integration process within gyroscopes and accelerometers, which leads to accumulated drift over time. This drift results in the miscalculation of orientation with respect to gravity in gyroscopes, causing incorrect cancellation of the gravity vector. In accelerometers, this incorrect cancellation of the gravity vector leads to a quadratic error in position estimation, since the resultant acceleration must be integrated twice to obtain the position. Given the drift inherent in IMU data, GNSS data can be integrated with IMU data to improve the accuracy and reliability of the robot localization system. [17]

2.1.4 GNSS Sensors

The GNSS (Global Navigation Satellite Systems) sensors are commonplace and are often encountered in applications that utilize GNSSs for autonomous navigation, GNSS ionosphere monitoring, GNSS reflectometry, among others purposes. The GNSS sensors currenly in operation include the United States with Global Positioning System (GPS), Russia with Global Navigation Satellite System (GLONASS), China with BeiDou Navigation Satellite System (BDS), and the European Union (EU) with Galileo. Each GNSS system has its own strengths and weaknesses, and since there is no one-size-fits-all solution, it has been suggested that combining all available GNSS systems can enhance navigation. Several studies have demonstrated that this approach leads to more precise positioning and supports remote sensing applications. However, the main challenges associated with the use of GNSS are the following: [18]

1. High-accuracy: Several applications require high-accuracy positioning. For example, military operations demand high-accuracy positioning, as even small errors can ruin entire systems. Imagine a scenario during wartime, if there were a slight positioning error, it could lead to mission failure or unintended collateral damage. That is why countries with GNSS capabilities continue to invest in improving positioning accuracy, even the improvements are marginal and require a huge sum of investment, which is especially critical for military applications.

2. Reliability: In the consumer market, users generally prefer to use low-cost receivers (e.g., u-blox receivers, smartphone-level GNSS chips) mainly due to economic considerations. Therefore, the usage of low-cost GNSS receivers must be accompanied by acceptable positioning accuracy. Thus, this such situation pose a major challenges for GNSS consumer-market manufacturers and scientists, especially those working to improve accuracy to make these systems suitable for adoption in consumer applications.

3. Natural threats: Complex environments (e.g., line of sights (LOS), atmospheric conditions, among others) that could degrade the quality of GNSS signals, which is

crucial for precise positioning. Thus, it becomes necessary to ensure stable and continuous precise GNSS positioning despite natural threats.

4. Intentional threats: The common intentional threats in GNSS are jamming and spoofing. Jamming is an intentional interference method with the aim of overloading receivers, thus rendering it unable to operate, and spoofing transmits fake signals, which leads GNSS to calculate a false position. [19]

Thus, to tackle these challenges, various augmentation systems have been developed by both governmental and private entities to enhance the GNSS signals. The example of augmentation systems: Real-Time Kinematic (RTK), Precise Point Positioning (PPP) or Differential/Wide-Area GPS (DGPS/WAGPS). [20]

Application	Land Survey	Machine Guidance	Natural Resources	Natural Hazards	Marine Navigation	Air Navigation
Category	RTK		PPP		DGPS/WAAS	
Coverage	Regional		Global		Coastal/Continental	
Purpose	Positioning		Monitoring		Navigation	
Communications	Terrestrial				Satellite	
Corrections	Commercial				Public	
Performance	Accuracy		Reliability		Integrity	
Method	Differential		Undifferenced		Differential	
Observable	Carrier phase		Carrier Phase		Pseudorange	
Precision	CENTIMETRE				METRE	

20 100 200 500

Reference Station Spacing (km)

Figure 2.2: An Overview of GNSS augmentation [20]

Figure 2.2 presents an overview of GNSS augmentation. According to this figure, GNSS augmentation can be divided into three categories based on coverage areas, as follows:

1. **Regional**: works in limited areas and relies on nearby ground-based stations.
2. **Global**: works in anywhere in the world and typically through satellite-only.
3. **Coastal/continental**: works across large land areas or regions near coastlines, but not truly global.

As seen in above picture, apart from the coverage differences, it can be inferred that regional coverage services offer the most precise positioning, typically at *cm*-level accuracy, which is suitable for applications such as land survey and machine guidance that use RTK. On the other hand, PPP, which provides global coverage, offers slightly lower accuracy but is well-suited for applications, for example, natural resources management and monitoring natural resources. DGPS/WAAS delivers *m*-level accuracy, which is the lowest among three, but it is still appropriate for marine and air navigation since it operates effectively within coastal/continental coverages.

Despite presenting detailed absolute positioning information relative to the Earth, unfortunately, GNSS and GNSS augmentation systems do not solve all the challenges associated with localization systems. If GNSS and GNSS augmentation systems could offer uninterrupted localization in the absence of signal-obstructing sources, then it would serve as an ideal solution for many applications that rely on accurate pose estimation. Therefore, to address these limitations, GNSS sensors are often integrated with other sensors (commonly with IMU sensors) to provide a more robust solution for localization systems.

Chapter 3

Robot Localization

As described in 1.2, localization is basically the process of determining a robot's pose within either known or unknown environments. Common challenges in localization include how to determine the robot's pose correctly and mitigate uncertainty in the environment. These issues have become the main topics, with many researchers proposing various solutions. However, unfortunately, to date, no perfect solution exists, as each localization approach has its own strengths and weaknesses; thus, the choice of methods largely depends on the specific requirements of the task, the acceptable trade-offs, and the intended objectives. Although numerous localization approaches exist, but this study does not aim to provide an in-depth comparison of these methods. Instead, it focuses solely on the specific localization method.

3.1 Environment Representations

There are various ways to represent environments, as outlined in 1.2. An environment representation consists of information about the space in which a robot operates. The environment representation is essential for a robot, particularly when it lacks prior knowledge of its surroundings, as its functionality become limited in such situations. Understanding the environment enables the robot to effectively plan its motion, avoid obstacles, perform navigation, and carry out other related tasks. Generally, the robot acquires this information through the use of sensors. However, unfortunately, sensor data is inherently noisy, thus, the robot must recursively maintain its belief with respect to the state of environment. Such noise can be mitigated through sensor fusion using estimation algorithms, such as the Extended Kalman Filter (EKF).

Various types of map representations are available. For example, the grid-based metric paradigm, which represents environments using evenly spaced grids and the topological paradigm, which represents environments as graphs, were both used by [21] for mapping indoor environment. [22] distinguished the three types of world models: geometric, topological, and semantic models. According to [22], the geometric model is directly derived

from perception data; the topological model provides an understanding of structure based on connectivity; and the semantic model is a symbolic model representation that contains information about objects, space properties and their relationship. Feature-based environment representations, such as line-based models, were utilized by [23] for navigation tasks in indoor environments. These are just a few examples of the many representations available.

3.1.1 Pose Estimation

Before delving into pose estimation using the Kalman Filter, this study will first examine Bayesian filters, as it is an essential tool for statistical estimation and for optimally integrating disparate sources of information. This process determines an optimal belief about the current system state by combining the prior belief (based on the previous state) with the current observations.

Mathematically, Bayes filters are described using conditional probability. However, before presenting the formal derivation, this study will first define all parameters involved in the Bayes filter as applied to localization. Following the *Markov assumptions*, the state is defined as a *complete* state, denoted by $\boldsymbol{x}_t$, which contains all necessary information about past events and states required to predict future states. Additionally, through interaction with the environment, the robot has access to two types of data, which can be categorized as follows: [24]

1. **Environment measurement data** presents details about a brief state of environment acquired over time. The notation for environment measurement data within a time interval is denoted as follows:

$$z_{t_1:t_2} = z_{t_1}, z_{t_1+1}, z_{t_1+2}, \ldots, z_{t_2} \tag{3.1}$$

   representing the set of all measurement data collected for $t_1 \leq t_2$.

2. **Control data** presents information about change in the system state. The control input at time t is denoted by u_t. The variable u_t represents the influence of the robot's action on its state within a time interval for $t_1 \leq t_2$, thus the sequence of control inputs is denoted as follows:

$$u_{t_1:t_2} = u_{t_1}, u_{t_1+1}, u_{t_1+2}, \ldots, u_{t_2} \tag{3.2}$$

The state x_t is sufficient to predict (potentially noisy) measurement z_t; thus, the resulting conditional probability distributions can be divided as follows:

1. **State transition probability**, denoted as $p(x_t \mid x_{t-1}, u_t)$, represent the probability of the system transitioning to state x_t given the previous state x_{t-1} and control input u_t. When the time index (t) is omitted, the transition can be rewritten as$p(x' \mid x, u)$, where x' is the successor state and x is the predecessor state.

2. **Measurement probability**, which is described as $p(z_t \mid x_t)$, represents the likehood of receiving observation z_t given that the system is in state x_t. When the time index (t) is omitted, this can be written as $p(z \mid x)$.

3. A *belief*, denoted as $bel(x_t)$, is a fundamental concept in robotics, as it defines the robot's internal knowledge about the *state* of the environment. It is represented by a conditional probability distribution that assigns a probability (or density value) to each possible hypothesis with respect to the true state. The belief can be divided into two types: prediction and correction steps.

3.2 The Kalman Filter

The Kalman filter is a special case of the Bayes filter, providing an exact solution to the Bayes filtering problem in the context of linear Gaussian systems subjected to Gaussian noise. In essence, the Kalman Filter is a specific implementation of the linear-quadratic estimator (LQE), as both are designed with the purpose of producing optimal estimates with respect to any quadratic function of the estimation error, even in the presence of noise. This technique was introduced by Rudolf E. Kalman in 1960. Nowadays, Kalman filtering is one of the most widely adopted across many fields, more importantly, *estimation* and *performance analysis*. [25]

For the sake of convenience, this equation (3.3) will be introduced in a simplified form, as it will be used extensively in the equations presented in the following paragraphs.

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{3.3}$$

As seen in equation (3.3), the probability density over the variable x is characterized by two parameters: the *mean*, denoted as μ and the *covariance*, denoted as Σ. These parameters must satisfy the following requirements: the mean μ must be a vector with the same dimensionality as the state x and the covariance matirx Σ must be symetric and positive-semidefinite. Additionally, the Kalman filter is parameterized by the *belief*, which is represented by the *mean* μ and the *covariance* Σ. As described in 3.1.1, there are three kinds of types that must be expressed as a linear function and be perturbed by Gaussian noise as follows:

1. **State transition probability**, denoted as $p(x_t \mid x_{t-1}, u_t)$, is described by the following equation:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{3.4}$$

x_t and x_{t-1} are state vectors and u_t is the control vector. A_t and B_t are matrices, where A_t is a square matrix of size $n \times n$, where n is the dimension of the state vector x_t and B_t is size $n \times m$, where m is the dimension of the control vector u_t. ϵ_t is the uncertainty in the state transition. It has the same dimensionality as the state vector, and its mean is zero; therefore, its covariance Σ is denoted as R_t.

The State transition probability $p(x_t \mid x_{t-1}, u_t)$ is defined by equation (3.4), thus, the *mean* of the posterior state is given by $A_t x_{t-1} + B_t u_t$ and its covariance is given by R_t using the equation (3.3)

$$p(x_t \mid x_{t-1}, u_t) = \det(2\pi R_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^\top R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\right\} \tag{3.5}$$

2. **Measurement probability**, denoted as $p(z_t \mid x_t)$, is described the following equation:

$$z_t = C_t x_t + \delta_t \tag{3.6}$$

C_t is a matrix of size $k \times n$, where n is the dimension of the measurement vector z_t. The measurement noise δ_t has the same dimensionality as the measurement vector, and its mean is zero; therefore, its covariance Σ is denoted as Q_t.

The measurement probability $p(z_t \mid x_t)$ is defined by equation (3.4), thus, the measurement probability is defined using the equation (3.3) as follows:

$$p(z_t \mid x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\right\} \tag{3.7}$$

3. A *belief* consists of two stages: the prediction step and correction step. However, for the sake of simplicity, thus, this case will use the initial belief $bel(x_o)$, its mean μ_o and its covariance Σ_0 using the equation (3.3) as follows:

$$bel(x_0) = p(x_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1}(x_0 - \mu_0)\right\} \tag{3.8}$$

As explained above in the three points, these provide the basic understanding necessary for the subsequent steps. Rather than explaining each step in detail, this section presents the complete equations of the Kalman filter as illustrated in **Table 1**. For a detailed explanation of how to derive the Kalman Filter leads to the specific forms presented in lines 1, 2, 3, 4, and 5, refer to [26].

Table 1: The Kalman filter algorithm for linear Gaussian state transitions and measurements

procedure KALMAN FILTER$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
1: $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
2: $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
3: $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
4: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$
5: $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$
6: **return** μ_t, Σ_t
end procedure

The Kalman filter algorithm, in which the belief $\overline{bel}(x_t)$ at time t is represented by the mean μ_t and the covariance Σ_t. The filter operates on two sets of parameters: the input at $t-1$, represented by μ_{t-1} and Σ_{t-1} and the output at t, represented by μ_t and Σ_t. To perform this update, these parameters are required: the control u_t and the measurement z_t.

3.3 The Extended Kalman Filter

Unfortunately, in practice, many robotic systems involve non-linear systems. For example, a robot navigating along a circular path exhibits motion that cannot be expressed in a linear state model. Therefore, the Extended Kalman Filter (EKF) is introduced to extend the traditional Kalman filter to handle such non-linearities by linearizing the state transition probability and the measurement probabilities. These probabilities are expressed in non-linear functions as follows:

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \tag{3.9}$$

$$z_t = h(x_t) + \delta_t \tag{3.10}$$

The detailed algorithm of the EKF is presented in the below table. But as for a detailed explanation of how the use of non-linear functions leads to the specific forms presented in Lines 1, 2, 3, 4, and 5, refer to [26].

procedure EXTENDED KALMAN FILTER$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
1: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
2: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3: $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
4: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
5: $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
6: **return** μ_t, Σ_t
end procedure

In many respects, the algorithm for the Kalman Filter is similar to that of the EKF. The key differences between the two are summarized in the table below as follows:

	Kalman filter	**EKF**
state prediction (Line 1)	$A_t \mu_{t-1} + B_t u_t$	$g(u_t, \mu_{t-1})$
measurement prediction (Line 4)	$C_t \bar{\mu}_t$	$h(\bar{\mu}_t)$

As shown in the table, the Kalman filter operates on linear system matrices A_t, B_t and C_t, whereas the EKF relies on the Jacobian matrices G_t and H_t.

Chapter 4

Perception System

The perception system is an essential component in mobile robots, enabling the robot to interpret and understand its surroundings. In this study, a camera sensor, particularly a stereo vision camera, is utilized as the primary sensor to detect line markings on the surfaces. These line markings serve as visual references for the robot to estimate its pose relative to the known environment, enabling it to navigate through the area. This becomes especially critical for local localization in environments where **GNSS signals are either unavailable or significantly degraded**.

4.1 Stereo Vision System

Since this study uses a simulated camera that replicates the physical camera used in the Turf Tank product, specifically a stereo vision system, so, it will include a detailed discussion of the stereo vision approach.

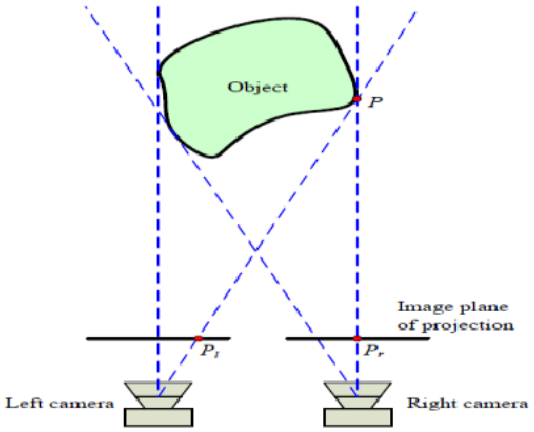


Figure 4.1: Two different positions of two cameras and their projection of image planes [27]

Fig. 4.1 depicts the basic geometry of a stereoscopic image captured by two identical cameras placed in different positions. This setup reflects the fundamental concept of stereo vision, which typically involves two cameras positioned on the left and right. This figure also shows that both cameras are aligned on the same plane in parallax sight. Additionally, it illustrates that the cameras are offset along the x-axis. For convenience in modeling the projection, the image planes are displayed in front of the cameras. [27]

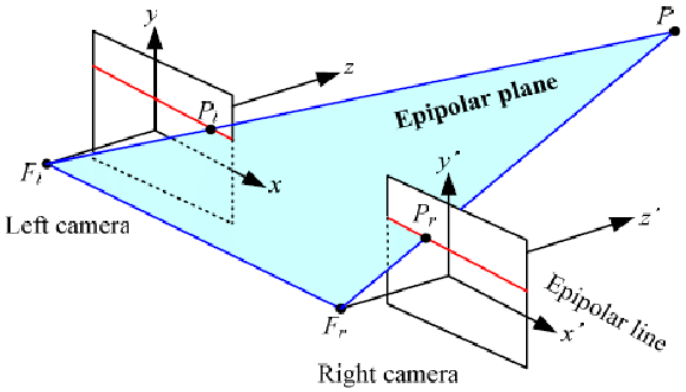


Figure 4.2: The epipolar geometry of stereoscopic vision [27]

Fig. 4.2 illustrates two different perspective views of an object point P from centers of the left and right cameras, denoted as F_l and F_r, respectively. It also shows the perspective projection of P onto the image planes of the left and right cameras, denoted as p_l and p_r, which are referred as a *conjugate pair*. From here, the plane that passes through both camera centers and the object point in the scene is known **epipolar** plane, and it corresponds to the point p_l and p_r in the image planes. [27]

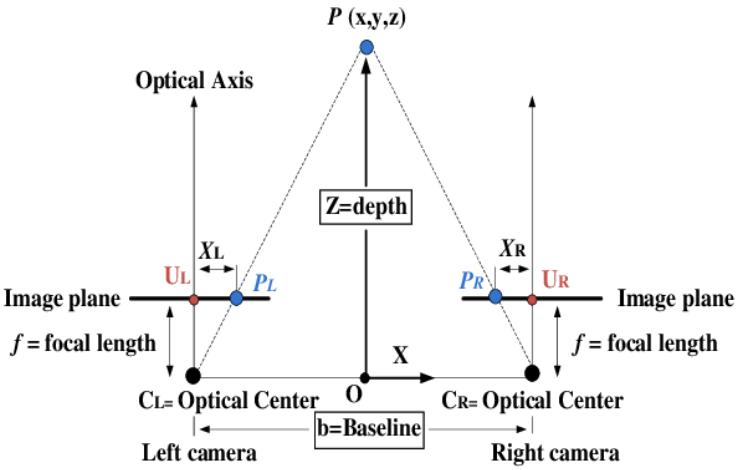


Figure 4.3: The epipolar geometry with parallel optical axes [27]

Fig. 4.3 shows the stereo camera coordinate system, which is thought to lie halfway between the left and right camera coordinate system. By applying the concept of similar triangles, as illustrated in Fig. 4.3, thus, the depth estimation equation can be derived under the assumption of ideal (pinhole) cameras as follows: [27]

$$Z = \frac{b * f}{x_L - x_R} = \frac{b * f}{d}$$

where the quantity ($d = x_L - x_R$), denoted as the disparity, b is the baseline, f is the focal length of the cameras.

