Marker-based FastSLAM on the AlphaBot2

João Gonçalves, Teresa Nogueira, Jacinto Pessoa and Mateus Ribeiro
Instituto Superior Técnico, University of Lisbon, Portugal

***Abstract*—This paper presents the implementation of the FastSLAM algorithm on the AlphaBot2 using a monocular camera and artificial ArUco markers. FastSLAM addresses the SLAM problem by combining particle filters and extended Kalman filters for efficient trajectory and landmark estimation. Our implementation highlights the advantages of FastSLAM as we also investigate the data association challenge by ignoring the unique IDs of ArUco markers, simulating environments with indistinguishable landmarks. Experimental results demonstrate that FastSLAM maintains reliable localization and mapping despite ambiguous data association. These results highlight FastSLAM's strengths and limitations in real-world conditions.**

***Index Terms*—AlphaBot2, ArUco Markers, FastSLAM, Data Association, Monocular Camera**

I. Introduction and Motivation

Simultaneous localization and mapping (SLAM) is a critical challenge in robotics, involving the concurrent construction of a map of an unknown environment while simultaneously determining the robot's pose within it. This dual problem is fundamental to enabling autonomous navigation, where a robot must operate without prior knowledge of its surroundings. The complexity of SLAM arises from the need to solve both the mapping and localization tasks concurrently, often in real-time, and with limited computational resources.

FastSLAM [1] is an influential algorithmic framework designed to address the SLAM problem efficiently, by leveraging particle filters to approximate the posterior distribution of a robot's trajectory and employing a set of extended Kalman filters (EKFs) to manage individual landmark estimations. This separation of the robot's path and the map's representation into distinct estimation problems reduces the overall computational complexity and enhances scalability [1].

In our work, we implement the FastSLAM 1.0 algorithm on the AlphaBot2 robot, utilizing artificial markers (ArUco). The cost-effective AlphaBot2 serves as an ideal testbed for real-world visual SLAM applications. We select ArUco markers due to their robustness and reliability in providing visual cues for accurate landmark recognition and localization [2].

Our implementation focuses not only on the integration of FastSLAM with the AlphaBot2 but also on addressing the data association problem, which is pivotal in SLAM. Data association involves correctly matching observed features with the corresponding landmarks in the map. Inaccurate data association can lead to significant errors in both the map and the robot's estimated trajectory [1], [3].

To explore the data association challenge, we experiment with neglecting the unique identifiers (IDs) of the ArUco markers. This scenario simulates conditions where landmarks cannot be distinctly identified, thereby forcing the algorithm to rely solely on spatial consistency and temporal tracking for association.

E-mails: {jrazevedogoncalves, maria.teresa.ramos.nogueira, jacinto.pessoa, mateusbandeiraribeiro}@tecnico.ulisboa.pt.

II. Theoretical Framework and Algorithms

In this section, we provide a brief explanation of the methods and algorithms used in our implementation. This includes a discussion about the FastSLAM algorithm and the approach to data association.

A. The FastSLAM Algorithm

Let x_t represent the robot's pose at time t, and $x_{0:t}$ the collection of robot poses up to time t. The control inputs up to time t are denoted by $u_{1:t}$, with u_t as the input at time t. Sensor measurements up to time t are $z_{1:t}$, with z_t as the measurement at time t. The map of features is $m_{1:M}$, where M is the number of landmarks.

Formally, FastSLAM aims to estimate the posterior of the robot's path (not just the current pose) and the environment map based on sensor measurements and control inputs:

$$p(x_{0:t}, m_{1:M} \mid z_{1:t}, u_{1:t}). \tag{1}$$

The dynamic Bayesian network representation of FastSLAM, depicted in Fig. 1, illustrates these dependencies, with nodes representing stochastic processes and edges representing the probabilistic dependencies. From this network, we can infer that poses behave according to a probabilistic law, named motion model, with density $p(x_t \mid x_{t-1}, u_t)$. Likewise, the measurements are governed by a (probabilistic) measurement model $p(z_t \mid x_t, m_{1:M})$.

