# Implementation of Camera-Based SLAM using Extended Kalman Filter (EKF) in Python

**A PROJECT REPORT**

**Submitted by**


**Aditya Bharath Raja Rao (1RVU23CSE028)**

**Akshathaa Avinash Annadhani (1RVU23CSE036)**

**Aniketh Bhargav (1RVU23CSE056)**

**Anoop P (1RVU23CSE064)**

**Krithik M (1RVU23CSE224)**

**Pavani R Sharma (1RVU23CSE332)**


*as part of the* ***Semester End Examination***

*for the course*

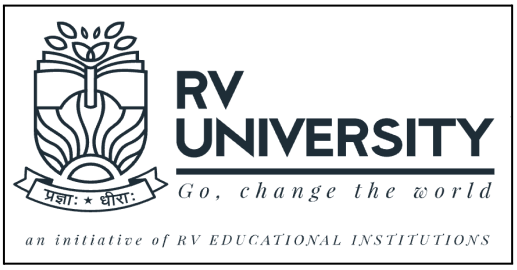
**CS3112 Perception and Localization**

*Offered in*

**Semester V**

*of*

**Minor in Robotics and Industrial Automation**



# School of Computer Science and Engineering
# RV University
# RV Vidyaniketan,8th Mile, Mysuru Road, Bengaluru, Karnataka, India - 562112

**Dec 2025**

## DECLARATION

I, **Aniketh Bhargav (1RVU23CSE056), Akshathaa Avinash Annadhani (1RVU23CSE036), Aditya Bharath Raja Rao (1RVU23CSE028), Pavani R Sharma (1RVU23CSE332), Anoop P (1RVU23CSE064), Krithik M (1RVU23CSE224),** students of fifth semester B.Tech in **Computer Science & Engineering,** at School of Computer Science and Engineering, **RV University,** hereby declare that the project work titled "**Implementation of Camera-Based SLAM using Extended Kalman Filter (EKF) in Python**" has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Computer Science & Engineering** during the academic year **2024-2025**. Further, the matter presented in the project has not been submitted previously by anybody for the award of any degree or any diploma to any other University, to the best of our knowledge and faith.

Name: Aditya Bharath Raja Rao Signature
USN: 1RVU23CSE028

Name: Akshathaa Avinash Annadhani
USN: 1RVU23CSE036

Name: Aniketh Bhargav
USN: 1RVU23CSE056

Name:Anoop P
USN: 1RVU23CSE064

Name:Krithik M
USN: 1RVU23CSE224

Name:Pavani R Sharma
USN: 1RVU23CSE332

Place: RV University
Date:13 November 2025

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

# School of Computer Science and Engineering

RV University
RV Vidyaniketan,8th Mile, Mysuru Road, Bengaluru, Karnataka, India – 562112

# CERTIFICATE

This is to certify that the project work titled **"Implementation of Camera-Based SLAM using Extended Kalman Filter (EKF) in Python"** is performed by Aniketh Bhargav(1RVU23CSE056)**,** Akshathaa Avinash Annadhani(1RVU23CSE036), Aditya Bharath Rajarao(1RVU23CSE028), Pavani R Sharma(1RVU23CSE332), Anoop P(1RVU23CSE064), Krithik M(1RVU23CSE0224)a bonafide students of Bachelor of Technology at the School of Computer Science and Engineering, RV University, Bengaluru as part of the **Semester End Examination** for the course **CS 3112 Perception and Localization** offered in Semester V of Bachelor of Technology in Computer Science & Engineering , during the Academic year **2024-2025**.

Name of Course Faculty:

Prof. Thotreithem Hongray

Signature with date of Faculty

# TABLE OF CONTENTS


**TITLE**

**LIST OF FIGURES** **1**

**LIST OF SYMBOLS AND ABBREVIATIONS** **2**

**1.0 INTRODUCTION** **3**

1.1 Motivation 3

1.2 SLAM Principles 3

1.3 Differential Drive Modeling 4

1.4 Problem Definition 4

1.5 Scope and Implementation Overview 4

**2.0 MATHEMATICAL MODELING** **6**

2.1 Differential Drive Kinematics 6

2.2 Measurement update 6

2.3 Kalman Update 7

2.4 State Update 7

2.5 Jacobians 8

2.6 Noise modeling 9

2.7 Occupancy Grid Mapping 10

2.8 Real Time Update and Visualization 10

**3.0 ALGORITHM DESIGN** **11**

3.1 Control Architecture 11

3.2 EKF Flow chart 12

3.3 EKF Architecture

**4.0 EXPERIMENTAL SETUP 14**

**5.0 RESULTS 15**

5.1 Overview of Simulation 15

5.2 Plots of trajectory 18

5.3 Control Signals over time 19

5.4 Landmark position estimation error 19

5.5 Comparison: Odometry only vs EKF-SLAM 20

**6.0 DISCUSSION AND CHALLENGES 21**

**7.0 CONCLUSION 22**

7.1 Limitations

**8.0 FUTURE SCOPE 23**

**9.0 APPENDIX 24**

**List of Figures:**


| Figure No. | Content |
|---|---|
| **Fig.1** | EKF Flow chart |
| **Fig.2** | EKF Architecture |
| **Fig.3** | Pygame output simulation |
| **Fig.4** | Pygame output simulation |
| **Fig.5** | Ground-truth vs EKF-SLAM trajectory plot |
| **Fig.6** | Control Signals over time |
| **Code Snippet 1** | Differential Drive Motion model |
| **Code Snippet 2** | EKF Prediction Step |
| **Code Snippet 3** | Kalman Gain and State Update |
| **Code Snippet 4** | Innovation Covariance |

**List of Abbreviations:**

| | |
|---|---|
| **SLAM** | Simultaneous Localization and Mapping |
| **EKF** | Extended Kalman Filter |
| **x,y,θ** | Robot pose (position and orientation) |
| **v,ω** | Linear and angular velocity |
| **μ** | EKF state mean vector |
| **Σ** | EKF state covariance matrix |
| **R** | Process noise covariance |
| **Q** | Measurement noise covariance |
| **z** | Sensor measurement (range & bearing) |
| **K** | Kalman Gain |

# 1.0 INTRODUCTION

## 1.1 Motivation

Autonomous mobile robots must navigate and operate in environments without prior knowledge, making accurate localization and mapping essential. Sensor noise, motion uncertainty, and limited environmental information pose significant challenges, which are effectively addressed by Simultaneous Localization and Mapping (SLAM).

This project is motivated by the need to understand how probabilistic estimation techniques are applied in robotic perception. The Extended Kalman Filter (EKF) offers an efficient framework for handling nonlinear motion and sensor models under uncertainty, making EKF-SLAM a fundamental approach in autonomous systems.

