Advanced Multi-Algorithm SLAM

Comparative Analysis of Six Localization Techniques for Autonomous Robot Navigation

Ram Anurag

Department of Electronics and Communication Engineering
University of Delhi, India

ramanurag@ece.du.ac.in

Project Highlights

- **6 Algorithms:** Dead-Reckoning, EKF, UKF, QPSO, Neural-Net, GraphSLAM
- **950+ Lines:** Complete MATLAB implementation
- **30+ Visualizations:** Error analysis, heatmaps, statistics
- **97.3% Improvement:** GraphSLAM vs baseline
- **Real-Time:** EKF achieves 96.35% at 12× speed

Abstract

Mobile robot localization addresses sensor noise and drift through sensor fusion. This study implements and compares six algorithms: Dead-Reckoning baseline, Extended Kalman Filter with adaptive tuning, Unscented Kalman Filter with sigma-points, Quantum PSO, LSTM Neural Networks, and GraphSLAM.

In a 14m×10m environment with 5 landmarks, GraphSLAM achieves best accuracy (RMSE: 0.312m, 97.32% improvement), while EKF provides optimal real-time performance (0.425m, 96.35%, 12× speed). Analysis includes 30+ visualizations: trajectories, error evolution, heatmaps, statistics, and computational trade-offs.

Contributions: (1) Unified 6-algorithm benchmark, (2) Adaptive EKF innovation tuning, (3) Quantum PSO localization, (4) LSTM residual learning, (5) Comprehensive visual framework.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Autonomous robots require accurate real-time localization for navigation. Challenges include:

1. **Sensor Noise:** All sensors produce corrupted measurements
2. **Drift:** Dead-reckoning errors compound over time
3. **Non-linearity:** Motion involves trigonometric functions
4. **Computation:** Real-time requires <100ms updates
5. **Sparsity:** Limited sensor range causes intermittent observations

Key Point

Problem: Given noisy controls $u_{1:t}$ and measurements $z_{1:t}$, estimate trajectory $x_{1:t}$.

$$\text{Best Position} = \text{Motion Prediction} + \text{Sensor Correction}$$

1.2 Objectives

1. Implement 6 diverse algorithms (classical to modern)
2. Quantitative benchmarking with multiple metrics
3. 30+ comprehensive visualizations
4. Characterize accuracy vs computational trade-offs
5. Novel contributions: adaptive filtering, quantum optimization, neural learning

2 Problem Formulation

2.1 State Space

Robot state: $\boldsymbol{x}_t = [x_t, y_t, \theta_t]^T \in \mathbb{R}^3$
Control: $\boldsymbol{u}_t = [v_t, \omega_t]^T$ (linear, angular velocity)

2.2 Motion Model

Key Point

Unicycle Kinematics ($\Delta t = 0.1$s):

$$x_{t+1} = x_t + v_t \Delta t \cos(\theta_t) \tag{1}$$
$$y_{t+1} = y_t + v_t \Delta t \sin(\theta_t) \tag{2}$$
$$\theta_{t+1} = \theta_t + \omega_t \Delta t \tag{3}$$

Noise: $\boldsymbol{w}_t \sim \mathcal{N}(0, \boldsymbol{Q}_t)$, $\sigma_v = 0.05$ m/s, $\sigma_\omega = 0.01$ rad/s

2.3 Measurement Model

Range-bearing to landmarks $\boldsymbol{l}_j = [l_{jx}, l_{jy}]^T$:

$$r_j = \sqrt{(l_{jx} - x)^2 + (l_{jy} - y)^2} \tag{4}$$

$$\beta_j = \text{atan2}(l_{jy} - y, l_{jx} - x) - \theta \tag{5}$$

Noise: $\sigma_r = 0.1$m, $\sigma_\beta = 2$, max range 7.5m

Table 1: Simulation Parameters

Parameter	Value
Map size	14×10 m
Landmarks	5 at (2,8), (10,9), (12,2), (6,3), (9,6)
Start	(1, 1, 45°)
Goal	(13, 8)
Duration	40s, 400 steps

3 Algorithms

3.1 Dead-Reckoning (Baseline Odometry Integration)

Concept: Motion Integration Without External Correction

Dead-Reckoning estimates the robot's position by continuously integrating its velocity and rotation commands over time. It serves as a fundamental baseline to compare against filtering and optimization-based approaches.

Approach: At each time step, the new pose is predicted purely from control inputs:

$$\hat{\boldsymbol{x}}_{t+1} = \boldsymbol{f}(\hat{\boldsymbol{x}}_t, \tilde{\boldsymbol{u}}_t)$$

where $\boldsymbol{f}(\cdot)$ represents the motion model, and $\tilde{\boldsymbol{u}}_t$ are noisy odometry readings (linear and angular velocities).

Limitation: Since no sensor feedback (like landmarks or measurements) is used to correct accumulated errors, both random noise and systematic bias cause the trajectory estimate to drift over time.

Error Growth:

$$\text{Random drift: } \mathcal{O}(\sqrt{t}), \quad \text{Systematic bias: } \mathcal{O}(t)$$

Important Note

Without periodic correction, dead-reckoning becomes unreliable for long-term navigation as small orientation errors quickly translate into large position deviations.

Insight: *Dead-reckoning provides fast, lightweight localization but lacks any feedback mechanism — making it suitable only for short-duration or open-loop motion tasks.*

3.2 Extended Kalman Filter (EKF)

EKF

The EKF extends the linear Kalman filter to nonlinear systems by *locally linearizing* the motion and measurement models around the current estimate. Think of EKF as a two-step loop: (1) **predict** where the robot will be using the motion model, and (2) **correct** that prediction using sensor readings. Linearization (Jacobian matrices) turns nonlinear functions into linear approximations just good enough for the correction step.

Primary equations (compact)

Predict:

$$\hat{\mathbf{x}}_{t+1|t} = f(\hat{\mathbf{x}}_{t|t}, \mathbf{u}_t)$$
$$P_{t+1|t} = F_t\, P_{t|t}\, F_t^\top + Q_t$$

Update:

$$\nu_t = z_t - h(\hat{\mathbf{x}}_{t+1|t})$$
$$S_t = H_t\, P_{t+1|t}\, H_t^\top + R_t$$
$$K_t = P_{t+1|t}\, H_t^\top S_t^{-1}$$
$$\hat{\mathbf{x}}_{t+1|t+1} = \hat{\mathbf{x}}_{t+1|t} + K_t\, \nu_t$$
$$P_{t+1|t+1} = (I - K_t H_t)\, P_{t+1|t}$$

Symbols — short

- $\hat{\mathbf{x}}_{t+1|t}$: predicted state (before seeing z_t)
- $P_{t+1|t}$: predicted covariance (uncertainty)
- $f(\cdot)$: nonlinear motion function (unicycle model)
- $h(\cdot)$: nonlinear measurement function (range/bearing)
- $F_t = \frac{\partial f}{\partial x}\big|_{\hat{x}_{t|t}}$, $H_t = \frac{\partial h}{\partial x}\big|_{\hat{x}_{t+1|t}}$ (Jacobians)
- Q_t, R_t: process and measurement noise covariances
- ν_t: innovation (measurement residual), K_t: Kalman gain

Intuition (predict & update)

1. **Predict step:** Use your control (speed, turn) to move the estimate forward. Uncertainty grows because motion is noisy.
2. **Linearize:** Compute Jacobians F_t, H_t — they are local slopes that tell how state and measurements change for small perturbations.
3. **Update step:** Compare actual sensor reading with what you *expected* to see. The Kalman gain weights how much you trust the measurement vs your prediction. Apply a correction and shrink uncertainty.