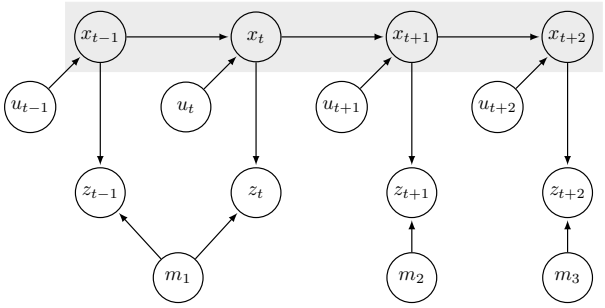

Fig. 1. Dynamic Bayes network of FastSLAM. Given the known robot path (shaded region), landmark positions are conditionally independent.

The key idea of FastSLAM exploits the fact that knowledge of the robot's path renders the individual landmark measurements independent [1]. FastSLAM decomposes the SLAM problem into one robot localization problem, and a collection of M landmark estimation problems. Thus, the exact factored posterior of Eq. (1) reads[1]

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}) \prod_{j=1}^{M} p(m_j \mid x_{0:t}, z_{1:t}). \tag{2}$$

[1]Here, we choose to omit the controls in the landmarks' posterior, since we assume the states $x_{0:t}$ are fully known.

The resulting algorithm for this Rao-Blackwellized particle filter is generically presented as follows:

Algorithm 1: Generic FastSLAM 1.0 Algorithm

Input: Particle set $\mathcal{X}_{t-1}$, control u_t, measurements z_t
Output: Updated particle set $\mathcal{X}_t$

$\mathcal{X}'_t = \emptyset$
for $k = 1$ **to** N **do**
  Retrieve a pose $x_{t-1}^{[k]}$ from the particle set $\mathcal{X}_{t-1}$
  Sample a new pose $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$
  for *each observed feature* z_t^i **do**
    Identify the correspondence j for the measurement z_t^i
    Incorporate the measurement z_t^i into the corresponding EKF, updating the mean $\mu_{j,t}^{[k]}$ and covariance $\Sigma_{j,t}^{[k]}$ of the j-th landmark
  Calculate the importance weight $w_t^{[k]}$ for the new particle
  Add the new particle $\{x_t^{[k]}, \mu_{j,t}^{[k]}, \Sigma_{j,t}^{[k]}, w_t^{[k]}\}$ to $\mathcal{X}'_t$
$\mathcal{X}_t = \emptyset$
for $k = 1$ **to** N **do**
  Sample $x_t^{[k]}$ from $\mathcal{X}'_t$ with probability $\propto w_t^{[k]}$
  Add $x_t^{[k]}$ to $\mathcal{X}_t$
return $\mathcal{X}_t$

The FastSLAM algorithm retrieves and samples poses for each particle based on the previous state and control input using the motion model. Observed features update corresponding landmark estimates using an EKF with the measurement model, and importance weights are calculated from the measurement likelihoods. Resampling focuses on the most probable particles, enhancing robustness to noise and uncertainties.

However, FastSLAM is not without limitations. Its performance heavily depends on the number of particles, sparsity of observations, dynamism of the environment and the accuracy of the motion and measurement models (which are robot specific). Additionally, resampling might lead to particle depletion, where a few particles dominate, reducing robustness. We will revisit these issues in future sections where data is analyzed.

The following subsections detail the baseline models used for both the known and unknown data association versions of the algorithm.

1) Motion Model: Given that the robot operates on a plane and has no odometry sensors, we utilized a Brownian motion model [4] to predict the robot's pose at time t, $[x', y', \theta']^T$, based on its previous pose at time $t-1$, $[x, y, \theta]^T$:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} -\frac{\hat{v}}{\hat{\omega}}\sin\theta + \frac{\hat{v}}{\hat{\omega}}\sin(\theta + \hat{\omega}\Delta t) \\ \frac{\hat{v}}{\hat{\omega}}\cos\theta - \frac{\hat{v}}{\hat{\omega}}\cos(\theta + \hat{\omega}\Delta t) \\ \hat{\omega}\Delta t + \hat{\gamma}\Delta t \end{bmatrix}. \quad (3)$$

For linear movement, the relation becomes:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \hat{v}\cos(\theta)\Delta t \\ \hat{v}\sin(\theta)\Delta t \\ \hat{\gamma}\Delta t \end{bmatrix}. \quad (4)$$

Here, $\hat{v}$ and $\hat{\omega}$ represent the velocities corrupted by noise:

$$\begin{cases} \hat{v} = v + \varepsilon_v \\ \hat{\omega} = \omega + \varepsilon_\omega \end{cases} \quad (5)$$

where v and ω are the commanded velocities of the robot, and ε_v and ε_ω are zero mean random variables that model control noise with variance δ_v and δ_ω, respectively. The variable $\hat{\gamma}$ models the degeneracy in the robot's rotation when it arrives at its final pose.

