Autonomous Systems Project Report
FastSlam on Pioneer 3DX - SFP

Dinis Alves da Silva 106261, Henrique Rodrigues 106362, Joana Teixeira 107520, Mafalda Lopes 106983

***Abstract*—We present a FastSLAM-based method for real-time localization and mapping of a Pioneer 3DX robot equipped with a web camera for ArUco marker (landmarks) detection. The system combines particle filtering for trajectory estimation with EKFs for landmark updates, supported by a resampling strategy to maintain particle diversity. Experimental evaluation across multiple particle counts shows that 40 to 50 particles provide the best balance between accuracy and efficiency. Results demonstrate that excessive particles can degrade performance due to weight concentration. The system runs in ROS with real-time visualization and integrated performance monitoring.**

I. Introduction

The Simultaneous Localization and Mapping (SLAM) problem, involves the task of constructing a map of an unknown environment using a moving robot while simultaneously estimating the robot's localization within it. FastSLAM is a probabilistic SLAM algorithm, based on particle filtration through weight resampling ,that addresses this challenge efficiently by decoupling pose estimation and landmark mapping into two separate problems. Using a particle filter for the robot localization problem and individual Extended Kalman Filters (EKFs) for the landmark mapping problem. Managing to reduce the exponential complexity of classic EKF-based methods to linear or even logarithmic complexity depending on the number of resources in the enviromnent.

In this project, we implemented and tested the FastSLAM algorithm on the Pioneer 3DX robot (SFP), with know correspondence, by using uniquely identifiable markers. The integration of a camera into our setup, allowed us to visually monitor what Arucos the robot perceived during operation. This real sensor data, odometry and camera data, made it possible to generate a map of the environment and a pose estimate for the robot. We then proceeded to interpreting localization behavior, understanding and calibrating sensor performance, validating map accuracy and identifying limitations.

II. Methods and Algorithms

A. FastSLAM Algorithm Overview

FastSLAM addresses the SLAM problem through a clever factorization that exploits the conditional independence property inherent in SLAM. Given a possible robot pose, represented by a weighted particle from the particle filter, an associated map can be formed for each particle through the now independent and separately solved landmark mapping problems maintained by a 2x2 dimensional EKF for each landmark (N landmarks). The algorithm's complexity, for M particles that track multiple hypothesis of the robot's pose and map can be written as $O(M \log N)$.

1) **Particle Filter:**
The key insight is that given the robot's pose, landmark observations become conditionally independent, allowing the posterior to be factored as:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) = p(x_{1:t} \mid z_{1:t}, u_{1:t}) \prod_{j=1}^{N} p(m_j \mid x_{1:t}, z_{1:t})$$

where $x_{1:t}$ represents the robot trajectory, $m = \{m_1, ..., m_N\}$ are the landmark positions, $z_{1:t}$ are observations, and $u_{1:t}$ are control inputs.
This factorization enables FastSLAM to use a particle filter for trajectory estimation combined with individual Extended Kalman Filters (EKFs) for each landmark, attached to each particle. Each particle k maintains a pose estimate $\mathbf{x}_t^{[k]}$, a set of landmark means and covariances $\{\mu_j^{[k]}, \Sigma_j^{[k]}\}$, and a weight $w_t^{[k]}$. The core of FastSLAM is this particle filter, which consists of multiple particles that help keep track of multiple hypotheses of the robot's pose and the environment's map, with associated uncertainties. Our implementation uses $K \in \{10, 25, 50, 75, 100\}$ particles.
2) **Motion Model:**
The motion model predicts the robot's new position based on its previous position and the control inputs. Each particle updates its pose using the probabilistic motion model $p(x_t \mid x_{t-1}, u_t)$, which captures the uncertainty in predicting x_t given x_{t-1} and u_t. The motion model performs the prediction step of the particle filter by updating each particle's pose using odometry data. It computes the change in position and orientation, adds Gaussian noise to account for motion uncertainty, and updates the particle's pose accordingly. This allows each particle to evolve over time while reflecting the inherent unpredictability of robot motion.
3) **Landmark Mapping:**
Within each particle, EKFs update the map. Each landmark has its own EKF, which updates its position and uncertainty based on new measurements. Since landmarks are static, the prediction step is trivial and the process covariance matrix is zero. The measurement function h given the landmark position m_{ct} and the robot pose x_t^k is linearized using a Taylor expansion:

$$h(m_{ct}, x_t^k) \approx \hat{z}_t^k + H_t^k (m_{ct} - \mu_{ct,t-1}^k)$$

Here, $\hat{z}_t^k$ is the predicted measurement, $\mu_{ct,t-1}^k$ is the current estimate of the landmark position, and H_t^k is the Jacobian of h. Using this linear approximation, the EKF updates the landmark's position and uncertainty.

