Simultaneous Localization and Mapping (SLAM) with FastSLAM using the Pioneer

Pedro Reis
107550
pedro.c.reis@tecnico.ulisboa.pt

Beatriz Marrucho Ferreira
106901
beatriz.marrucho.ferreira@tecnico.ulisboa.pt

Gonçalo Pinheiro
107543
goncalo.q.pinheiro@tecnico.ulisboa.pt

José Pedro Garcia
106356
jose.pedro.gomes.garcia@tecnico.ulisboa.pt

Group 27

***Abstract*—This project report presents the implementation and evaluation of the FastSLAM algorithm on the Pioneer P3-DX to address the SLAM problem, using ArUco markers as landmarks. A microsimulator and an ArUco detection algorithm were developed to test and tune the algorithm. The used approach combines data from the robot's onboard odometry and camera (detecting ArUco markers) to improve localization and map accuracy. To validate the method's performance, the system was tested in both simulated and real-world environments. The experimental results demonstrate the applicability of FastSLAM for accurate, real-time map construction and robot localization. The results were reliable and consistent across multiple runs using the same and similar ROSBags.**

I. Introduction and Motivation

The simultaneous localization and mapping, or *SLAM* for short, problem consists of a moving robot trying to recover a map of the environment it is in while simultaneously estimating its localization and orientation (pose) within the said map. [1] Fast-SLAM is a more modern approach to solving the SLAM problem more efficiently by breaking it down into separate, smaller problems: a robot localization problem and a collection of landmark estimation problems. [2]

Given this, the principal objective of this project is to make an implementation of the Fast-SLAM algorithm with known correspondence on the *Pioneer P3-DX* robot, with the use of *ArUco* markers as landmarks because of their uniquely identifiable features. The goal is to develop a system capable of generating an accurate map and estimating the robot's pose using a camera and odometry data.

II. Methods and Algorithms

A. Fast-SLAM Algorithm - Description

Fast-SLAM with known data association is a particle filter-based algorithm for solving the SLAM problem, usually used in environments with many landmarks, such as our project will have. It improves over the EKF-SLAM because the latter becomes too computationally expensive as the map grows. Furthermore, Fast-SLAM leverages a particle filter to represent multiple hypotheses of the robot's trajectory and uses independent Kalman filters for each landmark within each particle.

In Fast-SLAM, each particle maintains its map estimate. This simplifies the mapping process because, given the robot's path, the estimates of the landmarks become independent. After the new observation and the data association are made, the correspondence between the observation and the landmark is known, and the relevant landmarks in each particle are updated. In addition, the weight of each particle is adjusted based on how well its predicted observations match the real values, and the resampling step makes sure that the particles with more weight, which represent more probable hypotheses, are selected. This approach allows Fast-SLAM to effectively maintain accurate maps and robot trajectories, even in high-dimensional environments.

B. Fast-SLAM algorithm - Detailed Steps

1) Particle Filter and Robot Path: Each particle samples a new robot pose based on the previous pose control input, using the motion model

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

This model incorporates uncertainty in motion and control input u_t, allowing the algorithm to represent the distribution over possible trajectories. This formulation is critical in terms of tracking multiple hypotheses of the robot's path in the presence of sensor noise.

2) Motion Model: The motion model for Fast-SLAM predicts a new pose for the robot based on its previous pose and the control input. Usually it includes a kinematic model similar to:

$$x_t = x_{t-1} + v \cdot \Delta t \cdot \cos(\theta_{t-1}) + \mathcal{N}(0, \sigma_x^2) \tag{1}$$

$$y_t = y_{t-1} + v \cdot \Delta t \cdot \sin(\theta_{t-1}) + \mathcal{N}(0, \sigma_y^2) \tag{2}$$

$$\theta_t = \theta_{t-1} + \omega \cdot \Delta t + \mathcal{N}(0, \sigma_\theta^2) \tag{3}$$

Where v is the linear velocity, w is the angular velocity, and $\mathcal{N}$ represents the Gaussian noise or the motion uncertainty.