2) Measurement Model: Cameras provide a rich source of environmental data. Therefore, we opted for a 7D state representation for the landmarks, despite the robot being planar[2]. Each landmark state includes a coordinate $\mathbf{p} = [x\ y\ z]^T$ and a unit quaternion $\mathbf{q} = [q_w\ q_x\ q_y\ q_z]^T$ to represent the ArUco marker's orientation in space. Figure 2 illustrates the robot's kinematics.

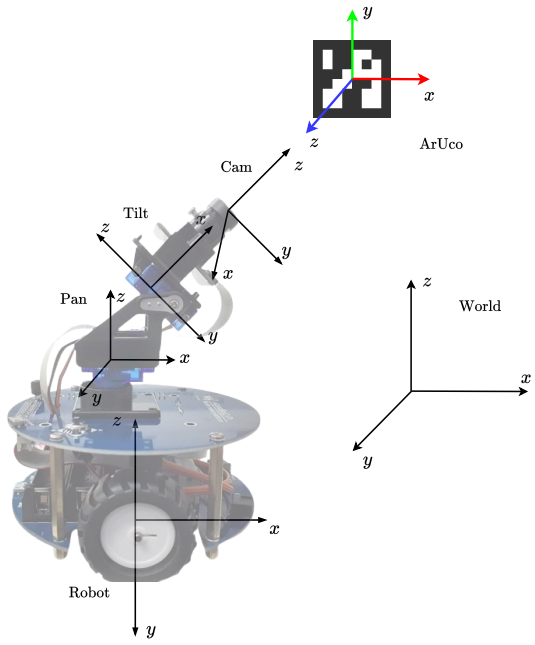

Fig. 2. AlphaBot2's kinematic model. This diagram illustrates the various reference frames used in the system.

Transformations from the ArUco's reference frame to the camera reference frame are handled by the `aruco_detect` library[3], resulting in measurements in the camera's frame. Thereby, we use the following superscripts to denote different reference frames: ^{c} for the camera frame, ^{t} for the tilt frame, ^{p} for the pan frame, ^{r} for the robot frame, and ^{w} for the world.

The transformation from the camera's frame to the world frame involves the following chain operation

$$\begin{bmatrix} \mathbf{p}^w \\ \mathbf{q}^w \end{bmatrix} = {}^w\mathbf{H}_c \begin{bmatrix} \mathbf{p}^c \\ \mathbf{q}^c \end{bmatrix} + \begin{bmatrix} \mathbf{h}_c \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{h}_r \\ \mathbf{0} \end{bmatrix} \quad (6)$$

where we define the matrix ${}^w\mathbf{H}_c = {}^w\mathbf{T}_r\ {}^r\mathbf{T}_p\ {}^p\mathbf{T}_t\ {}^t\mathbf{T}_c$ as the final camera to world transformation, $\mathbf{h}_c = [0\ 0\ 0.12]^T$ as the camera's height offset from the ground (in meters) and $\mathbf{h}_r$ the robot's coordinates in the world frame.

[2]The extra information is also usefull for the data association problem.
[3]Documentation http://wiki.ros.org/aruco_detect (last accessed 07/06/2024).

These transformations employ a classical block matrix construction in order to isolate the variables, incorporating quaternion-based rotations and reflections[4], as described in the literature [5], [6]. The individual transformation matrices were constructed as follows:

$$ {}^{t}\mathbf{T}_c = \text{blkdiag}\left(\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}, 1, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \right), \tag{7} $$

$$ {}^{p}\mathbf{T}_t = \text{blkdiag}\left(\begin{bmatrix} \cos(\theta_t) & 0 & \sin(\theta_t) \\ 0 & 1 & 0 \\ -\sin(\theta_t) & 0 & \cos(\theta_t) \end{bmatrix}, {}^{p}\mathbf{Mq}_t \right), \tag{8} $$

$$ {}^{r}\mathbf{T}_p = \text{blkdiag}\left(\begin{bmatrix} \cos(\theta_p) & -\sin(\theta_p) & 0 \\ \sin(\theta_p) & \cos(\theta_p) & 0 \\ 0 & 0 & 1 \end{bmatrix}, {}^{r}\mathbf{Mq}_p \right), \tag{9} $$

$$ {}^{w}\mathbf{T}_r = \text{blkdiag}\left(\begin{bmatrix} \cos(\theta_r) & -\sin(\theta_r) & 0 \\ \sin(\theta_r) & \cos(\theta_r) & 0 \\ 0 & 0 & 1 \end{bmatrix}, {}^{w}\mathbf{Mq}_r \right). \tag{10} $$

