REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
FACULTY OF ELECTRICAL-ELECTRONICS ENGINEERING
DEPARTMENT OF CONTROL AND AUTOMATION ENGINEERING

GRADUATION PROJECT

EKF-SLAM BASED INDOOR NAVIGATION

Student(s):
[Your Name(s) and Student Number(s)]

Advisor:
[Your Advisor's Name]

Istanbul, 2024

Chapter 1

ABSTRACT

The challenge of achieving robust autonomy for mobile robots in unknown, GPS-denied environments remains a cornerstone of modern robotics research. Simultaneous Localization and Mapping (SLAM) presents a solution to this fundamental problem, enabling a robot to operate without prior environmental knowledge. This thesis presents the comprehensive design, rigorous implementation, and systematic analysis of an integrated indoor navigation system built upon an Extended Kalman Filter (EKF) based SLAM algorithm. The core of this system is a probabilistic framework that fuses high-frequency odometry data from wheel encoders with exteroceptive sensor measurements, allowing it to concurrently build a consistent map of its surroundings while simultaneously tracking its own pose (position and orientation) within that map.

The mapping methodology is strategically twofold to serve distinct functions. For the state estimation component of SLAM, a feature-based map is generated. The perception module processes raw 2D laser scan data to extract salient, stable landmarks, which are modeled as circular features common in indoor settings. This is achieved through a robust pipeline involving point cloud clustering followed by a least-squares circle-fitting algorithm. A critical challenge in SLAM, the data association problem, is addressed using the Mahalanobis distance metric. This provides a statistically sound method for correlating new sensor observations to previously mapped landmarks by evaluating the consistency of the match against the system's current state uncertainty. In parallel, a dense representation of the environment is constructed as an occupancy grid. This grid map, populated using the robot's estimated poses and live sensor data, provides a rich environmental model suitable for subsequent navigation tasks.

This report thoroughly details the theoretical foundations of EKF-SLAM, the underlying mathematical models for motion and observation, the complete system architecture, and presents a quantitative evaluation of the system's performance. The system's ability to produce a consistent and accurate map of the environment is validated through a series of experiments in both high-fidelity simulations and real-world indoor scenarios, demonstrating its accuracy, robustness, and practical applicability as a foundation for autonomous navigation.

Keywords: EKF-SLAM, Simultaneous Localization and Mapping, Extended Kalman Filter, Mobile Robotics, State Estimation, Indoor Navigation, Landmark Detection, Data Association, Occupancy Grid, Circle Fitting, Laser Scanner.

ÖNSÖZ

Bu bitirme çalışmasının her aşamasında değerli bilgi ve tecrübeleriyle yol gösteren danışman hocamız **[Your Advisor's Name]**'e teşekkürlerimizi sunarız. Proje süresince desteklerini esirgemeyen Kontrol ve Otomasyon Mühendisliği Bölümü'ndeki tüm hocalarımıza ve arkadaşlarımıza teşekkür ederiz.

(If applicable) Bu çalışma, TÜBİTAK BİDEB [2209A / 2209B] programı tarafından desteklenmiştir. Desteklerinden dolayı TÜBİTAK'a teşekkür ederiz.

[Your Name(s)]

Haziran, 2024

Contents

List of Figures

List of Tables

Chapter 2

INTRODUCTION

2.1 Literature Survey

For a mobile robot to operate autonomously in an unknown or dynamic environment, it must answer two fundamental questions simultaneously: "Where am I?" and "What does my environment look like?". The process of answering these coupled questions is known as Simultaneous Localization and Mapping (SLAM). Without a map, a robot cannot determine its location, and without knowing its location, it cannot build a consistent map. This "chicken-and-egg" problem has been a central focus of robotics research for decades.

Early approaches to SLAM were dominated by probabilistic methods, with the Extended Kalman Filter (EKF) being one of the first and most influential solutions. The seminal work by Smith, Self, and Cheeseman established the use of the EKF to maintain a map of geometric features, where both the robot's pose and the feature locations were part of a single, large state vector. While powerful, this approach suffers from several key limitations. Its computational complexity is quadratic with respect to the number of landmarks, as the covariance matrix grows, making it unsuitable for large-scale environments. Furthermore, the EKF relies on a Gaussian noise assumption and can be prone to divergence if its linearization assumptions are strongly violated.

To address these limitations, various other SLAM paradigms have emerged. Particle Filters, as used in FastSLAM, offered a way to represent non-Gaussian, multi-modal probability distributions. This makes them more robust to ambiguity in data association but at a higher computational cost and susceptibility to particle deprivation. More recently, graph-based optimization methods (GraphSLAM) have become the standard for large-scale mapping. These methods formulate SLAM as a nonlinear least-squares problem, representing the robot's trajectory and map features as nodes in a graph connected by constraints from odometry and sensor measurements. Graph-based SLAM is generally more accurate and scalable than filter-based methods but is often performed as a batch optimization, making it less suitable for online, real-time applications without incremental extensions.

Despite these advancements, EKF-SLAM remains a foundational algorithm with significant value. It is an online, recursive filter that does not require storing the entire history of poses and measurements, making it suitable for systems with limited memory. For applications with a moderate number of distinct environmental features, such as the indoor navigation task addressed in this project, its computational efficiency and well-understood probabilistic formulation make it an excellent choice.

2.2 Objective of the Study

The objective of this project is to design, implement, and evaluate a comprehensive indoor navigation system for a differential drive mobile robot. The core of this system is an Extended Kalman Filter (EKF) based SLAM algorithm that enables robust localization, mapping, and autonomous navigation.

The system will be capable of:

State Estimation

Fusing wheel odometry data with external sensor measurements to provide a robust estimate of the robot's pose (θ, x, y). The state is propagated forward in time using a differential drive motion model based on wheel encoder feedback. This prediction is then corrected using landmark measurements in the EKF update step. The system explicitly models process noise to account for uncertainties in the motion model and measurement noise for sensor inaccuracies, which are used in the prediction and update steps of the filter.

Dual-Modality Mapping

Concurrently constructing two types of maps from sensor data, each serving a distinct purpose:

Sparse Landmark-Based Map

A map where the locations of distinct environmental features are estimated and tracked.

- **Feature Extraction**: A dedicated feature extraction component processes raw laser scan data to detect cylindrical landmarks. This process involves clustering scan points based on a proximity threshold, applying a least-squares circle fitting algorithm to each cluster, and filtering the results based on a predefined expected radius to identify valid landmarks.

- **EKF Integration**: The landmark positions are directly integrated into the EKF state vector, alongside the robot's pose. When new landmark observations are received, data association is performed using the Mahalanobis distance to determine if the observation corresponds to an existing landmark or a new, previously unseen one. New landmarks are dynamically added to the state vector, and the map is visualized using marker arrays.

Dense Occupancy Grid Map

A map which represents the environment as a grid of cells, each with a probability of being occupied, free, or unknown.

- **Generation**: This map is built within the main SLAM implementation. It uses the robot's current EKF-estimated pose and the raw LIDAR data.
- **Update Mechanism**: For each laser scan, a ray-casting algorithm updates the grid. Cells along the length of each laser beam up to its endpoint are marked as free space, while the cell at the beam's endpoint is marked as occupied. This allows the system to map not only obstacles but also open, navigable areas. The map is published as a standard occupancy grid message, which serves as the input for the planners.