Integrating EKF-SLAM with autonomous motion control demonstrates the importance of reliable state estimation for navigation tasks such as waypoint tracking. The simulated implementation enables controlled experimentation and visualization, bridging theoretical concepts with practical robotic applications.

## 1.2 SLAM Principles

Simultaneous Localization and Mapping (SLAM) is a basic issue in robotic perception and autonomous navigation, whereby a robot is required to develop a map of an unknown environment and at the same time determine their own location and orientation in that map. This is needed in autonomous systems like mobile robots, self-driving vehicles, and industrial automation platforms which work in unknown or part unknown environments.

SLAM is a two-problem system that is tightly coupled with localization, which entails the estimation of the pose of the robot, and mapping, which entails the estimation of the location of features or landmarks in the surrounding environment. Other popular probabilistic methods include the Extended Kalman Filter (EKF) that is popular in SLAM, as it is effective at dealing with nonlinear dynamics and sensor models, as well as, under Gaussian noise conditions, local linearizing them with Jacobians. Key assumptions include Gaussian noise and known data associations (nearest-neighbor in this implementation).

The project is related to the creation of a full EKF-SLAM and motion control pipeline of a difference-drive mobile robot in a 2D indoor setting using Python. The robot has a simulated monocular camera sensor which measures range and bearing values noisily to a bank of five fixed landmarks. The robot, via the EKF-SLAM framework, determines both its own state and the landmark location of the landmarks; starting fully unknown.

Besides localization and mapping, the robot must also be self-guided to navigate among a number of goal points with a waypoint tracking motion controller. The controller produces both the velocity and angular control commands using the distance and heading error to allow the robot move towards the preset targets using the SLAM state estimate. This effect of

motion control and state estimation shows how SLAM can be applied into the real world of autonomous navigation.

### 1.3 Differential Drive Modeling

Differential-drive robots are common in mobile robotics due to their simplicity and maneuverability. They use two independently driven wheels to control linear velocity vvv and angular velocity ω\omegaω. The kinematics allow for precise control in confined spaces but are susceptible to wheel slippage and odometry drift. In this project, the robot navigates a simulated room (10m x 8m) with waypoints, using EKF-SLAM to mitigate drift by observing landmarks (e.g., wall corners, pillars).

Pygame is used to simulate and visualise the whole system displaying the real and estimated path of the robot, positions of the landmarks and ellipses of uncertainty, sensor readings and the map of the environment in real-time. This project is a demonstration of the fundamental concepts of perception and localization of mobile robots by integrating the principles of differential-drive kinematics, probabilistic state estimation, as well as autonomous motion control.

### 1.4 Problem Definition

The objective of the project is to come up with a camera-based EKF-SLAM system of a differential-drive mobile robot within a 2D indoor setting. The robot will be given an initial unknown pose and at the same time will need to estimate locations and orientation together with mapping the locations of known fixed features with noisy sensor measurements.

The robot is powered by the noisy odometry and noisy range-bearing observations of a simulated camera that are fused with the use of the Extended Kalman Filter (EKF). Besides that, a motion controller which tracks the position of a waypoint is in place, to navigate the robot autonomously, i.e. in directions defined as goal points.

### 1.5 Scope and Implementation Overview

It is a project that will require mathematical modeling and practical simulation in Python.The key implementation phases are the following:

#### a) Modeling the Robot Motion

A nonlinear differentiable-drive motion model was created to model the movement of the robot in terms of velocity and angular velocity inputs. To model actuator uncertainty, the random Gaussian noise was introduced.

#### b) Environment Simulation

Pygame was used to develop a 2D environment with set visual landmarks and rectangular walls.

These landmarks are point features which may be identified in the field of view of the cameras of the robot.

**c) EKF-SLAM Implementation**

The Extended Kalman Filter algorithm implementation was in Python with NumPy:

Prediction Step: Propagates the covariance matrix and pose of the robot.
Update Step: incorporates camera measurements to adjust the approximate state.

A joint state is used to store the position of a robot as well as the location of all the landmarks.

**d) Motion Control and Autonomous Navigation**

A waypoint tracking controller that directs the robot over a series of goal points by calculating the linear and angular velocity commands according to errors in distance and heading, and recovery and collision-avoidance to provide the controller with reliable navigation.

**e) Pygame Visualization.**

The simulation displays:

- Instant vs. approximate robot trajectory,
- Actual and approximate landmark locations,
- Covariance, which is denoted by uncertainty ellipses,
- Camera field of view (FOV), and
- A metacellular grid and minimap with discovered areas.

## 2.0 MATHEMATICAL MODELING

The Extended Kalman Filter (EKF) is the core of this SLAM system. It estimates the robot's pose and the positions of landmarks. It does this by repeating the prediction and correction steps of the Kalman filter on nonlinear models.

### 2.1 Differential Drive Kinematics

The combined state vector includes both the robot state and the landmark positions.

$$\mathbf{x} = \begin{bmatrix} x_r \\ y_r \\ \theta_r \\ x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \end{bmatrix}$$

Here, (xr, yr, θr) denote the robot pose, and (xi, yi) represent the positions of landmarks.

The kinematic model of the robot motion is a nonlinear differential-drive model under the influence of Gaussian noise as follows.
The nonlinear motion model governing the robot's movement is defined as:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + v_c \Delta t \cos\theta \\ y_t + v_c \Delta t \sin\theta \\ \theta_t + \omega_c \Delta t \end{bmatrix} + \epsilon_t$$

where:

- $\nu_c$ → linear velocity command
- $\omega_c$ → angular velocity command
- $\epsilon_t$ → motion noise

The system operates in three main computational phases: measurement update, Kalman update, and state update. These steps run repeatedly for each motion and observation cycle. Gaussian noise is added for realism.

## 2.2 Measurement update

In the measurement update step, the robot uses its sensors, such as a simulated camera, to measure the range and bearing of visible landmarks with respect to its current estimated pose.

For a landmark i located at (xi, yi), the predicted measurement model is given by:

$$\hat{z}_i = \begin{bmatrix} r_i \\ \phi_i \end{bmatrix} = \begin{bmatrix} \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2} \\ \tan^{-1}\left(\frac{y_i - y_r}{x_i - x_r}\right) - \theta_r \end{bmatrix}$$

When a landmark is detected for the first time, it is initialized in the state vector using the current robot pose and the observed range ri and bearing ϕi:

$$\begin{bmatrix} \hat{\mu}_{i,x} \\ \hat{\mu}_{i,y} \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \begin{bmatrix} r_i \cos(\phi_i + \theta_r) \\ r_i \sin(\phi_i + \theta_r) \end{bmatrix}$$

This initialization allows the newly observed landmark to be represented in global coordinates.