Here, $\text{blkdiag}(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n)$ denotes a block diagonal matrix consisting of matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$ along the diagonal. Each block $\mathbf{A}_i$ is placed on the diagonal, with all other elements outside these blocks being zero. The transformation matrices, denoted by $\mathbf{Mq}$, are derived by first converting the rotation matrix (the first block of the respective transformation matrix) into a quaternion. Subsequently, this quaternion is used to construct a matrix that facilitates quaternion multiplication via standard matrix multiplication. The conversion from a quaternion to its corresponding matrix representation is given by:

$$ \mathbf{Mq} = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix}, \tag{11} $$

where $\mathbf{q} = [q_w \; q_x \; q_y \; q_z]^T$ is the quaternion obtained from the rotation matrix. This matrix form, $\mathbf{Mq}$, enables the quaternion product $\mathbf{q}_1\mathbf{q}_2$ to be computed as a matrix-vector product $\mathbf{Mq}_1\mathbf{q}_2$, effectively replicating the quaternion multiplication operation using matrix algebra [6].

Finally, the measurement model is obtained by reversing the order of operations, i.e.,

$$ \begin{bmatrix} \mathbf{p}^c \\ \mathbf{q}^c \end{bmatrix} = {}^{w}\mathbf{H}_c{}^{T} \left(\begin{bmatrix} \mathbf{p}^w \\ \mathbf{q}^w \end{bmatrix} - \begin{bmatrix} \mathbf{h}_r \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{h}_c \\ \mathbf{0} \end{bmatrix} \right), \tag{12} $$

since rotation matrices and quaternion matrices embedded in the diagonal block matrix are orthogonal (${}^{w}\mathbf{H}_c{}^{-1} = {}^{w}\mathbf{H}_c{}^{T}$).

3) Resampling Function: We employed a low-variance resampler with an effective sample size (ESS) threshold to improve the performance of the FastSLAM algorithm. The ESS is calculated as

$$ \text{ESS} = \frac{1}{\sum_{k=1}^{N} w_k^2}, \tag{13} $$

where w_k are the normalized weights of the particles. If the ESS falls below a chosen percentage of the total number of particles — the ESS *threshold* —, resampling is triggered.

[4]The reflections are propagated to the quaternions by multiplying the 3×3 reflection matrix by the 3×1 vector part of the quaternion, leaving the scalar part unchanged.

Low-variance resampling ensures that particles with higher weights are more likely to be selected multiple times, while those with lower weights are likely to be discarded. This method minimizes the variance introduced during the resampling process. The following pseudo-code snippet captures the essence of the algorithm:

Algorithm 2: Low-Variance Resampling with ESS

Input: Particles with weights
Output: Resampled particles

Normalize the weights of all particles
Calculate the ESS

if *ESS* $<$ *threshold* **then**
  Compute the cumulative sum of weights
  Choose a random starting point $r \sim \mathcal{U}(0, N^{-1})$
  for *each particle index* m *from 1 to* N **do**
    Calculate the threshold $U = r + (m-1)/N$
    while *the cumulative sum is less than* U **do**
      Move to the next particle
    Add the selected particle to the new particle array with equal weight
else
  return the original particle set

return the new particle set

B. Maximum Likelihood Data Association

While ArUco markers provide an ID for straightforward matching and their quantities in the environment are predetermined, this approach is limited in real-world scenarios where the number and identity of landmarks are unknown. Data association involves matching observed features with landmarks. One common approach to this problem is the individual compatibility maximum likelihood (ML) method [3], which selects the correspondence that will maximize the likelihood of the measurement given the current pose and landmark estimates. For each z_t^i, ML identifies the landmark j that maximizes $p(z_t^i \mid x_t, m_j)$.