3) EKF-Based Landmark Estimation: The landmarks are updated based on known data association. Within each particle, the map is updated using EKFs for each landmark. If a landmark c_t is observed for the first time, its position and covariance are initialized, respectively, as:

$$\mu_{c_t,t}^k = h^{-1}(z_t, x_t^k) \tag{4}$$

$$\Sigma_{c_t,t}^k = (H^{-1})^T Q_t H^{-1} \tag{5}$$

Here h^{-1} is the inverse measurement model, and H is the Jacobian of the observation function$h$. For previously observed landmarks, the predicted measurement is:

$$\hat{z}_t = h(\mu_{c_t,t-1}^k, x_t^k) \tag{6}$$

The innovation covariance and Kalman gain are, respectively, computed as:

$$Q = H\Sigma_{c_t,t-1}^k H^T + Q_t \tag{7}$$

$$K = \Sigma_{c_t,t-1}^k H^T Q^{-1} \tag{8}$$

The updated mean and covariance of the landmark estimate are, respectively:

$$\mu_{c_t,t}^k = \mu_{c_t,t-1}^k + K(z_t - \hat{z}_t) \tag{9}$$

$$\Sigma_{c_t,t}^k = (I - KH)\Sigma_{c_t,t-1}^k \tag{10}$$

4) Importance Weighting: Each particle is assigned an importance weight based on the likelihood of the observation given the predicted measurement:

$$w^k = |2\pi Q|^{-1/2} e^{\left(-\frac{1}{2}(z_t-\hat{z}_t)^T Q^{-1}(z_t-\hat{z}_t)\right)} \tag{11}$$

Particles that better explain the observation receive higher weights, increasing the likelihood of being selected in the next resampling.

5) Resampling: To prevent particle depletion and maintain a representative sample of the posterior distribution, resampling is done. Particles are selected with replacement, with probabilities proportional to their weights. This step ensures that computational resources are focused on the credible hypotheses.

However, excessive resampling can reduce diversity in the particle set, so it is typically triggered only when the effective number of particles drops below a certain threshold.

C. Landmarks - Description

In our project, we used ArUco markers as visual landmarks to help in the implementation of the algorithm. After the camera's calibration, it is easy for it to detect these high-contrast markers and consequently also uniquely identify them. The ArUco markers were statically placed throughout the robot's workspace; this way, the robot can make an accurate estimate of its position and also update its map as it moves through the environment.

D. Kabsch Algorithm - Bonus

To fairly evaluate the accuracy of estimated landmark positions, we use the Kabsch algorithm to align the estimated map with the ground-truth before computing error metrics. This step is important because our SLAM system does not estimate global pose but rather builds a map relative to the robot's initial frame, which can differ in orientation and position from the true frame.

The Kabsch algorithm computes the optimal rigid transformation (rotation + translation, no scaling) that minimizes the mean squared error between two sets of corresponding points. In our case, it aligns the estimated landmark positions to their known ground-truth positions.

The steps of the algorithm are:

1) Center both point sets by subtracting their centroids (mean positions).

2) Compute the covariance matrix between the centered sets.

3) Apply Singular Value Decomposition (SVD) to derive the optimal rotation matrix.

4) Calculate the translation vector needed to align the centroids.

5) Transform the estimated points using the computed rotation and translation.

This transformation ensures that SSE and RMSE are measured in a shared reference frame, making the evaluation of mapping accuracy meaningful and consistent, regardless of any global offset in the estimated map.

III. Implementation

A. Landmarks - ArUco Detection

This project, ArUco detection and pose estimation, was implemented with the help of the *OpenCV* library in Python. The purpose of this algorithm is to detect ArUco markers in a camera feed and estimate their pose for later use.

Initially, the program captures frames from a calibrated camera. Each image is converted to a greyscale to simplify the process and reduce the complexity. The algorithm then uses the *OpenCV* ArUco module to detect markers and previously retrieve their IDs.