Note: the perception process will be discussed in the following chapter 5.1.

Chapter 5

Methods

5.1 The Perception Processes

This the perception system is based on Turf Tank (see video in [28]), where Turf Tank has implemented the perception system that processes inputs based on the information from camera sensors facing backward. According to the video in [28], this configuration setup enables the robot to continuously detect lines on the ground surface behind it, allowing it to paint the lines in the forward direction aligned with the previously drawn lines, as detected by the backward facing camera. The overall the vision-based perception system is illustrated in Fig. 5.1, where the mobile robot is tasked with detecting linear features on surfaces within the simulation environment. The process of detecting straight lines using

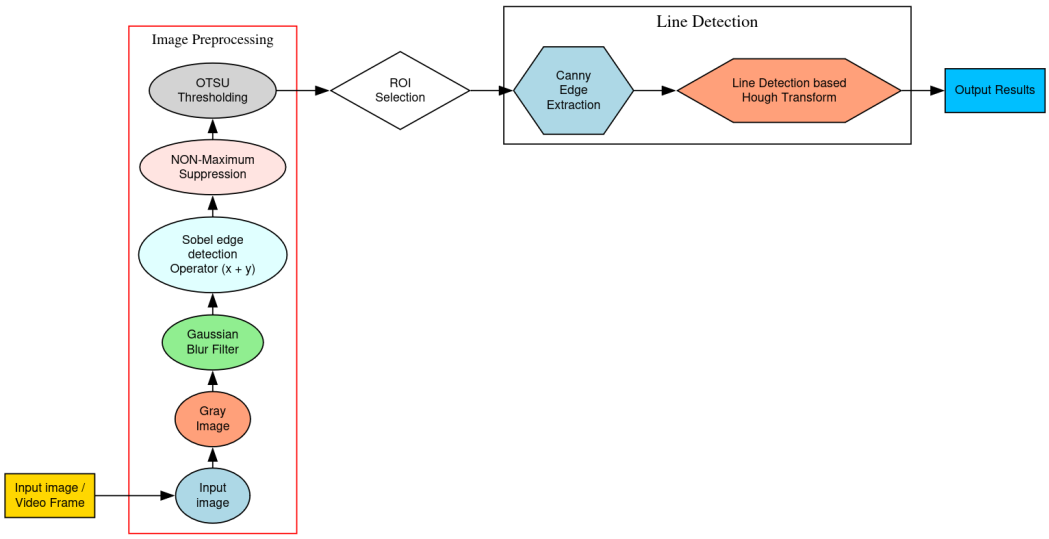


Figure 5.1: Overview of the perception system, modified from the image in [29]

the Hough Transform proceeds as follows: [29]

1. **Input Image**: This initial data is acquired from the surrounding environment using perception sensors, such as camera sensors. It contains meaningful visual information for further processing. However, raw image data is often insufficient due to various factors that may degrade its quality, such as noises and sensor-related errors. Thus, to address these issues, image enhancement referred as image pre-processing is required to improve the quality and reliability of the visual information.

2. **Grayscale Image**: As a perquisite for further processing, the raw data is first converted into a grayscale format. A grayscale image is a single-channel image that represent intensity values, eliminating the red, green, and blue (RGB) color channels found in the original image, making it suitable for subsequent steps such as edge detection.

3. **Noise Reduction**: After the original image is converted into grayscale format, but unfortunately, it may still suffer from unwanted information such as noise, which can adversely affect further processing steps. Therefore, a noise reduction technique is applied to enhance image quality by removing noise. A common approach for this task is the Gaussian filter. Although the Gaussian filter is helpful in preserving important features while smoothing the image, unfortunately, it suffers from unintended side effects, such as distorting parts of the image in regions with sudden changes in pixel intensity.

4. **Sobel Edge Detection Operator**: The Sobel operator is one of several commonly used edge detection algorithms. It approximates the gradient of image intensity in both horizantal and vertical directions. These gradients can be combined to compute the edge magnitude and orientation, thereby highlighting regions in the image where pixel intensities change rapidly.

5. **Non-Local Maximum Suppression**: The primary reason for using this technique is to determine an optimal threshold value that separates image intensities into two classes: white (foreground) and black (background).

6. **Otsu Thresholding**: The primary reason of using this technique to determine an optimal value (threshold value) that separates two intensities into two classes: white (foreground) and black (background)

7. **Region of Interest (ROI) Masking**: A region of interest is defined and applied to exclude irrelevant areas outside the region where lines are expected to appear. This step enhances detection accuracy and reduces computational load.

8. **Canny Edge Detection**: The Canny edge detection algorithm is applied to identify regions with significant intensity changes, which likely correspond to edges. In ad-

dition to detecting edges, the algorithm suppresses noise, improving the robustness of the results.

9. **Line Detection with the Hough Transform:** The masked edge image is then processed using the Hough Transform to detect straight lines. The resulting line segments are used by the mobile robot for localization tasks within the simulated environment.

5.1.1 Hough Transform for line

Before delving into the Hough Transform process, which will be discussed in the following subsection, this study will first present the Hough Transform in a mathematical form to provide a basic understanding.

The Hough Transform method was introduced by Paul Hough in 1962 [30] as a technique for detecting geometric shapes, most commonly straight lines, in binary or grayscale images. The fundamental idea behind the Hough Transform is to represent lines using a parametric form. In the standard Cartesian coordinate system, a line is typically expressed using the slope intercept form as follows:

$$y = mx + c \tag{5.1}$$

where collinear points in an image with coordinates (x, y) are described using slope m and intercept c. This equation can also be rewritten in a homogeneous form as follows:

$$Ay + Bx + 1 = 0 \tag{5.2}$$

where $A = -1/c$ and $B = m/c$. Thus, a line can be described by specifying a pair of parameters (A, B), which serve the coefficients in the homogeneous form of the line equation. Because Equation 5.2 is symmetric with respect to given parameters (x, y), thus it can represent the equation of a line given a point (x, y). Therefore, the same equation can be used to define both points (x, y) and lines (A, B) at the same time. When considering collinear points, Equation 5.2 can be rewritten as follows:

$$Ay_i + Bx_i + 1 = 0 \tag{5.3}$$

Additionally, the slope intercept c form of Equation (5.1), thus it can also be expressed as:

$$c = -mx_i + y_i \tag{5.4}$$

Unfortunately, this system is generally overdetermined with increments of mx_i, which requires more computations. To address this problem, Duda and Hart [31] proposed a solution to this problem by basing the mapping function on an alternative parameterization, known as the foot-of-normal form. This leads to the more robust polar Hough Transform, as follows:

$$\rho = x\cos\theta + y\sin\theta \tag{5.5}$$

where θ is the angle of the line normal to the line in an image and ρ is the distance between the origin and the point where the line intersects (see Fig. 5.2).

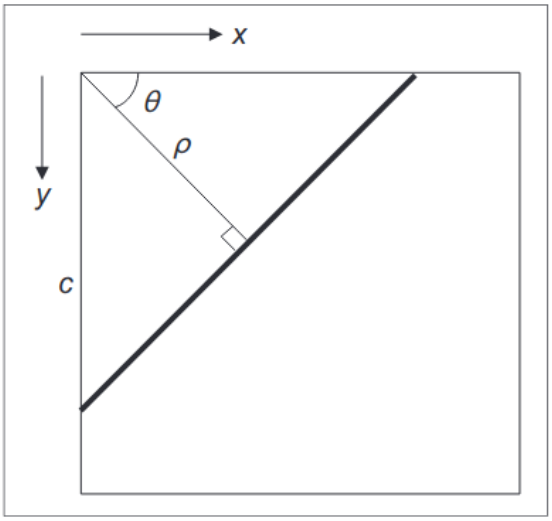


Figure 5.2: Overview of Polar consideration of a line [32]

Taking into account that two lines are perpendicular if the product of their slopes is -1 and the geometry described in Fig. 5.2. Thus, the equation could be described in Equation (5.6).

$$c = \frac{\rho}{\sin\theta}, \quad m = -\frac{1}{\sin\theta} \tag{5.6}$$

By referring equation (5.5), a different mapping function is introduced: points in the image space are cast into a 2D accumulator (see Fig. 5.3) array based on the values of ρ and θ parameters, forming *sinusoidal* curves. One advantage of this approach is that the values of ρ and θ parameters are bounded within a specific range, where $\theta \in [0°, 180°)$, and $\rho \in [-\sqrt{2}N, \sqrt{2}N]$, with N representing the size of the image.

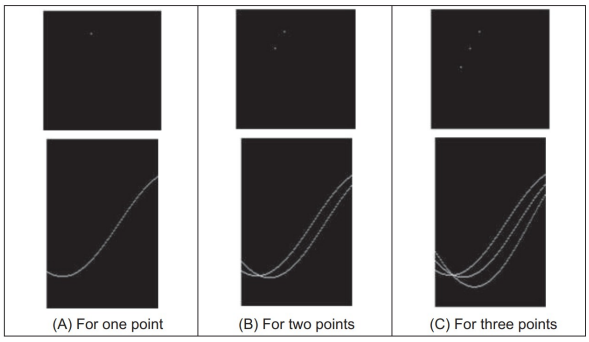


Figure 5.3: Accumulator space of the polar Hough transform [32]

Fig. 5.3 illustrates the polar Hough Transform accumulator spaces. The left, middle, and right images correspond to one, two, and three points in the image space, respectively. Each point casts a vote in the accumulator space, forming *sinusoidal* curve. As more points are added, more sinusoidal curves are generated. The intersection points of these curves indicate potential lines that pass through multiple points in the image.

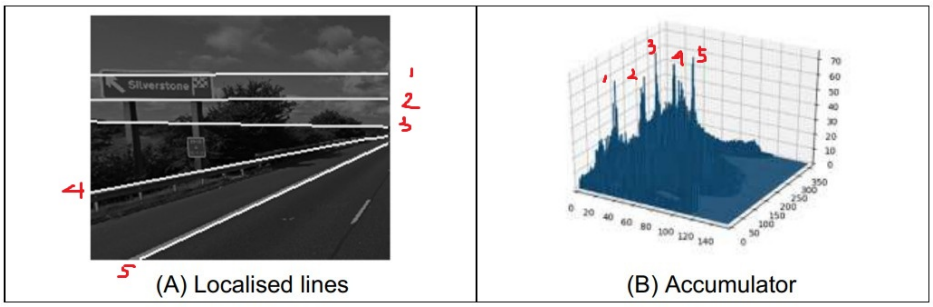


Figure 5.4: An example of applying the polar transform with the Hough Transform for lines [32]

The purpose of the Hough Transform for lines is to identify potential lines by looking at the tallest and most noticeable peaks in the plot. The higher peaks indicate that the more likely the line is in an image. To illustrate this concept, Figure 5.4 is used in this study. As shown in the figure, there are five noticeable peaks, indicating the presence of five localized lines in the image.

Note: In essence, the Hough transform method can be extended to arbitrary curves such as circles, ellipses, and others, as long as the parameterization is convenient. In other words, the Hough transform could be utilized for different shapes if parameterized effectively.

5.2 The Hough Transform process

The primary motivation for employing the Hough Transform lies in its effectiveness at detecting features that exhibit linear structures. Since the objects of interest in the environment appear as straight lines, this method is theoretically well-suited for such scenarios. The process of detecting a straight line using the Hough Transform (see Fig. 5.5) as follows: [33]

1. Identifying the edge contours of set of images (videos) from captured by camera sensors in space.
2. Involving binarizing the set of images and locating for non-zero coordinates in the binarized images.
3. The key data point (the location of non-zero coordinates within the binarized images) is transformed using the Hough Transform.
4. Define a threshold value and locate the point within the parameter space whose value exceeds the threshold value. In the corresponding Cartesian Space, these point pairs are collinear.
5. Finally, the output of transformed image contains the detected lines.

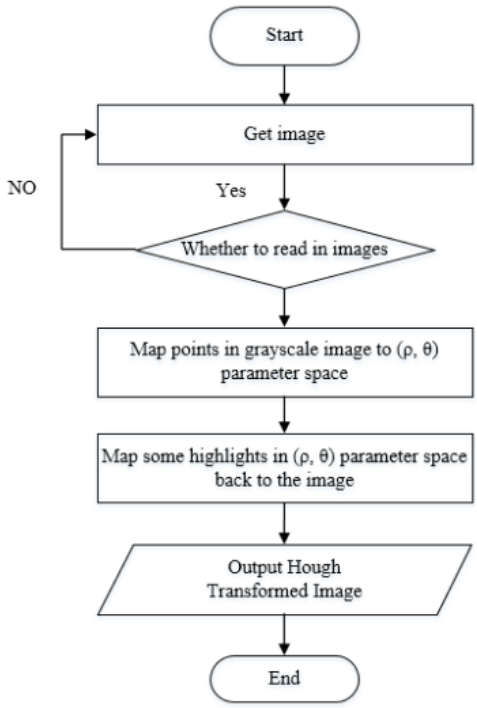


Figure 5.5: Overview of the perception system based on a modified image from [33]

The figure above illustrates the algorithm flowchart of the Hough transform method.

5.3 Odometry

The term "odometry" originates from the Greek words *hodos*, meaning "travel" or "journey," and *metron*, meaning "measure." In the context of robotics, odometry can be defined as the process of estimating a robot's pose and motion using onboard sensors (e.g., wheel encoders, IMU, GPS, Laser sensors, among others). Specifically, odometry provides an estimate of the robot's position and orientation over time, however, When this estimation is carried out incrementally over time without external references, the process is known as *dead reckoning*. One of the most widely used approaches to odometry in mobile robotics is based on the differential drive model, due to its simplicity and prevalence in practical platforms.

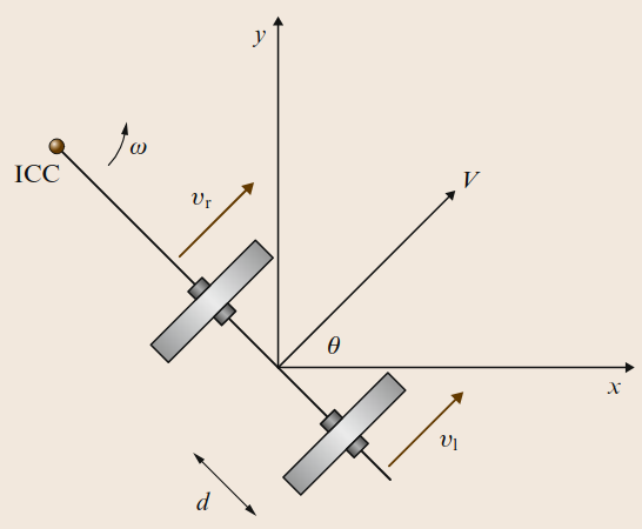


Figure 5.6: Differential Drive kinematics
[17]

As illustrated in Fig. 5.6, a differential drive robot consists of two independently controlled wheels mounted on a common axis. These wheels are assumed to be fixed to the robot's chassis and maintain continuous contact with the ground. When both wheels rotate at the same speed, the robot moves in a straight line. When the wheels rotate at different speeds, the robot follows a circular arc, rotating around a point called the *Instantaneous Center of Curvature* (ICC), which lies along the axis connecting the two wheels. The ground contact speeds on the left wheel and right wheel are denoted as v_l and v_r, respectively, and the wheels are separated by a distance $2d$. These parameters can be expressed in the following

mathematical equations:

$$\begin{aligned} \omega(R + \frac{l}{2}) &= V_r \\ \omega(R - \frac{l}{2}) &= V_l \end{aligned} \tag{5.7}$$

where l denote the distance between the centers and wheels, and let v_r and v_l represent the linear velocities of the right and left wheels, respectively, relative to the ground. R denote the signed distance from ICC to the midpoints between the wheels. The following equations 7.8 can be re-arranged to solve the rate of rotation ω about the ICC and the distance from the center of the robot to ICC R as follows:

$$\begin{aligned} \omega &= \frac{v_r - v_l}{2d} \\ R &= d\frac{v_r + v_l}{v_r - v_l} \end{aligned} \tag{5.8}$$

Based on the values of vr and v_l, the three motion cases can be distinguished as follows:

1. If $V_l = V_r$, then $R \to \infty$ and $\omega = 0$. Thus, the robot moves in a straight line without rotation.
2. If $V_l = -V_r$, then $R = 0$, and the robot rotates in place around the midpoint of the wheel axis.
3. If $V_l = 0$, then the robot rotates about the left wheel, with $R = \frac{l}{2}$. Similarly, with the case of if $V_r = 0$.