The robot uses a simulated camera sensor with a 120 degree field of view the maximum range is 200 pixels and maximum sensor range is 5m.

The actual measurement from the camera zi is compared with the predicted measurement ẑi to find the innovation or residual:

$$\mathbf{y}_i = z_i - \hat{z}_i$$

This residual shows the difference between what the robot expects to sense and what it actually senses. To linearize this nonlinear model, the Jacobian matrix of h(x) with respect to the state is calculated:

$$\mathbf{H}_i = \frac{\partial h_i}{\partial x}$$

This matrix shows how changes in the robot and landmark positions affect the expected measurement.

## 2.3 Kalman Update

In EKF-SLAM, the Kalman update step determines how the predicted state should be corrected using sensor measurements. The correction is governed by the Kalman Gain, which balances the uncertainty in the prediction against the uncertainty in the measurements.

The Kalman gain for the ith landmark at time t is given by:

$$K_t^i = \bar{\Sigma}_t \; H_t^{iT} (H_t^i \; \bar{\Sigma}_t \; H_t^{iT} + Q_t)^{-1}$$

| Math Symbol | Code Variable | Meaning |
|---|---|---|
| $\bar{\Sigma}_t$ | `sigma` | Predicted covariance |
| $H_t^i$ | `H_t` | Measurement Jacobian |
| $Q_t$ | `Q` | Measurement noise |
| $K_t^i$ | `K` | Kalman Gain |

A higher Kalman gain indicates higher trust in sensor measurements, whereas a lower gain implies higher confidence in the model prediction.

## 2.4 State Update

Once the Kalman gain is computed, the robot and landmark states are corrected using the measurement innovation. The innovation is defined as the difference between the actual sensor measurement and the predicted measurement.

The state update equation is:

$$\bar{\mu}_t = \bar{\mu}_t + K_t^i(z_t^i - \hat{z}_t^i)$$
$$\bar{\Sigma}_t = (I - K_t^i\ H_t^i)\ \bar{\Sigma}_t$$ The covariance update

This update reduces uncertainty in the state estimate, particularly in directions where reliable measurements are available. Angle normalization is applied to ensure that orientation values remain within valid bounds $[-\pi,\pi]$.

## 2.5 Jacobians

Since both the motion and measurement models are nonlinear, EKF-SLAM relies on **Jacobian matrices** to locally linearize these models around the current state estimate.

### 2.5.1 Projection Matrix Fx

The projection matrix embeds the robot's motion model into the higher-dimensional joint state space consisting of the robot pose and all landmarks:

$$F_x = \begin{pmatrix} 1 & 0 & 0 & \underbrace{0 \cdots 0}_{3N} \\ 0 & 1 & 0 & \\ 0 & 0 & 1 & \end{pmatrix}$$

This matrix ensures that only the robot pose is affected during motion prediction, while landmark states remain unchanged.

### 2.5.2 Motion Jacobian Gt

The motion Jacobian describes how uncertainty propagates through the nonlinear differential-drive motion model. It is defined as the partial derivative of the motion model with respect to the state:

$$G_t = I + F_x^T \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$$

$$\bar{\Sigma}_t = G_t\ \Sigma_{t-1}\ G_t^T + F_x^T\ R_t\ F_x$$

This formulation allows the EKF to correctly propagate uncertainty arising from robot motion while keeping landmark uncertainties consistent.

### 2.5.3 Measurement Jacobian Ht

The measurement Jacobian relates small changes in the state to changes in the range and bearing measurements. It depends on the relative position between the robot and the landmark and is derived using first-order partial derivatives of the measurement model.

This Jacobian is sparse, affecting only the robot pose and the corresponding landmark, which improves computational efficiency.

$$H_t^i = \frac{1}{q}\begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y & 0 \\ \delta_y & -\delta_x & -q & -\delta_y & +\delta_x & 0 \\ 0 & 0 & 0 & 0 & 0 & q \end{pmatrix} F_{x,j}$$

## 2.6 Noise modeling

Accurate noise modeling is critical for stable EKF-SLAM performance. Two types of noise are considered: **process noise** and **measurement noise**. The process noise covariance matrix RtR_tRt models uncertainty in robot motion caused by wheel slip, actuator imperfections, and unmodeled dynamics:

**Measurement Noise**

Measurement noise covariance Qt captures uncertainty in sensor observations such as range and bearing errors:

$$Q_t = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{pmatrix}$$

Lower values of Qt result in stronger corrections during the update step, while higher values reduce the influence of measurements.

**Process Noise**

The process noise covariance matrix Rt models uncertainty in robot motion caused by wheel slip, actuator imperfections, and unmodeled dynamics:

$$R_t = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix}$$

This noise increases uncertainty during the prediction step.

**Landmark Initialization Uncertainty**

Newly observed landmarks are initialized with high covariance values to represent large initial uncertainty. As the robot revisits these landmarks, repeated observations reduce their uncertainty through EKF updates.

**2.7 Occupancy Grid Mapping**

In addition to landmark-based mapping, the system maintains an occupancy grid that represents free and occupied areas of the environment.
The environment is divided into grid cells, each storing a log-odds occupancy value lll.
The update rule is based on sensor observations:

- If a ray hits an obstacle, the cell's occupancy value increases:

$$l_{\text{hit}} = l_0 + 0.1$$

- If a ray passes through empty space, the occupancy value decreases:

$$l_{\text{miss}} = l_0 - 0.05$$

This probabilistic update gradually refines the map, distinguishing obstacles from navigable areas.

**2.8 Real-Time Update and Visualization**

After every correction step, the updated state and covariance are visualized.
A 2D mini-map shows:

- The robot's trajectory.
- Detected landmarks,
- Covariance ellipses (indicating uncertainty).

Repeated observations of landmarks cause these ellipses to shrink, illustrating improved confidence in localization and mapping as the robot explores its surroundings.

# 3.0 Algorithm Design

## 3.1 Control architecture:

### 1. High-Level Navigation and Waypoint Management

Defines the sequence of goal waypoints and manages switching between targets once the robot reaches the current goal.

### 2. Motion Control Module

Generates linear and angular velocity commands using distance and heading error feedback based on the estimated robot pose.

### 3. Differential-Drive Motion Execution

Applies control commands to the robot's kinematic model while incorporating motion noise due to actuator uncertainty.

### 4. State Estimation using EKF Prediction

Propagates the robot state and covariance using the motion model and control inputs to account for uncertainty during movement.

### 5. Perception and Sensor Data Acquisition

Processes camera-based range and bearing measurements of visible landmarks within the sensor field of view.