4) **Weight Update:**
The Kalman gain K_t^k is calculated to determine the influence of the new measurement:

$$K_t^k = \Sigma_{ct,t-1}^k (H_t^k)^T \left[H_t^k \Sigma_{ct,t-1}^k (H_t^k)^T + Q_t \right]^{-1}$$

Here, $\Sigma_{ct,t-1}^k$ is the covariance of the landmark estimate from the previous step, and Q_t is the measurement noise covariance. The new mean $\mu_{ct,t}^k$ is updated with the new measurement z_t:

$$\mu_{ct,t}^k = \mu_{ct,t-1}^k + K_t^k (z_t - \hat{z}_t^k)$$

where z_t is the actual measurement observed by the robot, and $\hat{z}_t^k$ is the predicted measurement. Finally, the covariance $\Sigma_{ct,t}^k$ of the landmark position is updated to reflect the new estimate:

$$\Sigma_{ct,t}^k = (I - K_t^k H_t^k) \Sigma_{ct,t-1}^k$$

where I is the identity matrix. Particle weights are updated using the measurement likelihood based on the innovation and its covariance.

5) **Resampling:**
Resampling is performed to focus computational resources on the most likely hypotheses. Particles with higher importance factors are more likely to be selected, while particles with lower weights may be discarded. While performing a resample, the variance between particles in the particle set decreases, but the variance of the particle set as an estimator of the true belief increases over time. This means that with resampling, the particles will tend to better represent the possible states where the system can be. In order to improve results, resampling should only be done when the particle's importance factor is too high on a small number of particles. In this case, the particle set does not represent well the state of the system, and resampling helps maintain particle diversity while preventing degeneracy.

B. Probabilistic Motion Model

To improve realism and robustness over long trajectories where odometry drift accumulates, we implemented a probabilistic motion model that decomposes the robot movement into translation and rotation components each with appropriate noise modeling:

- $\delta_{\textbf{rot1}}$**:** The initial rotation needed to align the robot with the direction of motion

$$\delta_{rot1} = \text{atan2}(\Delta y, \Delta x) - \theta_{t-1} + \mathcal{N}(0, \sigma_{rot1})$$

- $\delta_{\textbf{trans}}$**:** The linear distance the robot travels along the aligned direction

$$\delta_{trans} = \sqrt{\Delta x^2 + \Delta y^2} + \mathcal{N}(0, \sigma_{trans})$$

- $\delta_{\textbf{rot2}}$**:** The final rotation needed to reach the desired orientation after translation

$$\delta_{rot2} = \theta_t - \theta_{t-1} - \delta_{rot1} + \mathcal{N}(0, \sigma_{rot2})$$

Each noise term is modeled as a zero-mean Gaussian distribution whose variance is a function of the motion magnitude. These variances increase proportionally to the robot's translational and rotational efforts, making the model sensitive to how aggressively the robot moves:

$$\begin{aligned}
\sigma_{rot1}^2 &= \alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2 \\
\sigma_{trans}^2 &= \alpha_3 \delta_{trans}^2 + \alpha_4 (\delta_{rot1}^2 + \delta_{rot2}^2) \\
\sigma_{rot2}^2 &= \alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2
\end{aligned}$$

The final pose update is computed by applying the noisy motion to the current particle pose (x, y, θ), resulting in a new hypothesized pose (x', y', θ'):

$$\begin{aligned}
x' &= x + \delta_{trans} \cdot \cos(\theta + \delta_{rot1}) \\
y' &= y + \delta_{trans} \cdot \sin(\theta + \delta_{rot1}) \\
\theta' &= \text{normalize}(\theta + \delta_{rot1} + \delta_{rot2})
\end{aligned}$$

This motion model enables each particle in the filter to explore slightly different possible robot poses, reflecting the uncertainty in motion execution.