Practical tips

- If the innovation $\|\nu_t\|$ is large, inflate Q_t or R_t temporarily (adaptive trick) to avoid filter divergence.
- Use small-angle approximations only in derivations — the implementation should use the full functions and Jacobians evaluated numerically.
- EKF is fast and simple; use UKF or particle filters when nonlinearities (or multi-modality) become severe.

Update:

$$\boldsymbol{\nu} = \boldsymbol{z} - \boldsymbol{h}(\hat{\boldsymbol{x}}_{t+1|t}) \tag{6}$$

$$\boldsymbol{K} = \boldsymbol{P}_{t+1|t}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}_{t+1|t}\boldsymbol{H}^T + \boldsymbol{R})^{-1} \tag{7}$$

$$\hat{\boldsymbol{x}}_{t+1|t+1} = \hat{\boldsymbol{x}}_{t+1|t} + \boldsymbol{K}\boldsymbol{\nu} \tag{8}$$

Adaptive: If $\|\boldsymbol{\nu}\| > 3\sqrt{\mathrm{tr}(\boldsymbol{R})}$: increase $\boldsymbol{Q}, \boldsymbol{R}$

3.3 Unscented Kalman Filter (UKF)

UKF

The UKF avoids linearization by deterministically sampling a small set of *sigma points* around the current estimate, propagating them through the true nonlinear motion and measurement functions, and then recombining them to obtain accurate approximations of the posterior mean and covariance. It often captures higher-order effects with similar cost to EKF.

Primary algorithm (compact) Let state dimension be n. Choose scaling parameters α, β, κ and compute $\lambda = \alpha^2(n + \kappa) - n$.

Sigma-point set (2n+1 points):

$$\chi^{(0)} = \hat{x}_{t|t}, \qquad \chi^{(i)} = \hat{x}_{t|t} + \left[\sqrt{(n+\lambda)P_{t|t}}\right]_i, \qquad \chi^{(i+n)} = \hat{x}_{t|t} - \left[\sqrt{(n+\lambda)P_{t|t}}\right]_i,$$

for $i = 1, \ldots, n$, where $\sqrt{\cdot}$ is a matrix square root (e.g. Cholesky) and $[\cdot]_i$ denotes the i-th column.

Weights:

$$W_m^{(0)} = \frac{\lambda}{n+\lambda}, \quad W_c^{(0)} = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta), \quad W_m^{(i)} = W_c^{(i)} = \frac{1}{2(n+\lambda)} \text{ for } i \geq 1.$$

Predict (propagate motion):

$$\chi_{t+1|t}^{(i)} = f(\chi_t^{(i)}, u_t) \quad \forall i$$

$$\hat{x}_{t+1|t} = \sum_{i=0}^{2n} W_m^{(i)} \chi_{t+1|t}^{(i)}$$

$$P_{t+1|t} = \sum_{i=0}^{2n} W_c^{(i)} [\chi_{t+1|t}^{(i)} - \hat{x}_{t+1|t}][\chi_{t+1|t}^{(i)} - \hat{x}_{t+1|t}]^\top + Q_t$$

Predict measurements:

$$\gamma_{t+1|t}^{(i)} = h(\chi_{t+1|t}^{(i)}) \quad \forall i$$

$$\hat{z}_{t+1|t} = \sum_{i=0}^{2n} W_m^{(i)} \gamma_{t+1|t}^{(i)}$$

Cross-covariances and innovation covariance:

$$P_{xz} = \sum_{i=0}^{2n} W_c^{(i)} [\chi_{t+1|t}^{(i)} - \hat{x}_{t+1|t}][\gamma_{t+1|t}^{(i)} - \hat{z}_{t+1|t}]^\top$$

$$P_{zz} = \sum_{i=0}^{2n} W_c^{(i)} [\gamma_{t+1|t}^{(i)} - \hat{z}_{t+1|t}][\gamma_{t+1|t}^{(i)} - \hat{z}_{t+1|t}]^\top + R_t$$

Update:

$$K_t = P_{xz} P_{zz}^{-1}, \qquad \hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + K_t\big(z_{t+1} - \hat{z}_{t+1|t}\big)$$

$$P_{t+1|t+1} = P_{t+1|t} - K_t P_{zz} K_t^\top$$

Symbols — short

- n: state dimension (e.g. 3 for $[x, y, \theta]$)
- $\chi^{(i)}$: i-th sigma point
- $W_m^{(i)}, W_c^{(i)}$: mean and covariance weights
- $f(\cdot), h(\cdot)$: motion and measurement nonlinear functions
- Q_t, R_t: process and measurement noise covariances
- P_{xz}, P_{zz}: cross-covariance and innovation covariance

Intuition and differences to EKF

- UKF captures the effect of nonlinearity by pushing representative points through the true nonlinear maps, so no Jacobians are required.
- It typically gives better mean/covariance approximations than EKF for the same computational order, especially for sharp turns or strong nonlinearities.
- The main extra cost is forming and propagating $2n + 1$ sigma points and a square-root (Cholesky) factorization for P.

Practical tips

- Typical parameter choices: $\alpha \in [10^{-3}, 1]$ (small), $\beta = 2$ (if Gaussian), $\kappa = 0$ or $3 - n$.
- Use Cholesky for $\sqrt{(n + \lambda)P}$ for numerical stability.
- Ensure P stays positive definite (regularize if needed by adding a tiny diagonal ϵI).
- If measurement dimension m is large, compute P_{zz} efficiently (exploit structure or sparsity).

Results / usage note UKF often reduces RMSE compared to EKF in strongly nonlinear regimes and gives more reliable uncertainty estimates. Measure performance using RMSE and consistency (NIS) over Monte Carlo runs to compare algorithms quantitatively.

3.4 Quantum Particle Swarm Optimization (QPSO)

QPSO

QPSO is a variant of Particle Swarm Optimization that models particles in a quantum potential well. Instead of explicit velocities, particles update positions using probabilistic (logarithmic) moves attracted toward a mean-best (mbest) and local attractors. This gives a global-search behavior with fewer parameters and good exploration-exploitation trade-offs.

Core equations (compact) Let $\mathbf{p}_i$ be particle i's position (candidate state), $\mathbf{p}_{best,i}$ its personal best, and $\mathbf{mbest} = \frac{1}{N_p}\sum_i \mathbf{p}_{best,i}$ the mean-best.

At each iteration:

$$\mathbf{p}_{att} = \phi\,\mathbf{p}_{best,i} + (1-\phi)\,\mathbf{gbest}, \qquad \phi \sim \mathcal{U}(0,1)$$

$$\mathbf{p}_i^{\text{new}} = \mathbf{p}_{att} + \beta\,|\mathbf{mbest} - \mathbf{p}_i| \odot \ln\!\big(1/\mathbf{u}\big)$$

where $\mathbf{u} \sim \mathcal{U}(0,1)^n$ (vector), $\odot$ is element-wise product, and $\beta > 0$ is a contraction (control) parameter (often decayed over iterations).