Path Planning

Utilizing the generated occupancy grid map to compute collision-free paths from a start to a goal configuration. The system implements and allows for the comparison of three distinct planning algorithms, each in its own component:

- **A***: A grid-based, best-first search algorithm that finds the optimal path by minimizing a cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the true cost from the start to the current node and $h(n)$ is a Euclidean distance heuristic to the goal. It explores an 8-connected grid and considers any cell with an occupancy value below 50 to be traversable.
- **RRT**: A randomized, sampling-based algorithm designed for quick exploration of high-dimensional spaces. It incrementally builds a tree from the start pose by randomly sampling points in the configuration space and steering the nearest tree node towards the sample by a fixed step size. A path is found when a node is grown within a specified goal tolerance of the target. The planning duration is logged for performance analysis.
- **Dijkstra**: A grid-based algorithm that finds the shortest path from a single source node to other nodes. It operates similarly to A* but without a heuristic

function (i.e., $h(n) = 0$), expanding outward from the start position based solely on the accumulated cost of travel.

The performance of the SLAM algorithm will be benchmarked against ground truth (in simulation) and pure odometry to quantify its accuracy and consistency. The path planning algorithms will be compared based on key performance metrics, including the final path length and the computation time required to find a solution.

Chapter 3

THEORETICAL BACKGROUND AND SYSTEM MODELING

3.1 The SLAM Problem

The problem of Simultaneous Localization and Mapping (SLAM) is a cornerstone and a fundamental challenge in mobile robotics and autonomous systems. It addresses the critical capability a robot requires to navigate an unknown environment when it cannot rely on an external positioning system, such as the Global Positioning System (GPS). In many practical scenarios—including indoor environments, underwater exploration, dense urban canyons, or planetary exploration—GPS is either unavailable or too imprecise to be useful. Therefore, the robot must rely solely on its own sensors to incrementally build a map of its surroundings while simultaneously using that very map to deduce its own location and orientation (its *pose*).

This concurrent process creates a fundamental "chicken-and-egg" dilemma, a cyclical dependency that lies at the heart of the SLAM problem:

- **Mapping requires accurate localization:** To build a coherent map, the robot must know its precise pose. When the robot's pose estimate is uncertain, the sensor measurements it collects (e.g., the distance to a wall or the location of a landmark) will be registered incorrectly in the map's frame of reference. This uncertainty leads to a distorted, inconsistent, and ultimately unusable map.
- **Localization requires a reliable map:** To accurately determine its pose, the robot needs a consistent and reliable map of the environment. It localizes itself by matching its current sensor readings to the features or structures already present in the map. If the map is inaccurate, this matching process will fail or produce an incorrect pose estimate, leading to the robot becoming "lost."

This inherent inter-dependency means the localization and mapping problems are tightly coupled and cannot be solved independently. They must be addressed concurrently. The core challenge of SLAM is to manage the uncertainties associated with both the robot's motion and its sensor measurements. Odometry errors from wheel encoders or inertial measurement units (IMUs) cause the robot's estimated pose to drift over time. Concurrently, all sensors, whether they be LiDAR, cameras, or sonar, are subject to noise, further complicating the task.

3.1.1 Probabilistic Formulation of SLAM

Given these inherent uncertainties, the SLAM problem is naturally framed within a probabilistic estimation framework. The goal of a SLAM system is to process a time series of control inputs (e.g., motor commands) and sensor measurements to compute a joint posterior probability distribution over the robot's trajectory and the map.

Let the history of the robot's poses be the trajectory $x_{1:k} = \{x_1, x_2, \ldots, x_k\}$, where x_k is the robot's pose at time k. Let the map of the environment be denoted by m. The available data consists of the sequence of control inputs, $u_{1:k} = \{u_1, u_2, \ldots, u_k\}$, and the sequence of sensor measurements, $z_{1:k} = \{z_1, z_2, \ldots, z_k\}$.

The complete SLAM problem, often referred to as **Full SLAM**, is to find the joint posterior distribution over the entire trajectory and the map:

$$P(x_{1:k}, m | z_{1:k}, u_{1:k}) \tag{3.1}$$

Solving for this full posterior provides the most accurate possible solution, as it utilizes all available information to estimate every pose in the robot's history. However, the dimensionality of this problem grows with every time step, making it computationally intractable for real-time applications, especially over long durations.

3.1.2 The Online SLAM Formulation

To make the problem computationally feasible for an operating robot, the **Online SLAM** problem was formulated. In this version, the focus shifts from the entire trajectory to the current state. The system marginalizes out the past poses, seeking to estimate the posterior over the current robot pose and the map:

$$P(x_k, m | z_{1:k}, u_{1:k}) \tag{3.2}$$

This simplification is the foundation for filter-based algorithms like the Extended Kalman Filter SLAM (EKF-SLAM). By focusing only on the current pose x_k, the complexity of the update step is kept bounded, enabling real-time performance. This approach sequentially updates the belief about the current state as each new measurement arrives. While this makes the problem tractable, it also means that by discarding past pose information, any errors in the state estimate can accumulate over time as past states are not corrected.

3.2 Probabilistic Robotics Foundation

The entire system is built upon the principles of probabilistic state estimation, which provides a framework for dealing with the inherent uncertainty in sensor data and robot motion. The core idea is to represent the robot's belief about its state and the world not as single, definite values, but as probability distributions. The estimation process is a recursive cycle of prediction and update, governed by Bayes' Theorem. For the continuous state space of the robot and landmarks, this is expressed in terms of probability density functions (PDFs):

$$f_X(x|z) = \frac{f_Z(z|x)f_X(x)}{f_Z(z)} \tag{3.3}$$

This can be broken down into the familiar Bayesian filtering cycle, which our system's Extended Kalman Filter (EKF) approximates for Gaussian distributions:

$$\underbrace{P(x_k|z_{1:k})}_{\text{posterior}} \propto \underbrace{P(z_k|x_k)}_{\text{likelihood}} \overbrace{\int P(x_k|x_{k-1})P(x_{k-1}|z_{1:k-1})dx_{k-1}}^{\text{prior}} \tag{3.4}$$

In our implementation, the system's entire belief is captured by a multivariate Gaussian distribution, defined by a mean state vector and a system covariance matrix. The state vector contains the robot's pose (θ, x, y) as well as the (x, y) coordinates of all observed landmarks. The covariance matrix represents the uncertainty across this entire state.

- **Prediction (Prior):** The prior is our belief about the current state before incorporating the latest sensor measurement. It is formed by applying a differential drive motion model to the previous posterior belief. This step, implemented in the EKF prediction function, uses odometry from wheel encoders to project the state forward in time. This prediction inherently increases the uncertainty in our estimate, a fact that is explicitly modeled by adding a defined process noise covariance to the system's covariance matrix.

- **Correction (Likelihood & Posterior):** When a new sensor measurement, z_k, provides fresh information about the environment, a correction is applied.
  - Raw laser scan data is first processed to extract landmark features (range and bearing).

  – Before correction, a data association step determines which landmark in the existing state vector corresponds to the new measurement. This is achieved probabilistically by calculating the Mahalanobis distance for each known landmark; the measurement is associated with the landmark that is closest in terms of statistical distance, provided it is below a set threshold. If no association is made, a new landmark is initialized in the state vector.
  – The likelihood, $P(z_k|x_k)$, quantifies how probable that measurement is, given a hypothetical state x_k. In the EKF update function, this is realized by comparing the actual measurement to an expected measurement calculated from the predicted state.
  – A Kalman gain is computed, which optimally weighs the uncertainty of the predicted state against the uncertainty of the sensor measurement. This gain is then used to update the predicted state and shrink the covariance matrix, resulting in the corrected posterior belief with reduced uncertainty.