Once a marker is detected, the pose (position and orientation relative to the camera) is estimated. This is done using the camera's intrinsic parameters and the known physical size of the marker. A translation vector gives the position, and the orientation is derived from the rotation vector.

To better visualize the detection, the program overlays onto the video feed. The distance between the marker and the camera is calculated using the translation vector:

$$r = \sqrt{x^2 + y^2 + z^2} \tag{12}$$

Additionally, the horizontal orientation (bearing) relative to the camera is computed as

$$\theta = \text{atan2}(x, y) \tag{13}$$

Both the pose and the marker's IDs are published via ROS topics, allowing other nodes to use this information. After the

program finishes, a 2D plot of the marker position in the $XZ-plane$.

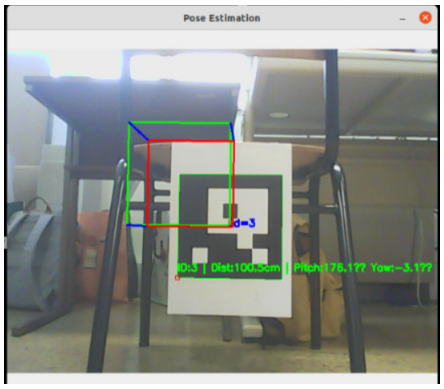

Fig. 1. ArUco detection through the camera feed

B. Microsimulator

For testing the algorithm developed, our group, created a microsimulator. This does not rely on real-world data; instead, it uses values of odometry, bearing, and range that closely resemble the real values.

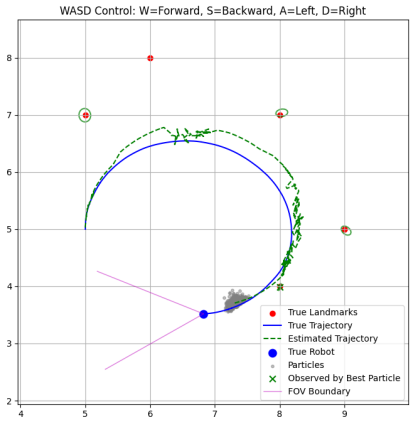

Fig. 2. Microsimulator final graphs.

Image 2 shows a result of the microsimulator. We have a map with the landmarks, predicted and actual trajectory of the robot, and the particles, all at the end of a circular path.

C. Fast-SLAM - Algorithm Implementation

The implementation of the Fast-SLAM algorithm in this project follows the structure below:

1) Particle Initialization: The particles are initialized based on the robot's initial pose, which is retrieved from the ROS topic `/pose`. Within each particle, we have a copy of the robot's pose vector (x, y, θ), an empty map where we will store landmark estimation information, and the initial weight of 1.0.

Furthermore, each particle independently maintains its hypotheses of the environment and the robot's trajectory. This will guarantee robustness against noisy observations.

2) Motion Model: The motion model updates each particle's pose based on odometry measurements. These changes, in position and orientation, are computed using the provided control inputs (linear and angular velocity). Besides that, Gaussian noise is added to both of these velocities, with the standard deviations proportional to their magnitudes. With these noise estimations added, the kinematic model adjusts the particle's pose, (x, y, θ); if the angular velocity, w, is significant, it is used a circular motion, but if not, it is applied a straight-line movement.

Let the robot pose at time t and the control input be:

$$\mathbf{x}_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}; \quad \mathbf{u}_t = \begin{bmatrix} v \\ \omega \end{bmatrix}$$

where:

- v is the linear velocity,
- ω is the angular velocity.