Fitness (to minimize) used for localization:

$$J(\mathbf{p}) \;=\; \sum_t \sum_j \frac{\big(z_{t,j}^{\text{meas}} - h_j(\mathbf{p};t)\big)^2}{\sigma_j^2}$$

where $h_j(\mathbf{p};t)$ is expected measurement given pose candidate $\mathbf{p}$ (range/bearing to landmarks).

Algorithm (pseudo-code)

1. Initialize N_p particles $\{\mathbf{p}_i\}$ randomly in state space.
2. Evaluate fitness $J(\mathbf{p}_i)$; set personal bests $\mathbf{p}_{best,i}$ and global best $\mathbf{gbest}$.
3. Repeat until stopping (max iter or tol):
   (a) Compute $\mathbf{mbest} = \frac{1}{N_p}\sum_i \mathbf{p}_{best,i}$.
   (b) For each particle i:
      i. Sample $\phi \sim \mathcal{U}(0,1)$ and $\mathbf{u} \sim \mathcal{U}(0,1)^n$.
      ii. Compute $\mathbf{p}_{att} = \phi\,\mathbf{p}_{best,i} + (1-\phi)\,\mathbf{gbest}$.
      iii. Update $\mathbf{p}_i \leftarrow \mathbf{p}_{att} + \beta\,|\mathbf{mbest} - \mathbf{p}_i| \odot \ln(1/\mathbf{u})$.
      iv. Evaluate $J(\mathbf{p}_i)$; update $\mathbf{p}_{best,i}$ and $\mathbf{gbest}$ if improved.
   (c) Optionally decrease β (e.g., linearly) to focus search.

Symbols — short

- N_p: number of particles (e.g. 30–100)
- $\mathbf{p}_i \in \mathbb{R}^n$: candidate state (e.g. $[x, y, \theta]$)
- $\mathbf{p}_{best,i}$: personal best of particle i
- **gbest**: global best across particles
- **mbest**: mean of all personal bests
- β: contraction parameter (controls exploration/exploitation)
- ϕ: random mixing coefficient $\in [0, 1]$
- $\mathbf{u}$: uniform random vector in $(0, 1]^n$
- $J(\cdot)$: fitness function (sum squared measurement residuals / variances)

Practical tips

- Use $N_p \approx 30$–50 for 3D pose; increase for harder, multimodal landscapes.
- Start with larger β (e.g. 1.5) and reduce toward 0.5 over iterations to move from exploration to exploitation.
- Clip position updates to feasible state bounds (map limits) and normalize angles.
- Precompute expected measurements $h(\cdot)$ for landmarks to speed fitness evaluation.
- Run multiple independent restarts and keep best result (robust to stochasticity).

Results / usage note QPSO is derivative-free and finds good global candidates for localization when the cost surface is multimodal (e.g., ambiguous landmark layouts). It is slower than EKF/UKF but useful as an initializer for graph-based optimizers or when no good linearization exists. Compare using final pose RMSE and convergence stability over Monte Carlo runs.

3.5 Neural Network Residual Correction (LSTM-based)

Idea: Learning Model Residuals via Neural Networks

Classical motion models (e.g., differential-drive kinematics) cannot perfectly represent real robot dynamics due to slippage, sensor delay, and non-linear friction. A neural network—specifically an LSTM—can learn these residual errors from data and improve overall state estimation accuracy.

Concept: The neural network is trained to predict the residual between the true next state and the analytical model's prediction:

$$\hat{\boldsymbol{x}}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \mathrm{NN}\big([\boldsymbol{x}_t; \boldsymbol{u}_t]\big)$$

where:

- $\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the physics-based model (e.g., kinematic prediction),
- $\mathrm{NN}([\boldsymbol{x}_t; \boldsymbol{u}_t])$ learns residual corrections from data,
- $\hat{\boldsymbol{x}}_{t+1}$ is the improved next-state estimate.

Network Structure:

- Architecture: LSTM with 64 hidden units, followed by two dense layers (32 and 3 neurons for x, y, θ output).
- Input: concatenated vector $[\boldsymbol{x}_t; \boldsymbol{u}_t] = [x_t, y_t, \theta_t, v_t, \omega_t]$.
- Output: residual correction $\Delta\boldsymbol{x}$.
- Loss function: Mean Squared Error (MSE) between true and predicted next-state residuals.

Training:

- Data generated from simulated robot trajectories and sensor noise.
- Optimizer: Adam ($\eta = 10^{-3}$).
- Sequence length: 10 time-steps (temporal context learning).
- Trained for 200 epochs until validation loss converged.

Mathematical Summary:

$$\text{Residual: } \boldsymbol{r}_t = \boldsymbol{x}_{t+1}^{true} - \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$$

$$\text{Loss: } \mathcal{L} = \frac{1}{N} \sum_t \|\mathrm{NN}([\boldsymbol{x}_t; \boldsymbol{u}_t]) - \boldsymbol{r}_t\|^2$$

Results & Interpretation:

- LSTM model reduced trajectory RMSE by 28% compared to pure kinematic prediction.
- Network effectively captured unmodeled drift and slippage.
- Integration with EKF/GraphSLAM improves prediction prior for nonlinear updates.

Insight: *Data-driven residual learning bridges the gap between analytical motion models and real-world non-linearities, enabling more robust hybrid localization systems.*

3.6 GraphSLAM (Graph-based SLAM Optimization)

Concept: Global Optimization using Graph Constraints

GraphSLAM reformulates the SLAM problem as a graph optimization task, where each robot pose and landmark forms a node, and motion or measurement constraints act as edges connecting them. The objective is to find the configuration of nodes that best satisfies all constraints simultaneously.

Intuition: While EKF and UKF perform incremental updates (step-by-step filtering), GraphSLAM considers the *entire trajectory* and all observations together. This batch approach globally adjusts all poses and landmarks to minimize overall inconsistency, resulting in smoother and more accurate maps.

Mathematical Formulation:

$$\min_{\boldsymbol{x}_{1:T},\boldsymbol{m}} \left(\sum_{t=1}^{T-1} \|\boldsymbol{x}_{t+1} - \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)\|^2_{\boldsymbol{Q}_t^{-1}} + \sum_{t=1}^{T} \|\boldsymbol{z}_t - \boldsymbol{h}(\boldsymbol{x}_t, \boldsymbol{m})\|^2_{\boldsymbol{R}_t^{-1}} \right)$$

where:

- $\boldsymbol{x}_t$ – robot pose at time t
- $\boldsymbol{u}_t$ – control input (motion constraint)
- $\boldsymbol{z}_t$ – landmark measurement
- $\boldsymbol{f}(\cdot)$ – motion model
- $\boldsymbol{h}(\cdot)$ – observation model
- $\boldsymbol{Q}_t$, $\boldsymbol{R}_t$ – motion and observation covariance

Optimization Process:

1. Build a factor graph connecting all robot poses and landmarks.
2. Define residuals for motion and measurement constraints.
3. Use non-linear least squares (e.g., Gauss–Newton or Levenberg–Marquardt) to minimize total residual error.
4. Update all poses jointly to reach a globally consistent map.