C. Kabsch Algorithm for Map Alignment

The Kabsch algorithm is used to compute the optimal rotation and translation that best align two sets of corresponding points in 3D space, minimizing the root mean square deviation (RMSD) between them. In our Fast SLAM framework, we employ the Kabsch algorithm primarily to align estimated landmark positions from particles with ground truth for evaluation.

Given two corresponding point sets $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ (estimated landmarks) and $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ (ground truth), the goal is to find the optimal rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ that minimize:

$$\text{RMSD} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \|\mathbf{q}_i - (\mathbf{R}\mathbf{p}_i + \mathbf{t})\|^2}$$

a) Centroid Computation: Compute the centroids of both point sets:

$$\bar{\mathbf{P}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{p}_i, \quad \bar{\mathbf{Q}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{q}_i$$

b) Centering the Points:

$$\mathbf{p}_i' = \mathbf{p}_i - \bar{\mathbf{P}}, \quad \mathbf{q}_i' = \mathbf{q}_i - \bar{\mathbf{Q}}$$

c) Covariance and SVD: Construct the covariance matrix and apply singular value decomposition:

$$\mathbf{H} = \sum_{i=1}^{n} \mathbf{p}_i' \mathbf{q}_i'^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

d) Optimal Rotation and Translation:

$$\mathbf{R} = \mathbf{V}\text{diag}(1, 1, \det(\mathbf{V}\mathbf{U}^T))\mathbf{U}^T, \quad \mathbf{t} = \bar{\mathbf{Q}} - \mathbf{R}\bar{\mathbf{P}}$$

Once the optimal transformation is computed, we rotate and translate the estimated landmarks to align them with the ground truth, effectively bringing both into the same reference frame. This alignment is critical for quantitatively evaluating map accuracy. Since Fast SLAM relies on particles that maintain separate hypotheses, global drift often causes mismatches in absolute coordinates. The Kabsch alignment corrects for this drift, ensuring shape-consistent comparisons. This allows accurate computation of RMSE between the estimated and ground truth landmark positions, independent of global frame discrepancies.

III. Implementation

A. Camera Model and Coordinate Transformation

We place our camera on board the Pionner to detect ArUco markers in the image plane and estimate their 3D positions in the robot frame. This coordinate transformation process relies on a camera projection model, where 3D world points are mapped to 2D image coordinates via:

$$\tilde{U} = M_{\text{int}} M_{\text{ext}} \tilde{X}_w = P\tilde{X}_w$$

- $\tilde{U} = [u, v, 1]^T$: homogeneous image coordinates,
- $\tilde{X}_w = [x_w, y_w, z_w, 1]^T$: 3D point in the world frame,
- $M_{\text{int}} = K$: intrinsic matrix (camera calibration),
- $M_{\text{ext}} = [R \mid t]$: extrinsic matrix (camera pose in world),
- $P = M_{\text{int}} M_{\text{ext}}$: full camera projection matrix.

To retrieve 3D information from the 2D image, we invert the transformation chain:

$$\tilde{X}_c = M_{\text{int}}^{-1}\tilde{U}, \quad \tilde{X}_w = M_{\text{ext}}^{-1}\tilde{X}_c$$

This allows us to project image coordinates back into the robot frame for mapping and localization.

B. ArUco Landmark Detection Implementation

The detection pipeline is implemented using the OpenCV ArUco library. After converting the captured image from camera feed to grayscale,improving detection performance, markers are detected and identified, using OpenCV's `aruco.detectMarkers` function to detect marker corners and extract their IDs. This allows us to estimate their 3D pose relative to the camera. The resulting translation vector is then transformed into the robot frame using the fixed camera-to-robot transformation explained above. Finally, Cartesian coordinates are converted to polar form to yield a range and bearing measurement relative to the robot, which is used as an observation in the Fast SLAM update.

$$r = \sqrt{x^2 + y^2}, \quad \theta = \arctan 2(\frac{x}{y})$$

Where r is the range to the landmark, and θ is the bearing relative to the robot's forward axis.

This procedure enables the robot to identify the landmark's ID and determine its relative position in the robot's coordinate frame. These range and bearing measurements serve as observation inputs for the Fast SLAM algorithm.