Let v_l and v_r, denote as the time-dependent linear velocities of the left and right wheels, respectively. By defining the point midway between the two wheels as the origin of the robot's body frame, and letting θ_t represent the orientation of the robot with respect to the x-axis of a global Cartesian coordinate system, thus the robot's pose can be determined as follows:

$$\begin{aligned} x(t) &= \int V(t)\cos[\theta(t)]dt \\ y(t) &= \int V(t)\sin[\theta(t)]dt \\ \theta(t) &= \int \omega(t)dt \end{aligned} \tag{5.9}$$

5.4 Visual Odometry

Visual Odometry (VO) is the process of estimating the pose of an agent (e.g., a mobile robot, a vehicle, among others) using a stream of images obtained from one or more cameras attached to the agent. The basic working principle of VO is based on the incremental estimation of an agent's pose by analyzing image sequences captured by camera sensors.

The advantages of using this method is that VO provides more accurate trajectory estimates, with the relative position error ranging approximately 0.1% to 2%. Thus, VO could become an excellent alternative to conventional odometry or serve as a complementary approach-particularly in GPS-denied scenarios, where GPS can not provide accurate pose estimation; in such cases, VO comes to the rescue. [34]

Despite the advantages provided by VO, it still has weaknesses that could be detrimental to pose estimation. Since VO is based on visual perception, it is heavily dependent on the ability of cameras to operate effectively in dynamic environments, which may hinder the perception process. For example, in outdoor environments, unpredictable conditions such as sunlight or shadows can complicate the localization process.

Note: The table below provides a comparison of commonly used sensors in the localization process. [35]

Sensor/technology	**Advantages**	**Disadvantages**
Wheel odometry	Relatively simple in estimating agent's pose Enables high sampling rates with short-term accuracy Inexpensive	Wheel slippage causes position drift Accumulation of errors over time Accurate numerical differentiation is required to estimate acceleration, velocity, and position even in the presence of noise
IMU	Not subject to interference outages	Position drift Have long-term drift errors
GPS/GNSS	Provides absolute position with known value of error No error accumulation over time	Can not operate efficiently in indoor, underwater, and closed areas
Cameras	Store a significant amount of meaningful information within an image High level of localization accuracy Low-cost solution	Necessitates image-processing and data-extraction techniques Since processing images requires expertise, this could be expensive

Chapter 6

Simulation Tools

6.1 The Robot Operating System (ROS)

The Robot Operating System (ROS) was originally developed at Stanford University as part of the STAIR project and was later expanded by the robotics startup, Willow Garage. Its design philosophy emphasizes several key principles of ROS as follows: [36]

1. **Peer-to-peer**: Its purpose is to promote decentralized architecture that enables flexible communication and collaboration between nodes and one another. In addition, the decentralized architecture allows the system to operate independently; if one node fails, other nodes can continue to function normally. This contrasts with the centralized architecture, where the failure of a single node can cause the entire system to fail.

2. **Tools-based**: The purpose of this is to provide various tools libraries to achieve specific goals. For example, ROS provides tools to perform navigation tasks, perception tasks, localization tasks, and more.

3. **Multi-lingual**: Since there are various programming languages in the programming realm, it is important to accommodate the user's preference language. This allows the integration of various components into a single component, thus enhancing accessibility and usability. For example, the ROS libraries for motion planning are written in C++, but users can write their code in Python and seamlessly integrate it with the C++ libraries.

4. **Thin**: ROS encourages design code that is as thin as possible, so that code can be re-used with different robot frameworks.

5. **Free and Open Source**: ROS is publicly available and distributed under the BSD license, allowing users to use it in both commercial and non-commercial projects.

ROS1 became one of the most widely adopted frameworks in the robotics community, mainly due to its extensive library of tools and packages. However, as time flies by, the adoption of ROS1 has further advanced, and ROS1 has begun to show its limitations. Consequently, ROS2 was introduced to address these limitations. Table [37] compares the major architectural differences between ROS1 and ROS2.

Category	**ROS1**	**ROS2**
Network Transport	TCP/UDP	DDS
Network Architecture	*roscore*	Peer-to-peer discovery
Platform Support	Linux, Windows, macOS	Linux, Windows, macOS
Client Support	*rospy* and *roscpp*	Underlying C libraries (*rcl*)
Node vs. Process	Single node per process	Multiple nodes per process
Threading Model	Callback handlers	Callback queues, Swappable executor
Node State Management	None	Lifecycle nodes
Embedded Systems	*rosserial*	micro-ROS
Parameter Access	XML-RPC	Service calls
Parameter Types	Types inferred when assigned	Types declared and enforced

6.1.1 Fundamental Concepts of ROS2

A node is the basic computational unit in ROS2, which responsible for communication with other nodes (see Fig. 6.1).

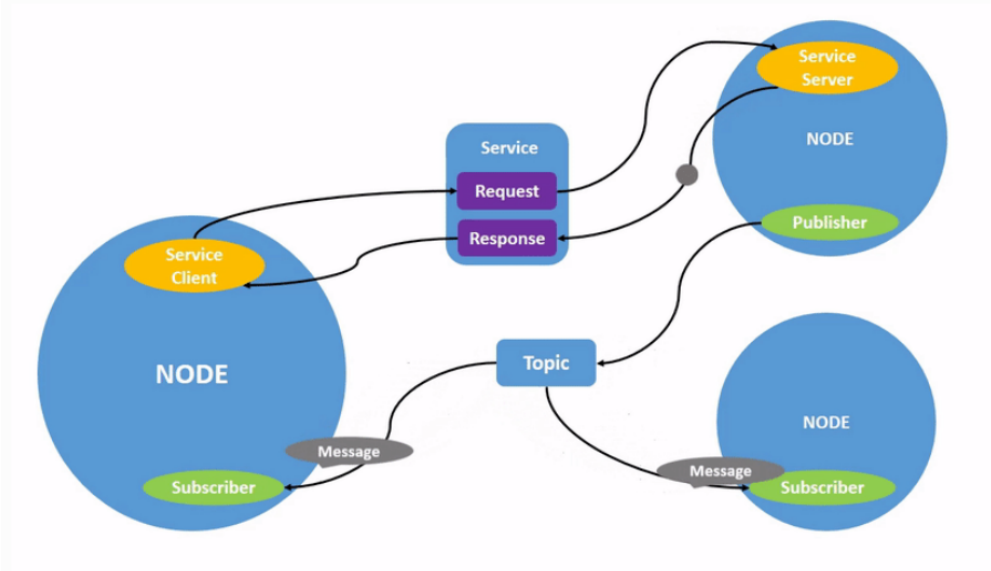


Figure 6.1: ROS2 node graph
[38]

The process communication mechanism in ROS2 interfaces is as follows: [38]

1. **Topics**: Topics are implemented in ROS2 to act as a bus for nodes to transport messages from one node to other nodes. As illustrated in Fig. 6.1, it specifically shows that the topic interacts between subscriber and publisher nodes. The subscribers only need to know what, and which topics are relevant without the need to know who the publishers are. In ROS2, topics are usually implemented for streaming data purposes such as information about robot states or image processing from sensors.

2. **Services**: Services are implemented in ROS2 based on request-and-response communication. This framework is a type of one-way transport system. Services are essentially where one node (client) makes a request to another node (server) and the server receives the request and responds to the client. Services in ROS are formatted in asynchronous communication.

3. **Actions**: Actions are goal-oriented tasks that can be monitored or canceled (preempted). The actions framework is essentially the same as the services framework, but it includes preemption and feedback interfaces.

6.2 Gazebo Simulator

Gazebo is a 3D Robotics Simulator that, when coupled with ROS2, has become increasingly popular in research, design, and development related to robotic systems as it offers high-performance and realistic simulation environments. Additionally, Gazebo also provides realistic sensor feedback, closely resembling that of the real world, and is designed to replicate dynamic environmental attributes such as mass, velocity, friction, gravity, and more attributes. Although Gazebo is designed as accurately as possible to reflect behaviors in real-world situations, it does not mean that it is intended to become the replacement for real-world testing as there are limitations in Gazebo and the unpredictable environment in the real world. [39]

The development of Gazebo's architecture has gone through repeated trial and error, enabling the extension of its features and improvements in terms of performance as outlined in Fig. 6.2.

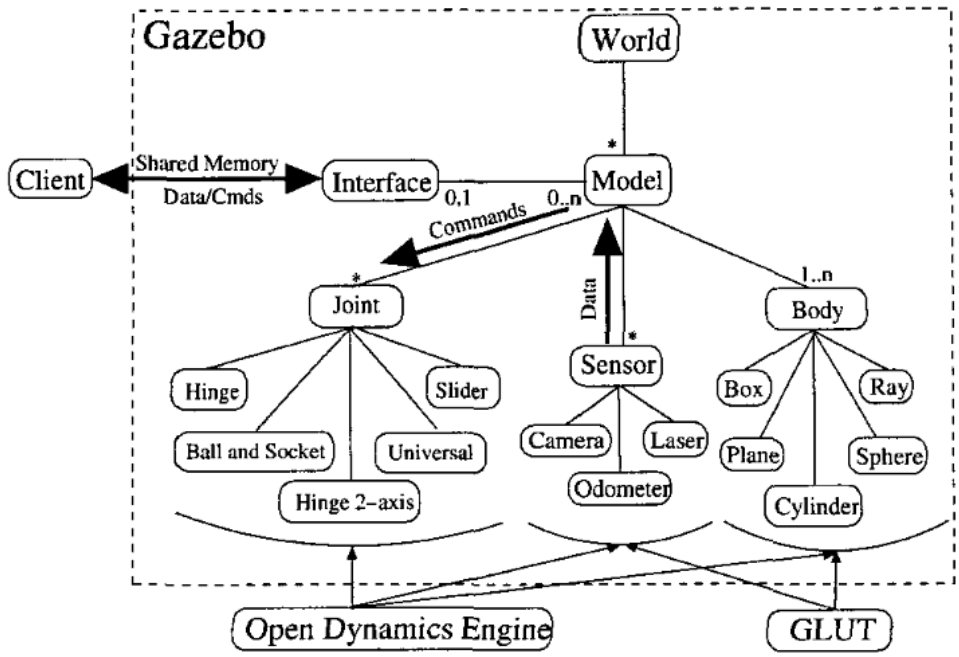


Figure 6.2: Gazebo's Architecture
[39]

The above pictures illustrate Gazebo's components, and their interactions as follows: [39]

1. **World**: A complete environment consisting of multiple models.
2. **Model**: Basically, any object that represents physical representations, which encompasses as follows:
   (a) **Bodies** represent the physical structure of a model in the form of geometric shapes such as boxes, spheres, cylinders, planes, and more.
   (b) **Joints** are basically the mechanisms that connect bodies to form kinematic and dynamic relationships. There are various types of joints such as hinge, ball and socket, slider, universal, hinge 2-axis, and more.
   (c) **Sensors** are used for data collection purposes. Examples of sensors are cameras, lasers, odometers, LIDAR, and more.
3. **Interface**: Basically, communication, in which the client can access and send commands to instruct the model, for instance, move joints, request sensor data, and so on.
4. **Client**: Interacts with the interface in Gazebo through shared memory to send commands or receive data to/from the interface.

Additionally, Open Dynamic Engine, a physic engine, is utilized in Gazebo to offer dynamic and kinematic simulations that are associated with various attributes such as mass, joints, collision, rotational function, and more. Gazebo also offers visualization tools such as OpenGL and GLUT. [39]

Another benefit of using Gazebo is that Gazebo is an open-source and cross-platform, supporting mainstream operating systems such as Linux, Windows, and macOS. In addition, Gazebo also can be seamlessly integrated with cloud services through app.gazebosim.org, enabling users to access it anywhere and anytime as long as they have internet access. However, Gazebo has several weaknesses such as a particular demand for power when simulating large resources such as large environments or multiple robots, which necessitate powerful machines, steep learning curves, and more. [40]

6.3 RViz2

RViz2 stands for Robot Visualization, which is a general-purpose 3D visualization environment using ROS2, which is developed and maintained by the Open-Source Robotics Foundation (OSRF). RViz2 is used to analyze and visualize a variety of data types streaming including sensor data, robot models, and others through ROS2, while emphasizing the three-dimensional nature of the data. The core components of Rviz2 as follows: [41]

1. **Displays**: An element enables users to visualize something (e.g., the robot model) in the 3D visualization space.
2. **Configurations**: This element allows users to adjust properties within Rviz2 and enable them to load and save configuration files.
3. **View Panel**: This enables users to adjust the scene based on the camera perspective (e.g., orbital camera, first-person camera, and more) in the visualization window.
4. **Coordinate Frames**: display the relationship between different frames within the robot system.

Chapter 7

Implementation

7.1 The Implementation of Simulation

In this study, the implementation of simulation through libraries within ROS2, Gazebo, and RViz2. This study first introduces the robot model that will become central to the implementation in the simulation. The robot model is equipped with various sensors that are important for this study, such as GPS, IMU, wheel encoders, and cameras sensors. The robot model in this study is based on the Yahboom ROSMASTER X3, which was forked from Github [42] and then adapted into ROS2 in this study, as their original model was based on ROS1. The robot model for this study is described in Fig. 7.1.

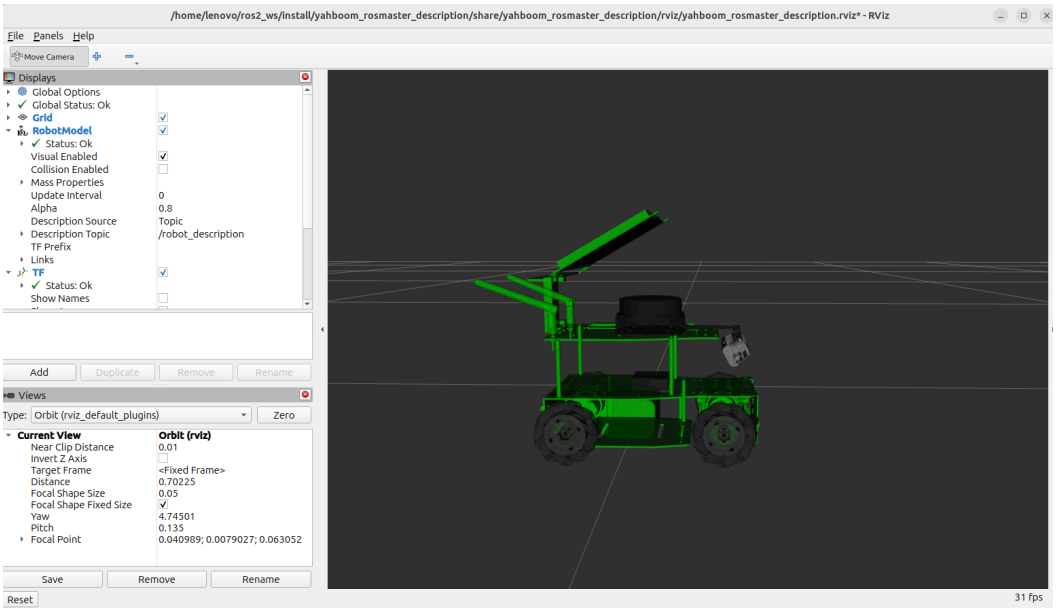


Figure 7.1: The robot model

According to the specifications [42], the Yahboom ROSMATER X3 is a mobile robot equipped with four Mecanum wheels, which enable it to move in any direction be backward, forward, sideways, or rotationally. Additionally, it is equipped with LiDAR (e.g., YPLIDAR4 ROS), depth cameras (e.g., Astra Pro depth camera), IMU, motor encoders, and optionally a GPS module (which can be added to the robot chassis). As the robot chassis is wide, it allows for the installation of multiple sensors, making it ideal for robotic localization tasks. This is why the Yahboom ROSMASTER X3 is used as the robot model in this study's simulation. A comprehensive overview of the sensors utilized in the simulation is presented in Fig. 7.2.