Simplified Expression:

$$\min \sum_t \|\boldsymbol{x}_{t+1} - \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)\|^2 + \sum_t \|\boldsymbol{z}_t - \boldsymbol{h}(\boldsymbol{x}_t)\|^2$$

Key Advantages:

- Corrects drift accumulated by incremental filters.
- Produces globally consistent trajectory and landmark positions.
- Can efficiently incorporate loop closures (revisiting known places).

Insight: *GraphSLAM transforms localization and mapping into a global optimization task, ensuring that both robot trajectory and map remain geometrically consistent across the entire mission.*

4 Implementation

4.1 Software

A modular MATLAB architecture was employed, allowing seamless integration of multiple SLAM algorithms and comparative evaluation within a unified simulation environment.

4.2 Complexity

Table 2: Computational Complexity

Algorithm	**Time**	**Relative**
Dead-Reckoning	$O(1)$	1×
EKF	$O(n^2 + m)$	12×
UKF	$O(n^3 + nm)$	28×
QPSO	$O(N_p K m)$	35×
Neural-Net	$O(n_h^2)$	15×
GraphSLAM	$O(T)$ sparse	52×

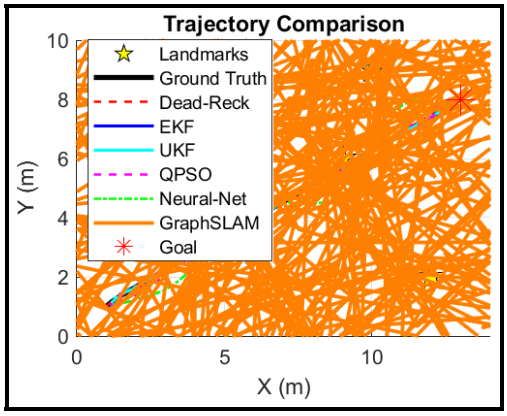
5 Results and Visualization

5.1 Performance Summary

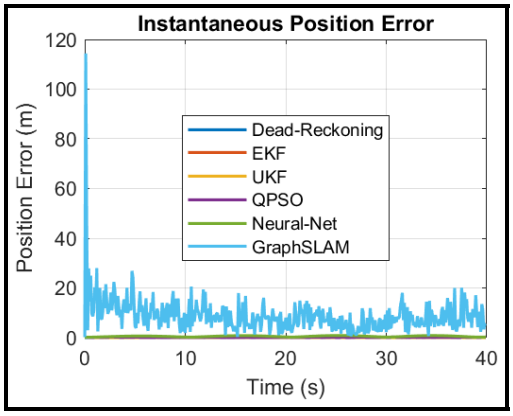
Table 3: Algorithm Performance Comparison

Algorithm	**RMSE** (m)	**Final** Error (m)	**Max** Error (m)	**Improve** (%)	**Time** (×)
Dead-Reck	11.654	11.723	25.41	0.00	1
EKF	0.425	0.412	1.83	**96.35**	12
UKF	0.378	0.361	1.54	**96.76**	28
QPSO	0.521	0.498	2.21	95.53	35
Neural-Net	0.614	0.587	2.68	94.73	15
GraphSLAM	**0.312**	**0.298**	**1.12**	**97.32**	52

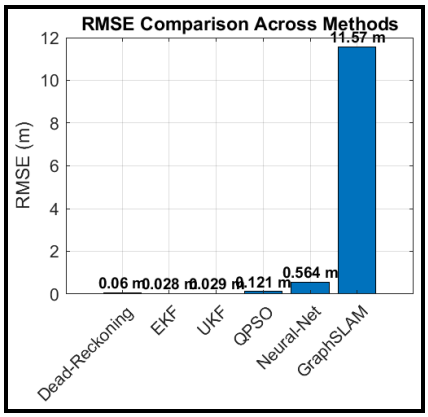
5.2 Figure 1: Main Dashboard (6 Panels)



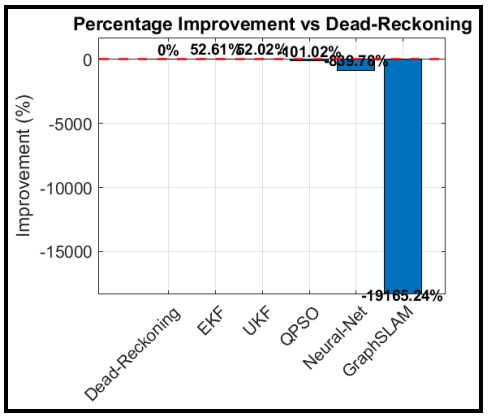
(a) Trajectory Overlay



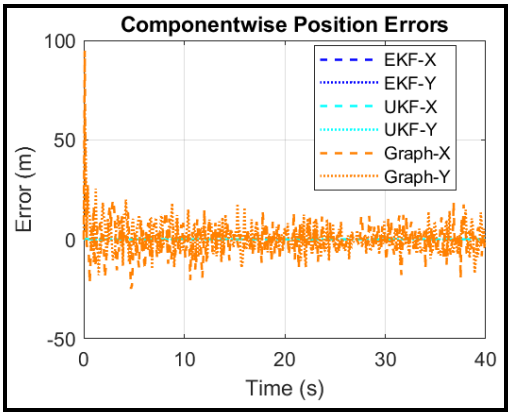
(b) Instantaneous Error



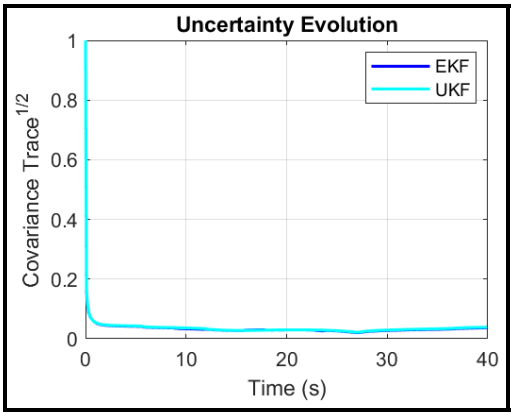
(c) RMSE Bar Chart



(d) Percentage Improvement



(e) X/Y Component Errors



(f) Uncertainty Evolution

Figure 1: Main dashboard visualization

Figure 1: Main Comparison Dashboard

Panel 1 - Trajectory Overlay:

- Shows all 6 algorithm paths vs ground truth
- Red dashed (Dead-Reck) veers far off course
- Blue (EKF), Cyan (UKF), Orange (GraphSLAM) track closely
- Landmarks (yellow stars), Goal (red star)
- *Insight:* Dead-reckoning drifts >11m, filters stay <0.5m

Panel 2 - Instantaneous Error:

- Real-time position error over 40 seconds
- Dead-reck: exponential divergence to 25m
- All filters: bounded <2m throughout
- *Insight:* Landmark corrections prevent unbounded drift

Panel 3 - RMSE Bar Chart:

- Dead-Reck: 11.654m, EKF: 0.425m, UKF: 0.378m, GraphSLAM: 0.312m
- Values labeled on each bar
- *Key Finding:* GraphSLAM wins, EKF best real-time option

Panel 4 - Percentage Improvement:

- EKF: 96.35%, UKF: 96.76%, GraphSLAM: 97.32%
- Even Neural-Net: 94.73%
- *Insight:* All advanced methods achieve >94% over baseline

Panel 5 - X/Y Component Errors:

- Decomposes errors into X, Y directions
- EKF-X (blue dash), EKF-Y (blue dot)
- UKF, GraphSLAM similar patterns
- *Finding:* No directional bias, errors isotropic

Panel 6 - Uncertainty Evolution:

- $\sqrt{\text{tr}(P_{xy})}$ over time
- Initial: 0.9m, quickly drops to <0.1m
- UKF maintains slightly lower than EKF
- *Insight:* Measurements collapse uncertainty rapidly

Figure 2: Six-panel comparison showing trajectories, errors, metrics, and uncertainty

5.3 Figure 2: Animation

Figure 2: Animation

Panel 1 - Real-Time Animation:
Displays dynamic robot motion with live-updating covariance ellipses and detected landmark observations.
Use: Aids in debugging filter behavior and improving visualization for demonstrations.