The motion model includes proportional Gaussian noise:

$$v' = v + \epsilon_v, \quad \epsilon_v \sim \mathcal{N}(0, \sigma_v^2)$$
$$\omega' = \omega + \epsilon_\omega, \quad \epsilon_\omega \sim \mathcal{N}(0, \sigma_\omega^2)$$

where:

$$\sigma_v = \texttt{motion\_noise[0]} \cdot \max(|v|, 0.1)$$
$$\sigma_\omega = \texttt{motion\_noise[1]} \cdot \max(|\omega|, 0.01)$$

Therefore, we can apply the following equations to the motion model described above:

if $|w| > 10^{-6}$ **then**
  $\theta_{\text{new}} \leftarrow \text{wrap\_angle}(\text{pose}[2] + w \cdot dt)$
  $\theta_{\text{mid}} \leftarrow (\text{pose}[2] + \theta_{\text{new}})/2$
  $\text{distance} \leftarrow v \cdot dt$
  $x \leftarrow \text{pose}[0] + \text{distance} \cdot \cos(\theta_{\text{mid}})$
  $y \leftarrow \text{pose}[1] + \text{distance} \cdot \sin(\theta_{\text{mid}})$
else
  $x \leftarrow \text{pose}[0] + v \cdot \cos(\text{pose}[2]) \cdot dt$
  $y \leftarrow \text{pose}[1] + v \cdot \sin(\text{pose}[2]) \cdot dt$
  $\theta_{\text{new}} \leftarrow \text{pose}[2]$
end if
return $[x, y, \theta_{\text{new}}]$

In the pseudocode above, `dt` is the time step. The noise added to the linear and angular velocities is, respectively, named `v_noise` and `w_noise`, and their standard deviations are controlled by `motion_noise` parameters. The `wrap_angle` function ensures that the angle remains between $[-\pi, \pi]$. The variables x, y, and `theta_new` are the updated position and orientation of the particle. The current pose of the particle is given by (pose[0], pose[1], pose[2]).

3) Landmark Processing: The landmark data is received from the ROS topics `/markers_pose` and `/markers_id`. Each particle updates its set of landmarks based on the observations made by the robot's sensors.

During the update process, the system detects each landmark and either updates an existing one or initializes a new one. If the **landmark is new,** it is initialized using the observed range and bearing, but if the **landmark is already known,** the landmark is updated via an Extended Kalman Filter (EKF), which predicts the expected observation ($\hat{z}$) using the current estimate, computes the Jacobian of the measurement model, updates the mean and covariance using the Kalman gain, and applies the angular thresholding and Mahalanobis distance filtering to reject inconsistent measurements.

This update of the already known landmarks includes the following: Predicting the expected observation, $\hat{z}$, using the current estimate of the landmark's position and the particle's pose. The predicted measurement z_{pred} for a landmark at (landmark[0], landmark[1]) from a particle at (pose[0], pose[1], pose[2]) is:

1: $dx \leftarrow landmark[0] - pose[0]$
2: $dy \leftarrow landmark[1] - pose[1]$
3: $r \leftarrow \text{hypot}(dx, dy)$
4: bearing $\leftarrow$ wrap_angle(atan2$(dy, dx) - pose[2]$)
5: $z\_pred \leftarrow [r, \text{bearing}]$

In addition, it also computes the Jacobian matrix H of the measurement model. This will be essential for linearizing the non-linear observation function. The matrix is given by:

$$H = \begin{bmatrix} \frac{dx}{q} & \frac{-dy}{q} \\ \frac{dy}{q} & \frac{dx}{q} \end{bmatrix}$$

Here, dx and dy are the differences between the particle's pose and the landmark's position, and $q = dx^2 + dy^2$.

To take the uncertainties into account in both the landmark's position and the sensor noise, the system calculates the prediction covariance S, using the formula: $S = H\Sigma H^T + R$. Where Σ is the covariance matrix of the landmark's position estimate, and R is the measurement noise covariance matrix defined as $R = np.diag(measurement\_noise)^2$.

Following this, the mean and covariance of the landmark are updated using the **Kalman gain**. This is a process that integrates new measurement information to improve the landmark's estimated position and reduce uncertainty. To maintain the integrity of the estimation, the system filters out inconsistent measurements. This can be done in two different ways: angular thresholding and Mahalanobis distance filtering. With this, we make sure only reliable measurements are used in the updates.