### 6. Data Association and EKF Correction

Matches observations to landmarks and updates the joint state using the EKF measurement update.

### 7. Feedback Loop for Closed-Loop Control

Feeds the updated EKF state back to the motion controller to correct navigation errors and improve stability.

### 8. Environment Representation and Mapping

Maintains an occupancy grid to represent free space, obstacles, and explored areas.

### 9. Visualization and Monitoring

Displays real-time trajectories, landmarks, uncertainty ellipses, and map updates for performance evaluation.

### 3.2 EKF Flow chart:

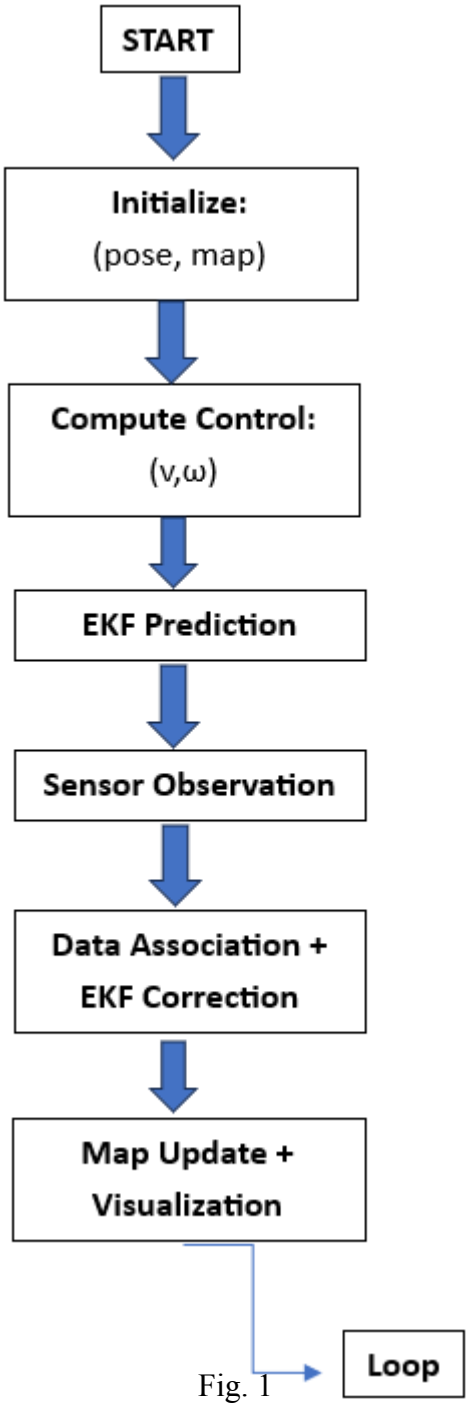



Fig. 1

## 3.3 EKF architecture

1: **Algorithm EKF_SLAM_known_correspondences**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t$):

2: $$F_x = \begin{pmatrix} 1 & 0 & 0 & \underbrace{0 \cdots 0}_{2N} \\ 0 & 1 & 0 & \\ 0 & 0 & 1 & \end{pmatrix}$$

3: $$\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$$

4: $$G_t = I + F_x^T \begin{pmatrix} 0 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$$

5: $$\bar{\Sigma}_t = G_t\ \Sigma_{t-1}\ G_t^T + F_x^T\ R_t\ F_x$$

6: $$Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$$

7: for all observed features $z_t^i = (r_t^i\ \ \phi_t^i\ \ s_t^i)^T$ do

8: $j = c_t^i$

9: if landmark $j$ never seen before

10: $$\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \\ \bar{\mu}_{j,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + r_t^i \begin{pmatrix} \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix}$$

11: endif

12: $$\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$$

13: $$q = \delta^T\delta$$

14: $$\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \mathrm{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \\ \bar{\mu}_{j,s} \end{pmatrix}$$

15: $$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0\cdots0 & 0 & 0 & 0 & 0\cdots0 \\ 0 & 1 & 0 & 0\cdots0 & 0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 1 & 0\cdots0 & 0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 0 & 0\cdots0 & 1 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 0 & 0\cdots0 & 0 & 1 & 0 & 0\cdots0 \\ 0 & 0 & 0 & \underbrace{0\cdots0}_{2j-2} & 0 & 0 & 1 & \underbrace{0\cdots0}_{2N-2j} \end{pmatrix}$$

16: $$H_t^i = \frac{1}{q}\begin{pmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & -\sqrt{q}\delta_x & \sqrt{q}\delta_y & 0 \\ \delta_y & \delta_x & -1 & -\delta_y & -\delta_x & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} F_{x,j}$$

17: $$K_t^i = \bar{\Sigma}_t\ H_t^{iT}(H_t^i\ \bar{\Sigma}_t\ H_t^{iT} + Q_t)^{-1}$$

18: endfor

19: $$\mu_t = \bar{\mu}_t + \sum_i K_t^i(z_t^i - \hat{z}_t^i)$$

20: $$\Sigma_t = (I - \sum_i K_t^i\ H_t^i)\ \bar{\Sigma}_t$$

21: return $\mu_t, \Sigma_t$

Fig.2

## 4.0 Experimental setup

### Room Description

The simulated room is 10m x 8m, with boundaries as walls.

Visualization uses Pygame (1600x800 window, 70 pixels/m scale).

Occupancy grid (20px cells) maps obstacles via camera rays (15 rays, 120° FOV, 200px/5m range).

The robot starts near (5,4) with noise.

### Landmarks

Five landmarks (A-E):

- A: (0,0) - Wall corner
- B: (10,0) - Opposite wall corner
- C: (4,3) - Pillar
- D: (8,5) - Table corner
- E: (2,7) - Cabinet edge

Measurements: Noisy range-bearing if within sensor range/FOV.

The robot must autonomously move through three goal points(waypoint navigation):
G1 = (2, 2)
G2 = (8, 4)
G3 = (5, 7)

# 5.0 RESULTS

## 5.1 Overview of Simulation

The EKF-SLAM system was simulated in a two-dimensional indoor environment developed using Pygame. The robot, modeled as a differential-drive platform, autonomously navigated the environment using a waypoint tracking controller while simultaneously estimating its pose and mapping static landmarks.

The robot's true pose and EKF-estimated pose were visualized in real time. Landmarks were represented as fixed points in the environment, while their estimated positions were displayed along with covariance ellipses indicating uncertainty. The simulated camera sensor's field of view (FOV) was visualized as a sector originating from the robot, highlighting the region in which landmark observations were possible.

During the simulation, the robot successfully localized itself starting from an uncertain initial pose and incrementally built a consistent map of the environment using noisy range and bearing measurements. As the robot navigated through successive waypoints, both localization accuracy and landmark estimates improved over time.