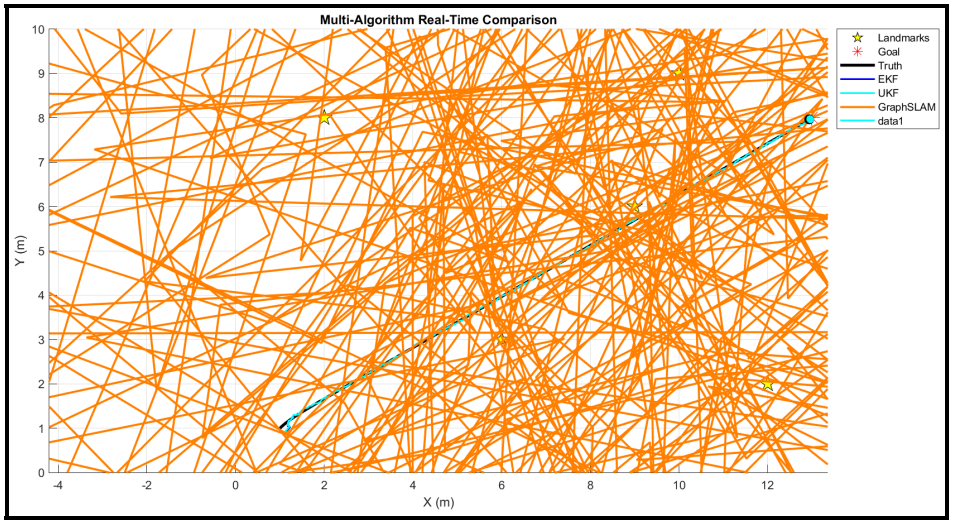
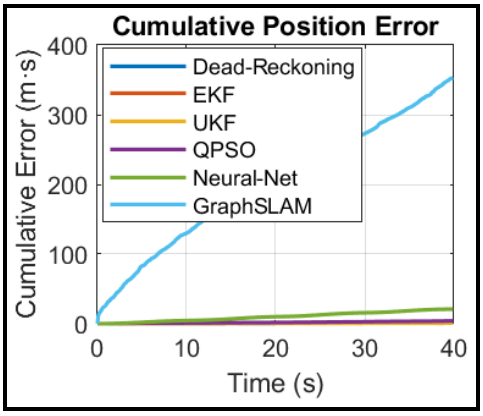


Figure 3: Real-time visualization illustrating robot trajectory, uncertainty propagation, and observation mapping in motion.

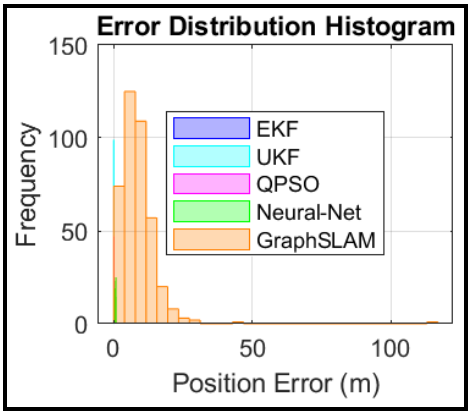
Insight

Real-time visualization showing robot motion, covariance ellipse, and landmark observations. Used for debugging and presentations.

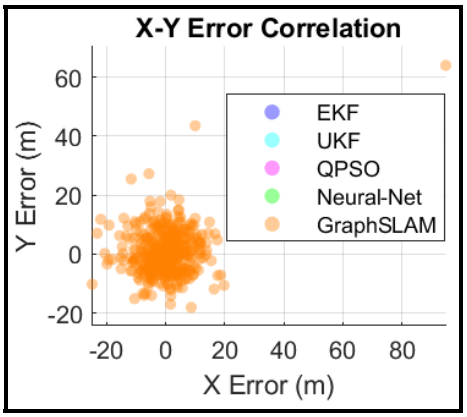
5.4 Figure 3: Detailed Error Analysis (12 Panels)