3.3 The Extended Kalman Filter (EKF)

The standard Kalman Filter provides an optimal, recursive solution for state estimation, but its applicability is strictly limited to linear systems with Gaussian noise distributions. In robotics, however, nearly all meaningful problems involve nonlinearities. The motion of a robot arm, the kinematics of a wheeled robot, or the relationship between landmarks and sensor measurements (e.g., bearing and range) are inherently nonlinear processes.

The Extended Kalman Filter (EKF) is an extension of the Kalman Filter designed to handle these nonlinear systems. The core idea of the EKF is not to solve the nonlinear problem directly, but to approximate it as a series of linear problems. It achieves this by repeatedly linearizing the nonlinear models around the current best estimate of the state using a first-order Taylor series expansion. At each timestep, a new linear model is generated, to which the standard Kalman Filter equations can be applied.

Consider a system with a nonlinear state transition function $g(\cdot)$ and a nonlinear measurement function $h(\cdot)$:

$$x_k = g(x_{k-1}, u_{k-1}, w_{k-1}) \tag{3.5}$$

$$z_k = h(x_k, v_k) \tag{3.6}$$

Here, w_{k-1} and v_k represent the process and measurement noise, which are assumed to be zero-mean Gaussian random variables, $w_k \sim \mathcal{N}(0, Q)$ and $v_k \sim \mathcal{N}(0, R)$.

The fundamental challenge is that when a Gaussian distribution (representing our belief about the state) is passed through a nonlinear function, the resulting distribution is no longer Gaussian. The EKF circumvents this by using the Jacobians of the nonlinear functions to create a local linear approximation. These Jacobians are the matrices of all first-order partial derivatives:

- $A_k = \frac{\partial g}{\partial x}|_{\hat{x}_{k-1}, u_{k-1}, 0}$: The Jacobian of the motion model with respect to the state.
- $W_k = \frac{\partial g}{\partial w}|_{\hat{x}_{k-1}, u_{k-1}, 0}$: The Jacobian of the motion model with respect to the process noise.
- $H_k = \frac{\partial h}{\partial x}|_{\tilde{x}_k, 0}$: The Jacobian of the measurement model with respect to the state.
- $V_k = \frac{\partial h}{\partial v}|_{\tilde{x}_k, 0}$: The Jacobian of the measurement model with respect to the measurement noise.

For many common models where the noise is simply additive (i.e., $g(x, u, w) = g'(x, u) + w$ and $h(x, v) = h'(x) + v$), the Jacobians W_k and V_k become identity matrices, simplifying the equations. However, the general form is important for more complex models.

Prediction Step

The prediction step, also known as the "time update," projects the state and its uncertainty forward in time.

First, the state estimate is propagated forward using the full nonlinear motion model. This is done to get the most accurate possible a priori estimate, referred to as $\tilde{x}_k$ or $\hat{x}_k^-$:

$$\hat{x}_k^- = g(\hat{x}_{k-1}, u_{k-1}, 0) \tag{3.7}$$

Next, the covariance of the state is propagated. We cannot simply pass the covariance matrix Σ_{k-1} through the nonlinear function g, as this would result in a non-Gaussian distribution. Instead, the covariance is projected forward using the linearized model defined by the Jacobian A_k. The process noise is transformed by its Jacobian W_k:

$$\Sigma_k^- = A_k \Sigma_{k-1} A_k^T + W_k Q_{k-1} W_k^T \tag{3.8}$$

Update Step

The update step, or "measurement update," corrects the predicted state based on a new measurement z_k.

The core of the update is the "innovation" or "residual," which is the difference between the actual sensor measurement z_k and the predicted measurement $h(\hat{x}_k^-, 0)$:

$$\text{innovation} = z_k - h(\hat{x}_k^-, 0) \tag{3.9}$$

The Kalman Gain K_k is then computed. This gain is a crucial matrix that determines how much the innovation should influence the new state estimate. It optimally weighs the uncertainty of the prediction (Σ_k^-) against the uncertainty of the measurement (R), as transformed by their respective Jacobians:

$$K_k = \Sigma_k^- H_k^T (H_k \Sigma_k^- H_k^T + V_k R_k V_k^T)^{-1} \tag{3.10}$$

Finally, the state estimate and its covariance are updated by incorporating the measurement. The a posteriori state estimate $\hat{x}_k$ is calculated by adding the innovation, weighted by the Kalman gain, to the predicted state:

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0)) \tag{3.11}$$

The covariance is also updated to reflect the reduction in uncertainty resulting from the measurement:

$$\Sigma_k = (I - K_k H_k)\Sigma_k^- \tag{3.12}$$

This updated state $\hat{x}_k$ and covariance Σ_k then serve as the input for the prediction step at the next time index, continuing the recursive cycle.

Assumptions and Limitations

While the EKF is a powerful and widely used tool for nonlinear state estimation, its performance is predicated on several key assumptions. The violation of these assumptions can lead to poor performance or even filter divergence.

- **Gaussianity:** The EKF fundamentally assumes that the uncertainty of the state can be adequately represented by a Gaussian distribution at every time step. While the prediction and update steps are designed to produce a Gaussian output, if the true underlying distribution becomes highly non-Gaussian (e.g., bimodal after seeing an ambiguous landmark), the EKF's single-mean approximation will be a poor representation of the true belief.

- **Linearity Assumption:** The accuracy of the EKF hinges on how well the first-order Taylor expansion approximates the true nonlinear functions. If the system is highly nonlinear, or if the uncertainty (Σ_k) is large, the linearization around the mean may not be a representative approximation of the function's behavior across the full spread of the uncertainty. This linearization error is a primary source of sub-optimality in the EKF.

- **Differentiability:** The approach requires that the functions $g(\cdot)$ and $h(\cdot)$ are differentiable so that the Jacobians can be computed. This excludes systems with true discontinuities.

3.4 Robot Motion and Odometry Model

The motion of the differential drive robot serves as our state transition model, $g(\cdot)$, which is the foundation of the EKF's prediction step. The relationship between the robot's body velocity (twist) and the angular velocities of its wheels is defined by its kinematics. The body twist $V_b = (\dot{\theta}, v_x, v_y)^T$ describes the instantaneous motion of the robot's reference frame. The twist of the body frame can be expressed in the frame of each wheel using the adjoint transformation matrix, A_{ib}. For a wheel i:

$$V_i = A_{ib} V_b \tag{3.13}$$

For our differential drive robot with left and right wheels, and a half track-width of d, the adjoints for transforming from the body to the wheel frames are:

$$A_{lb} = \begin{bmatrix} 1 & 0 & 0 \\ -d & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, A_{rb} = \begin{bmatrix} 1 & 0 & 0 \\ d & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

This gives the twists in the left and right wheel frames:

$$V_l = A_{lb} V_b = \begin{bmatrix} \dot{\theta} \\ v_x - d\dot{\theta} \\ v_y \end{bmatrix} \tag{3.15}$$

$$V_r = A_{rb} V_b = \begin{bmatrix} \dot{\theta} \\ v_x + d\dot{\theta} \\ v_y \end{bmatrix} \tag{3.16}$$