In the end, each particle's weight is **updated based on the likelihood of the observed measurment**, using the following Gaussian probability distribution:

$$w = \frac{1}{\sqrt{(2\pi)^2|S|}} \exp(-\frac{1}{2}(z - \hat{z})^T S^{-1}(z - \hat{z})) \tag{14}$$

In the formula above, z represents the difference between the observed and the predicted measurements. This weighting mechanism assings higher weights to the particles that have better correspondence with the sensor observations.

4) Resampling: To prevent degeneracy, where only a few particles dominate the representation, resampling is triggered when the effective number of particles (N_{eff}) falls below the threshold previously defined. This number is calculated with:

$$N_{eff} = \frac{1}{\Sigma_{m=1}^{M}(w_t^{(m)})^2} \tag{15}$$

This ensures that the resampling is only performed when the importance factor of some particles is significantly higher than others.

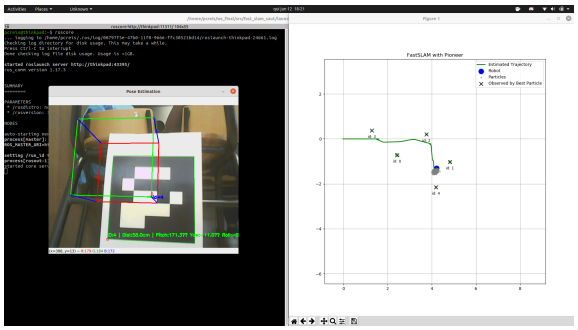

Fig. 3. Algorithm implementation - Final Result

In image 3, we have a camera feed side by side with the final result of the Fast-SLAM algorithm implementation.

IV. Experimental Results

A. ROSBags

To see if our implementation worked, the group recorded several ROSBags. Different ROSBags trajectories were made in different environments, but every one with 5 landmarks.

Firstly, used a linear trajectory up and down the fifth floor corridor of the north tower of IST, and also an L-shaped trajectory in the same corridors. We also tested in the LSDC4 room on the fifth floor, where we made a circular trajectory starting and ending almost at the same point. In this room, we made an L-shaped trajectory too.

B. Visualization of Results

For a first evaluation of the results obtained by the developed algorithm, no metrics were computed. This was done by having a general knowledge of the map the robot was on and comparing it to the camera feed and the *RViz* representation.

To make the best possible comparison, we took note of several key details: if in the final map the localization of all the landmarks made sense relative to each other in terms of distance and position, and if the trajectory presented in the final map was accurately represented, whether it was back and forth, L-shaped, or circular.

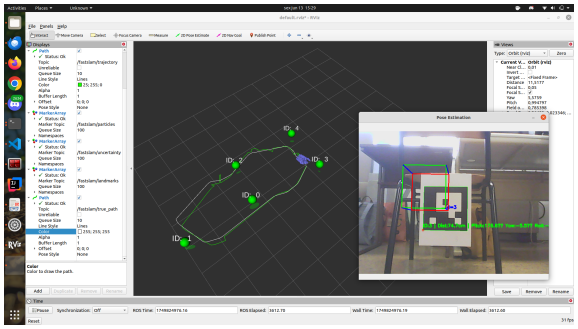

Fig. 4. Algorithm implementation - RViz observation

C. Ground-truth

We started by measuring the distance between the robot's initial position and the landmarks, given that we were able to trace the ground truth positions of each of the landmarks. Then we ran the algorithm and got the estimated positions from it, which we later used to fine-tune the parameters to obtain the best estimate possible.