Fig. 1. ArUco marker detection from the robot's camera. Each marker is identified with its estimated angle (green) and distance (purple) relative to the robot's odometry frame.

C. FastSLAM Implementation

1) Particle Initialization: Each particle starts with a pose at the origin and an empty map of landmarks. This initial pose defines the reference frame for the particle. The weight is initialized to 1.0 and later updated based on how well the particle's predictions align with sensor observations. Motion noise parameters are also set during initialization to account for uncertainty in the robot's movements.

2) Motion Model (Particle Prediction Step): The motion model performs the prediction step of the particle filter by updating each particle's pose using odometry data. It computes the change in position and orientation, adds Gaussian noise to account for motion uncertainty, and updates the particle's pose accordingly. This allows each particle to evolve over time while reflecting the inherent unpredictability of robot motion.

Each particle k maintains a pose estimate $x_t^{[k]}$ and individual landmark estimates $\{\mu_j^{[k]}, \Sigma_j^{[k]}\}$ for each observed landmark j. At each time step, the new pose is sampled from the motion model:

$$x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$$

3) Landmark Handling in FastSLAM:

a) Measurement Model: Each ArUco marker j detected by the robot's sensors is represented by its global position $\mathbf{m}_j = [x_j, y_j]^T$. Given the robot pose $\mathbf{x}_t = [x, y, \theta]^T$, the observation model relates the robot's position to the expected measurement of landmark j:

$$\mathbf{h}_j(\mathbf{x}_t, \mathbf{m}_j) = \begin{bmatrix} d \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{(x_j - x)^2 + (y_j - y)^2} \\ \text{atan2}(y_j - y, x_j - x) - \theta \end{bmatrix} \quad (1)$$

where d represents the Euclidean distance between the landmark and the robot, and ϕ represents the bearing angle to the landmark relative to the robot's orientation.

b) Landmark Update Process: The landmark update process involves detecting landmarks within the sensor's range and field of view, then either updating an existing landmark or creating a new landmark if it has not been observed before, using the Extended Kalman Filter (EKF). For each detected landmark, the system performs the following steps:

- Jacobian Computation: The Jacobian matrix of the measurement function with respect to the landmark position is crucial for the EKF update. It is computed as:

$$\mathbf{H}_j = \begin{bmatrix} \frac{\partial d}{\partial x_j} & \frac{\partial d}{\partial y_j} \\ \frac{\partial \phi}{\partial x_j} & \frac{\partial \phi}{\partial y_j} \end{bmatrix} = \begin{bmatrix} \frac{\Delta x}{\sqrt{q}} & \frac{\Delta y}{\sqrt{q}} \\ -\frac{\Delta y}{q} & \frac{\Delta x}{q} \end{bmatrix} \tag{2}$$

  where $\Delta x = x_j - x$, $\Delta y = y_j - y$, and $q = \Delta x^2 + \Delta y^2$. This Jacobian quantifies how small changes in the landmark position affect the predicted measurements.

- Innovation Covariance: The measurement prediction covariance combines the landmark position uncertainty with the sensor noise:

$$\mathbf{S}_j^{[k]} = \mathbf{H}_j \mathbf{\Sigma}_{j,t-1}^{[k]} \mathbf{H}_j^T + \mathbf{Q} \tag{3}$$

  where $\mathbf{\Sigma}_{j,t-1}^{[k]}$ is the covariance matrix of landmark j's position estimate for particle k, and $\mathbf{Q}$ is the measurement noise covariance matrix based on the sensor's noise characteristics.

- EKF Landmark Correction Step: For each observed landmark, the standard EKF update equations are applied:

$$\mathbf{K}_j^{[k]} = \mathbf{\Sigma}_{j,t-1}^{[k]} \mathbf{H}_j^T (\mathbf{S}_j^{[k]})^{-1} \tag{4}$$

$$\boldsymbol{\mu}_{j,t}^{[k]} = \boldsymbol{\mu}_{j,t-1}^{[k]} + \mathbf{K}_j^{[k]} \boldsymbol{\nu}_j^{[k]} \tag{5}$$

$$\mathbf{\Sigma}_{j,t}^{[k]} = (\mathbf{I} - \mathbf{K}_j^{[k]} \mathbf{H}_j) bol \Sigma_{j,t-1}^{[k]} \tag{6}$$

  where $\mathbf{K}_j^{[k]}$ is the Kalman gain, and $\boldsymbol{\nu}_j^{[k]} = \mathbf{z}_t - \hat{\mathbf{z}}_{j,t}^{[k]}$ is the innovation vector representing the difference between the actual observation and the predicted measurement.