For conventional wheels, the non-holonomic constraint dictates that there is no lateral slipping, so the velocity in the wheel's y-direction is zero ($v_{yi} = 0$), and the velocity in the x-direction is $v_{xi} = r\dot{\phi}_i$, where r is the wheel radius and $\dot{\phi}_i$ is its angular velocity. Substituting these constraints gives the inverse kinematics, which determine the wheel speeds needed for a desired body twist:

$$\dot{\phi}_l = \frac{v_x - d\dot{\theta}}{r} \tag{3.17}$$

$$\dot{\phi}_r = \frac{v_x + d\dot{\theta}}{r} \tag{3.18}$$

For odometry, we use the forward kinematics to compute the body twist from measured wheel rotations. In our system, wheel encoders measure the change in wheel angles, which are used to find the angular velocities $(\dot{\phi}_l, \dot{\phi}_r)$ over a timestep Δt. These are the inputs to the forward kinematics equation to compute the body twist that the robot experienced.

$$V_b = \begin{bmatrix} \omega_b \\ v_{bx} \\ v_{by} \end{bmatrix} = \begin{bmatrix} -r/(2d) & r/(2d) \\ r/2 & r/2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\phi}_l \\ \dot{\phi}_r \end{bmatrix} \tag{3.19}$$

This calculated twist is our control input, $u_k = V_b \Delta t$. It is then integrated to update the robot's pose from its previous state estimate, $\hat{x}_{k-1}$, giving our nonlinear state transition function $g(\cdot)$. The integration is handled differently depending on the robot's angular velocity. If the angular velocity ω_b is nearly zero (the robot is moving straight):

$$g(\hat{x}_{k-1}, u_k) = \begin{bmatrix} \hat{x}_{k-1,x} + v_{bx}\Delta t \cos(\hat{x}_{k-1,\theta}) \\ \hat{x}_{k-1,y} + v_{bx}\Delta t \sin(\hat{x}_{k-1,\theta}) \\ \hat{x}_{k-1,\theta} \end{bmatrix} \tag{3.20}$$

If ω_b is non-zero (the robot is turning):

$$g(\hat{x}_{k-1}, u_k) = \begin{bmatrix} \hat{x}_{k-1,x} + \frac{v_{bx}}{\omega_b}(\sin(\hat{x}_{k-1,\theta} + \omega_b \Delta t) - \sin(\hat{x}_{k-1,\theta})) \\ \hat{x}_{k-1,y} + \frac{v_{bx}}{\omega_b}(-\cos(\hat{x}_{k-1,\theta} + \omega_b \Delta t) + \cos(\hat{x}_{k-1,\theta})) \\ \hat{x}_{k-1,\theta} + \omega_b \Delta t \end{bmatrix} \tag{3.21}$$

Because the EKF requires a linear model for propagating uncertainty, this non-linear function $g(\cdot)$ must be linearized at each timestep. This is done by computing its Jacobian, $A_k = \frac{\partial g}{\partial x}\Big|_{\hat{x}_{k-1}, u_k}$. This matrix describes how small changes in the prior state affect the predicted state. For the differential drive model, the Jacobian takes the form:

$$A_k = \begin{bmatrix} 1 & 0 & \frac{\partial g_x}{\partial \theta} \\ 0 & 1 & \frac{\partial g_y}{\partial \theta} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

where the partial derivative terms depend on whether the robot is turning or moving straight. This Jacobian is critical for the prediction step of the EKF, as it is used to transform the system covariance: $\Sigma_k = A_k \Sigma_{k-1} A_k^T + Q_k$, where Q_k is the process noise.

3.5 Sensor Measurement Model

The sensor measurement model, $h(\cdot)$, is a function that predicts the expected sensor reading given the current state estimate of the robot and landmarks. While the system's primary sensor is a laser scanner, the measurement model operates on a more abstract feature: the position of cylindrical landmarks detected in the environment. A dedicated component processes raw laser data to identify these landmarks and provides their position relative to the robot in Cartesian coordinates. For use in the filter, these are converted into a range-bearing format, $z_k = (r, \phi)^T$, as sensor noise is more naturally expressed in these polar coordinates.

Given a robot pose (x_r, y_r, θ_r) and a landmark position (x_m, y_m) from the state vector, the non-linear measurement model $h(\cdot)$ predicts the expected measurement as:

$$h(\hat{\xi}_k) = \begin{bmatrix} \sqrt{(x_m - x_r)^2 + (y_m - y_r)^2} \\ \text{atan2}(y_m - y_r, x_m - x_r) - \theta_r \end{bmatrix} \tag{3.23}$$

This predicted measurement is assumed to be corrupted by sensor noise, which is modeled as a zero-mean Gaussian, $v_t \sim \mathcal{N}(0, R)$.

Because the measurement model is non-linear, it must be linearized around the current state estimate to be used in the Extended Kalman Filter framework. This linearization is the Jacobian of the measurement function, denoted H_k. For a state vector containing the robot pose and N landmarks, this Jacobian is a sparse 2x(3+2N) matrix where the only non-zero entries correspond to the robot's pose and the specific landmark j being observed. The Jacobian for this observation is:

$$H_k^j = \begin{bmatrix} -\frac{\Delta x}{\sqrt{d}} & -\frac{\Delta y}{\sqrt{d}} & 0 & \dots & \frac{\Delta x}{\sqrt{d}} & \frac{\Delta y}{\sqrt{d}} & \dots \\ \frac{\Delta y}{d} & -\frac{\Delta x}{d} & -1 & \dots & -\frac{\Delta y}{d} & \frac{\Delta x}{d} & \dots \end{bmatrix} \tag{3.24}$$

where $\Delta x = x_{m_j} - x_r$, $\Delta y = y_{m_j} - y_r$, and $d = \Delta x^2 + \Delta y^2$ are all computed using the current state estimate. Together, the non-linear model $h(\cdot)$ and its Jacobian H provide the necessary components for the EKF to perform its measurement update step, correcting the state estimate with new sensory information.

Chapter 4

SYSTEM ARCHITECTURE AND IMPLEMENTATION

4.1 EKF-SLAM Algorithm

In EKF-SLAM, the state vector is augmented to include not only the robot's pose but also the positions of all observed landmarks.

$$\xi = [\underbrace{x_r, y_r, \theta_r}_{\text{robot pose}}, \underbrace{m_{1,x}, m_{1,y}}_{\text{landmark 1}}, \dots, \underbrace{m_{N,x}, m_{N,y}}_{\text{landmark N}}]^T \tag{4.1}$$

The corresponding covariance matrix Σ represents the uncertainty in this entire augmented state. Crucially, it contains not only the uncertainty of the robot pose and landmark positions (on its diagonal blocks) but also the cross-correlation terms between every pair of elements (in its off-diagonal blocks). These correlations are the key to SLAM; observing a known landmark reduces the uncertainty in the robot's pose, which in turn, due to the stored correlations, reduces the uncertainty of other, unobserved landmarks. When the filter is initialized, the robot's pose covariance is typically set to zero (assuming a known start position), while the landmark covariance terms are initialized to infinity, reflecting complete uncertainty about the map.