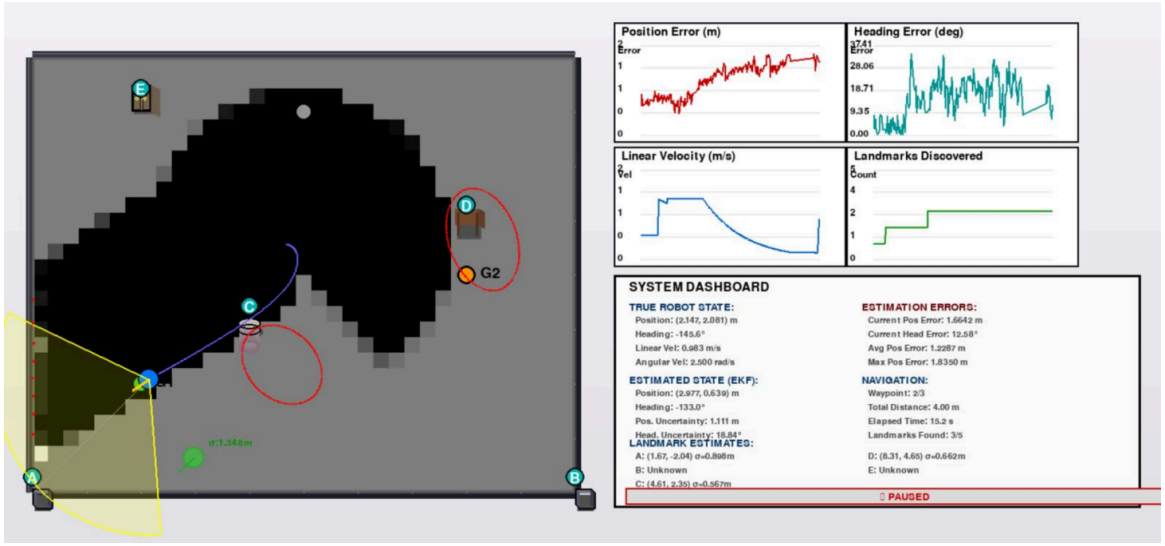
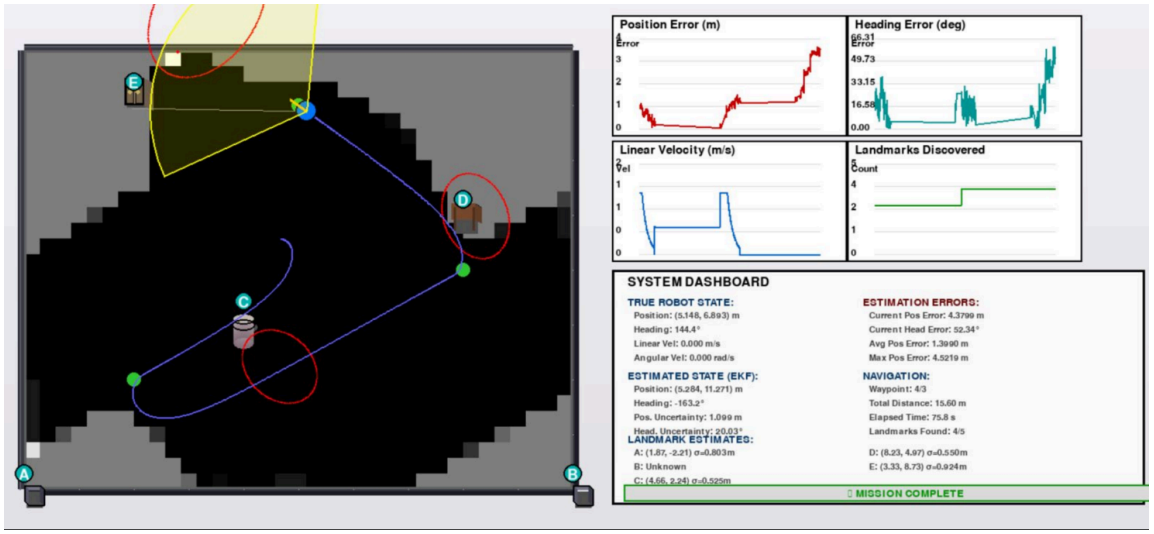
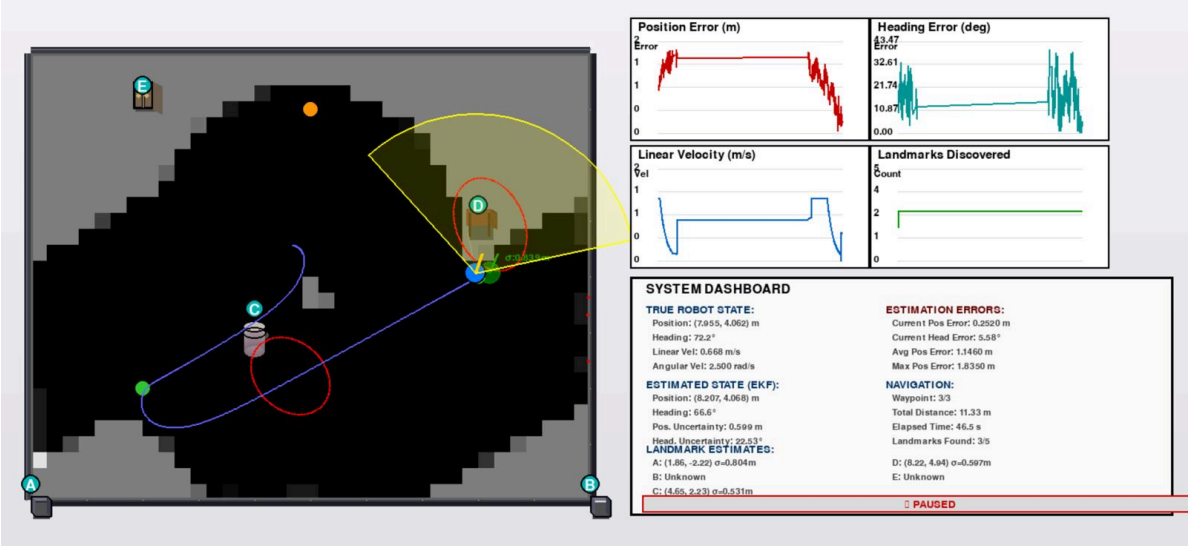