To achieve this, we calculate the Mahalanobis distance between the observed feature and each landmark. The Mahalanobis distance accounts for the variance in the measurements and is given by:

$$ D^2 = (z_t^i - \hat{z}_j)^T Z_j^{-1} (z_t^i - \hat{z}_j), \tag{14} $$

where $\hat{z}_j$ is the predicted measurement of landmark j and Z_j is the predicted covariance matrix. The landmark with the smallest Mahalanobis distance is chosen, maximizing the likelihood of the correct association.

III. Implementation

This section details our implementation of the FastSLAM algorithm on the AlphaBot2. We cover key aspects such as noise parametrization and measurement covariance estimation. We also discuss practical considerations and optimizations that were essential for real-world performance.

A. Project Structure within ROS

Figure 3 illustrates the nodes and topics used in our project, highlighting the communication flow between different parts of the system. We note that the AlphaBot2's control node had to be made from scratch as briefly explained in Section VI.

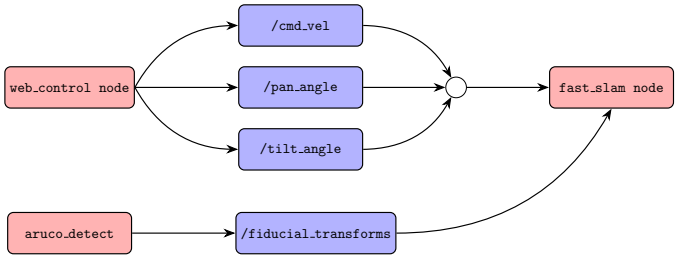

Fig. 3. Diagram showing the interaction between nodes and topics within the AlphaBot2 ROS structure.

B. Motion Model Noise Estimation

The control noise is modeled as a Gaussian random variable with zero mean and variance:

$$\delta_u = \begin{bmatrix} \delta_v \\ \delta_\omega \end{bmatrix}, \tag{15}$$

with $\delta_v = 0.01$ [m/s]^2 and $\delta_\omega = 0.04$ [rad/s]^2, determined heuristically. The noise was determined through over 30 trials to ensure the application of the Central Limit Theorem. We evaluated the time the robot took to traverse 48 cm and perform a full rotation (2π rad), calculating the linear and angular velocities for each case. The variances were then obtained using the following formula:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2, \text{ for } N = 30, \tag{16}$$

where $\bar{x}$ is the average controlled velocity. The degeneracy $\hat{\gamma}$ is modelled as a biased postmovement turn as described in [4].

C. Measurement Model Covariance Matrix

The measurement model covariance matrix was determined heuristically through the following steps:

1) Calibration: Calibration was performed using an 8×6 checkboard with 0.035 m squares[5] at a resolution of 400×304 pixel^2, allowing for an optimal framerate of 30 FPS during real-time operation. Accurate calibration is crucial for the correct estimation of ArUco marker poses, and subsequentially, covariance estimation for the model.

2) Estimation: To allow for accurate and dynamic noise covariance determination for each observation, various samples were taken at four different distances (60 cm, 120 cm, 180 cm, 240 cm) and three different angles (-45º, 0º, 45º) from the target ArUco markers. This was done for all seven variables under study $(x, y, z, q_x, q_y, q_z, q_w)$, and their respective variances were calculated. We then performed a regression analysis using a second-degree polynomial model, for each variable, represented by the equation:

$$f(r,\theta) = \alpha_0 + \alpha_1 \cdot \theta + \alpha_2 \cdot r + \alpha_3 \cdot \theta^2 + \alpha_4 \cdot r \cdot \theta + \alpha_5 \cdot r^2. \tag{17}$$

The choice of this model was based on goodness of fit parameters, such as R-squared, adjusted R-squared, and RMSE.

[5]As described in the following ROS tutorial (last accessed 07/06/2024).

D. Heuristic for Data Association

The Mahalanobis distance squared, D^2, follows a chi-squared distribution [3]. To ensure robust data association, a chi-squared test with 7 degrees of freedom (equal to the feature vector dimension) is applied. For a 95% confidence level, true correspondences fall below this threshold 95% of the time. If the Mahalanobis distance exceeds this threshold, the observation is considered a non-match, reducing false associations.

A compatibility checker further rules out false alarms. It sums the individual compatibilities of each observation with all landmarks. If the sum is null, the observation is marked as a new landmark. This helps identify and discard spurious measurements, ensuring only reliable data updates the map.