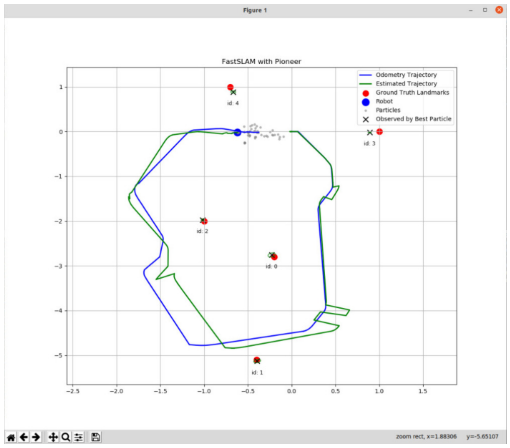

Fig. 5. Algorithm implementation - comparison with the Ground Truth

D. Fine-Tuning & Metrics

To improve the accuracy and robustness of our SLAM algorithm, we performed systematic experiments using the same set of recorded ROSBags while varying the following parameters:

1) Number of Particles – to balance accuracy and computational cost.

2) Measurement Noise Covariance – which influences EKF updates for landmarks.

3) Motion Model Noise – affecting the prediction of the robot's pose.

Each combination of parameters was tested to assess its effect on the quality of the resulting map.

We used the following quantitative metrics to compare the performance of each parameter configuration:

- **SSE (Sum of Squared Errors)** – the sum of squared distances between each estimated landmark (after rigid alignment via the Kabsch algorithm) and its corresponding ground-truth position. Lower values indicate better alignment and accuracy.

$$\text{SSE} = \sum_{i=1}^{N} \left\| \hat{l}_i - l_i \right\|^2 \tag{16}$$

Where $\hat{l}_i$ is the estimated position (aligned), and l_i is the ground truth.

- **RMSE (Root Mean Squared Error)** – provides an average error per landmark, offering a more intuitive scale of the mapping error.

$$\text{RMSE} = \sqrt{\frac{\text{SSE}}{N}} \tag{17}$$

The data below was obtained using the above metrics:

1) We varied the number of particles while keeping motion noise fixed at [0.1, 0.05] and measurement noise at [0.2, 0.1]. As shown, increasing the number of particles from 50 to 500 resulted in a steady improvement in accuracy, as indicated by a decreasing RMSE.

TABLE I
SSE AND RMSE FOR DIFFERENT PARTICLE COUNTS (ROSBAG1)

Particles	SSE	RMSE
50	0.0338	0.0823
100	0.0327	0.0809
200	0.0306	0.0783
500	0.0320	0.0800

TABLE II
SSE AND RMSE FOR DIFFERENT PARTICLE COUNTS (ROSBAG2)

Particles	SSE	RMSE
50	0.0059	0.0343
100	0.0053	0.0327
200	0.0058	0.0341
500	0.0053	0.0327

The improvement from 200 to 500 particles was marginal, suggesting that beyond 200 particles, the filter reaches a point of diminishing returns. Thus, 200 particles may offer a good balance between accuracy and computational cost.

2) We fixed the particle count at 500 and motion noise at [0.1, 0.05], and tested different values for measurement noise. Interestingly, the lowest RMSE (0.0797) occurred with the highest measurement noise [0.3, 0.3], while the lowest noise ([0.1, 0.05]) slightly worsened performance.

TABLE III
SSE AND RMSE FOR DIFFERENT MEASUREMENT NOISE VALUES (ROSBAG1)

Measurement Noise [R]	SSE	RMSE
[0.3, 0.3]	0.0317	0.0797
[0.2, 0.1]	0.0320	0.0800
[0.1, 0.05]	0.0338	0.0822

This suggests that overly confident measurements (i.e., low noise) can make the filter more sensitive to errors or outliers,

TABLE IV
SSE AND RMSE FOR DIFFERENT MEASUREMENT NOISE VALUES (ROSBAG2)

Measurement Noise [R]	SSE	RMSE
[0.3, 0.3]	0.0044	0.0297
[0.2, 0.1]	0.0053	0.0327
[0.1, 0.05]	0.0060	0.0347

while more conservative noise values help smooth out these effects. Therefore, moderate to high measurement noise may produce more stable landmark estimates.