Algorithm 1 EKF SLAM Loop (Pseudo Code)

```
1: function EKF_SLAM_Loop(previous_state, previous_covariance, odometry_input, measurements)
2:     // Prediction Step
3:     $predicted\_state \leftarrow$ motion_model($previous\_state$, $odometry\_input$)   ▷ Predict state using motion model $g(\cdot)$
4:     $A \leftarrow$ jacobian_of_motion_model($previous\_state$, $odometry\_input$)   ▷ Linearize the motion model
5:     $predicted\_covariance \leftarrow A \cdot previous\_covariance \cdot A^T +$ process_noise_Q   ▷ Propagate uncertainty
6:     // Update Step
7:     $corrected\_state \leftarrow predicted\_state$
8:     $corrected\_covariance \leftarrow predicted\_covariance$
9:     for each measurement $z\_i$ in $measurements$ do
10:        $landmark\_id \leftarrow$ data_association($z\_i$, $corrected\_state$, $corrected\_covariance$)
11:        if $landmark\_id$ is new then
12:            initialize_new_landmark($corrected\_state$, $corrected\_covariance$, $z\_i$)
13:        end if
14:        $predicted\_measurement \leftarrow$ measurement_model($corrected\_state$, $landmark\_id$)   ▷ Predict measurement using $h(\cdot)$
15:        $H \leftarrow$ jacobian_of_measurement_model($corrected\_state$, $landmark\_id$)   ▷ Linearize measurement model
16:        $K \leftarrow corrected\_covariance \cdot H^T \cdot \Big(H \cdot corrected\_covariance \cdot H^T +$ measurement_noise_R$\Big)^{-1}$   ▷ Compute Kalman Gain
17:        $corrected\_state \leftarrow corrected\_state + K \cdot \big(z\_i - predicted\_measurement\big)$   ▷ Correct the state
18:        $corrected\_covariance \leftarrow \big(I - K \cdot H\big) \cdot corrected\_covariance$   ▷ Update the covariance
19:    end for
20:    return $corrected\_state$, $corrected\_covariance$
21: end function
```

Algorithm Breakdown

The EKF-SLAM algorithm is a two-step process that recursively estimates the state and covariance.

Prediction

The prediction step uses the robot's odometry to estimate its new position after a motion.

- **State Prediction (Line 3):** The system uses the odometry model, $g(\cdot)$, to project the previous state estimate forward. The control input for this model is a body twist computed from wheel encoder readings.

- **Covariance Prediction (Lines 4-5):** The motion model is linearized by computing its Jacobian, A_t, with respect to the state. This matrix is then used to propagate the covariance estimate forward, incorporating process noise, Q, to model the uncertainty in the robot's motion. During this step, the parts of the state vector and covariance matrix corresponding to the landmarks remain unchanged, as the map is assumed to be static.

Update (Correction)

The update step corrects the predicted state using sensor measurements of landmarks. This phase involves more than a standard EKF update.

- **Data Association (Line 9):** Before incorporating a measurement, the system must determine which landmark it corresponds to. In our implementation, this is done by computing the Mahalanobis distance between the measurement and the predicted location of each known landmark. The measurement is associated with the closest landmark, provided the distance is below a set threshold.

- **Landmark Initialization (Lines 10-12):** If a measurement cannot be associated with any existing landmark, it is assumed to be a new landmark, which is then initialized and added to the state vector and covariance matrix. The new landmark's global position is estimated by inverting the measurement model using the current robot pose estimate.

- **Measurement Correction (Lines 14-18):** For each associated measurement, a correction is applied. The system first computes a theoretical measurement using the non-linear measurement model $h(\cdot)$. This model is linearized by computing its Jacobian, H. The Kalman gain K is then calculated, which optimally weighs the prior uncertainty against the measurement noise R. Finally, the state and covariance are updated using this gain and the difference between the actual and theoretical measurement. This process is performed sequentially for each observed landmark.

4.2 Landmark-Based Mapping

The landmark-based map provides a sparse representation of the environment by detecting and tracking the positions of distinct cylindrical objects. This map is not a separate data structure but is directly embedded within the EKF state vector, tightly coupling the process of localization and mapping. The creation and maintenance of this map involve a pipeline of feature extraction followed by probabilistic data association within the SLAM filter.

Feature Extraction

A dedicated feature extraction node processes the raw 2D laser scan data to identify potential landmarks. This process involves several steps:

- **Clustering:** Points from the laser scan are first converted from polar to Cartesian coordinates. The algorithm then iterates through the points, grouping consecutive points into clusters if the Euclidean distance between them is below a fixed threshold. This process also handles the "wrap-around" case where the last point and the first point in a 360-degree scan may belong to the same object. Clusters containing fewer than a minimum number of points are discarded.

- **Circle Fitting:** A circle is fitted to each valid cluster of points using a least-squares method to estimate the landmark's center (a, b) and radius R. This is based on an algebraic fit which solves for the parameters of the circle equation:

  $$(x-a)^2 + (y-b)^2 = R^2 \tag{4.2}$$

  The implementation calculates the centroid of the cluster's points, shifts the coordinates to this centroid, and then uses Singular Value Decomposition (SVD) to robustly solve for the circle parameters.

- **Filtering and Publishing:** The fitted circles are then filtered based on their radius. Only circles whose estimated radius is within a small tolerance of a predefined, expected landmark radius are considered valid. The relative Cartesian coordinates of these validated landmarks are then published for use by the main EKF-SLAM node.

Data Association and Map Management

When the SLAM node receives a set of landmark detections, it must associate each measurement with an existing landmark in the map or identify it as a new one. This is a critical step, as incorrect associations can cause the filter to diverge.

- **Mahalanobis Distance:** The association is achieved by calculating the Mahalanobis distance between the new measurement z_i and the predicted measurements $\hat{z}_k$ for all existing landmarks in the map.

  $$D_M(z_i, \hat{z}_k) = \sqrt{(z_i - \hat{z}_k)^T \Psi_k^{-1} (z_i - \hat{z}_k)} \tag{4.3}$$

where $\Psi_k = H_k \Sigma_k^- H_k^T + R$ is the innovation covariance. This statistical distance is superior to Euclidean distance as it accounts for the current uncertainty of both the robot's pose and the landmark's estimated position.

- **Association Logic:** The measurement is associated with the landmark that has the smallest Mahalanobis distance, provided it is below a certain validation threshold. If a valid association is found, the measurement is used to perform a standard EKF correction on the state of the robot and that specific landmark. If no existing landmark provides a match below the threshold, the measurement is considered to be of a new, previously unseen landmark. A new landmark is then initialized in the state vector, and its position is calculated by inverting the measurement model. This allows the map to be built incrementally as the robot explores its environment.

4.3 Occupancy Grid Mapping

While the EKF component of a SLAM system maintains a sparse, feature-based map for localization, a complementary representation is often required for tasks such as navigation and path planning. For this, an occupancy grid map is generated in parallel. This method provides a dense, geometric representation of the environment, which is essential for determining collision-free paths and for human-interpretable visualization of the robot's surroundings.

The map is discretized into a 2D grid of fixed-size cells. Each cell, m_i, stores the probability that it is occupied by an obstacle, conditioned on all sensor measurements up to the current time, $P(m_i|z_{1:k})$. This approach rests on a key simplifying assumption: the states of all cells are conditionally independent given the robot's trajectory. This means that knowing the state of one cell gives no information about the state of another, provided the robot's pose is known. This assumption makes the problem computationally tractable, as it allows the state of each cell to be estimated independently.