The process of associating measurements with landmarks and identifying new landmarks is outlined in the following algorithm:

Algorithm 3: Maximum Likelihood Data Association

Input: Measurement z_t, predicted landmark estimates $\{\hat{z}_j\}$, predicted covariance matrices $\{Z_j\}$
Output: Associated landmarks and new landmarks

for *each observed feature* z_t^i **do**
  for *each landmark* $\hat{z}_j$ **do**
    Compute the Mahalanobis distance, D^2

Find the landmark $\hat{z}_j$ with the smallest D^2

if $D^2 < \chi^2$ **then**
  Associate z_t^i with landmark $\hat{z}_j$

for *each* z_t^i *where compatibility checker equals 0* **do**
  Mark z_t^i as a potential new landmark

return the associated landmarks and new landmarks

E. Observation Outlier Rejection

Additionally, an observation outlier rejection step is implemented using the Mahalanobis distance for the algorithm for the known data association case. Before proceeding to the update step, the algorithm checks if the Mahalanobis distance between the given observation and the matched landmark is below the threshold described in the previous section. If it is not, the observation is discarded.

IV. Results

This section outlines the results from our transition from micro-simulator testing to real-world trials, highlighting the practical application of our FastSLAM implementation.

A. Results in the Micro-Simulator

To facilitate testing and validation of the implementation without the necessity of collecting real data, a micro-simulator was developed. This simulator provides essential guidance on key parameterization values, including the number of particles and the appropriate timing for resampling.

To effectively simulate our algorithm, it is essential to understand the Raspberry Pi Camera Module 2's sensor model. The camera's field of view (FOV) is determined using:

$$\text{FOV} = 2 \cdot \arctan\left(\frac{\text{sensor size}}{2 \cdot \text{focal length}}\right), \tag{18}$$

which, as per the Sony IMX219 sensor datasheet[6], results in a horizontal FOV of approximately 62.2 degrees and a vertical FOV of 48.8 degrees. Additionally, when configured at a resolution of 400×304 pixel^2, the camera can detect ArUco markers of 14 cm up to about 3 meters away. This forms the basis of the sensor model used in our simulations.

1) Number of Particles Analysis: We determined the optimal particle count for our FastSLAM implementation through micro-simulator tests (multiple Monte Carlo runs for various numbers of particles). Table I shows that root mean square error (RMSE) of the particle's path relative to the true path decreases as particle numbers increase, stabilizing at about $N = 100$ particles. Higher counts do not significantly improve accuracy but increase computational demand. This optimal count was thus chosen for an efficient real-world application.

TABLE I
RMSE AS A FUNCTION OF PARTICLE NUMBER.

N	10	20	50	100	200	500
RMSE	0.350	0.280	0.200	0.150	0.149	0.140

2) Determining the Effective Sample Size Threshold: Our simulations confirmed the effectiveness of employing a low-variance resampler with an ESS threshold in FastSLAM. Setting the threshold to about 85% of the maximum particle count effectively prevents early convergence of particles to incorrect positions, a critical aspect in maintaining diversity among particle hypotheses. Figure 4 illustrates the significant reduction in premature convergence and particle impoverishment when this threshold is applied.

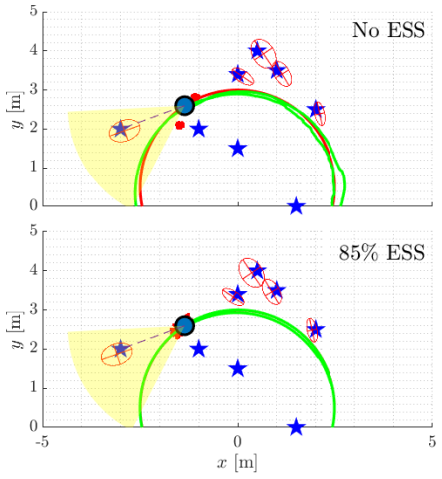

Fig. 4. Demonstration of ESS threshold impact on particle distribution and convergence. This figure illustrates the 2D projection of the simulation to enhance visual clarity. Key elements are marked as follows: blue stars denote the true positions of landmarks; red ellipses show the averaged landmark estimates from particles, including uncertainty; the averaged path is in green, contrasted by the actual path in red. Particles are depicted as small red circles. The robot's true position is indicated by a large blue circle, and its FOV is represented by a yellow cone.