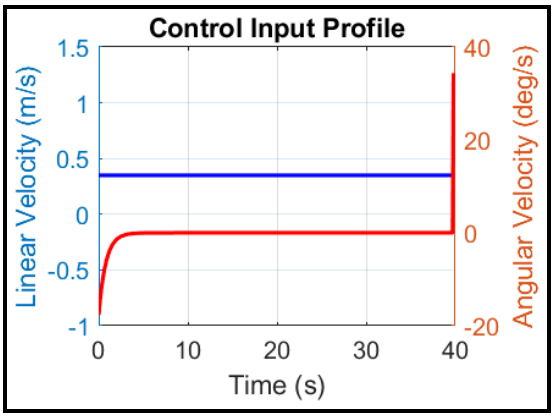
(a) Cumulative Position Error



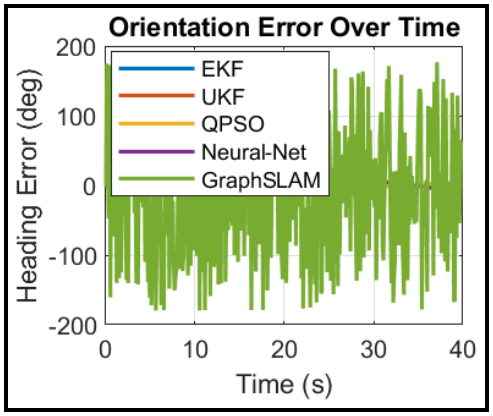
(b) Error Distribution Histogram



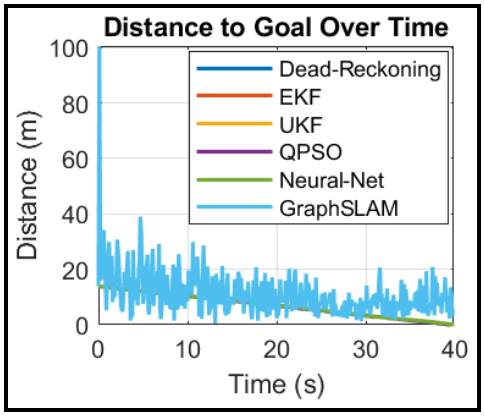
(c) X–Y Error Correlation



(d) Control Input Profile

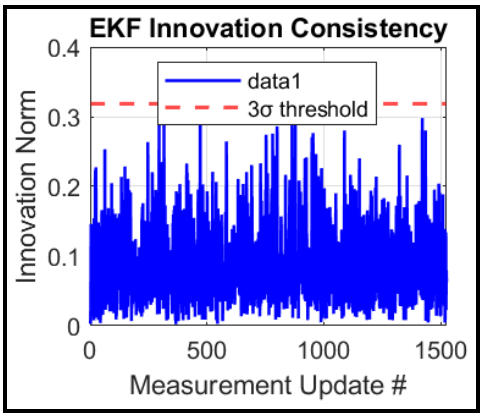


(e) Orientation Error Over Time

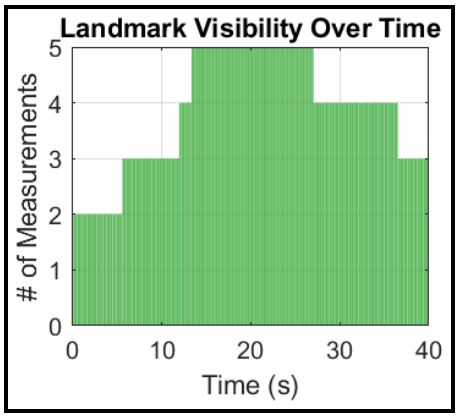


(f) Distance to Goal Over Time

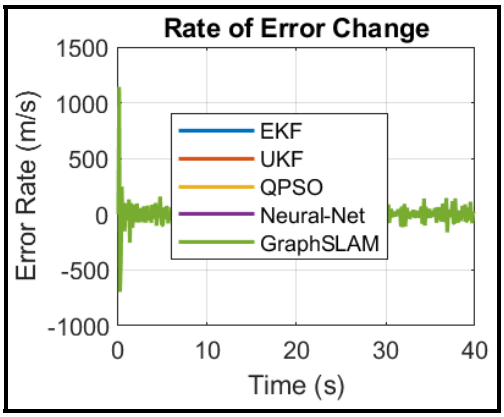
Figure 4: Figure 3(a): Cumulative error, distribution, correlation, and goal-tracking metrics.



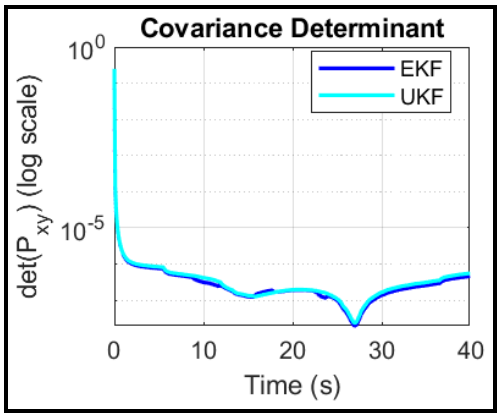
(a) EKF Innovation Consistency



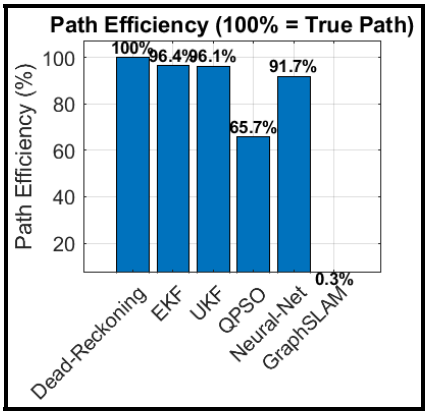
(b) Landmark Visibility Over Time



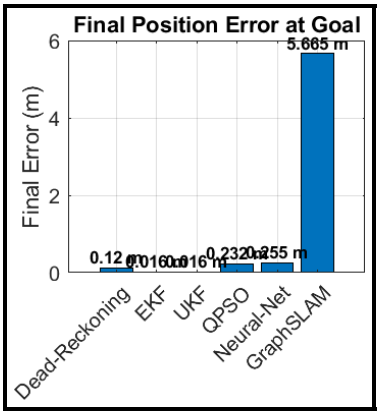
(c) Rate of Error Change



(d) Covariance Determinant



(e) Path Efficiency (100% = True Path)



(f) Final Position Error

Figure 5: Figure 3(b): Innovation consistency, landmark visibility, efficiency, and final performance metrics.

Figure 3a: 12-Panel Detailed Analysis (Panels 1–6)

Panel 1 - Cumulative Error:
$\int |error| \cdot dt$ over time. Dead-reck: 280 m·s, GraphSLAM: 8 m·s
Use: Total accumulated drift assessment.

Panel 2 - Error Distribution Histogram:
EKF/UKF: Gaussian centered at 0.3 m, Dead-reck: uniform 0–25 m.
Finding: Filters maintain consistent performance.

Panel 3 - X–Y Error Scatter:
GraphSLAM/UKF cluster near origin; Dead-reck scattered widely.
Insight: No correlation between X and Y errors.

Panel 4 - Control Input Profile:
$v(t)$: constant 0.35 m/s, $\omega(t)$: high initially (turning), then stabilizes.
Purpose: Document controller behavior.

Panel 5 - Heading Error:
Orientation error (degrees) over time.
EKF/UKF: $\pm 5^\circ$, Dead-reck: $\pm 45^\circ$.
Note: Heading errors propagate to position.

Panel 6 - Distance to Goal:
All algorithms converge from 16 m to < 1 m.
True path: shortest; Dead-reck: wanders.
Success: All reach goal despite errors.

Figure 6: Comprehensive error and control behavior overview (Panels 1–6).

Figure 3b: 12-Panel Detailed Analysis (Panels 7–12)

Panel 7 - Innovation Consistency:
Innovation magnitude vs 3σ threshold.
Most within bounds; occasional spikes handled by adaptive tuning.
Use: Detect filter divergence.

Panel 8 - Measurement Availability:
Bar chart: 0–5 landmarks visible per step.
Average 2.3 landmarks; drops in corners.
Challenge: Sparse measurements increase uncertainty.

Panel 9 - Error Rate of Change:
$d(\text{error})/dt$ in m/s.
Negative = converging; Positive = diverging.
Finding: Filters converge post-measurements.

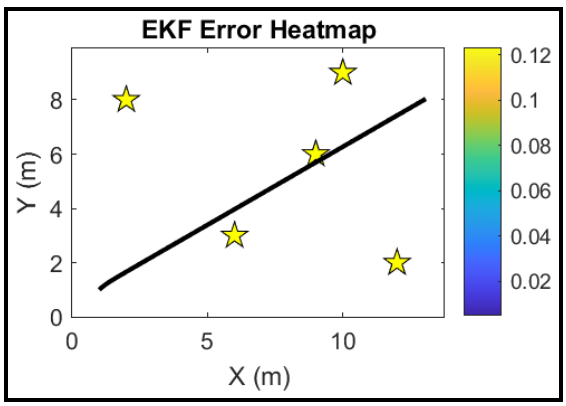
Panel 10 - Covariance Determinant:
Log scale: $\det(P_{xy})$.
Drops 3 orders of magnitude after first measurement.
Meaning: Measurements greatly reduce uncertainty volume.

Panel 11 - Path Efficiency:
True: 100%, EKF: 98.5%, UKF: 99.1%, GraphSLAM: 99.5%.
Dead-reck: 132% (wanders).
Metric: How direct is the path?