A primary challenge in updating these probabilities directly is their bounded nature. Probabilities are confined to the interval $[0, 1]$, which can lead to numerical instability and saturation. For example, once a cell's probability approaches 1.0, it requires an overwhelming amount of contradictory evidence to reduce its value significantly. The filter becomes overly confident and loses its ability to adapt to changes in the environment.

Log-Odds Representation

To resolve these issues, the standard approach is to work with the log-odds representation of probability instead of the probability itself. The log-odds $l(m_i)$ for a cell m_i is defined as the logarithm of the ratio between the probability of being occupied and the probability of being free:

$$l(m_i) = \log \frac{P(m_i|z_{1:k})}{1 - P(m_i|z_{1:k})} \tag{4.4}$$

This transformation elegantly maps the probability from the bounded interval $[0, 1]$ to the entire real number line $(-\infty, \infty)$. A log-odds value of 0 corresponds to a probability of 0.5 (completely unknown), with large positive values indicating high confidence in occupancy and large negative values indicating high confidence in the cell being free. This formulation avoids saturation and simplifies the update rule. To recover the probability from the log-odds value, the inverse transformation is used:

$$P(m_i|z_{1:k}) = 1 - \frac{1}{1 + \exp(l(m_i))} \tag{4.5}$$

The Log-Odds Update Rule

A significant advantage of the log-odds representation is that the Bayesian update, which is multiplicative in probability space, becomes a simple and efficient addition in log-odds space. The belief about a cell m_i at time k is updated as follows:

$$l_{k,i} = l_{k-1,i} + \text{inverse\_sensor\_model}(m_i, x_k, z_k) - l_0 \tag{4.6}$$

Here, $l_{k-1,i}$ is the previous log-odds value of the cell, and the term inverse_sensor_model$(\cdot)$ provides the new information gained from the latest sensor measurement z_k, given the robot's current pose estimate x_k. The term l_0 represents the prior belief about occupancy, which is the log-odds corresponding to $P = 0.5$ (i.e., $l_0 = 0$). Subtracting the prior prevents it from being counted repeatedly in the recursive update.

Inverse Sensor Model for Laser Scanners

The inverse sensor model defines how a sensor measurement affects the belief about the map cells. For a 2D laser scanner, which provides a set of range-and-bearing measurements, the model works by tracing each laser beam through the grid. For a single beam from measurement z_k taken at pose x_k:

- The grid cells that the laser beam **passes through** on its way to its endpoint are considered free space. Their log-odds values are updated by adding a predefined value, l_{free}. This value is negative, thereby decreasing the cell's log-odds and increasing its probability of being free.

- The single grid cell in which the laser beam **terminates** is considered occupied. Its log-odds value is updated by adding l_{occ}, a positive value that increases its probability of being occupied.

Cells that are outside the sensor's field of view remain unchanged. In practice, the magnitude of l_{occ} is often greater than that of l_{free}, reflecting a higher confidence in an occupied reading (a single hit is strong evidence) compared to a free-space reading (a beam passing through might have simply missed a small or transient obstacle).

It is critical to note that the entire grid mapping process is fundamentally dependent on the accuracy of the robot pose estimate x_k provided by the SLAM filter (e.g., the EKF). If the localization is inaccurate, sensor data will be registered incorrectly, leading to a blurred, smeared, and inconsistent map that is unusable for navigation. Therefore, the quality of the occupancy grid is a direct reflection of the quality of the simultaneous localization solution.

4.4 Fusing SLAM with Occupancy Grid Mapping

The two mapping approaches—landmark-based EKF and occupancy grid—are not independent; they are fused to create a robust and detailed representation of the environment. The landmark-based EKF provides a globally consistent estimate of the robot's pose, which is essential for building an accurate, dense occupancy grid. The sparse landmark map provides long-term stability and corrections, while the occupancy grid offers detailed geometric information about all obstacles and free space, which is critical for navigation tasks like path planning.

The Problem of Odometric Drift

If the occupancy grid were built using only raw odometry data, it would quickly become distorted. As the robot moves, small errors in wheel encoders accumulate, causing the odometry pose to drift from the true pose. This drift is unbounded and grows over time. When plotting sensor data using this drifting reference frame, features like walls and obstacles would appear smeared, duplicated, or misaligned in the grid map, rendering it useless for reliable navigation. The system visualizes this issue by publishing both the uncorrected odometry path and the corrected SLAM path, making the accumulated drift apparent.

The Solution: A Globally Consistent Frame

To prevent this, the system leverages the pose estimate $\hat{\xi}_k = (\hat{x}_k, \hat{y}_k, \hat{\theta}_k)$ from the EKF-SLAM filter. This pose has been corrected by landmark observations and is anchored to a globally consistent map frame. The fusion occurs within a dedicated callback that triggers whenever new laser scan data is received. The first action in this callback is to retrieve the most current, corrected pose from the EKF state.

Each laser scan, which is measured in the robot's local frame, is then transformed into the global map frame using this corrected pose before being used to update the occupancy grid. For a laser scan point $p_{\text{robot}} = [x_{\text{scan}}, y_{\text{scan}}]^T$ in the robot's frame, its position in the map frame p_{map} is calculated as:

$$p_{\text{map}} = \begin{bmatrix} \cos(\hat{\theta}_k) & -\sin(\hat{\theta}_k) \\ \sin(\hat{\theta}_k) & \cos(\hat{\theta}_k) \end{bmatrix} p_{\text{robot}} + \begin{bmatrix} \hat{x}_k \\ \hat{y}_k \end{bmatrix} \tag{4.7}$$

By using the SLAM-corrected pose for this transformation, we ensure that all sensor data is integrated into a single, coherent global frame, free from the cumulative errors of odometry.

Grid Update Mechanism

The occupancy grid itself is a 2D array where each cell holds a value representing the probability that it is occupied. The grid is updated using a ray-casting technique:

- For each beam in a laser scan, a ray is traced from the SLAM-corrected robot position out to the measured distance in the direction of the beam.
- All cells that the ray passes through are updated to have a higher probability of being **free space**.
- The single cell at the endpoint of the ray (if it is within the sensor's maximum range) is updated to have a higher probability of being **occupied**.

This process is repeated for every beam in the scan and for every new scan received. Over time, this carves out the navigable free space in the environment while building up a solid representation of walls and obstacles. The completed map is then periodically published as a standard occupancy grid message for use by other system components, such as the path planners.

4.5 System Hardware and Software Platform

The experimental validation of this project was conducted on a TurtleBot3 Burger mobile robot. This platform was chosen for its accessibility, well-supported ROS integration, and suitable sensor suite for indoor SLAM.

- **Hardware Platform:** The TurtleBot3 Burger is a compact, two-wheeled differential drive robot. Its key hardware components include:
  - **Actuation:** Two Dynamixel XL430-W250-T servo motors with integrated wheel encoders, providing both motion and proprioceptive feedback (odometry).
  - **Sensing:** A 360° 2D laser distance sensor (LDS-01) is the primary exteroceptive sensor for mapping and landmark detection.
  - **Computation:** An onboard Raspberry Pi 4 serves as the main computer, running the ROS framework and all high-level navigation nodes.
  - **Low-Level Control:** An OpenCR 1.0 board interfaces with the motors and sensors, providing a direct hardware abstraction layer.
- **Software Framework:** The entire system is built upon the Robot Operating System (ROS 2). ROS provides the necessary middleware for communication between different software modules (nodes), including message passing, service calls, and parameter management. The modularity of ROS allows the SLAM algorithm, mapping processes, and path planners to run as independent yet interconnected nodes.