4) Particle Weight Update: The particle weights are updated based on the likelihood of the observed measurements given each particle's landmark estimates. For each particle k and observed landmark j, the measurement likelihood is computed using the multivariate Gaussian distribution:

$$w_{j,t}^{[k]} = \frac{1}{\sqrt{(2\pi)^n |\mathbf{S}_j^{[k]}|}} \exp\left(-\frac{1}{2} (\boldsymbol{\nu}_j^{[k]})^T (\mathbf{S}_j^{[k]})^{-1} \boldsymbol{\nu}_j^{[k]} \right) \tag{7}$$

where n is the measurement dimension (typically 2 for range-bearing sensors), and $|\mathbf{S}_j^{[k]}|$ is the determinant of the innovation covariance matrix.

By continuously updating landmark estimates and particle weights through this process, FastSLAM maintains a set of particles that best represent both the robot's trajectory and the map of the environment, effectively handling the inherent uncertainties in simultaneous localization and mapping.

5) Resampling: The implemented algorithm triggers resampling only when the effective sample size falls below half the total particle count, specifically when $n_{eff} < \frac{M}{2}$. The effective number of particles is computed as:

$$n_{eff} = \frac{1}{\sum_{m=1}^{M} (w_t^{(m)})^2}$$

This criterion prevents unnecessary resampling by activating the process only when weight concentration becomes problematic, indicating that a few particles dominate the importance distribution while others carry negligible weights.

The resampling strategy employed is systematic resampling with low variance characteristics. This approach maintains stochastic selection where each particle's probability of inclusion in the new set correlates directly with its normalized weight. However, the method preserves diversity by ensuring that even particles with relatively small weights retain some probability of selection, preventing complete elimination of potentially valuable hypotheses.

The particle population remains fixed throughout the resampling operation. When particles with minimal weights are excluded from the new generation, higher-weighted particles compensate by appearing multiple times in the resampled set. This mechanism concentrates computational effort on the most promising trajectory hypotheses while maintaining the algorithm's ability to explore alternative solutions through the retained diversity of the particle ensemble.

D. Dynamic Measurement Covariance Matrix

In computer vision-based SLAM systems, measurement noise characteristics vary based on marker visibility, distance, and geometric properties. This section presents a dynamic covariance matrix that adapts to these varying conditions through error propagation analysis.

1) Pixel-Level Error Modeling: Pixel detection accuracy depends on marker size in the image. The pixel error model is:

$$\sigma_{\text{pixel}} = \sigma_{\text{base}} + \frac{k}{\text{marker}_{\text{pixels}}} \tag{8}$$

where σ_{base} is the system's noise floor, k is a calibration parameter, and $\text{marker}_{\text{pixels}}$ is the marker size in pixels.

2) Angular Measurement Uncertainty: The bearing angle to a detected marker is computed using the pinhole camera model:

$$\theta = \arctan\left(\frac{\mu - c_x}{f} \right) \tag{9}$$

The angular uncertainty is derived through error propagation:

$$\sigma_\theta = \left| \frac{\partial \theta}{\partial \mu} \right| \cdot \sigma_{\text{pixel}} = \frac{\sigma_{\text{pixel}}}{f} = \frac{\sigma_{\text{base}}}{f} + \frac{k}{f \cdot \text{marker}_{\text{pixels}}} \tag{10}$$

For practical implementation, the bearing covariance is modeled as:

$$\sigma_{\text{bearing}}(\text{marker}_{\text{pixels}}) = \alpha + \frac{\beta}{\text{marker}_{\text{pixels}}} \tag{11}$$

3) Range Measurement Uncertainty: Range estimation relies on the known physical size of markers and their apparent size in the image:

$$\text{distance} = \frac{\text{marker} \cdot f}{\text{pixel}} \tag{12}$$

The range uncertainty is derived through error propagation, considering that pixel errors have quadratic effects on distance:

$$\sigma_d = \left| \frac{\partial d}{\partial \text{pixel}} \right| \cdot \sigma_{\text{pixel}} = \frac{d^2 \cdot \sigma_{\text{pixel}}}{\text{marker}_{\text{pixels}} \cdot f} \tag{13}$$