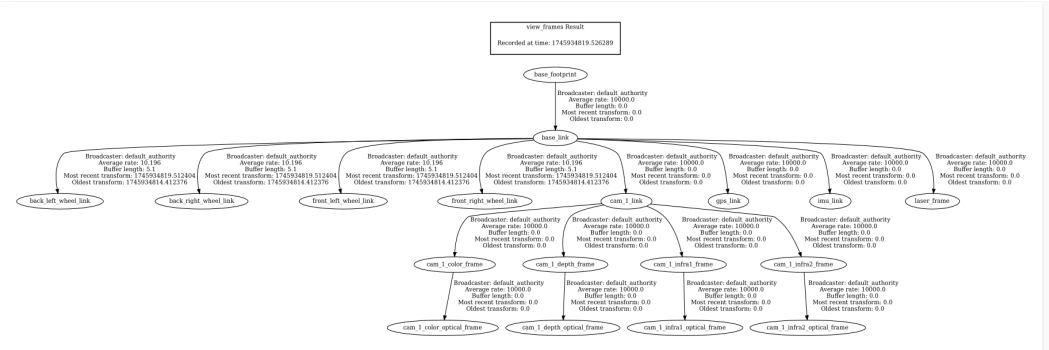


Figure 7.2: The robot model's frames

The robot model's frames describe a structure that mirrors the real-world mobile robot, for detailed frames of the robot model in simulation are as follows:

1. *base_footprint*: represents the root frame, which describes contact with the ground and connects to the *base_link* node. It is typically a fixed frame used for localization and navigation purposes.

2. *base_link*: Represents the chassis of the robot model in the simulation. This is the central link to which all other components (sensors, actuators) are connected.

3. *wheel_link*: Represents the wheels in the robot model, enabling the robot's mobility. In this simulation, the robot model is represented by mecanum wheels, which consist of four wheels in front and back on the left and right sides that allow it to move forward, backward, or rotate.

4. *cam1_link*: Represents the frame of camera sensors. Its frame consists of depth, infrared, color, and optical frames.

5. *gps_link*: Represents the frame of GPS sensors, which measures robot's pose relative to the global map.

6. *imu_link*: Represents the frame of IMU sensors, which measures the robot's orientation, acceleration, and angular velocity.

7. *laser_frame*: Represents the frames of LiDAR sensors, which measures the distances to objects. Its frame is used for SLAM tasks.

As outlined in REP 105, which defines the rules for coordinate frame conventions for mobile robots, the following standards are illustrated in Fig. 7.3

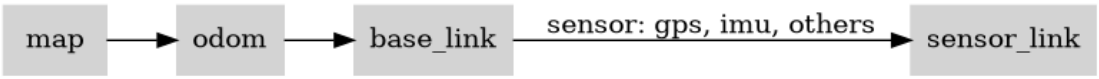


Figure 7.3: Coordinate frame conventions

The frame flow defines the structure of coordinate frame conventions as follows:

1. *map* serves as a global reference frame and it is typically fixed.
2. *odom* serves as the local reference frame. The transformation from *map* to *odom* is generally provided by ROS packages such as AMCL or other related localization packages.
3. *base_link* is the reference frame attached to the robot's chassis. It typically moves relative to *odom* through sensor fusion using ROS packages as the robot navigates.
4. *sensor_link* is a reference frame attached to the sensor, such as the camera, lidar, IMU, or other related sensors.

In this study, the simulation uses the **robot_localization** package, which describes the implementation of the Extended Kalman Filter (EKF) through the **ekf_localization_node** and the **navsat_transform_node**. These nodes for the **ekf_localization_node** are defined in the simulation as follows (see Fig. 7.4 and Fig. 7.5):

- **ekf_localization_node_odom** performs sensor fusion by integrating data from various sensors, including wheel encoders via the */wheel/odometry* topic, the IMU via the */imu/data* topic, the camera via the */cam1/depth* topic, and the GPS via */gps/data*. The fused output is published on the */odom* topic in the *nav_msgs/msg/Odometry* message format and represents the robot's estimated cumulative pose over time.
- **ekf_localization_node_map** focus on estimating the robot's pose in the global map frame. This node serves as the ground truth of the robot's pose in the simulation.

The other node for the **navsat_transform_node** is described as follows (see Fig. 7.6):

- **navsat_transform_node** performs the estimation of the robot's pose with respect to the global reference frames with the Adaptive Monte Carlo (AMCL) method.

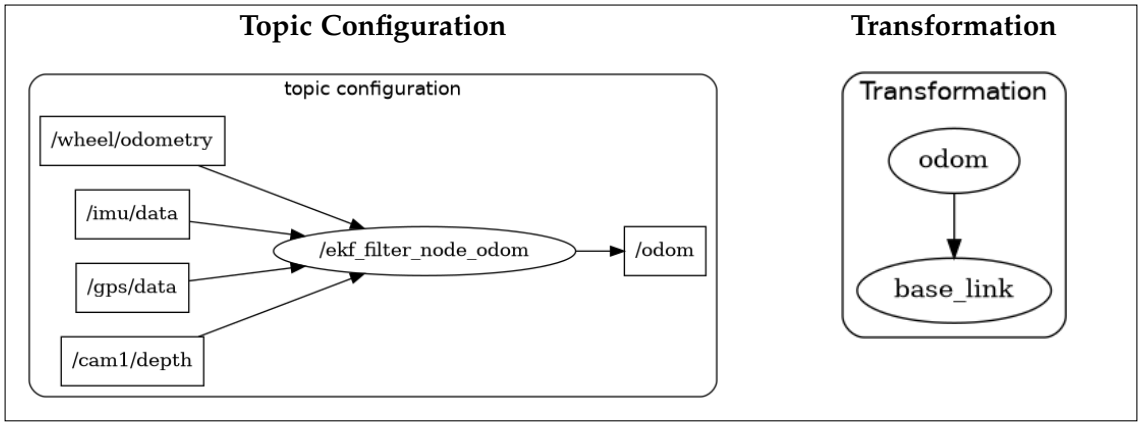


Figure 7.4: ekf_localization_node_odom node

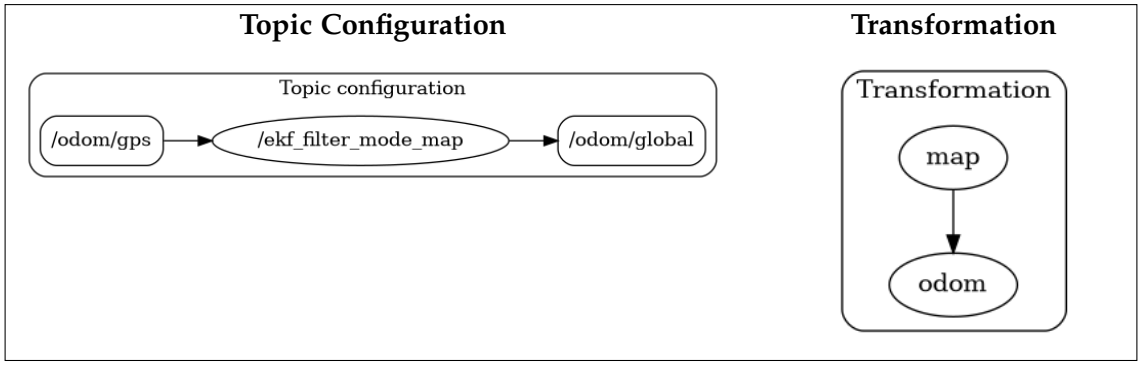


Figure 7.5: ekf_localization_node_map node

In order for the GPS node to function correctly in the simulation, accurate mapping and localization are required. This process is commonly known as SLAM (Simultaneous Localization and Mapping) techniques. However, in this study, SLAM is not discussed in detail, as it falls outside the scope of this work.

For sake of simplicity, SLAM techniques, especially the AMCL method, are only briefly presented to provide the context in this work. Since this study utilizes the **Nav2** packages, thus, the AMCL method, which is a component of **Nav2**, is used to localize the robot within its known surrounding environment.

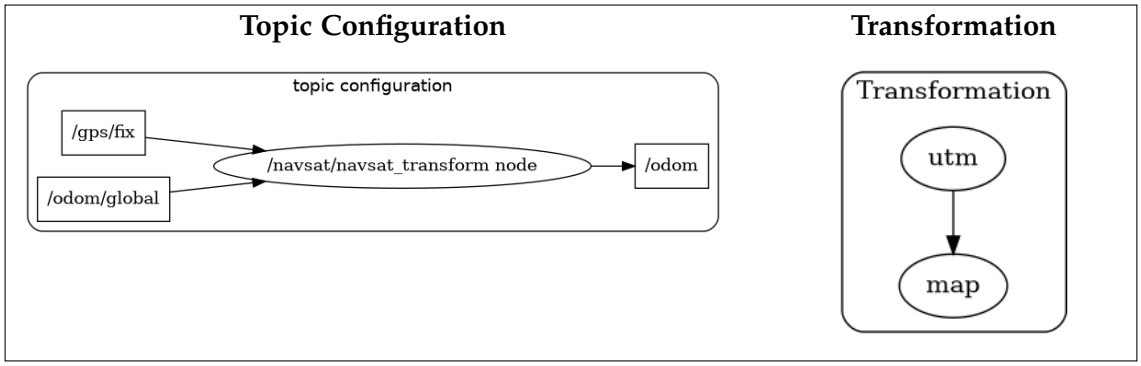


Figure 7.6: navsat_transform_node node

The diagram illustrates the flowchart of **navsat_transform_node**. In this simulation, the map coordinates with respect to the surface of the Earth are described using **UTM** coordinate system. Measuring absolute orientation is required when simulating the robot model using **robot_localization** package with the GPS sensor. To measure absolute orientation, one can use IMUs with magnetometers, as described in the *odom/global* topic, and matching techniques over a known map through the */gps/fix* topic using the AMCL method.

7.2 Alternative Implementation Method

This part of the dispensation is due to the challenges in simulating ROS2, particularly with visual odometry, which is difficult to run effectively in simulation. This is because these systems rely heavily on capturing features from unique surface textures and patterns. To achieve reasonable performance, a photorealistic simulator such as Unreal Engine or Isaac Sim is required. Unfortunately, the laptop used for the simulation does not have a GPU graphics card, particularly one with Nvidia drivers. Therefore, the final simulation will be conducted in MATLAB, using the generated lines are straight and that the implementation closely resembles what is offered in the perception system, without performing perception system simulation. The overall implementation of Autonomous Navigation System Architecture is described in Fig. 7.7.

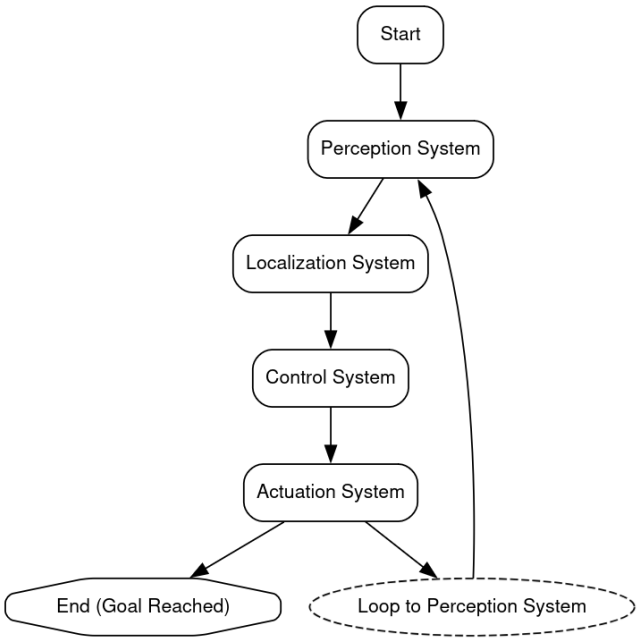


Figure 7.7: The overview of Autonomous Navigation System Architecture

The diagram represents the system architecture of the mobile robot, which consists of four systems, including perception, localization, control, and actuation systems. The mobile robot first undergoes the system initialization, which serves as the beginning of a continuous perception-action loop that continues until the robot reaches its goal. During the perception stage, the mobile robot collects input data from perception sensors (e.g., camera sensors), including line detection data.

In the following stage, the collected sensor data are then fed into the localization systems, which is responsible for estimating the robot's pose relative to the environment. This estimation enables the mobile robot to navigate based on the collected information. In addition, the localization system performs the prediction and correction steps with the EKF method that allow the mobile robot to navigate more accurately. This is crucial in real-world scenarios, where unpredictable dynamic situations are commonplace, making it impossible for the mobile robot to follow the actual path perfectly. Therefore, this system aims to minimize errors caused by factors such as noise.

Based on the data from the localization system, the control system translates this information into motor commands that enable the mobile robot to perform tasks. The control system acts as the processing unit, interpreting the information and sending appropriate signals to the actuation system to carry out the desired movements.

Lastly, the actuation system is responsible for executing the commands received from the control system, resulting in the mobile robot moving accordingly. This process loops continuously until a specific condition is met, for example, the mobile robot reaches its goal.

7.2.1 The Implementation of Localization Process

Since there are differences between simulation using ROS2, Gazebo, and RViz2 compared to MATLAB, thus, it is essential to describe the requirements that need to be met to achieve the desired goals in this study. Unlike ROS2, Gazebo, and RViz2, which often allow a plug-and-play approach using existing packages, MATLAB typically requires more customization and development from scratch.

Section 7.2 has outlined how mobile robots are generally expected to operate in terms of the overall architecture of the autonomous system. However, this study focuses more specifically on the localization system. Therefore, it is essential to provide a more detailed explanation of how the localization system is intended to function, particularly within the MATLAB environment.

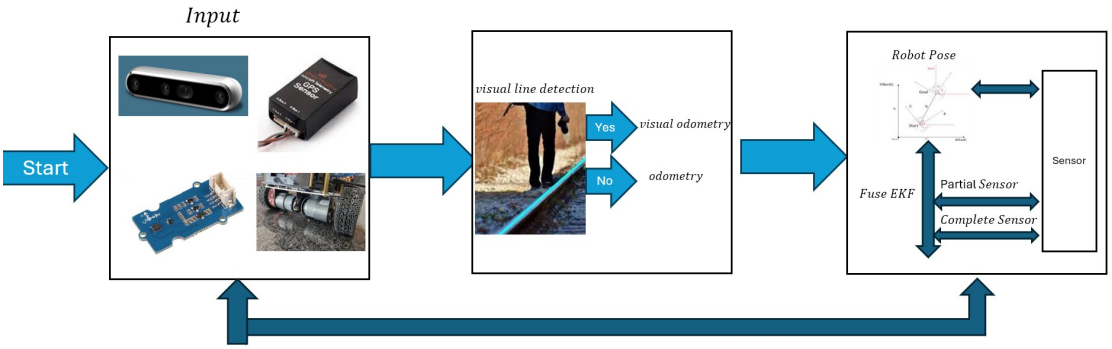


Figure 7.8: The overview of Localization System Architecture

Fig. 7.8 illustrates a basic overview of the approach adopted in this study. The localization process is divided into three main phases as follows:

- **The initial phase**, where this phase consists of various sensors, including wheel encoders, IMU, and GNSS sensors. The goal is to gather environmental data through camera sensor data and produce a preliminary estimate of the robot's pose through wheel encoders, IMU, and GNSS sensors.
- **The second phase**, where this phase focuses on refining the robot's pose when perception data is available, but non-perception system are not processed in this component, but will be handled in the final phase, along with refined data from the perception system.
- **The final phase**, where this phase performs sensor fusion using the EKF method to integrate and update the robot's pose. The process runs iteratively in a loop until the goal is reached. In accordance with the criteria defined in this study, the localization

process is classified as **complete** if it involves the perception system (e.g., successful line detection via camera sensor). In contrast, if the system operates without using camera-based perception, the process is classified as **incomplete**. However, this does not indicate a **system failure**; the robot remains functional and capable of reaching its goal, albeit through a reduced sensor configuration.

The 2D kinematic model for differential drive robot, where the robot's position is described by x and its heading direction by the unit vector h. The forward velocity v and angular velocity w define how the robot moves and turns, while R is 90 degrees rotation matrix, that produces a vector perpendicular to h. The motion is described by:

$$\dot{x} = vh \tag{7.1}$$

This eq. 7.1 means that the velocity of the robot is in the direction of the heading vector h, scaled by the forward speed v. The heading changes is described as follows:

$$\dot{h} = wRh \tag{7.2}$$

where Rh is the rotational direction vector of the heading 90 degrees, giving the perpendicular direction to h. This eq. 7.2 shows that the heading changes at a rate proportional to w. This formulation ensures that the heading remains a unit vector while describing the robot's translation and rotation in a simple, compact form. The next step is to describe the discrete-time kinematic model for the differential drive robot derived from eq. 7.1 and 7.2. In discrete form, using a timestep ΔT (written as dt), then approximate the next state by adding the derivative times the step size to the current state, which is described as follows:

$$x_{n+1} = x_n + \Delta T v_n h_n \tag{7.3}$$

and

$$h_{n+1} = h_n + \Delta T w_n R h_n \tag{7.4}$$

This formulation uses simple Euler integration, making it computationally efficient for real-time control, though very large time steps or high angular velocities may require more accurate methods to preserve the unit length of the heading vector.

The next step is to describe the discrete-time kinematic model for the differential drive robot in **state-space form**. As shown that the position update is described by eq. 7.3 and the heading update is described by eq. 7.4. Thus, these equations are described in the state-space form as follows:

$$z(n) = \begin{bmatrix} x(n) \\ h(n) \end{bmatrix} \tag{7.5}$$

which combines both the robot's position and heading into one variable. The state update equation then becomes:

$$z(n+1) = \begin{bmatrix} x(n) + \Delta T v(n) h(n) \\ h(n) + \Delta T w(n) R h(n) \end{bmatrix} \tag{7.6}$$

This eq. 7.6 can be written compactly as follows:

$$z(n+1) = f(x(n), h(n)) = f(z(n)) \tag{7.7}$$

where $z(n)$ is the robot's state at time step n, which includes position $x(n)$ and heading $h(n)$. To obtain the velocities needed for this update, the forward velocity v from wheel odometry is described as follows:

$$v = r \cdot \frac{w_R + w_L}{2} \tag{7.8}$$

where r is the radius of each wheel and w_R and W_L are angular velocities (rotational speeds) of the right and left wheels. The angular velocity w from wheel odometry is described as follows:

$$v = r \cdot \frac{w_R - w_L}{D} \tag{7.9}$$

where D is the distance between the centers of the right and left wheels. With $v(n)$ and $w(n)$ are known, the state update become as follows:

$$z(n+1) = f(x(n), h(n)) = f(z(n), v(n), w(n)) \tag{7.10}$$

This eq. 7.10 defines a model suitable for use in a Kalman Filter, where noisy odometry measurements are fused with other sensor data to improve the accuracy of state estimation.