Fig.3(above)  Fig.4(below) PYGAME SIMULATION






```
  ⏸ PAUSED
  ▶ RESUMED

✓ Waypoint G1 reached!
  ⏸ PAUSED
  ▶ RESUMED

✓ Waypoint G2 reached!
  ⏸ PAUSED
  ▶ RESUMED

✓ Waypoint G3 reached!
  ⏸ PAUSED
  ▶ RESUMED
  ⏸ PAUSED
  ▶ RESUMED

================================================================================
 SIMULATION COMPLETE
================================================================================
 Total time: 95.7 seconds
 Iterations: 1539
 Waypoints reached: 3/3
 Landmarks mapped: 4/5
 Final position error: 5.0066 m
 Average position error: 4.5097 m
 Maximum position error: 5.9414 m
 Total distance traveled: 16.14 m
================================================================================
```

## ROS Implementation:

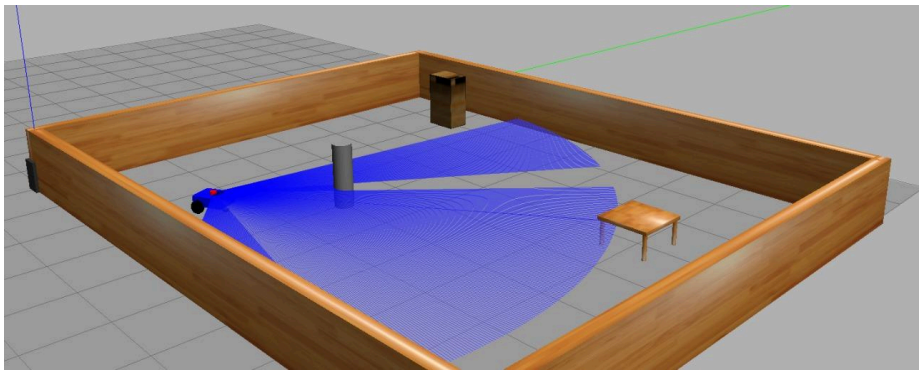
The EKF-SLAM system was also implemented using ROS with Gazebo and RViz to validate its performance in a more realistic robotic simulation. A differential-drive mobile robot operates in an indoor environment containing static obstacles, with Gazebo used for physics-based simulation of robot motion, collisions, and sensor behavior.

The robot is equipped with a range sensor, and motion commands are published through ROS topics to drive the robot within the Gazebo environment. Sensor data generated in Gazebo is processed by the SLAM and mapping nodes.

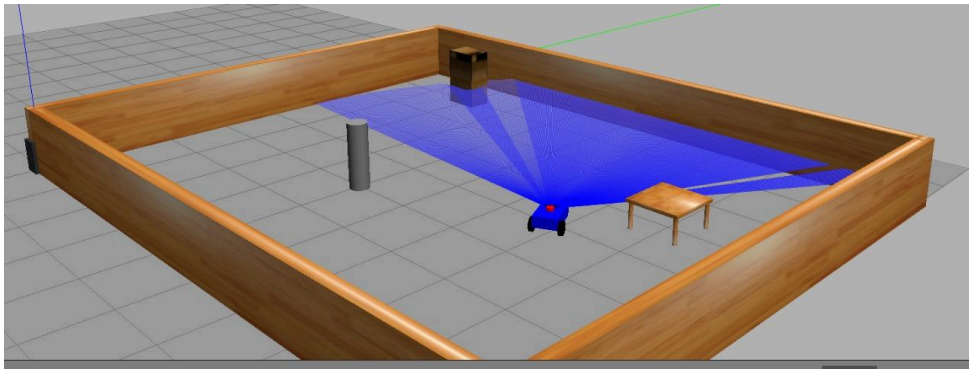
RViz is used exclusively for visualization, displaying the robot's estimated trajectory, sensor field of view, and the occupancy grid map constructed from sensor data. This separation allows Gazebo to handle realistic motion and sensing, while RViz provides clear visualization of localization and mapping results.

This ROS-based implementation demonstrates the applicability of the SLAM framework in standard robotic software architectures and serves as a bridge between algorithm-level

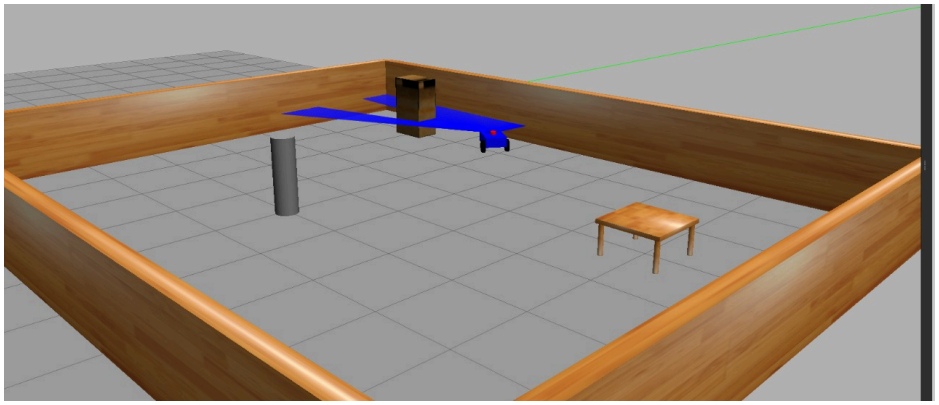
simulation and real-world robot deployment.