For computational efficiency, the range covariance is modeled as a quadratic function:

$$\sigma_{\text{range}}(\text{distance}) = \alpha \cdot d^2 + \beta \tag{14}$$

The dynamic model accounts for perspective geometry effects where pixel errors amplify quadratically with distance, and larger markers enable better sub-pixel accuracy through improved signal-to-noise ratios in corner detection algorithms. The final dynamic measurement covariance matrix adapts to observation quality by combining distance and marker size dependencies:

$$\mathbf{Q} = \begin{bmatrix} \sigma^2_{\text{range}}(\text{distance}) & 0 \\ 0 & \sigma^2_{\text{bearing}}(\text{marker}_{\text{pixels}}) \end{bmatrix} \tag{15}$$

This dynamic approach automatically weights measurements based on their expected accuracy, improving filter performance by trusting high-quality observations from close, large markers while appropriately downweighting distant or small marker detections.

IV. Experimental Results

A. Experimental Setup and Ground Truth

Our FastSLAM implementation was evaluated using real-world data collected with a Pioneer 3DX robot equipped with a web camera in an indoor laboratory environment. The experimental setup consisted of 10 ArUco markers strategically placed throughout the corridor, with the robot following distinct trajectory patterns recorded as ROSbags.

Ground truth data was established through manual measurement and mapping of marker positions relative to the robot's starting position. The coordinate system was carefully calibrated, with trajectory ground truth consisting of waypoints representing the robot's true path. Camera calibration was performed using a standard checkerboard pattern to ensure accurate ArUco pose estimation.

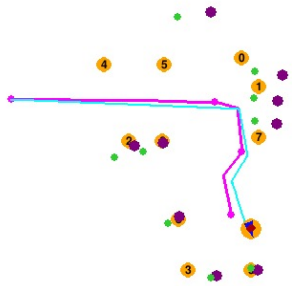

Fig. 2. FastSLAM experimental results showing trajectory estimation and landmark mapping performance of a ROSBag

Legend:

- Blue Line: Odometry-based trajectory (wheel encoders only)
- Purple Line: Ground truth trajectory
- Yellow Markers (0-10): Ground truth ArUco marker positions
- Purple Markers: FastSLAM estimated marker positions
- Green Markers: Kabsch alignment reference points

B. Evaluation Metrics

Absolute Trajectory Error (ATE):

$$\text{ATE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \min_j ||p_i^{SLAM} - p_j^{GT}||^2}$$

where p_i^{SLAM} represents SLAM-estimated positions and p_j^{GT} represents ground truth positions.

Mean Squared Position Error (MSP):

$$\text{MSP} = \frac{1}{N} \sum_{i=1}^{N} \min_j ||p_i^{est} - p_j^{GT}||^2$$

Root Mean Square Error (RMSE) after Kabsch Alignment:

$$\text{RMSE} = \sqrt{\frac{1}{M} \sum_{j=1}^{M} ||L_j^{est} - (R \cdot L_j^{GT} + t)||^2}$$

Effective Sample Size and Real-Time Performance:

$$n_{eff} = \frac{1}{\sum_{k=1}^{K} (w_k^{norm})^2}, \quad \text{RTF} = \frac{t_{dataset}}{t_{wall}}$$

C. Performance Analysis

The experimental evaluation was conducted using 40 particles over a dataset spanning approximately 120 seconds with 1,918 total updates. The system achieved a successfully identified and tracked all visible Aruco markers in the environment.

Trajectory Estimation Accuracy: The FastSLAM algorithm demonstrated robust trajectory estimation with an average ATE of 0.144 meters compared to ground truth. While the current instantaneous ATE reached 0.256 meters during

challenging sections of the trajectory, the system maintained consistent performance throughout the experiment. The MSP metric showed an average value of 0.523 m², indicating reasonable position estimation accuracy given the challenging indoor environment and visual-only sensing modality.