7.2.2 The Hough Transform

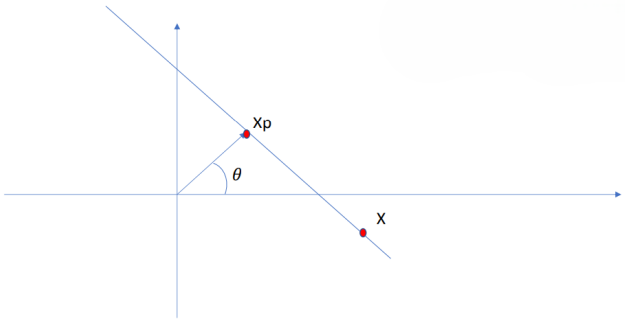


Figure 7.9: The Hough Transform

This Fig. 7.9 illustrates the concept behind the Hough Transform in this study to detect straight lines in an image. The key idea is to represent a line using the polar form:

$$r = x \cdot cos\theta + y \cdot sin\theta \tag{7.11}$$

where r represents the perpendicular distance from the origin to the line and θ represents the angle between the x-axis and the perpendicular vector.

$$n = [cos(\theta), sin(\theta)] \tag{7.12}$$

Equation 7.12 represents the unit normal vector to the line, which defines the direction perpendicular to the line. As shown in the image above, particularly, X_p is the closet point on the line to the origin. This point is described by the following equation:

$$X_p = r \cdot n \tag{7.13}$$

Consider X as an arbitrary point that lies on the line. The direction vector $X_p - X$ lies along the line, and therefore must be orthogonal to the normal vector n. This implies that the dot product between $X_p - X$ and n is zero, which can be expressed as:

$$< X_p - X, n >= 0 \tag{7.14}$$

By substituting $X_p = r \cdot n$, then the equation can be expressed as follows:

$$< r \cdot n - X, n >= 0 \tag{7.15}$$

Using linearity of the dot product, then the equation can be expressed as follows:

$$r < n, n > - < X, n >= 0 \tag{7.16}$$

Since n is a unit vector, <n, n> $= 1$, thus the equation can be simplified as follows:

$$r =< X, n > \tag{7.17}$$

The explanation above introduces the concept of the Hough Transform for this study. In the next step, this study delves into a more detailed formulation of the Hough Transform, specifically focusing on how it applies to points lying on a line. First, the equation starts with the implicit line equation as follows:

$$< x - p, n >= 0 \tag{7.18}$$

where x represents an arbitrary point on the line, p is a reference point on line, and n is the unit normal vector perpendicular to the line. As shown in this eq. 7.18 that the condition $< x - p, n >= 0$ ensures that the vector from p to x is orthogonal to n, which mathematically defines the line.

In the following step, this study introduces an alternate equation that parameterize a point x on the line, described as follows:

$$x = a \cdot R_{90} \cdot n + p = a \cdot v + p \tag{7.19}$$

where R_{90} is the matrix that rotates a vector by 90 degrees, and a isn an arbitrary scalar. Since a adjusts the length along this direction, every point on the line can be expressed

as a displacement along v (the line direction) from a reference point p. It moves toward the Hough formulation using the eq. 7.12, thus this equation can expressed as $n' = [cos(\theta), sin(\theta)]$, and the closet point to the origin on the line is $x_p = r \cdot n'$, where r is the perpendicular distance to the origin. As explained in the eq. 7.19, thus any point x on the line can be expressed as $x = a \cdot v + p$. Additionally, the condition that x lies on the line means the vector from x_p to x must be orthogonal to n', thus the equation can be written as follows:

$$< x_p - x, n' >= 0 \tag{7.20}$$

Equation 7.20 satisfies the condition outlined in Equation 7.18, and can therefore be expanded as follows:

$$< r \cdot n' - a \cdot R_{90} \cdot n + p, n' >= 0 \tag{7.21}$$

By expanding and simplifying using properties of the unit vector, we find that $n' = n$, and thus we can get $r =< p, n' >$. This confirms that for any point x on the line, the same r value is recovered. This is fundamental to the Hough Transform, where all points on the same line yield the same (r, θ) pair in the Hough space.

The next step is to introduce how the Hough Transform representation of a line behaves under 2D rotation and translation. As explained above, in the Hough transform, a line is represented by the parameters r and θ, where θ represents the orientation of the line's normal vector $n' = [cos(\theta), sin(\theta)]$, and r is the perpendicular distance from the origin to the line. The closet point on the line to the origin, x_p, which can be expressed as $x_p = r \cdot n'$. The equation of a point x on the line, after **rotation and translation**, becomes $x = R(a \cdot v + p) + t$, where $v = R_{90} \cdot n$ is the direction vector of the line, R is a 2D rotation matrix, t is a translation vector, and a is a scalar.

As explained in eq. 7.20 and 7.21, where the condition $< x_p - x, n' >= 0$, which expands with rotation and translation as follows:

$$\langle x_p - a \cdot R_{90} \cdot R \cdot n + R \cdot p + t,\ n' \rangle = \langle r \cdot n' - a \cdot R_{90} \cdot R \cdot n + R \cdot p + t,\ n' \rangle = 0 \tag{7.22}$$

Assuming $n' = R \cdot n$, the resulting expression simplifies to $r =< R \cdot p + t, n' >$. This equation describes how the new Hough distance r, after applying the transformation, depends on the rotated and translated defining point and the normal vector. Furthermore, this study also compares the parameters before and after transformation:

- **Before** the transformation, the normal vector is $n'_1 = n$ and the distance is $r_1 =< p, n'_1 >$
- **After** the transformation, the normal vector becomes $n'_2 = R \cdot n$ and the distance is $r_2 =< R \cdot p + t, n'_2 >$.

If there is no rotation ($R = I$) and the translation t is along the line (e.g., $t = c \cdot R_{90} \cdot n$), then the value of r **remains unchanged**.

As explained above, particularly how to determine the rotation matrix R and translation vector t from two Hough Transforms (before and after rotation and translation) of the same line in 2D space as follows:

- **Before Rotation and Translation**, a line is represented in Hough space by parameters θ_1 and r_1. Prior to any transformation, the line's orientation is defined by its normal vector $n' = [\cos(\theta_1), \sin(\theta_1)] = n$, and the angle θ_1 can be computed via $\theta_1 = atan2(\cos(\theta_1), \sin(\theta_1)) = atan2(n_2, n_1)$. The perpendicular distance from the origin to the line, denoted r_1, can be expressed as $r_1 =< p, n' >$, where p is any point on the line. For convenience, this point may be chosen as $r_1 \cdot n'$.

- **After Rotation and Translation**, the normal vector becomes $n'_2 = [\cos(\theta_2), \sin(\theta_2)] = R \cdot n$ and its new Hough distance is $r_2 =< R \cdot p + t, n'_2 >$. To reconstruct this transformed states, this can be rewritten as $R \cdot p + t = r_2 \cdot n'_2 + r' \cdot R_{90} \cdot n$, where $R_{90} \cdot n$ is a vector along the line direction and r' is an arbitrary scalar parameter. Solving for t, we get $t = r_2 \cdot n'_2 + r' \cdot R \cdot R_{90} \cdot n - R \cdot p$. Since r' is arbitrary, the translation vector t can not be uniquely determined from Hough parameters alone. The rotation matrix R, on the other hand, is uniquely determined and has the standard 2D form as follows:

$$R = \begin{bmatrix} \cos(d\theta) & -\sin(d\theta) \\ \sin(d\theta) & \cos(d\theta) \end{bmatrix} \tag{7.23}$$

  Applying this rotation matrix to the original vector n yields the new normal n'_2, as shown below:

$$R \cdot n = \begin{bmatrix} n_1 \cos(d\theta) & -n_2 \sin(d\theta) \\ n_1 \sin(d\theta) & n_2 \cos(d\theta) \end{bmatrix} = n'_2 \tag{7.24}$$

The line orientation and distance can be used within a Kalman filter to estimate motion involving both rotation and translation. In this study, the transformed Hough distance r_2, which is defined as the inner product as follows:

$$r_2 =< R \cdot p + t, n'_2 > \tag{7.25}$$

Since $n'_2 = R \cdot n$, we can substitute to get:

$$\begin{aligned} r_2 &= \langle R \cdot p + t,\ R \cdot n \rangle \\ &= \langle R \cdot p + t + a \cdot R \cdot R_{90} \cdot n,\ R \cdot n \rangle \\ &= \langle R \cdot p,\ R \cdot n \rangle + \langle t,\ R \cdot n \rangle \\ &= \langle p,\ n \rangle + \langle t,\ R \cdot n \rangle \\ &= r_1 + \langle t,\ R \cdot n \rangle \end{aligned} \tag{7.26}$$

For Kalman Filtering purposes, this leads to two key steps:

- **Angle update**: The change in line orientation corresponds to the negative of the turning angle (e.g., the vehicle or sensor turning direction).

- **Distance update**: Given the original distance r_1, the new distance r_2 can be described as follows:

$$r_2 = r_1 + < t, R \cdot n > \tag{7.27}$$

Plugging this into distance update: To express translation in a form for filtering, it can be parameterized as:

$$r_2 = r_1 + t_1 \tag{7.28}$$

because $< t_1 \cdot R \cdot n, R \cdot n >= t_1$ and $< t_2 \cdot R_{90} \cdot R \cdot n, R \cdot n >= 0$. This equation allows for estimation of translation and orientation from sequences of line observations.

7.2.3 Line-Following Controller

Figure 7.10: The line Following Controller

Fig. 7.10 describes a line-following controller for the robot, where the goal is to keep the robot aligned and moving along a line in a 2D plane. The line is characterized by a normal vector n, and the robot's position relative to the line is represented by a translation vector t, which is decomposed into two orthogonal components:

- $t_1 R_n$: represents the component along the normal direction to the line.
- $t_2 R_{90} R_n$: represents the component tangential to the line, e.g., perpendicular to the normal.

In this context, the robot's knowledge of its state is limited: it only know t_1, which is the perpendicular distance to the line, and θ, which is the angle between the robot's heading and the line direction, inferred from the rotation matrix, R. The controller then uses a proportional control law to correct both angular deviation and lateral displacement. Specifically, the angular velocity $d\theta = -k_1\theta - k_2 t_1$, where k_1 and k_2 are feedback gains. This control law causes the robot to reduce both its heading error θ and its distance from the line t_1, ensuring it converges to and follows the desired path.

The angular rate ω is modeled as

$$d\theta = -k_1\theta - k_2 t_1,$$

and the displacement-like variable t_1 evolves according to

$$dt_1 = v \sin(\theta) \approx v\theta,$$

where the small-angle approximation $\sin(\theta) \approx \theta$ is used. Differentiating t_1 twice gives

$$d^2 t_1 = v\, d\theta,$$

which, after substituting the expression for $d\theta$ and replacing θ with $\frac{t_1}{v}$, yields:

$$d^2 t_1 = -k_1 dt_1 - v k_2 t_1 \tag{7.29}$$

This is a second-order homogeneous linear differential whose characteristic polynomial is $s^2 + k_1 s + v k_2 = 0$. According to the Routh-Hurwitz stability criterion, the system will stable if $k_1 > 0$ and $v k_2 > 0$.

7.2.4 Kalman Filter for 3D Pose

The following step describes a Kalman Filter setup for estimating the robot's 3D pose, where the state vector is defined as follows:

$$(t_2, t_1, \theta) \tag{7.30}$$

where t_1 and t_2 represent two positional components and θ represents the orientation. The continuous-time prediction motion model is formulated as follows:

$$\dot{\mathbf{x}} = f(x, v, \omega) = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix} \tag{7.31}$$

where v is forward speed, and w is angular velocity. The first two terms update position based on heading, the third updates orientation. Using Euler integration with a timestep h, the discrete-time update becomes:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h \begin{bmatrix} v_n\cos(\theta_n) \\ v_n\sin(\theta_n) \\ \omega_n \end{bmatrix} \tag{7.32}$$

This eq. 7.32 is the basic **prediction step** in the Kalman Filter. The next step is measurements, which come from a Hough transform-based vision system, which provides the measured orientation θ and a line measurement, which is described as follows:

$$r_2 = r_1 + < t, R_\theta \cdot n > \tag{7.33}$$

where $n = [0\ 1]$ is the unit vector and R_θ is the rotation matrix that rotates n by θ:

$$R_\theta n = \begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \end{bmatrix} \tag{7.34}$$

Assuming the reference line is the horizontal x-axis, the dot product $< [t_2,\ t_1], R_\theta n >$ evaluates to:

$$-t_2 \sin(\theta) + t_1 \cos(\theta) \tag{7.35}$$

Thus, the measurement equation becomes:

$$h = r_1 + \begin{bmatrix} \theta \\ -t_2 \sin(\theta) + t_1 \cos(\theta) \end{bmatrix} \tag{7.36}$$

This eq. 7.36 express the measurement vector in terms of the robot's state. The next step is to analyze the observability of the 3D Kalman filter for the robot pose estimation by linearizing the system at the operating point where

$$\theta = 0,\ t_1 = 0 \tag{7.37}$$

At this point, the prediction model can be expressed from

$$\dot{x} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ w \end{bmatrix} \tag{7.38}$$

into the Jacobian $\frac{\partial f}{\partial x}$ as follows:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 0 & -v\sin(\theta) \\ 0 & 0 & v\cos(\theta) \\ 0 & 0 & 0 \end{bmatrix} \tag{7.39}$$

At the operating point $\theta = 0$ as described in eq. 7.54, thus, $\sin(\theta) = 0$ and $\cos(\theta) = 1$, so:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & v \\ 0 & 0 & 0 \end{bmatrix} \tag{7.40}$$

The discrete-time state transition matrix F is obtained using the Euler approximation,

$$F = I + h\frac{\partial f}{\partial x} \tag{7.41}$$

and its complete form can be expressed as follows:

$$F = I + h\frac{\partial f}{\partial x} = I + h \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & v \\ 0 & 0 & 0 \end{bmatrix} \tag{7.42}$$

where I is the identity matrix and h is the timestep. For the measurement model, the Jacobian is computed from the measurement equations derived earlier in 7.36. Its form can be expressed as follows:

$$H = \frac{\partial h}{\partial x} = \begin{bmatrix} 0 & 0 & 1 \\ -\sin(\theta) & cos(\theta) & -t_2\cos(\theta) - t_1 sin(\theta) \end{bmatrix} \tag{7.43}$$

and evaluated at the chosen operating point in eq. 7.54, where $\theta = 0$ and $t_1 = 0$. Thus, the form is as follows:

$$H|_{\theta=0,t_1=0} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & -t_2 \end{bmatrix} \tag{7.44}$$

State-transition linearization around the continuous dynamic $\dot{x} = f(x,u) = [v\cos\theta, v\sin\theta, w]^T$, thus

$$A \equiv \left.\frac{\partial f}{\partial x}\right|_{\theta=0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & v \\ 0 & 0 & 0 \end{bmatrix} \tag{7.45}$$

Discrete-time $F = I + hA$ as follows:

$$F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & hv \\ 0 & 0 & 0 \end{bmatrix} \tag{7.46}$$

From the linearized observability stack, as follows:

$$\mathcal{O} = \begin{bmatrix} H \\ HF \\ HF^2 \end{bmatrix} \tag{7.47}$$

Notice that the first column of H in eq. 7.44 is $[0;0]$ or zero. Multiplying H by F and F^2 or higher powers will never introduce non-zero elements into the first column of any block HF^k. Therefore, the entire first state direction lies in the null space of $\mathcal{O}$, implying that $\mathcal{O}$ cannot have full rank. Equivalently, the vector as follows:

$$v_{null} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{7.48}$$

Hence rank $\mathcal{O} = 2 < 3$.

Another way to check observability is to note that since the measurement equation is expressed in eq. 7.28, **note** that t_2 is shown in the second row of eq. 7.44, does not appear in eq. 7.28, thus, this means that r is independent of t_2, making t_2 unobservable because the measurements provide no direct information about t_2. Consequently, the estimation should be reduced to a 2D kalman filter.