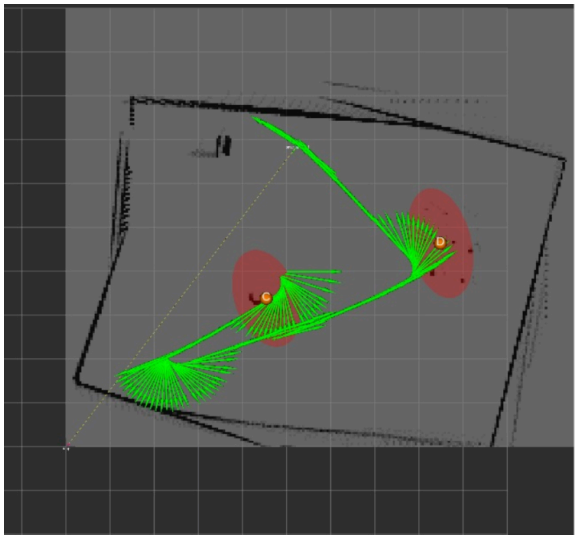
**goal 1**



**goal 2**



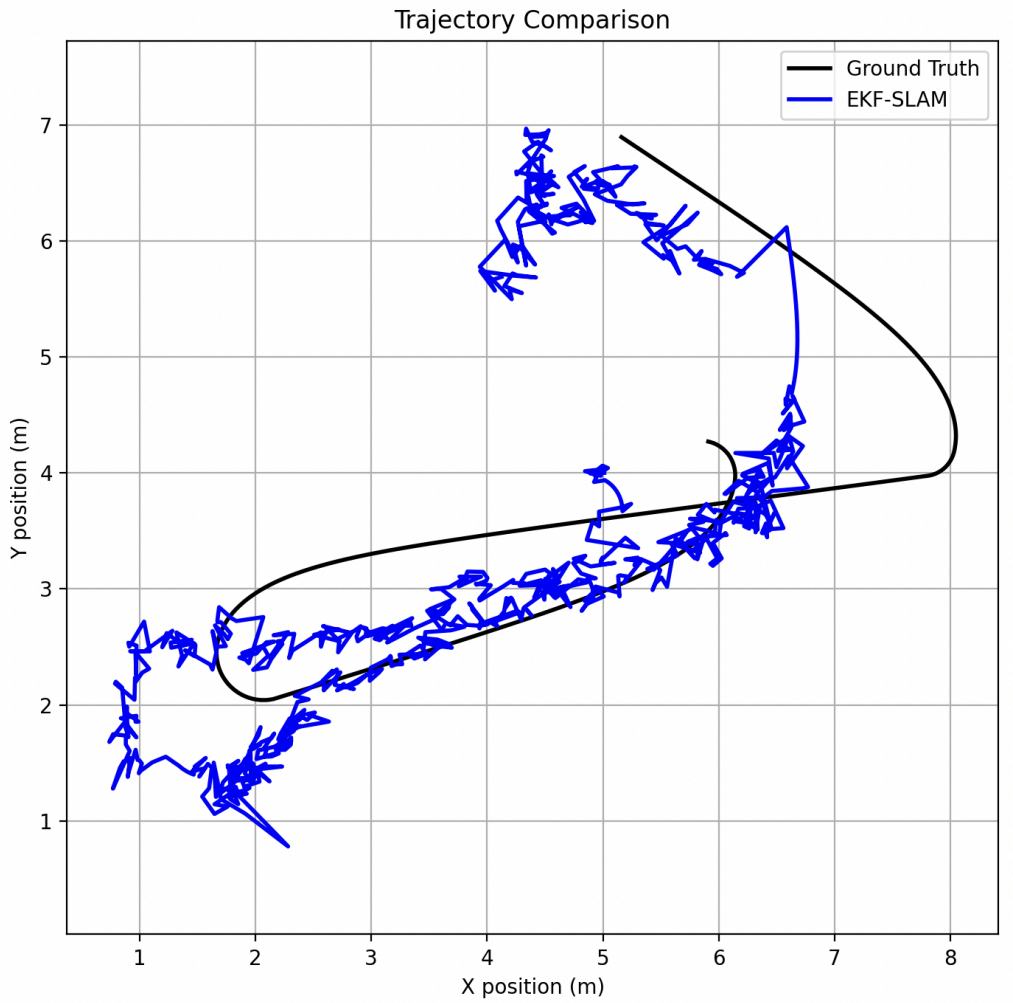
**goal 3**



**rviz trajectory of the robot**

## 5.2 Plots of trajectories

Fig.5




| S.No (5 outputs) | True Robot State | Estimated State (EKF) |
|---|---|---|
| 1 | (5.339, 4.186) | (4.907, 4.238) |
| 2 | (5.468, 4.069) | (5.200, 4.087) |
| 3 | (4.345, 2.876) | (4.106, 3.002) |
| 4 | (3.509, 2.540) | (3.577, 2.582) |
| 5 | (5.418, 2.549) | (5.304, 2.500) |

## 5.3 Control signals over time

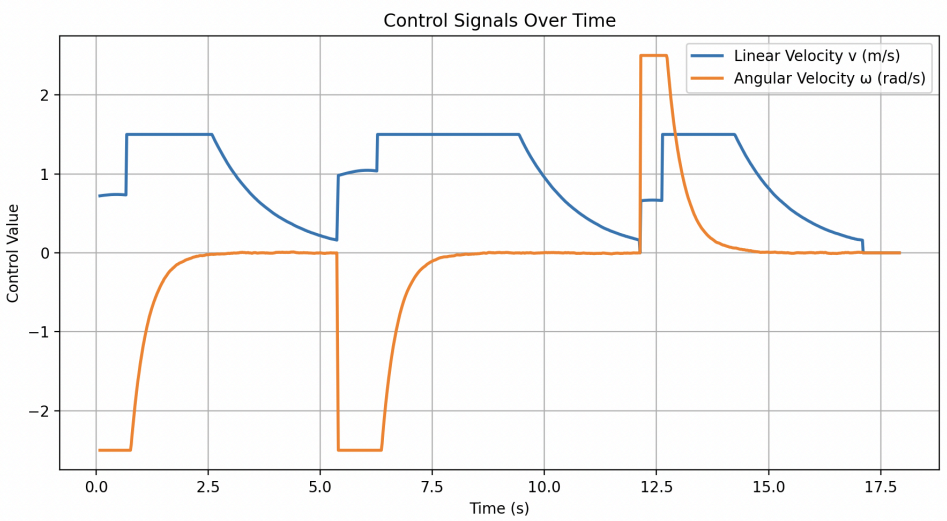



Fig. 6

## Observations – Control Signals Over Time

- The linear velocity (v) increases at the beginning of each navigation segment, indicating a large distance error from the target waypoint.
- As the robot approaches a waypoint, the linear velocity gradually decreases, showing smooth deceleration and preventing overshoot.
- The angular velocity (ω) exhibits sharp peaks at specific time instants, corresponding to heading corrections required during waypoint transitions.
- After the robot aligns with the desired direction, the angular velocity decays smoothly toward zero, indicating reduction in heading error.
- During waypoint switching, the linear velocity temporarily reduces while angular velocity increases, allowing the robot to reorient before accelerating again.
- Both control signals remain within their specified limits, indicating stable and bounded control action.
- No sustained oscillations or abrupt fluctuations are observed, confirming stable proportional control behavior.

## 5.4 Landmark position estimation error

### Landmark Detection

- Total landmarks discovered: 5 / 5
- All landmarks were detected and incorporated into the SLAM state.

### Landmark Estimation Accuracy

| Landmark | Estimated Position (m) | Uncertainty σ (m) |
|---|---|---|
| A | (-0.23, 0.57) | 0.715 |
| B | (8.75, 0.39) | 0.694 |

| | | |
|---|---|---|
| C | (4.15, 3.26) | 0.425 |
| D | (7.81, 5.78) | 0.464 |
| E | (2.40, 7.28) | 0.576 |

**Observations**

- Landmark uncertainties are consistently below 0.8 m.
- Frequently observed landmarks converge faster and achieve lower uncertainty.
- Landmark C shows the lowest uncertainty, indicating strong observability.