3) 3D Simulation Results: In this section, we present the results of our 3D simulations to validate the measurement model and data association techniques previously discussed. The following simulation in Fig. 5 helps illustrate how our methods perform in a realistic three-dimensional environment.

[6]Sony IMX219 sensor datasheet (last accessed 07/06/2024).

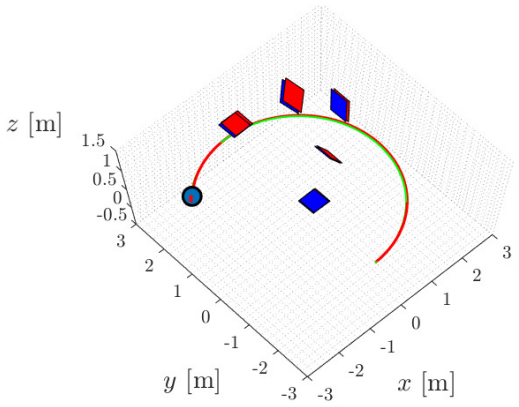

Fig. 5. 3D results of the final FastSLAM algorithm with unknown data association. Red squares represent average estimate, blue is the ground truth.

B. Results on Real-World Data

Given the constraints associated with the robot, such as short battery life and faulty movement drivers (as detailed in Section VI), we conducted our tests in small environments, limiting the number of ArUco markers to a maximum of six, each placed at various orientations and heights. Below, we provide an overview and comparative analysis of our implementation choices, supported by real data. All plots are hand made by us, each containing specific visual elements to aid interpretation: (i) highest weight particle is represented by a large blue circle marker, (ii) particles are shown in small red circles, (iii) markers are rectangular and colored in pink, (iv) the robot's path history is green, (v) the robot and camera's orientation are displayed in red and black, respectively.

1) Pan and Tilt: Given the limited FOV and detection range discussed in Section IV-A, we enhance detection by using the pan and tilt functionalities of the camera, as seen in Fig. 6.

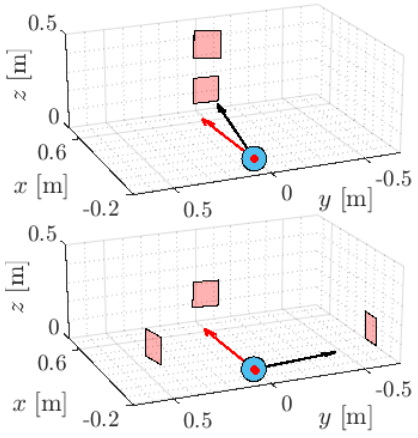

Fig. 6. Use of pan and tilt during test procedure. The first plot showcases the use of tilt, the second one the use of pan, in real-world data.

2) Outlier Rejection: Our outlier rejection protocol enabled us to discard poorly taken measurements, which were often the result of the robot's erratic movements, especially the oscillation that occurs when the robot stops. Due to its instability, the robot tends to wobble similarly to Shakey the robot, with inertia contributing significantly to this effect. In Fig. 7, we provide the same map, one made with outlier rejection and another without.

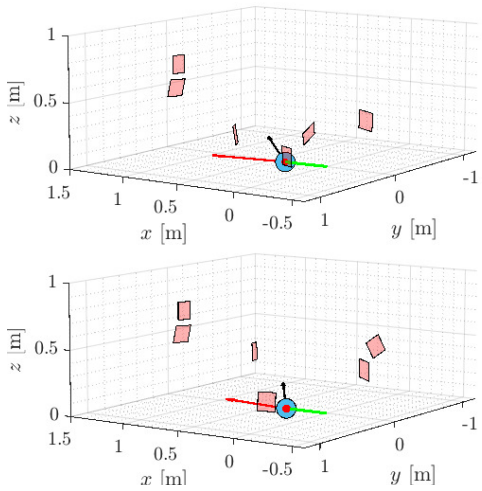

Fig. 7. Effect of outlier rejection. The top map displays results without the implementation of outlier rejection, while the bottom map incorporates it, demonstrating improved accuracy.

The distortion is more pronounced in the first map compared to the second. As detailed in Section III, the algorithm discards measurements exceeding a Mahalanobis distance of $\chi^{-2}_{95\%}(7) \approx 18.4753$. This statistical threshold ensures only accurate environmental data points are retained, improving the clarity and reliability of the second map's mapping process.