7.2.5 Kalman Filter for 2D pose

The following step describes a Kalman filter setup for estimating a robot's 2D pose, where state vector is reduced to position along one axis and orientation. The state is defined as follows:

$$x = \begin{bmatrix} t_1 \\ \theta \end{bmatrix} \tag{7.49}$$

where t_1 is a position coordinate and θ is orientation. The continuous-time prediction motion model is formulated as follows:

$$\dot{\mathbf{x}} = \begin{bmatrix} v\sin(\theta) \\ \omega \end{bmatrix} \tag{7.50}$$

where $v\sin(\theta)$ describes the velocity component along the t_1 direction and w is the angular velocity. Using Euler integration with a timestep h, the discrete-time update becomes:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h \begin{bmatrix} v_n \sin(\theta_n) \\ \omega_n \end{bmatrix} \tag{7.51}$$

This eq. 7.32 predicts the next state from the previous one. The next step is measurements, which come from a Hough transform-based vision system, which provides the measured orientation θ and a line measurement, which is described as follows:

$$r_2 = r_1 + < t, R_\theta \cdot n > \tag{7.52}$$

where $n = [0\ 1]$ is the target line normal vector and R_θ is the rotation matrix that rotates n by θ:

$$R_\theta n = \begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \end{bmatrix} \tag{7.53}$$

In this reduced model, the dot product is mainly related to t_1, simplifying the measurement relationship. The next step is to analyze the observability of the 2D Kalman filter for the robot pose estimation by linearizing the system at the operating point where

$$\theta = 0,\ t_1 = 0 \tag{7.54}$$

yields the state transition Jacobian $F = I + hA$ as follows:

$$F = I + h\frac{\partial f}{\partial x} = I + h \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix} \tag{7.55}$$

where h is the timestep and v is the forward velocity.The measurement matrix H becomes the I_{2x2} identity. The discrete observability matrix is as follows:

$$\mathcal{O} = \begin{bmatrix} H \\ HF \end{bmatrix} = \begin{bmatrix} I \\ I + hA \end{bmatrix}. \tag{7.56}$$

With A above and $v \neq 0$, thus this stacked matrix and null space has rank 2, thus rank $\mathcal{O} = 2 = 2$, indicating that both state (t_1 and θ) variables are directly observed.

7.2.6 Determining R and t from Hough transforms ($R = I$ Case)

The following step is the process of determining the rotation matrix $R = I$ and translation vector t from Hough transform measurements in a specific case where $R = I$ (no rotation). **Before transformation**, the line's unit normal vector is described as follows:

$$n_1' = [\cos(\theta_1),\ \sin(\theta_1)] \tag{7.57}$$

and the perpendicular distance from the origin is described as follows:

$$r_1 =< p,\ n_1' > \tag{7.58}$$

where p is a point on the line, which we can choose $p = r_1 n_1'$. **After transformation**, the normal vector remains unchanged since $R = I$ (no rotation), thus the equation can be described as follows:

$$n_2' = [\cos(\theta_1),\ \sin(\theta_1)] = Rn = In = n \tag{7.59}$$

Thus, the new perpendicular distance from the origin is:

$$r_2 =< R \cdot p + t,\ n_2' > \tag{7.60}$$

where $R \cdot p + t$ is the transformed point. Additionally, we may choose $R \cdot p = p + t = r_2 \cdot n_2' + r' \cdot R_{90} \cdot n$, where R_{90} rotates n by 90 degrees. Thus, this leads to:

$$t = r_2 \cdot n + r' \cdot R_{90} \cdot n - p \tag{7.61}$$

However, when substituted back into the measurement equation, the term involving r' vanishes because n is orthogonal to $R_{90} \cdot n$. This means that r_2 is independent of r'. As a result, the translation perpendicular to the line cannot be determined from the these measurements, making part of t unobservable in this case.

7.2.7 Camera Perspectives

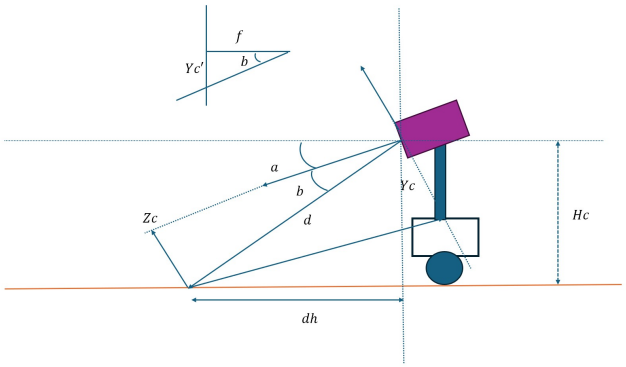


Figure 7.11: The Camera's Perspective

Fig. 7.11 illustrates the steps involved in updating the robot's pose as it detects and follows line markings for painting. In this study, a camera is mounted on the robot and oriented to face backward–opposite to the direction of the robot's forward movement. As the robot moves, the camera captures images of the environment behind it, allowing it to detect lines on the surfaces. Since the robot is moving forward while the environment remains stationary, the lines and features in the captured images appear to shift in the opposite direction. This apparent motion is a projection of the robot's actual movement through the environment.

As shown in Fig. 7.11, The geometric relationship between the camera, its viewing angle, and the observed object is obtained by combining camera projection principles with trigonometric constraints. In the top-left of Fig. 7.11, the pinhole camera model is depicted: the vertical displacement in the image plane, denoted by Y_c', is related to the actual vertical displacement Y_c in the camera frame and the depth Z_c via the projection equation as follows:

$$Yc' = f \cdot \frac{Y_c}{Z_c} \tag{7.62}$$

where f is the camera's focal length. The angle b between the camera's optical axis and the object's vertical projection as follows:

$$\tan(b) = \frac{Y_c'}{f} \tag{7.63}$$

On the right, the larger as shown in Fig. 7.11, this diagram illustrates the object located at a horizontal distance dh from the camera's base, with the camera mounted at height H_c. The slant distance d from the camera to the object, denoted by d, which is obtained using the Pythagorean theorem as follows:

$$d = \sqrt{Y_c^2 + Z_c^2} \quad \text{or} \quad d = \sqrt{d_h^2 + H_c^2} \tag{7.64}$$

In the camera frame, as follows:

$$d = \sqrt{d_h^2 + H_c^2} \tag{7.65}$$

Angle a and b represent the tilt of the camera and the vertical deviation in the image (viewing offset), respectively. Accordingly, the the equation can be described as follows:

$$d \cdot sin(a + b) = H_c \tag{7.66}$$

while in the camera frame, the vertical component is described as follows:

$$d \cdot sin(b) = Y_c \tag{7.67}$$

From the eq. 7.63 and 7.66, these equations can be described as follows:

$$b = \arctan(\frac{Y_c'}{f}), \quad d = \frac{H_c}{\sin(a + b)}, \tag{7.68}$$

and by substituting these equations with eq. 7.62, we obtain as follows:

$$Z_c^2 = \frac{d^2}{1 + (\frac{Y_c'}{f})^2} \tag{7.69}$$

This eq. 7.69 enables to compute camera coordinates from the image coordinates using:

$$X_c' = \frac{X_c}{Z_c} \quad , \quad Y_c' = \frac{Y_c}{Z_c} \tag{7.70}$$

The transformation from world coordinates (X_g, Y_g, Z_g) to camera coordinates (X_c, Y_c, Z_c) is a **linear** mapping, generally achieved through a rotation and translation matrix. However, mapping from the world coordinates to normalized image-plane normalized coordinates (X_c', Y_c') involves the **nonlinear** transformation because this process involves division by Z_c, which is shown in the eq. 7.70. The next step involves addressing the linearization and observability issues to reduce to reduce the system to the 2D Kalman Filter.

The projection mapping from 3D camera coordinates to normalized image coordinates, as described in 7.70, is based on perspective projection. This projection introduces geometric distortion, meaning that straight lines in the 3D world may not necessarily appear as straight lines in the image. Consequently, when the robot detects features, particularly lines, in the image, we need to **invert** this mapping to recover (X_c, Y_c, Z_c) from (X_c', Y_c'). This process requires solving for depth Z_c using the nonlinear relation derived as follows:

$$Z_c = \frac{d}{\sqrt{1 + (\frac{Y_c'}{f})^2}}, \quad d = \frac{Hc}{sin(a + \arctan{(\frac{Y_c'}{f})})} \tag{7.71}$$

because of the $\arctan(\cdot), \sin(\cdot)$, and the $\sqrt{\frac{1}{1+(\cdot)^2}}$ term, the measurement function $h(x)$, which maps state x to measurements, is inherently nonlinear. Therefore, since the Kalman Filter assumes linear measurements, the function must be linearized using the EKF Kalman filter in this study. As explained in Subsection 7.2.4, the 3D Kalman filter exhibits unobservable state directions under the given linearization and measurement model. A practical solution is to reduce the state to the observable subspace using a 2D Kalman filter, as demonstrated in Section 7.2.5, where the states are proven to be observable.

Since t_2 is unobservable, the t_2 component is removed, and the reduced state becomes as follows:

$$x_r = \begin{bmatrix} t_1 \\ \theta \end{bmatrix} \tag{7.72}$$

In this context, the subscript r denotes quantities of the **reduced-order system** obtained after removing the unobservable state. The continuous-time prediction motion model is formulated as follows:

$$\dot{t_1} = v \sin(\theta), \quad \dot{\theta} = \omega \tag{7.73}$$

Using Euler integration with a timestep h, the discrete-time update becomes:

$$x_{r,k+1} = f_r(x_{r,k}, u_k) = \begin{bmatrix} t_{1,k} + hv\sin(\theta_k) \\ \theta_k + h\omega_k \end{bmatrix} \tag{7.74}$$

and linearize to form the Jacobian $Fr = \frac{\partial f_r}{\partial x_r}$ as follows:

$$F_r = \begin{bmatrix} 1 & hv\cos(\theta) \\ 0 & 1 \end{bmatrix} \tag{7.75}$$

Process noise model: select $Q_r(2X2)$ according to assumed noise in v, w, or model uncertainty. The next step is the measurement model where, we retain the same measurement set but omit t_2; thus the linearized measurement becomes as follows:

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \theta \\ -t_2\sin(\theta) + t_1\cos(\theta) \end{bmatrix} \tag{7.76}$$

Dropping t_2 (treat it as unknown/unestimated), thus the form becomes as follows:

$$h_r(x_r) = r_1 + \begin{bmatrix} \theta \\ t_1\cos(\theta) + c(\hat{t}_2, \theta) \end{bmatrix} \tag{7.77}$$

if measurement is just nominally t_1, then we may linearize so that as follows:

$$H_r \approx I_{2X2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7.78}$$

for the case t_1 and θ are directly observed after omitting t_2. The complete EKF algorithm is outlined as follows:

Prediction:

$$\begin{aligned} \hat{x}_{r,k+1|k} &= f_r\left(\hat{x}_{r,k|k}, u_k\right) \\ P_{k+1|k} &= F_r P_{k|k} F_r^\top + Q_r \end{aligned} \tag{7.79}$$

Linearize measurement (evaluate Jacobian H_r at $\hat{x}_{r,k+1|k}$). For the identity H case, $H_r = I$.

Update Measurement:

$$\begin{aligned} K_{k+1} &= P_{k+1|k} H_r^\top \left(H_r P_{k+1|k} H_r^\top + R\right)^{-1} \\ \hat{x}_{r,k+1|k+1} &= \hat{x}_{r,k+1|k} + K_{k+1}\left(z_{k+1} - h_r\left(\hat{x}_{r,k+1|k}\right)\right) \\ P_{k+1|k+1} &= \left(I - K_{k+1} H_r\right) P_{k+1|k} \end{aligned} \tag{7.80}$$

Chapter 8

Results

This section presents an overview of the simulation process and the results generated from its simulation. The simulation was primarily carried out in MATLAB, where custom-defined lines were used to represent what the camera would detect in a real or simulation scenario. Nonetheless, in this scenario, no actual camera or simulated camera was used; rather, the predefined lines were designed to closely mimic expected camera outputs using the Hough transform.

8.1 The simulation without source errors

This simulation was carried out with the assumption that the source errors would not affect the system using the EKF method, and it was ensured that everything worked perfectly under ideal conditions. When everything had worked as expected, then this simulation became the baseline for further evaluating the accuracy of the localization system under conditions where source errors would be present.

The setup involved the mobile robot starting at the point and heading along with x-axis toward the target point. This simulation steers the mobile robot to follow the straight path from its start point $(0, 0)$ to the target point $(10, 0)$ by continuously correcting its heading and lateral position errors. The robot's state in this simulation can be described as follows:

$$x_k = \begin{bmatrix} X_k \\ Y_k \\ \theta_k \end{bmatrix} \tag{8.1}$$

where X_k and Y_k were positions in world frame and θ was the heading angle (yaw). This simulation assumed that the control inputs at step k were linear speed, denoted by v_k, and angular speed, denoted by w_k.

Nonlinear motion model as follows:

$$\begin{aligned} X_{k+1} &= X_k + v_k \Delta t \cos\theta_k \\ Y_{k+1} &= Y_k + v_k \Delta t \sin\theta_k \\ \theta_{k+1} &= \theta_k + w_k \Delta t \end{aligned} \tag{8.2}$$

The next step was to make the robot follow the straight line from the start point $\mathbf{x_s}$ at $(0, 0)$ to the target point $\mathbf{x_e}$ at $(10, 0)$. The unit vector for the path direction was described as follows:

$$\mathbf{u} = \frac{\mathbf{x_e} - \mathbf{x}_s}{\|\mathbf{x_e} - \mathbf{x}_s\|} \tag{8.3}$$

where the unit vector along the the path $\mathbf{u}$ was obtained by normalizing the vector from $\mathbf{x_e}$ and $\mathbf{x_s}$. The normal vector to the path was described as follows:

$$\mathbf{n} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{u} \tag{8.4}$$

where the perpendicular unit vector $\mathbf{n}$ was obtained rotated $\mathbf{u}$ by 90 degrees. The projection of the current point $\mathbf{x}$ onto the desired path, denoted by $\mathbf{x_p}$ was computed by as follows:

$$\mathbf{x_p} = \mathbf{x} - (\mathbf{n^T x})\mathbf{n} \tag{8.5}$$

The next step was to ensure the projection remains within the path segment; thus it was clamped to $\mathbf{x_s}$ when it fell before the start $\mathbf{x_s}$, or to $\mathbf{x_e}$ when it extended past the target. Finally, the look-ahead point was obtained by shifting projection forward along the path by the distance γ in the direction of $\mathbf{u}$, using the following:

$$\mathbf{x_T} = \mathbf{x_p} - \gamma \mathbf{u} \tag{8.6}$$

This 8.6 served as the reference for the guidance controller to steer the mobile robot smoothly toward the path. However, in the guidance control, error vector to look-ahead point was described as follows:

$$\mathbf{e} = \mathbf{x_T} - \mathbf{x} \tag{8.7}$$

represented the displacement from the robot's current position to the look-ahead point. The desired linear speed v was set to be smaller than the magnitude of the error vector $||\mathbf{e}||$. In this simulation, v was capped to 2 m/s. The heading error θ_{err} was obtained from the following equation:

$$\theta_{err} = \arcsin((\mathbf{u} \times \mathbf{h})_z) \tag{8.8}$$

The next step was to compute the lateral position error, which was described as follows:

$$t1_{err} = \mathbf{n}^T(\mathbf{x} - \mathbf{x_p}) \tag{8.9}$$

The angular velocity command w was then computed using the control law described as follows:

$$w = -k_\theta \theta_{err} - k_t t1_{err} \tag{8.10}$$

In this simulation, both gains k_θ and k_t were set to 2. Finally, w was saturated as described as follows:

$$|w| \leq \frac{v}{L} \tag{8.11}$$

ensured that the steering rate remained within the physical limits given the current speed v and the mobile robot length L.

In this simulation, the EKF estimated the robot state as described in eq. 8.1, where

- **Prediction** based on odometry inputs v_k and w_k
- **Update** from Hough Transform measurements of the line the robot follows.

1. EKF Prediction Step (Odometry):

Given current state $x_k = [X_k, Y_k, \theta_k]^T$ described in 8.1, the predicted state after time Δt, as follows:

The **discrete motion model**:

$$x_{k+1} = f(x_k, u_k) + w_k \tag{8.12}$$

with

$$f(x_k, u_k) = \begin{bmatrix} X_k + v_k \Delta t \cos\theta_k \\ Y_k + v_k \Delta t \sin(\theta) \\ \theta_k + w_k \Delta t \end{bmatrix} \tag{8.13}$$

Linearized motion around the predicted state to get F_k as follows:

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{x_k, u_k} = \begin{bmatrix} 1 & 0 & -v_k \Delta t \sin(\theta_k) \\ 0 & 1 & v_k \Delta t \cos(\theta_k) \\ 0 & 0 & 1 \end{bmatrix} \tag{8.14}$$

The covariance prediction P_{k+1} as follows:

$$P_{k+1} = F_k P_k F_k^T + Q_K \tag{8.15}$$

where the process noise covariance matrix, denoted by Q_k, was described as follows:

$$Q_k = G_k Q_c G_k^T \tag{8.16}$$

where

$$G_k = \left.\frac{\partial f}{\partial u}\right|_{x_k, u_k} = \begin{bmatrix} \Delta t \cos(\theta_k) & 0 \\ \Delta t \sin(\theta_k) & 0 \\ 0 & \Delta t \end{bmatrix} \tag{8.17}$$

and

$$G_c = \begin{bmatrix} \sigma_X^2 & 0 & 0 \\ 0 & \sigma_Y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{8.18}$$

In this simulation, Q_k was used as tuned constant, thus $w_k \sim \mathcal{N}(0, Q)$ where $Q = G_c$. This parameter could be tuned up according to your choice. However, since this section discussed the simulation without noise, therefore Q was **set at very small but nonzero** (e.g., 10^{-6} to 10^{-8}).