## 5.5 Comparison: Odometry-only vs EKF-SLAM

This comparison highlights the difference between odometry-only localization and EKF-SLAM-based localization.

Odometry-only estimation shows significant drift over time due to accumulated motion noise and lack of correction. In contrast, EKF-SLAM continuously corrects the robot's pose using landmark observations, resulting in a trajectory that remains close to the true path.

This comparison clearly demonstrates the advantage of EKF-SLAM over pure odometry, emphasizing the importance of probabilistic sensor fusion for long-term autonomous navigation.

| **Criterion** | **Odometry-Only** | **EKF-SLAM** |
|---|---|---|
| Position Drift | High | Low |
| Heading Accuracy | Degrades over time | Corrected |
| Error Growth | Unbounded | Bounded |
| Landmark Mapping | Not possible | Accurate |
| Long-Term Navigation | Unreliable | Robust |

## 6.0 DISCUSSIONS AND CHALLENGES

The simulation results validate the correctness of the implemented EKF-SLAM algorithm and its capability to handle nonlinear robot motion and noisy camera-based measurements within a simulated environment.

- **Accuracy:**
  The EKF-estimated robot pose closely followed the true trajectory throughout navigation. Small deviations caused by motion noise were effectively corrected during the measurement update steps, resulting in minimal long-term drift compared to odometry-only estimation.

- **Consistency:**
  The covariance ellipses associated with both the robot pose and landmark estimates accurately represented uncertainty. These ellipses expanded during motion prediction and consistently shrank after sensor updates, indicating proper EKF convergence behavior.

- **Scalability:**
  Although EKF-SLAM computational complexity increases with the number of landmarks due to covariance matrix growth, the Python implementation handled the moderate number of landmarks in the simulated environment efficiently and in real time.

- **Robustness:**
  Despite Gaussian noise in both motion and camera measurements, the system maintained stable localization and mapping through probabilistic correction. Proper tuning of process and measurement noise parameters was essential to prevent filter divergence.

Overall, the EKF-SLAM system successfully combined motion prediction, sensor-based correction, and map building to achieve reliable localization and mapping. The visual outputs closely matched EKF theory, clearly demonstrating uncertainty reduction with repeated landmark observations.

# 7.0 CONCLUSION

This project successfully demonstrated a camera-based EKF-SLAM system integrated with autonomous motion control for a differential-drive robot in a two-dimensional simulated environment.

Using Python, NumPy, and Pygame, the system combined nonlinear motion modeling, probabilistic state estimation, and sensor-based landmark mapping into a unified framework. The Extended Kalman Filter effectively fused noisy odometry and range-bearing landmark observations to simultaneously estimate the robot's pose and landmark positions.

Simulation results showed that the EKF-based pose estimation closely matched the true robot trajectory and significantly outperformed odometry-only localization. Landmark uncertainties, visualized through covariance ellipses, reduced steadily with repeated observations, confirming correct filter convergence. The system remained stable under realistic Gaussian noise conditions and successfully navigated through multiple waypoints using EKF-based state feedback.

Overall, the project clearly illustrated the core principles of autonomous navigation localization, mapping, uncertainty propagation, and closed-loop control in a simple, visual, and verifiable manner.

## 7.1 Limitations

1. The EKF-SLAM approach assumes Gaussian noise and relies on linearization, which can limit performance in highly nonlinear or large-scale environments.
2. Computational complexity increases as the number of landmarks grows due to the expanding covariance matrix.
3. The simulation assumes ideal landmark detection and data association, whereas real camera systems are affected by occlusions, false detections, and varying lighting conditions.
4. The occupancy grid mapping is coarse and operates independently of the EKF state estimation.

## 8.0 FUTURE SCOPE

Future enhancements can focus on improving realism, scalability, and applicability of the SLAM system:

1. **Visual Feature-Based SLAM:**
   Replace ideal point landmarks with real image features using detectors such as ORB, SIFT, or SURF for realistic camera-based perception.

2. **3D SLAM Extension:**
   Extend the system to three dimensions by incorporating depth or stereo camera inputs, enabling full 6-DOF pose estimation.

3. **ROS and Hardware Integration:**
   Deploy the EKF-SLAM algorithm on real robotic platforms such as TurtleBot, Duckiebot, or ESP32-based robots using ROS for real-time sensor fusion and mapping.

4. **Advanced SLAM Techniques:**
   Investigate Particle Filter SLAM or Graph-Based SLAM methods to handle larger and more complex environments.

# Appendix

## Appendix 1: Simulation and EKF Parameters

The following parameters were used in the EKF-SLAM simulation to model robot motion, sensor uncertainty, and environment constraints. These values were chosen to ensure stable filter convergence and realistic behavior.

Robot Motion Parameters

- Linear velocity v: proportional to distance error
- Angular velocity ω: proportional to heading error
- Time step Δt: fixed simulation update interval

Process Noise Parameters (R)

- Linear velocity noise variance
- Angular velocity noise variance

These parameters model actuator uncertainty such as wheel slip and control inaccuracies.

Sensor Parameters

- Sensor type: Simulated monocular camera
- Field of View (FOV): 120°
- Number of sensor rays: 15
- Maximum sensing range: finite range threshold

Measurement Noise Parameters (Q)

- Range measurement variance
- Bearing measurement variance

These values represent uncertainty in camera-based landmark detection.

## Appendix 2: Key Code Snippets

A.2.1 Differential Drive Motion model(Prediction Step)

Code Snippet 1

```
if abs(w) > 1e-4:  # Circular motion
    # Arc motion equations
    dx = -(v/w) * np.sin(theta) + (v/w) * np.sin(theta + w*dt)
    dy = (v/w) * np.cos(theta) - (v/w) * np.cos(theta + w*dt)
    dtheta = w * dt
else:  # Straight line motion
    dx = v * np.cos(theta) * dt
    dy = v * np.sin(theta) * dt
    dtheta = 0.0
```

A.2.2 EKF Prediction Step
Code Snippet 2

```
sigma_bar = G_t @ sigma @ G_t.T + R_t
```

A.2.3 Kalman Gain and State Update
Code Snippet 3

```
# Kalman gain: K = Sigma * H^T * S^{-1}
try:
    K = sigma @ H_t.T @ np.linalg.inv(S)
except np.linalg.LinAlgError:
    continue

# State update: mu = mu + K * innovation
mu = mu + K @ innovation
mu[2, 0] = normalize_angle(mu[2, 0])
```

A.2.4 Innovation Covariance
Code Snippet 4

```
# Innovation covariance
S = H_t @ sigma @ H_t.T + Q
```