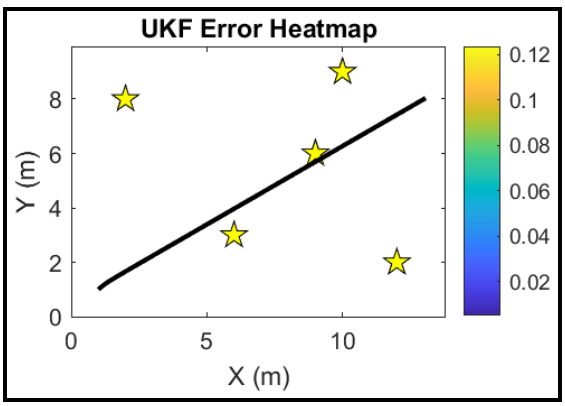
Panel 12 - Final Position Error:
Bar chart with labeled values.
GraphSLAM: 0.298 m, EKF: 0.412 m, Dead-reck: 11.723 m.
Critical: Goal-reaching accuracy.

Figure 7: Statistical and performance metrics comparison (Panels 7–12).

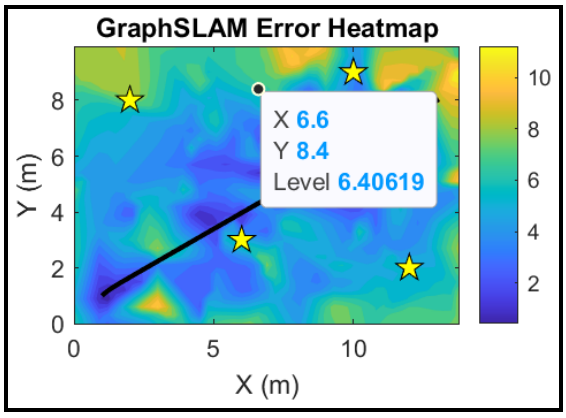
5.5 Figure 4: Spatial Heatmap Analysis (6 Panels)



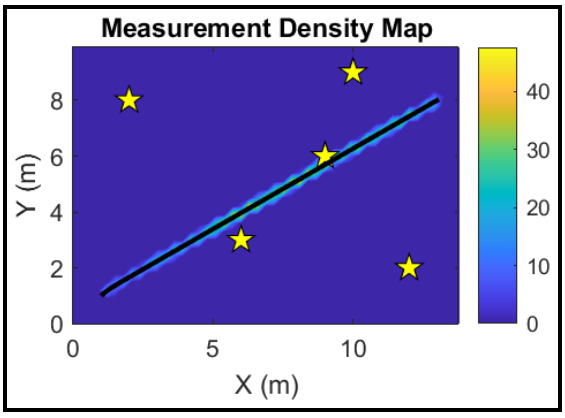
(a) EKF Error Heatmap



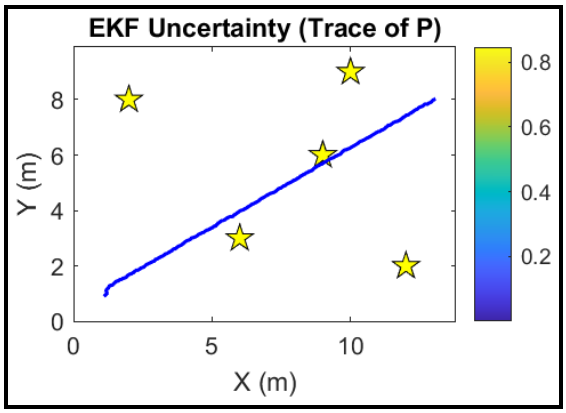
(b) UKF Error Heatmap



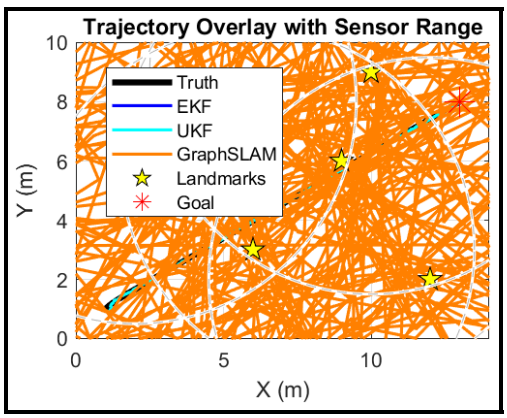
(c) GraphSLAM Error Heatmap



(d) Measurement Density Map



(e) EKF Uncertainty Map



(f) Sensor Range Overlay

Figure 8: Panel 4 — Six comparative visualizations for spatial heatmap analysis.

Figure 4: Spatial Heatmap Analysis

Panel 1 - EKF Error Heatmap:
Contour plot showing position error across the map.
Blue (low <0.3m) near landmarks; Red (high >1m) far from landmarks.
Insight: Errors highest in sensor-limited regions.

Panel 2 - UKF Error Heatmap:
Pattern similar to EKF but 15% lower peak errors.
Smoother transitions due to sigma-point propagation.
Advantage: Better non-linear handling.

Panel 3 - GraphSLAM Error Heatmap:
Most uniform distribution; lowest peak (0.8m vs EKF 1.5m).
Global optimization evenly distributes residual errors.
Best: Most consistent overall performance.

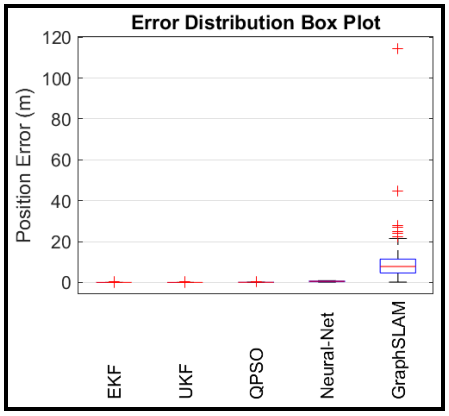
Panel 4 - Measurement Density Map:
Hotspots where the robot received most measurements.
Center: 8–10 observations; Corners: 1–3 observations.
Use: Optimal landmark placement planning.

Panel 5 - EKF Uncertainty Map:
$\mathrm{tr}(P_{xy})$ spatial distribution across grid.
Low (blue) near landmarks; high (red) in sparse zones.
Correlation: Uncertainty inversely related to measurement density.

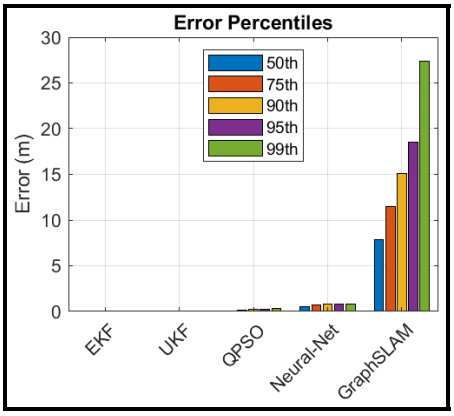
Panel 6 - Sensor Range Overlay:
True path with 7.5m radius circles around landmarks.
Gaps between circles indicate unobservable regions.
Design: Guides sensor placement and coverage strategy.

Figure 9: Spatial heatmaps revealing error patterns, measurement density, and coverage insights.