2. EKF Update Step (Hough Transform):

The measurement in this simulation came from detecting the line in camera space via Hough Transform $\rho_k = X_k \cos(\theta_k) + Y_k \sin(\theta_k)$. The measurement equation as follows:

$$z_k = \begin{bmatrix} \rho_k \\ \psi_k \end{bmatrix} = h(x_k) + v_k \tag{8.19}$$

where

$$h(x_k) = \begin{bmatrix} X_k \cos(\theta_k) + Y_k \sin(\theta_k) \\ \psi_k \end{bmatrix} \tag{8.20}$$

Linearized measurement to get H_k as follows:

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{x=x_k} = \begin{bmatrix} \cos(\theta_k) & \sin(\theta_k) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8.21}$$

ψ_k represented the measured orientation of a detected line, and $v_k \sim \mathcal{N}(0, R)$ represented the measurement noise.

$$R = \begin{bmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\psi^2 \end{bmatrix} \tag{8.22}$$

This parameter R could be tuned up according to your choice. However, since this section discussed the simulation without noise, therefore R was **set at very small but nonzero** (e.g., 10^{-6} to 10^{-8}). The next step was to use the measurement to update the system value, described as follows:

$$\begin{aligned} \mathbf{y}_{k+1} &= \mathbf{z}_k - h(\mathbf{x}_k) \\ S_{k+1} &= H_k P_k H_k^T + R \\ K_{k+1} &= P_k H_k^T S_k^{-1} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + K_k \mathbf{y}_k \\ P_k &= (I - K_k H_k) P_k \end{aligned} \tag{8.23}$$

The results of the simulation without source errors were shown in the images below.

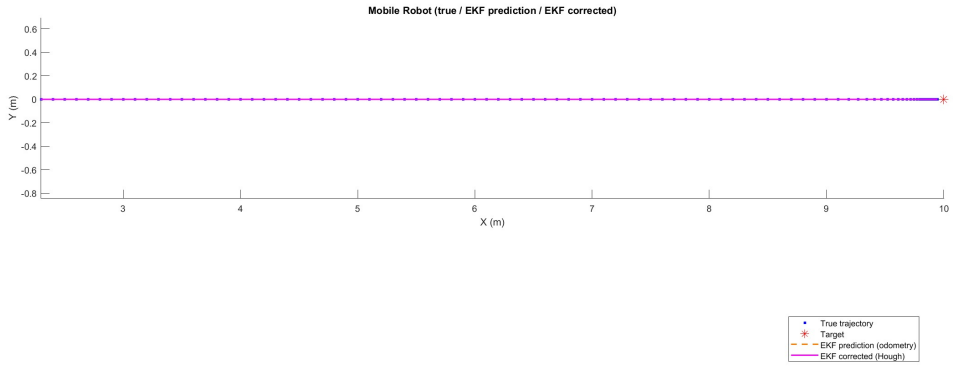


Figure 8.1: The Mobile Robot (True Trajectory, EKF prediction, EKF corrected) under perfect conditions

Fig. 8.1 described the robot's navigation toward the target point (red star) using the steering controller and EKF (prediction and correction steps) for localization.

- The **blue points** represented the *true trajectory* of the robot, generated by the simulated ground-truth motion.
- The **orange dashed lines** represented EKF's **predicted path** based solely on odometry readings.
- The **magenta solid line** represented the EKF-corrected path based on fused odometry from the prediction step and Hough transform from the measurement step. This reflected actual path the robot navigated.

The results in Fig. 8.1 confirmed satisfactory performance, as the simulation was intended to produce a straight-line trajectory, and the outcome aligned precisely with that intention.

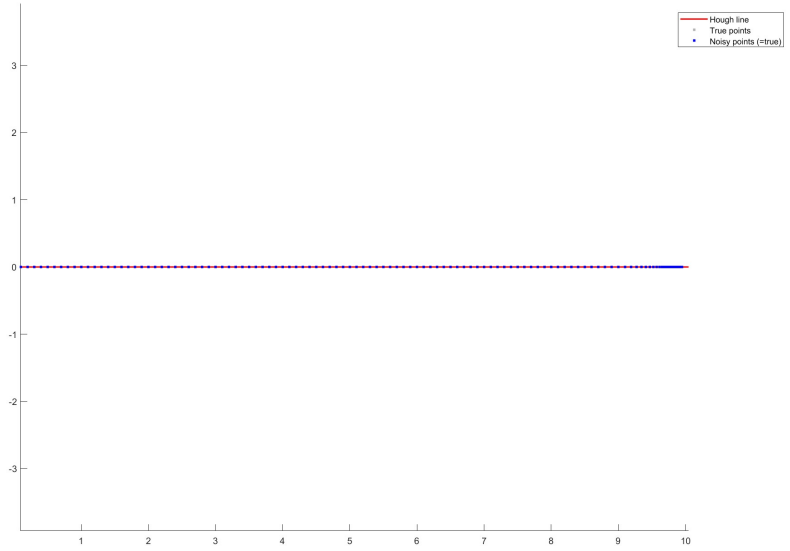


Figure 8.2: Detected line in camera coordinates using Hough transform under perfect conditions

Fig. 8.2 described the points projected onto the camera's image plane over time during the simulation.

- The **red line** represented the best-fit line detected using the Hough transform.
- The **light gray dots** represented the true feature points to the trajectory points perceived by the camera under free-noise conditions.
- The **blue dots** represented the noisy feature points (e.g., lightning condition) that camera actually measured. In this case, the simulation was performed without noise perturbation.

Since the simulation was conducted under ideal conditions without noise, the results shown in Fig. 8.2 were perfectly satisfactory.

8.2 The simulation with source errors

Source errors, such as noise, were added to this simulation to more accurately reflect the conditions commonly encountered in the real world, thus making it more realistic. This scenario was assumed to be more representative of actual deployment conditions, despite being conducted in a simulated environment. In this case, the implementation followed the procedure described in section 8.1, with the difference that noise was added to this

simulation. The results of the simulation with source errors were shown in the images below.

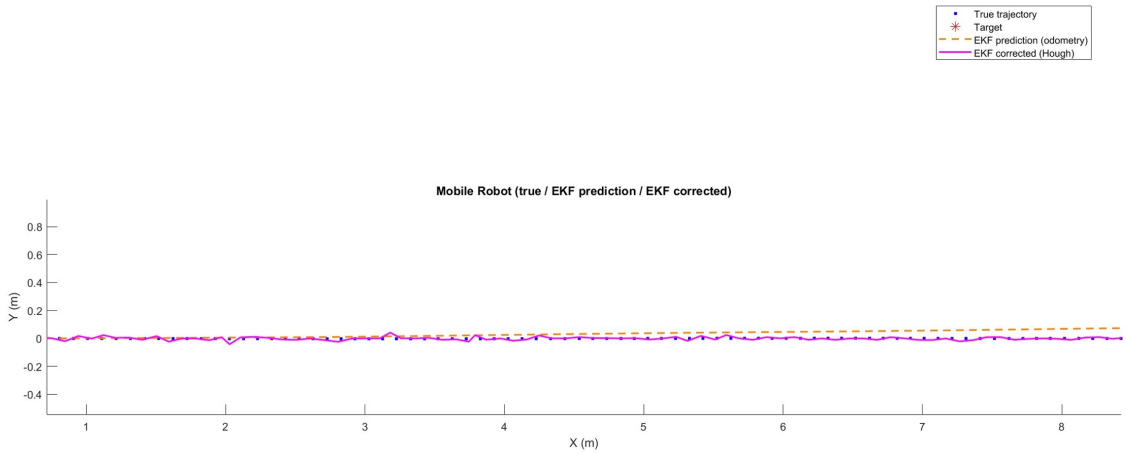


Figure 8.3: The Mobile Robot (True Trajectory, EKF prediction, EKF corrected) under noisy conditions

Fig. 8.3 illustrated the simulated mobile robot navigating toward a fixed target (red star) under uncertain conditions. The blue trajectory represented the robot's true path, generated from the ground-truth model. The orange dashed lines showed the EKF's prediction in this setup, based solely on noisy odometry, which gradually drifted from the true path due to cumulative errors. The magenta line depicted the EKF's corrected trajectory, based on fused odometry from the prediction step and the Hough transform from the measurement step. Under noisy conditions, this process updated and realigned the robot's movement toward the ground truth. This reflected actual path the robot navigated.

As described in section 8.1, the process noise covariance matrix Q and measurement noise covariance R could be tuned as required. In this simulation, Q and R were set to 0.1 under noisy conditions, whereas under noise-free conditions (section 8.1), they were set to 1×10^{-6}. The results showed that the trajectory of the robot model (orange dashed line) exhibited a slight deviation from the path in gradual step. The trajectory (magenta line) closely followed the blue true trajectory, indicating that the correction process significantly reduced the positional drift present in the orange dashed line. The result was considered satisfactory, as it demonstrated that a trajectory under noise becomes curved, irregular, and significantly deviates from a straight path.

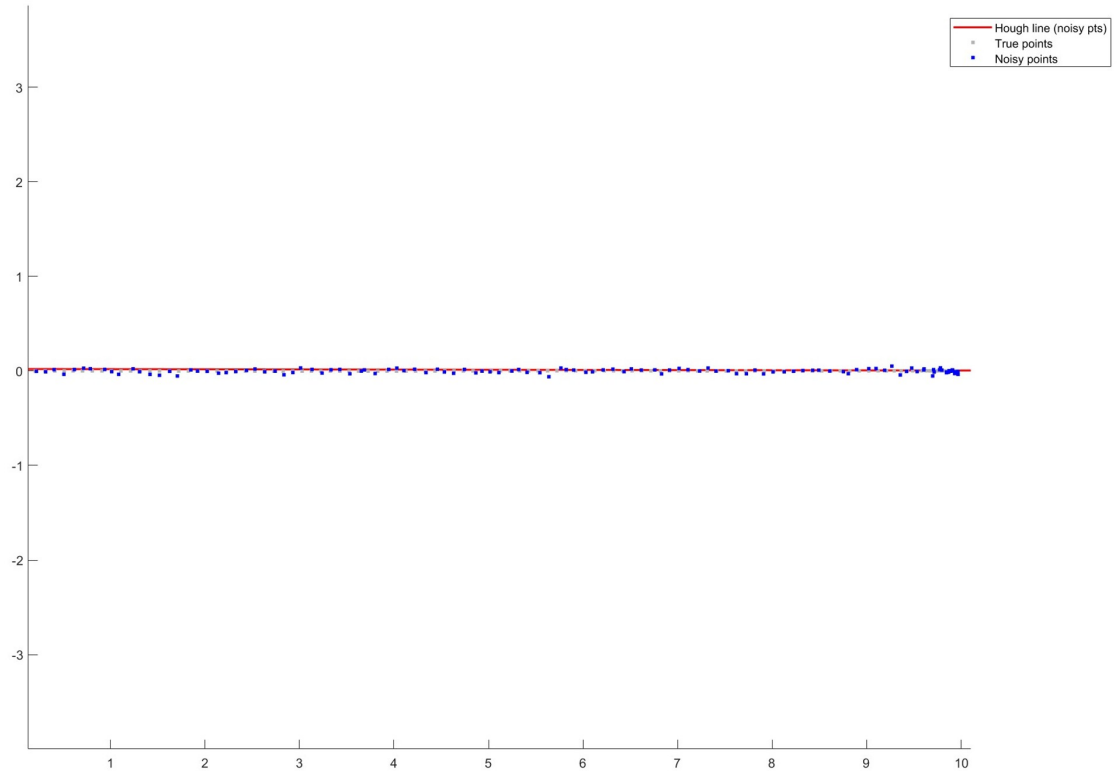


Figure 8.4: Detected line in camera coordinates using Hough transform under noisy conditions

Fig. 8.4 described the robot model's path under noise within the camera's view. The results showed that the red lines represented the best-fit line using the Hough Transform. The light gray dots represented represented the true feature points to the trajectory points perceived by the camera under free-noise conditions. The blue dots represented the noisy measurements under external environments such as lightning conditions.

8.3 The simulation with source errors with added GNSS sensors

In this case, the setup involved GNSS sensors, odometry, and camera sensors, where the EKF prediction used odometry data, and the EKF measurement update incorporated **GNSS and camera** sensor data.

The general EKF measurement update as follows:

$$\begin{aligned} \mathbf{y}_{k+1} &= \mathbf{z}_k - h(\mathbf{x}_k) \\ S_{k+1} &= H_k P_k H_k^T + R \\ K_{k+1} &= P_k H_k^T S_k^{-1} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + K_k \mathbf{y}_k \\ P_k &= (I - K_k H_k) P_k \end{aligned} \tag{8.24}$$

Since GNSS directly measured the the absolute position, thus, the measurement model under **GNSS**:

$$z_{k,gnss} = \begin{bmatrix} x \\ y \end{bmatrix} + w_{k,gnss} \tag{8.25}$$

Measurement Jacobian:

$$H_{k,gnss} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{8.26}$$

$w_{k,gnss} \sim \mathcal{N}(0, R)$ represented the measurement noise.

$$R_{gnss} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{8.27}$$

For the measurement under the Hough transform, as follows:

$$z_{k,hough} = \begin{bmatrix} \rho_k \\ \psi_k \end{bmatrix} = h(x_k) + v_k \tag{8.28}$$

Nonlinear measurement function:

$$h(x_k) = \begin{bmatrix} X_k \cos(\theta_k) + Y_k \sin(\theta_k) \\ \psi_k \end{bmatrix} \tag{8.29}$$

Measurement Jacobian:

$$H_{k,hough} = \begin{bmatrix} \cos(\theta_k) & \sin(\theta_k) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8.30}$$

$w_{k,hough} \sim \mathcal{N}(0, R)$ represented the measurement noise.

$$R_{hough} = \begin{bmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\psi^2 \end{bmatrix} \tag{8.31}$$

In this simulation, GNSS was used to correct position drift, and the Hough transform was used to correct line distance and heading in the measurement system, while the prediction system relied on odometry.

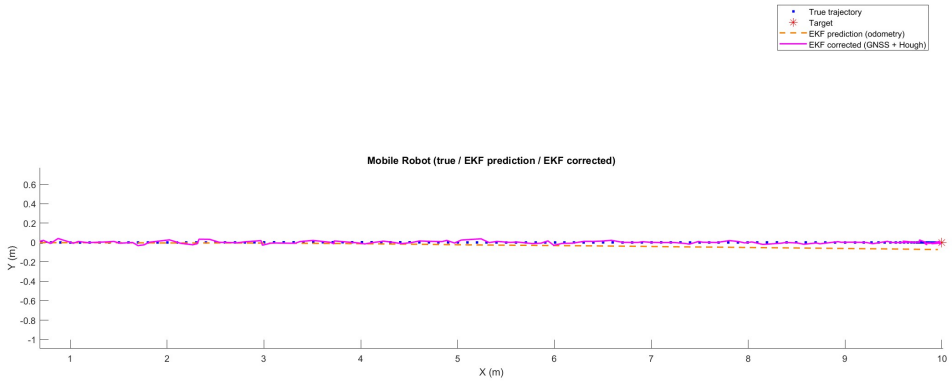
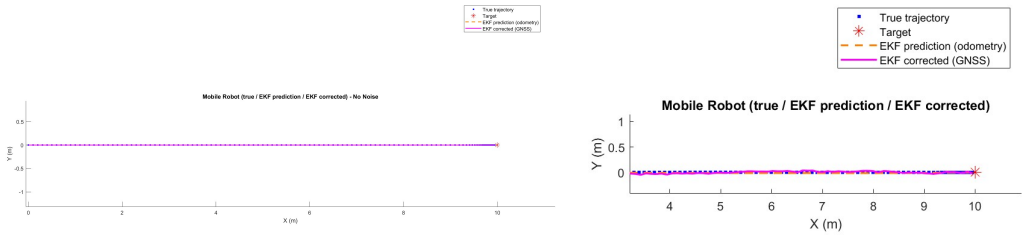


Figure 8.5: The Mobile Robot (True Trajectory, EKF prediction, EKF corrected) under noisy conditions

Fig. 8.5 illustrated the trajectory tracking performance of the mobile robot. The blue dotted line represented the true trajectory of the robot, while the red asterisk marked the target destination. The orange dashed line indicated the EKF prediction based solely on odometry data, which exhibited a small but noticeable drift below the true path due to accumulated odometry errors. In this simulation, Q and R were set to 0.04 under noisy conditions, whereas under noise-free conditions, they were set to 1×10^{-6}. The magenta solid line showed the EKF corrected trajectory using GNSS and Hough transform data as measurement step fused with odometry as prediction step, which closely aligned with the true trajectory, demonstrating the effectiveness of sensor fusion in mitigating drift and improving positional accuracy. Overall, the results confirmed that incorporating external measurements into the EKF improved localization performance compared to the results from section. 8.1 and 8.2.

8.3.1 The simulation with added GNSS sensors (without camera sensors) for measurements

In this simulation, this study first investigated the prediction step using odometry sensors and the measurement step using **GNSS sensors** under both noise-free and noisy conditions.