Chapter 5

PATH PLANNING

Once a consistent occupancy grid map is built, the robot can navigate the environment. This requires a path planner to find a sequence of waypoints from the robot's current location to a specified goal. This project implements and compares three classical algorithms for this purpose. Each planner is an independent component that subscribes to the occupancy grid map, the robot's current pose, and goal requests. Upon finding a valid path, it publishes a sequence of poses for the robot to follow. The following sections detail the theory and implementation of each planner.

5.1 A* Algorithm

A* is an informed or "best-first" search algorithm that finds the least-cost path from a start node n_{start} to a goal node n_{goal}. It efficiently explores the graph by prioritizing nodes that appear to be on the best path. This is achieved by evaluating each node n with the function:

$$f(n) = g(n) + h(n) \tag{5.1}$$

where:

- $g(n)$ is the exact, known cost of the path from the start node to the current node n.
- $h(n)$ is a heuristic function that estimates the cost of the cheapest path from n to the goal. A common heuristic for grid maps is the Euclidean distance: $h(n) = \sqrt{(n_x - n_{\text{goal},x})^2 + (n_y - n_{\text{goal},y})^2}$.

A* maintains a priority queue of nodes to visit, ordered by the lowest $f(n)$ value. For the heuristic to be admissible (and for A* to guarantee an optimal path), $h(n)$ must never overestimate the actual cost to the goal. In our system, the algorithm operates on a discretized grid, where traversable cells have an occupancy value below a set threshold, and movement is permitted to 8-connected neighbors.

Algorithm 2 A* Algorithm (Pseudo Code)

```
1: function A_STAR(start, goal, map)
2:     open_set ← PriorityQueue()
3:     open_set.add(start, 0)
4:     came_from ← {}
5:     g_score ← {node : $\infty$  for all nodes in map}
6:     g_score[start] ← 0
7:     f_score ← {node : $\infty$  for all nodes in map}
8:     f_score[start] ← heuristic(start, goal)
9:     while open_set is not empty do
10:        current ← open_set.pop_lowest_f_score()
11:        if current = goal then
12:            return reconstruct_path(came_from, current)
13:        end if
14:        for each neighbor of current do
15:            tentative_g_score ← g_score[current] + dist(current, neighbor)
16:            if tentative_g_score < g_score[neighbor] then
17:                came_from[neighbor] ← current
18:                g_score[neighbor] ← tentative_g_score
19:                f_score[neighbor] ← tentative_g_score + heuristic(neighbor, goal)
20:                if neighbor not in open_set then
21:                    open_set.add(neighbor, f_score[neighbor])
22:                end if
23:            end if
24:        end for
25:    end while
26:    return failure                                        ▷ No path found
27: end function
```

Pseudo-code Explanation

The algorithm begins by initializing data structures (Lines 2-8): an ‘open_set‘ priority queue to track nodes to be explored, a ‘came_from‘ map to reconstruct the final path, and maps to store the ‘g_score‘ (cost from start) and ‘f_score‘ (total estimated cost) for each node. The main loop (Line 10) continues as long as there are potential nodes to explore. In each iteration, it selects the node with the lowest ‘f_score‘ from the ‘open_set‘ as the ‘current‘ node to expand. If this node is the goal, the path is reconstructed and returned (Lines 12-14). Otherwise, the algorithm examines each of the current node’s neighbors (Line 16). For each neighbor, it calculates the cost of the path to it through the ‘current‘ node. If this new path is better than any previously recorded path to that neighbor (Line 18), the ‘came_from‘, ‘g_score‘, and ‘f_score‘ maps are updated with this new, better path, and the neighbor is added to the ‘open_set‘ for future consideration. If the ‘open_set‘ becomes empty before the goal is reached, it means no path exists.

5.2 Rapidly-exploring Random Tree (RRT)

RRT is a sampling-based algorithm designed for efficient pathfinding in complex, high-dimensional spaces. Unlike grid-based searchers, RRT builds a tree of reachable states by randomly sampling the configuration space. The algorithm proceeds as follows:

1. **Sample:** Generate a random state q_{rand} in the configuration space.
2. **Find Nearest:** Find the node in the existing tree, q_{near}, that is closest to q_{rand}.
3. **Steer:** Generate a new state, q_{new}, by taking a step of a fixed size ϵ from q_{near} in the direction of q_{rand}. The steer function can be defined as:

$$q_{\text{new}} = q_{\text{near}} + \frac{q_{\text{rand}} - q_{\text{near}}}{\|q_{\text{rand}} - q_{\text{near}}\|}\epsilon \tag{5.2}$$

4. **Check Collision:** If the path from q_{near} to q_{new} is collision-free, add q_{new} to the tree as a child of q_{near}.

The algorithm repeats until a node is added that is within a certain tolerance of the goal. While not optimal, RRT is probabilistically complete. In our implementation, a visualization of the explored tree is also published for analysis.

Algorithm 3 RRT Algorithm (Pseudo Code)

```
1: function RRT(start, goal, max_iterations)
2:     tree ← initialize_tree_with(start)
3:     for $i$ from 1 to max_iterations do
4:         $q_{rand}$ ← sample_random_point()
5:         $q_{near}$ ← find_nearest_node_in_tree(tree, $q_{rand}$)
6:         $q_{new}$ ← steer($q_{near}$, $q_{rand}$, step_size)
7:         if not is_collision($q_{near}$, $q_{new}$) then
8:             add_node_and_edge(tree, $q_{near}$, $q_{new}$)
9:             if distance($q_{new}$, goal) < goal_threshold then
10:                return reconstruct_path_from_tree(tree, $q_{new}$)
11:            end if
12:        end if
13:    end for
14:    return failure                                    ▷ No path found
15: end function
```

Pseudo-code Explanation

The RRT algorithm begins by initializing a tree with the start node (Line 2). The main loop then runs for a specified maximum number of iterations or until a path is found (Line 3). Inside the loop, a random point is sampled from the valid map area (Line 4), and the nearest node in the tree to this random point is found (Line 5). The ‘steer‘ function then generates a new candidate node by taking a fixed-size step from the nearest node toward the random point (Line 6). A collision check is performed to ensure the straight-line path to this new node is clear (Line 8); if it is, the new node is added to the tree (Line 9). After a node is successfully added, the algorithm checks if this new node is within a given tolerance of the goal (Line 10). If it is, the path is reconstructed by tracing back from the goal node to the start through its parents in the tree, and the search successfully terminates (Line 11).

5.3 Dijkstra’s Algorithm

Dijkstra’s algorithm is a classic graph search algorithm that finds the shortest path from a single source node to all other nodes in a graph with non-negative edge weights. It is an uninformed search algorithm, meaning it does not use a heuristic to guide its search. It can be seen as a special case of A* where the heuristic is always zero, $h(n) = 0$. The evaluation function is simply:

$$f(n) = g(n) \tag{5.3}$$

Dijkstra’s explores the state space by expanding outward from the start node, always visiting the un-visited node with the lowest accumulated cost $g(n)$. Because it explores uniformly in all directions, it is often less efficient than A* for point-to-point pathfinding but is guaranteed to find the shortest path. Its implementation in our system is nearly identical to A*, operating on the same 8-connected grid structure.