Landmark Mapping Performance: Landmark estimation accuracy was evaluated using Kabsch alignment to eliminate global coordinate frame differences. The system achieved an average RMSE of 0.095 meters with a current RMSE of 0.290 meters. The relatively low average error demonstrates the algorithm's ability to maintain accurate landmark estimates over time, while the higher instantaneous values reflect the natural uncertainty fluctuations during dynamic robot motion.

D. Computational Efficiency

The core FastSLAM algorithm demonstrated efficient computational performance, with the actual SLAM processing requiring only 13.04 ± 9.08 ms per update cycle. However, the overall system achieved a Real-Time Factor (RTF) of 0.532×, processing data at an effective rate of 15.96 Hz against the target 30 Hz. This performance gap reveals an important distinction between core algorithm efficiency and total system overhead.

Core SLAM Algorithm Performance: Timing analysis of the essential SLAM components revealed efficient processing: - Motion model update: 3.77 ± 3.03 ms - Landmark processing: 8.28 ± 4.40 ms - Resampling: 0.44 ± 4.60 ms - Weight calculation: 0.01 ± 0.02 ms - **Total core SLAM:** 13.04 ± 9.08 ms per cycle

These timing results indicate that the FastSLAM algorithm itself operates at approximately 77 Hz (1000ms ÷ 13ms), well above real-time requirements. The motion model and landmark updates dominate computational cost, as expected for particle filter SLAM.

Metrics and Evaluation Overhead: The performance bottleneck arises from the comprehensive metrics evaluation system rather than the core SLAM algorithm. The difference between the 13ms SLAM processing time and the observed 62.7ms total cycle time (corresponding to 15.96 Hz) indicates approximately 50ms of overhead per cycle from:

- Absolute Trajectory Error (ATE) calculation using KDTree nearest-neighbor search across trajectory points
- Mean Squared Position (MSP) computation requiring distance calculations to all ground truth positions
- Kabsch alignment with Singular Value Decomposition (SVD) for landmark registration
- Real-time trajectory interpolation and ground truth comparison
- Comprehensive timing statistics collection and periodic file I/O operations

This metrics overhead was intentionally implemented to provide detailed performance analysis during evaluation. In a production deployment, disabling the extensive metrics collection would allow the system to operate much closer to the theoretical 77 Hz capability, easily achieving real-time performance requirements.

The results demonstrate that FastSLAM can indeed operate in real-time when computational resources are focused on the core algorithm rather than evaluation infrastructure. The observed performance limitation is an artifact of the experimental setup rather than an algorithmic constraint.

E. Limitations and Observations

Several limitations were observed during evaluation. The Real-Time Factor below 1.0 indicates computational overhead that could impact performance in time-critical applications. The instantaneous error metrics showed higher variance than average values, suggesting sensitivity to rapid robot motion and challenging viewing angles.

The reliance on visual markers limits applicability to environments where such fiducials cannot be deployed. The current implementation requires known data association through ArUco IDs, which simplifies the correspondence problem but reduces generalizability to natural landmark scenarios.

The particle count of 40 proved sufficient for this environment scale and complexity, demonstrating that effective SLAM can be achieved without extensive computational resources when using structured landmarks with known correspondence.

Evaluating the accuracy of our results was hard due to the lack of reliable ground truth, measured with little precision with a tape measure. Especially in the trajectory ground truth where we left a trail of papers marking robot passage, and measured distances from one paper to the next, leading to cumulative errors. This made it challenging to confirm whether the estimated trajectory and map were correct. We relied on visual consistency and qualitative alignment with expected marker positions to assess performance.

The visualization demonstrates FastSLAM's ability to maintain accurate landmark estimates despite significant odometry drift. The blue trajectory shows cumulative errors typical of wheel odometry, while the close correspondence between ground truth (yellow) and estimated (purple) marker positions validates the algorithm's mapping performance. Kabsch alignment (green markers) enables fair trajectory comparison by accounting for coordinate frame differences.

V. Conclusions

This project successfully implemented a FastSLAM-based system for real-time localization and mapping using odometry and ArUco marker detection.

Although the system remained real-time up to 100 particles, the configuration with 40 to 50 particles provided the best balance between robustness, accuracy, and computational efficiency.

Overall, the project gave us a solid understanding of FastSLAM in practice and showed its effectiveness using visual and odometry data,under real-world conditions.