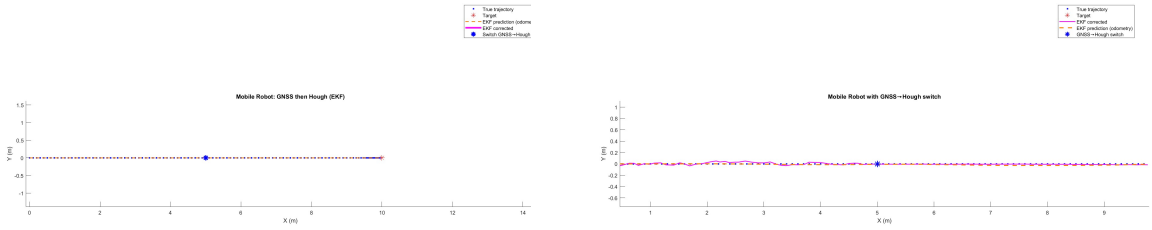
(a) Free-noise conditions **(b)** Noisy conditions

Figure 8.6: The Mobile Robot (True Trajectory, EKF prediction, EKF corrected) with GNSS measurement under different conditions

Fig. 8.6 illustrated the comparison of the mobile robot's trajectory under different noise conditions. In this simulation, Q and R were set to 0.04 under noisy conditions, whereas under noise-free conditions, they were set to 1×10^{-6}. The left image indicated ideal conditions with no noise affecting the measurement and prediction steps, showing that the mobile robot moved in the straight line along the X-axis with no deviation along the Y-axis. On the other hand, the right image showed that the robot's trajectory was affected by GNSS measurement noise and odometry noise in the prediction step, resulting in deviations. Nonetheless, the mobile robot still moved close to the true trajectory (blue dots), as illustrated by the magenta solid line. This demonstrated that GNSS could improve localization accuracy compared to camera sensors in the measurement step.

8.3.2 The simulation with added GNSS sensors and camera sensors for measurements

The next step was to implement a scenario in which the GNSS worked intermittently. In this case, the measurements used GNSS whenever possible; however, when GNSS signals were lost, the mobile robot switched to using the Hough transform for the measurement step. The implementation was carried out under both noise-free and noisy conditions. In this simulation, the GNSS worked until at the point $(5, 0)$, afterward, the GNSS no longer functioned, so the system in measurements switched to the Hough transform. Additionally, in this simulation, Q was set to 0.04, R for GNSS was set to 0.04 and R for the Hough Transform was set to 0.001 under noisy conditions, whereas under noise-free conditions, they were set to 1×10^{-6}.



(a) Free-noise conditions

(b) Noisy conditions

Figure 8.7: The Mobile Robot (True Trajectory, EKF prediction, EKF corrected) under different conditions and intermittent pattern

Fig. 8.7 described the comparison of the mobile robot's trajectory under different noise conditions with an intermittent pattern. In both cases, the robot's actual trajectory was represented by the magenta solid line. This magenta solid line showed the EKF-corrected trajectory obtained through an adaptive measurement strategy for the intermittent pattern.

- When GNSS signals were **available**, they were used to update the measurement model.
- when GNSS became **unavailable**, the mobile robot switched to the camera-based measurement model.

As shown in Fig. 8.7, the blue star indicated the point where the mobile robot switched to the Hough transform due to signal loss. The left image showed that the estimated trajectory and the corrected trajectory almost perfectly overlapped with the true path under noise-free conditions, indicating that the EKF was accurately implemented with prediction using odometry and measurements following the adaptive strategy: using GNSS when it functioned and the Hough transform when GNSS was unavailable. On the other hand, the right image showed slight deviations from the true path, but the EKF still maintained good tracking performance with only minor discrepancies, demonstrating its robustness in handling sensor noise.

8.4 The comparison of results

This section discussed the comparison of results as described in the numerical values using the Root Mean Square (RMS) to validate the visualization results.

```
num_trials = 20; % number of trials
rms_errors = zeros(num_trials, 1); % store RMS from each trial

for trial = 1:num_trials
    % Compute RMS for this trial
    rms_errors(trial) = sqrt(mean((y_vals_real - y_vals_ekf).^2));
end

% Compute combined RMS across all trials
overall_rms_error = sqrt(mean(rms_errors.^2));

% Display results
disp(['RMS Error for each trial: ', num2str(rms_errors')]);
disp(['Overall RMS Error across ', num2str(num_trials), ' trials: ', num2str(overall_rms_error)]);
```

Figure 32: The RMS functions

Fig. 32 outlined the implementation of the RMS functions, where `y_vals_real` parameter described the actual line values (ground truth) over time and `y_vals_ekf` parameter represented the EKF-estimated line values over time. Then, the difference from `y_vals_real - y_vals_ekf` represented the error at each points, which is computed by the `rms_error` parameter as the square root of the average of the squared errors. Additionally, in this simulation, the variable `num_trials` specified the number of trials (20 in this case).

Method	**Without Noisy Conditions (20 trials)**	**Under Noisy Conditions (20 trials)**
Pure Odometry	11.9827	14.1542
Pure GNSS	7.3084	9.8025
Odometry + Camera	8.8924	11.3175
Odometry + Camera + GNSS	5.1051	6.6124

Table 1: Comparison of different localization methods under varying noise conditions (RMS values).

The table 1 presented the performance of localization systems using different sensors under both ideal and noisy conditions, where lower RMS values indicated better accuracy and higher values indicated worse accuracy. Under both ideal and noisy conditions, pure odometry achieved errors of 11.9827 under ideal conditions and 14.1542 under noisy conditions, while pure GNSS recorded errors of 7.3084 and 9.8025, respectively. The combination of odometry and camera sensors resulted in errors of 8.8924 without noise, 11.3175 with noise. However, by adding GPS to this combination, the error was significantly reduced at 5.1051 without noise, 6.6124 with noise.

The results showed that pure odometry exhibited the highest error, mainly due to cumulative drift over time, while pure GNSS significantly reduced errors compared to pure odometry. However, although odometry had the highest error, when combined with camera sensors it reduced the errors, but this combination was still not better than GNSS alone, as visual odometry was sensitive to environmental factors. The best performance

was achieved by combining odometry, camera, and GNSS, which provided the most robust and accurate localization, even under noisy conditions, compared to all other configurations listed in the table.

Chapter 9

Conclusion

In this study, the perception system was simulated to approximate what a backward-facing camera would detect when identifying lines using the Hough Transform. This approach closely mirrored the behavior of an actual or simulated camera in real-world scenarios, accurately detecting lines under both ideal conditions and in the presence of source errors, without relying on a physical camera. Additionally, the perception system contributed to the localization system, which in this study relied on odometry, GNSS, and camera sensors. These inputs were integrated through sensor fusion using the EKF method.

In this simulation, different scenarios were designed to investigate whether the use of sensor fusion could improve the robustness and accuracy of the mobile robot's localization system. The results showed that sensor fusion combining odometry, camera, and GNSS data achieved the best performance overall. However, the pure GNSS showed that it had better performance compared to sensor fusion that used odometry and camera sensors. This result was expected because GNSS, such as the RTK-GPS used by Turf Tank in their products, measures absolute position with centimeter-level accuracy. In contrast, odometry suffers from cumulative errors due to drift, and the camera is sensitive to environmental factors, which can negatively impact localization performance.

The results, particularly the visualizations, showed that under ideal conditions, the robot model was able to follow the straight trajectory closely aligned with the true path. However, when noise was introduced, the robot's trajectory exhibited irregularities and deviations, highlighting the impact of noise on the mobile robot's performance. Overall, the results were considered satisfactory, as they demonstrated the effectiveness of sensor fusion using odometry, camera, and GNSS sensors, reflecting real-world performance under both noise-free and noisy conditions. These findings indicate the potential applicability of the proposed approach to real-world robotic systems operating under uncertainty.

9.1 Future Works

The potential to apply the results from simulation to real-world applications is still subject to unpredictable variables. Therefore, what works perfectly in the simulation setup cannot be conclusively generalized to real-world scenarios. It is thus advisable to conduct testing using actual hardware as part of future work. This would not only corroborate the simulation outcomes but also assist in recognizing actual constraints and unexpected obstacles in real-world contexts. In addition to applying the simulation results to practical scenarios, alternative methods such as machine learning approaches or others could also be explored.

In addition to test on actual hardware, simulation can be conducted using photorealistic simulators, particularly the perception systems that depend on realistic environments. These simulations require computers equipped with latest NVIDIA GPUs, which are especially important in robotics applications that rely on visual input.

Bibliography

[1] Tian Youjin et al. "A Robust Lane Detection Method Based on Vanishing Point Estimation". In: *Procedia Computer Science* 131 (2018). Recent Advancement in Information and Communication Technology: pp. 354–360. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.04.174. URL: https://www.sciencedirect.com/science/article/pii/S1877050918305489.

[2] Weiyu Hao. "Review on lane detection and related methods". In: *Cognitive Robotics* 3 (2023), pp. 135–141. ISSN: 2667-2413. DOI: https://doi.org/10.1016/j.cogr.2023.05.004. URL: https://www.sciencedirect.com/science/article/pii/S2667241323000186.

[3] Jungang Guan et al. "Real-Time Straight-Line Detection for XGA-Size Videos by Hough Transform with Parallelized Voting Procedures". In: *Sensors* 17.2 (2017), p. 270. ISSN: 1424-8220. DOI: 10.3390/s17020270. URL: https://www.mdpi.com/1424-8220/17/2/270.

[4] Leandro Fernandes and Manuel Oliveira. "Real-time line detection through an improved Hough transform voting scheme". In: *Pattern Recognition* 41 (Sept. 2008), pp. 299–314. DOI: 10.1016/j.patcog.2007.04.003.

[5] Wei Wang, Hui Lin, and Junshu Wang. "CNN based lane detection with instance segmentation in edge-cloud computing". In: *J. Cloud Comput.* 9.1 (May 2020). ISSN: 2192-113X. DOI: 10.1186/s13677-020-00172-z. URL: https://doi.org/10.1186/s13677-020-00172-z.

[6] Shoudong Huang and Gamini Dissanayake. "Robot Localization: An Introduction". In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. John Wiley & Sons, Ltd, 2016, pp. 1–10. ISBN: 9780471346081. DOI: https://doi.org/10.1002/047134608X.W8318.

[7] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. 2nd. Cambridge, MA: The MIT Press, 2011, pp. 101–103. ISBN: 9780262015356.

[8] M.V.V. RadhaKrishna, M. Venkata Govindh, and P. Krishna Veni. "A Review on Image Processing Sensor". In: *Journal of Physics: Conference Series* 1714.1 (Jan. 2021), p. 012055. DOI: 10.1088/1742-6596/1714/1/012055. URL: https://dx.doi.org/10.1088/1742-6596/1714/1/012055.

[9] Stefan May, Kai Pervoelz, and Hartmut Surmann. "3D Cameras: 3D Computer Vision of Wide Scope". In: *Vision Systems*. Ed. by Goro Obinata and Ashish Dutta. Rijeka: IntechOpen, 2007. Chap. 11. DOI: 10.5772/4988. URL: https://doi.org/10.5772/4988.

[10] M. Bertozzi and A. Broggi. "GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection". In: *Trans. Img. Proc.* 7.1 (Jan. 1998), pp. 62–81. ISSN: 1057-7149. DOI: 10.1109/83.650851. URL: https://doi.org/10.1109/83.650851.

[11] Jens Christian Andersen, Nils A. Andersen, and Ole Ravn. "Vision Assisted Laser Scanner Navigation for Autonomous Robots". In: *Experimental Robotics: The 10th International Symposium on Experimental Robotics*. Ed. by Oussama Khatib, Vijay Kumar, and Daniela Rus. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 111–120. ISBN: 978-3-540-77457-0. DOI: 10.1007/978-3-540-77457-0_11. URL: https://doi.org/10.1007/978-3-540-77457-0_11.

[12] Luca Iocchi and Daniele Nardi. "Hough Localization for mobile robots in polygonal environments". In: *Robotics and Autonomous Systems* 40 (July 2002), pp. 43–58. DOI: 10.1016/S0921-8890(02)00207-5.

[13] N. Sukumar and P. Sumathi. "A Robust Vision-based Lane Detection using RANSAC Algorithm". In: *2022 IEEE Global Conference on Computing, Power and Communication Technologies (GlobConPT)*. 2022, pp. 1–5. DOI: 10.1109/GlobConPT57482.2022.9938320.

[14] Jing-Ming Guo and Herleeyandi Markoni. "Deep Learning Based Lane Line Detection and Segmentation Using Slice Image Feature". In: *2021 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*. 2021, pp. 1–2. DOI: 10.1109/ISPACS51563.2021.9651012.

[15] Yijun Zhou, Jianan Zhao, and Chen Luo. "A novel method for reconstructing general 3D curves from stereo images". In: *The Visual Computer* 37 (July 2021). DOI: 10.1007/s00371-020-01959-6.

[16] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. 2nd. Cambridge, MA: The MIT Press, 2011, pp. 115–116. ISBN: 9780262015356.

[17] Gregory Dudek and Michael Jenkin. "Inertial Sensors, GPS, and Odometry". In: Jan. 2008, pp. 477–490. ISBN: 978-3-540-23957-4. DOI: 10.1007/978-3-540-30301-5_21.

[18] Yury V. Yasyukevich, Baocheng Zhang, and Venkata Ratnam Devanaboyina. "Advances in GNSS Positioning and GNSS Remote Sensing". In: *Sensors* 24.4 (2024). ISSN: 1424-8220. DOI: 10.3390/s24041200. URL: https://www.mdpi.com/1424-8220/24/4/1200.

[19] Katarina Radoš, Marta Brkić, and Dinko Begušić. "Recent Advances on Jamming and Spoofing Detection in GNSS". In: *Sensors* 24.13 (2024). ISSN: 1424-8220. DOI: 10.3390/s24134210. URL: https://www.mdpi.com/1424-8220/24/13/4210.

[20] Canadian Geodetic Reference System Committee (CGRSC). *GNSS Augmentation*. https://cgrsc.ca/resources/gnss-augmentation/. Accessed: 2025-07-30. 2025.

[21] Sebastian Thrun and Arno Bü. "Integrating grid-based and topological maps for mobile robot navigation". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*. AAAI'96. Portland, Oregon: AAAI Press, 1996, pp. 944–950. ISBN: 026251091X.

[22] Raja Chatila and Jean-Paul Laumond. "Position referencing and consistent world modeling for mobile robots". In: vol. Vol. 2. Apr. 1985, pp. 138–145. DOI: 10.1109/ROBOT.1985.1087373.

[23] A. Garulli et al. "Mobile robot SLAM for line-based environment representation". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. 2005, pp. 2041–2046. DOI: 10.1109/CDC.2005.1582461.

[24] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Cambridge, Mass.: MIT Press, 2005, pp. 19–33. ISBN: 0262201623, 9780262201629.

[25] M. S. Grewal and A. P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB®*. 3rd. Hoboken, NJ: John Wiley & Sons, 2008.

[26] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Cambridge, Mass.: MIT Press, 2005, pp. 39–64. ISBN: 0262201623, 9780262201629.

[27] Raad Thaher and Zaid Hussein. "Stereo Vision Distance Estimation Employing SAD with Canny Edge Detector". In: *International Journal of Computer Applications* 107 (Dec. 2014), pp. 38–43. DOI: 10.5120/18735-9977.

[28] Turf Tank. *Marking Soccer Rugby Fields with Turf Tank*. 2025. URL: https://www.youtube.com/watch?v=cTrxOl3FSrA (visited on 06/02/2025).

[29] Muhammad Awais Javeed et al. "Lane Line Detection and Object Scene Segmentation Using Otsu Thresholding and the Fast Hough Transform for Intelligent Vehicles in Complex Road Conditions". In: *Electronics* 12.5 (2023). ISSN: 2079-9292. URL: https://www.mdpi.com/2079-9292/12/5/1079.

[30] Paul Hough. "Method and Means for Recognizing Complex Patterns". US Patent US3069654A. Dec. 1962.

[31] Richard O. Duda and Peter E. Hart. "Use of the Hough transformation to detect lines and curves in pictures". In: 15.1 (Jan. 1972), pp. 11–15. ISSN: 0001-0782. DOI: 10.1145/361237.361242. URL: https://doi.org/10.1145/361237.361242.

[32] Mark Nixon and Alberto Aguado. *Feature Extraction and Image Processing for Computer Vision*. 4th. eBook ISBN: 9780128149775. Academic Press, 2019. ISBN: 9780128149768.

[33] Biao Chen, Bangfeng Ding, and Jiangtao Wang. "Application of an Improved Hough Transform and Image Correction Algorithm in ACC". In: *Journal of Physics: Conference Series* 1621 (Aug. 2020), p. 012044. DOI: 10.1088/1742-6596/1621/1/012044.

[34] Davide Scaramuzza and Friedrich Fraundorfer. "Visual Odometry [Tutorial]". In: *IEEE Robotics Automation Magazine* 18.4 (2011), pp. 80–92. DOI: 10.1109/MRA.2011.943233.

[35] MO Aqel et al. "Review of visual odometry: types, approaches, challenges, and applications". In: *SpringerPlus* 5.1 (2016), p. 1897. DOI: 10.1186/s40064-016-3573-7.

[36] M. Quigley et al. "ROS: an open-source Robot Operating System". In: *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*. 2009.

[37] S. Macenski et al. "Robot Operating System 2: Design, Architecture, and Uses In The Wild". In: *Science Robotics* 7.66 (2022).

[38] Open Robotics. *ROS 2 Documentation: Humble*. https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html. Accessed: 2025-04-12. 2025.

[39] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Cat. No.04CH37566. IEEE. Sendai, Japan, 2004.

[40] G. Sim. *Gazebo Sim*. https://gazebosim.org/home. Accessed: 2025-04-12. 2025.

[41] M. Quigley, B. Gerkey, and W. D. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media, Inc., 2015.

[42] Yahboom Technology. *ROSMASTER X3 - ROS Robot Car*. https://github.com/YahboomTechnology/ROSMASTERX3. Accessed: 2025-05-02. 2023.