3) Unknow Data Association: Below we provide our first attempt at the unknow data association problem:

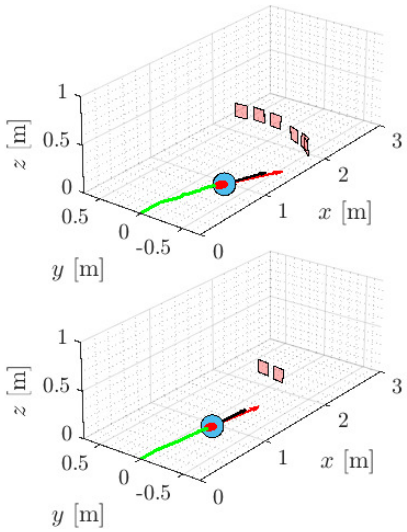

Fig. 8. Landmark creation problem. The top image illustrates the scenario without any adjustments, while the bottom image shows the corrected setup. The real environment contains only two markers.

There is a noticeable difference between the first and second images. Due to our robot's instability when halted, the Mahalanobis distance often exceeds our initial threshold, leading to the erroneous creation of non-existent landmarks. For the second image, we adjusted the threshold to four times the standard $\chi^{-2}_{95\%}(7)$. This modification ensures the display of the correct number of landmarks.

Concerned about the risks of indefinitely increasing the threshold — which could lead to incorrect measurement associations — we opted to enhance our robot's stability by adding weights and clothespins for better balance during stops. The improved results are demonstrated in Fig. 9.

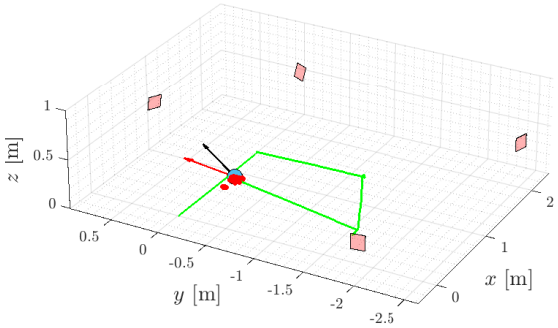

Fig. 9. Correct data association with threshold of $\chi^{-2}_{95\%}(7)$ for a balanced robot. The map has four landmarks in four different corners.

This result culminates all of the discussion beforehand, showing both landmark estimation and a larger route to showcase localization.

V. Conclusions

In conclusion, this paper has presented the successful implementation of the FastSLAM algorithm on the AlphaBot. Through systematic testing, we highlighted the critical role of outlier rejection in enhancing map accuracy and reliability. Despite initial challenges related to the robot's stability and associated measurement inaccuracies, modifications to the Mahalanobis distance threshold and physical adjustments to the robot's hardware significantly improved data integrity and mapping precision with unknown data association. Our findings affirm the efficacy of FastSLAM in real-world conditions, while also underscoring the importance of robust error handling and system stability for optimal performance. Future work will concentrate on refining the robot's control accuracy and precision, and on developing a dynamic threshold that can adapt to inconsistencies in the environment.

VI. Further Considerations

Initially, the robot's movements were erratic, and the control interface was not sufficiently responsive for precise operations. To overcome these challenges, we developed a custom driver tailored to our requirements, making the robot usable for our experiments. This driver is now a critical system component of our work. Additionally, we created a dedicated wiki[7] with detailed configuration guidelines to support the "Autonomous Systems" curricular unit, ensuring students can efficiently set up and use the AlphaBot2 robot for their projects.

References

[1] M. Montemerlo and S. Thrun, *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*. Springer, 2007, ch. 1–4.

[2] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, "Fiducial Markers for Pose Estimation," *Journal of Intelligent & Robotic Systems*, 2021.

[3] A. J. Cooper, "A Comparison of Data Association Techniques for Simultaneous Localization and Mapping," S.M. Thesis, MIT, 2005.

[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005, ch. 2–6.

[5] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2009, ch. 2–3.

[6] L. Carlone, V. Macchia, F. Tibaldi, and B. Bona, "Quaternion-based EKF-SLAM from Relative Pose Measurements: Observability Analysis and Applications," *Robotica*, 2015.

[7] Available in https://github.com/Kons-5/ROS-for-the-Alphabot2/wiki.