3) With particle count and measurement noise fixed, we varied motion noise. The best RMSE (0.0762) was achieved with motion noise [0.2, 0.1], indicating that a slightly higher noise model can better accommodate unmodeled dynamics or real-world variability in odometry.

TABLE V
SSE AND RMSE FOR DIFFERENT MOTION MODEL NOISE VALUES (ROSBAG1)

Motion Model Noise [Q]	SSE	RMSE
[0.3, 0.3]	0.0322	0.0803
[0.2, 0.1]	0.0290	0.0762
[0.1, 0.05]	0.0320	0.0800

TABLE VI
SSE AND RMSE FOR DIFFERENT MOTION MODEL NOISE VALUES (ROSBAG2)

Motion Model Noise [Q]	SSE	RMSE
[0.3, 0.3]	0.0050	0.0317
[0.2, 0.1]	0.0054	0.0329
[0.1, 0.05]	0.0053	0.0327

Lower motion noise did not consistently improve accuracy, possibly because it underestimates uncertainty and leads to overconfident prediction steps.

V. CONCLUSIONS

A. ArUco Marker Detection Errors

During the landmark detection process, we encountered two significant limitations:

First, we were unable to obtain accurate bearing angles from the *ArUco* markers, which made it considerably more difficult to estimate correct poses. Upon investigation, we discovered that this issue stemmed from our initial camera calibration method. We had used the standard *ArUco* chessboard pattern, which resulted in poor bearing angle estimations. After switching to calibration with a *ChArUco* board, we observed a substantial improvement in the accuracy of the bearing angles.

Second, we faced issues with estimating the distance from the camera to the *ArUco* markers. This problem was compounded by conflicts between *OpenCV* dependencies and the *ArUco* library, which significantly hindered our development progress. Unfortunately, we were unable to resolve these issues in time, leading to incorrect landmark position estimates.

B. FastSlam limitations

While our FastSLAM implementation produced accurate and consistent results, it has a few natural limitations. It assumes known landmark IDs, which may be affected by marker visibility or occlusion. The system's performance depends on proper tuning of motion and measurement noise, though this was addressed through systematic testing. Loop closure is not yet implemented, which could lead to minor drift over long runs. Additionally, higher particle counts increase computational load, but offer better accuracy when needed. Overall, the system performs well within the intended conditions and offers a solid foundation for further improvements.

C. Final conclusion and comments

This project successfully demonstrated the implementation of the FastSLAM algorithm on the Pioneer P3-DX using visual landmarks (ArUco markers). Through a combination of simulation, real-world testing, and systematic parameter tuning, we were able to achieve accurate and consistent mapping and localization. The use of SSE and RMSE, along with Kabsch alignment, provided meaningful metrics for evaluating map quality across different parameter configurations.

We found that moderate values of measurement and motion noise led to better results, and that increasing the number of particles improved accuracy up to a practical limit. While some challenges were encountered with marker detection and pose estimation, the system showed robustness in most test environments.

Looking forward, further improvements could include implementing loop closure for global consistency, improving data association under ambiguity, and extending the system to larger-scale or dynamic environments. Overall, this project provided valuable hands-on experience in SLAM, probabilistic robotics, and real-world system integration.

REFERENCES

[1] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, E. Nebot, "FastSLAM: An Efficient Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association," Phil. Trans. Roy. Soc. London, vol. A, pp. –, 2003.

[2] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," Phil. Trans. Roy. Soc. London, vol. A, pp. –, 2002.

[3] Durrant-Whyte, Hugh, and Tim Bailey. "Simultaneous localization and mapping: part I." IEEE Robotics & Automation Magazine 13.2 (2006): 99-110.

[4] Montemerlo, Michael, Thrun, Sebastian. "FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics". Springer Tracts in Advanced Robotics, vol. 27, Springer-Verlag Berlin Heidelberg, 2007.