Algorithm 4 Dijkstra's Algorithm (Pseudo Code)

```
 1: function DIJKSTRA(start, goal, map)
 2:     dist ← { node : $\infty$  for all nodes in map}
 3:     prev ← { node : null  for all nodes in map}
 4:     dist[start] ← 0
 5:     $Q$ ← PriorityQueue(all nodes in map)
 6:     while $Q$ is not empty do
 7:         $u$ ← $Q$.extract_min_dist()
 8:         if $u$ = goal then
 9:             return reconstruct_path(prev, u)
10:         end if
11:         for each neighbor $v$ of $u$ do
12:             alt ← dist[u] + length(u, v)
13:             if alt < dist[v] then
14:                 dist[v] ← alt
15:                 prev[v] ← u
16:                 $Q$.decrease_priority(v, alt)
17:             end if
18:         end for
19:     end while
20:     return failure                                  ▷ No path found
21: end function
```

Pseudo-code Explanation

Dijkstra's algorithm initializes by setting the distance to all nodes to infinity except for the start node, which is set to 0 (Lines 2-4). A priority queue is created containing all nodes, prioritized by their distance value (Line 5). The main loop (Line 7) runs as long as there are nodes in the priority queue. In each step, the node 'u' with the smallest distance is extracted (Line 8). If 'u' is the goal, the path is found (Lines 9-11). If not, the algorithm considers each neighbor 'v' of 'u' (Line 12). It calculates the distance to 'v' through 'u' (Line 13). If this new path is shorter than the previously known distance to 'v' (Line 14), the distance and predecessor maps are updated (Lines 15-16), and the priority of 'v' in the queue is updated to reflect this shorter path (Line 17). This process, known as relaxation, is repeated until the goal is extracted from the queue.

Chapter 6

REALISTIC DESIGN CONSTRAINTS

The development and implementation of this project were subject to several realistic constraints that influenced the design choices and system architecture. These constraints span technical, environmental, and logistical domains.

- **Technical Constraints:**
  - **Computational Power:** The entire SLAM and navigation stack runs on an onboard Raspberry Pi 4. This limitation necessitates the use of computationally efficient algorithms. EKF-SLAM was chosen over more resource-intensive methods like particle filters or full graph-based optimization due to its lower memory footprint and predictable, real-time update steps.
  - **Sensor Accuracy and Noise:** The LDS-01 LiDAR sensor, while effective, has inherent limitations in range and precision, and is susceptible to noise from reflective or transparent surfaces. Similarly, wheel odometry is prone to drift from wheel slippage, uneven floors, and calibration errors. The probabilistic nature of the EKF is designed specifically to handle and mitigate such noise.
  - **Power Consumption:** The robot's operational time is limited by its battery capacity. All algorithms must be efficient not only in computation but also in power usage to maximize the duration of experiments.
- **Environmental Constraints:**
  - **Feature Availability:** EKF-SLAM relies on the consistent re-observation of distinct landmarks. The system's performance is therefore dependent on the structure of the environment. In feature-poor environments (e.g., long, empty corridors), the filter's ability to correct for drift is diminished.

  - **Dynamic Obstacles:** The current implementation assumes a static environment. The presence of unmodeled dynamic obstacles (e.g., people walking) can introduce spurious landmark detections or corrupt laser scans, which can negatively impact the map and localization accuracy.

- **Logistical and Financial Constraints:**
  - **Budget:** The project was developed with a limited budget, which influenced the choice of the cost-effective TurtleBot3 platform. This constraint precludes the use of higher-precision, more expensive sensors that could potentially improve performance.
  - **Development Time:** The project timeline was constrained by the academic semester, requiring a focus on implementing and validating a core set of reliable algorithms rather than exploring a wider range of more complex, experimental techniques.

Chapter 7

RESULTS AND DISCUSSIONS

This section is reserved for the presentation and analysis of the experimental results obtained from both simulation and real-world tests.

7.1 Simulation Results

Simulations were conducted in a controlled environment to validate the core algorithms and provide a baseline for performance. *When inserting your results, discuss how the EKF-SLAM trajectory closely follows the ground truth, especially in areas where the odometry-only path shows significant drift. Analyze the final landmark map for accuracy, noting the final estimated positions and their covariance ellipses.*

7.2 Real-World Experimental Results

The system was deployed on a physical TurtleBot3 platform in a laboratory environment with physical cylindrical landmarks. *When presenting your results, discuss the challenges encountered, such as non-ideal sensor data or unexpected wheel slip. Analyze the final map and trajectory, highlighting how the SLAM filter successfully built a coherent map despite these real-world imperfections.*

7.3 Performance Benchmarks

To quantitatively evaluate the system, several metrics were recorded.

- **Localization Accuracy:** The Absolute Trajectory Error (ATE) was used to measure the difference between the estimated path and the ground truth path in simulation. *When presenting your ATE table, comment on the magnitude of the improvement provided by SLAM over pure odometry.*

- **Path Planner Performance:** The three path planning algorithms were compared on a representative map generated by the SLAM system. *In your analysis, discuss the trade-offs between the algorithms. Note that A* should provide the shortest path with less exploration than Dijkstra's, while RRT may find a longer, less smooth path more quickly, especially in open spaces.*

Chapter 8

CONCLUSION

This project successfully demonstrated a complete EKF-SLAM based navigation stack for an indoor mobile robot. The system effectively integrates wheel odometry and laser scan data to build both a sparse landmark map and a dense occupancy grid. The results from simulation and real-world experiments show a significant improvement in localization accuracy when using EKF-SLAM compared to relying on pure wheel odometry, successfully correcting for drift over long trajectories. The generated maps proved suitable for use with classical path planning algorithms—A*, RRT, and Dijkstra—enabling the robot to navigate autonomously from a start to a goal location.

Future work could explore more advanced data association techniques to improve robustness in cluttered environments, such as implementing a Joint Compatibility Branch and Bound (JCBB) algorithm. Additionally, integrating other sensor modalities like an IMU could improve the motion model's accuracy. Finally, transitioning from EKF-SLAM to a modern graph-based SLAM system using frameworks like g2o or GTSAM could offer better scalability and consistency for larger, more complex environments.

Appendix A

LIST OF SYMBOLS AND ABBREVIATIONS

A.1 Symbols

Table A.1: List of Symbols

Symbol	Description
x_k	State vector at time k
$\hat{x}_k$	Estimated state vector at time k
ξ_t	Augmented state vector (robot pose + map)
μ	Mean of a Gaussian distribution
Σ	Covariance matrix of a Gaussian distribution
u_k	Control vector (body twist) at time k
z_k	Measurement vector at time k
$g(\cdot)$	Nonlinear state transition function (motion model)
$h(\cdot)$	Nonlinear measurement function (sensor model)
A_k	Jacobian of the motion model w.r.t. state
H_k	Jacobian of the measurement model w.r.t. state
Q	Process noise covariance
R	Measurement noise covariance
K_k	Kalman Gain at time k
T_{ab}	Homogeneous transformation from frame b to a
V_b	Body twist of the robot
ϕ_l, ϕ_r	Left and right wheel angles

A.2 Abbreviations

Table A.2: List of Abbreviations

Abbreviation	**Description**
SLAM	Simultaneous Localization and Mapping
EKF	Extended Kalman Filter
ROS	Robot Operating System
IMU	Inertial Measurement Unit
RRT	Rapidly-exploring Random Tree
ATE	Absolute Trajectory Error
PDF	Probability Density Function

Bibliography