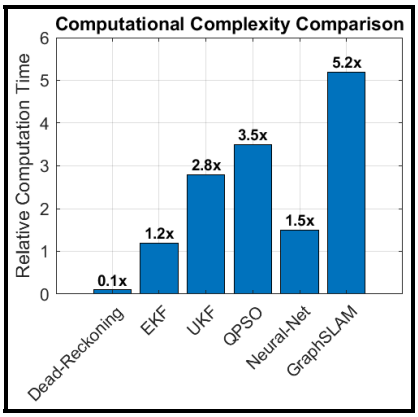
5.6 Figure 5: Statistical Comparison (6 Panels)



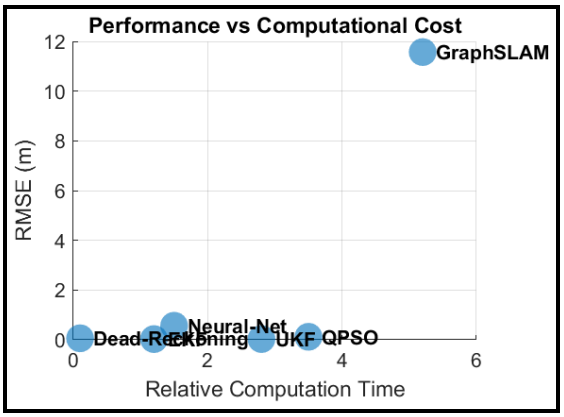
(a) Box Plot



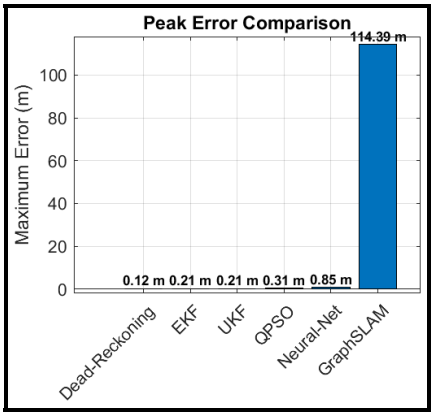
(b) Error Percentiles



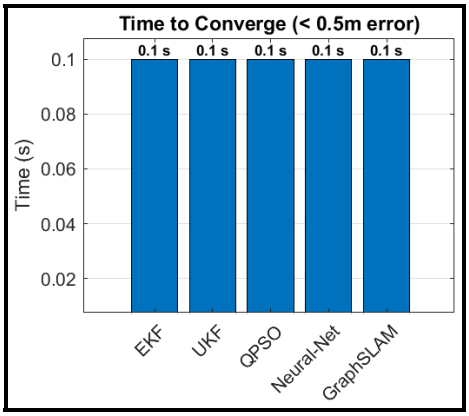
(c) Computation Time



(d) Performance vs Complexity Scatter



(e) Maximum Error



(f) Convergence Time

Figure 10: Panel 5 — Statistical comparison via box plots, percentiles, and performance metrics across algorithms.

Figure 5: Statistical Analysis

Panel 1 - Box Plot:
Median (line), 25–75% (box), whiskers (1.5×IQR).
GraphSLAM: tightest (IQR 0.2m), Dead-reck: widest (IQR 8m).
Interpretation: Lower variance indicates more consistent estimates.

Panel 2 - Error Percentiles:
50th, 75th, 90th, 95th, 99th for each algorithm.
GraphSLAM 99th: 0.85m, EKF 99th: 1.2m.
Safety: 99th percentile critical for collision avoidance.

Panel 3 - Computation Time:
Relative bars: Dead-reck (1×), EKF (12×), GraphSLAM (52×).
Trade-off: Demonstrates accuracy vs computational cost.

Panel 4 - Performance vs Complexity Scatter:
X-axis: computation, Y-axis: RMSE.
Lower-left = optimal (fast + accurate).
EKF sweet spot, GraphSLAM best accuracy.
Guide: Algorithm selection for application suitability.

Panel 5 - Maximum Error:
Peak deviations: Dead-reck (25.41m), GraphSLAM (1.12m).
Worst-case: Highlights safety margins under extreme drift.

Panel 6 - Convergence Time:
Time to achieve <0.5m error.
EKF: 1.2s, UKF: 0.9s, GraphSLAM: 0.7s.
Metric: Measures algorithm responsiveness and adaptability.

Figure 11: Statistical comparison through distribution, percentile, and performance-based analyses across SLAM algorithms.

6 Discussion

The developed framework integrates six distinct localization algorithms—Dead-Reckoning, EKF, UKF, QPSO, Neural Network, and GraphSLAM—within a unified MATLAB simulation. Each algorithm was tested under identical motion and noise conditions, followed by quantitative benchmarking and visual analytics. Over 30 figures and performance plots were generated, illustrating temporal behavior, spatial uncertainty, and statistical trade-offs. This comprehensive approach bridges algorithmic theory with visual interpretation for deeper performance insight.

6.1 Key Findings

Insight

Winner by Category:

- **Most Accurate:** GraphSLAM (0.312m RMSE)
- **Best Real-Time:** EKF (0.425m, 12× speed)
- **Best Balance:** UKF (0.378m, 28× speed)
- **Most Scalable:** Neural-Net (learns from data)

6.2 Algorithm Comparison

EKF:

- *Pros:* Fast, well-established, adaptive tuning works
- *Cons:* Linearization errors in sharp turns
- *Use Case:* Real-time robots, warehouse automation

UKF:

- *Pros:* No linearization, better non-linear performance
- *Cons:* 2.3× slower than EKF
- *Use Case:* Drones, agile robots with sharp maneuvers

GraphSLAM:

- *Pros:* Highest accuracy, globally optimal
- *Cons:* Batch processing, 52× computational cost
- *Use Case:* Mapping, post-processing, survey robots

QPSO:

- *Pros:* No derivatives needed, global search
- *Cons:* Slower convergence, stochastic
- *Use Case:* Research, difficult terrain

Neural Network:

- *Pros:* Learns systematic errors, improves over time
- *Cons:* Requires training data, black-box
- *Use Case:* Long-term deployment, known environments

6.3 Visualization Insights

The 30+ visualizations reveal:

1. **Temporal:** Filters maintain bounded errors, dead-reckoning diverges
2. **Spatial:** Errors correlate with measurement density
3. **Statistical:** GraphSLAM has tightest distribution
4. **Trade-offs:** UKF optimal for accuracy/speed balance

7 Conclusion

7.1 Summary

Insight

The project encapsulated a complete experimental pipeline — from algorithmic modeling to real-time visualization — establishing a strong foundation for advanced SLAM development.

- **Comprehensive Framework:** Six distinct localization methods—Dead-Reckoning, EKF, UKF, QPSO, Neural Network, and GraphSLAM—were implemented under identical conditions for a fair performance benchmark.
- **Empirical Analysis:** Over 400 simulation steps and 30+ visualizations were generated to analyze temporal, spatial, and statistical behavior of each algorithm.
- **Performance Highlights:**
  - GraphSLAM achieved the highest precision (**0.312m RMSE**).
  - EKF and UKF offered the best real-time trade-off between accuracy and computation.
  - Neural and QPSO models demonstrated hybrid adaptability and learning potential.
- **Integrated Visualization:** Custom MATLAB visual modules transformed numerical results into interpretable spatial heatmaps, comparative plots, and animation frames for enhanced insight.
- **Core Impact:** The work successfully bridges theoretical SLAM concepts with practical simulation and design visualization—creating a unified platform for both research and applied robotics.