Feature-Based Simultaneous Localization and Mapping (SLAM) using ArUco Markers

Ahmed Alghfeli Hassan Alhosani Reem Almheiri

I. Abstract

This report presents a feature-based Simultaneous Localization and Mapping (SLAM) system for a 3 Degree of Freedom (DoF) Turtlebot robot. The system utilizes ArUco markers for feature detection. The robot's motion is modeled using a motion model, and observations are incorporated to refine the robot's pose and map estimation. The proposed approach aims to accurately estimate the robot's position and map the environment in real-time.

II. Introduction

In the field of Simultaneous Localization and Mapping (SLAM), various techniques are employed to accurately estimate the position and map the environment. This project focuses on a feature-based SLAM approach, specifically utilizing markers of the ArUco type. By leveraging computer vision techniques, the aim is to detect and track the position of ArUco markers within the environment. This introductory section provides an overview of the project's objectives and highlights the significance of employing ArUco markers as the primary feature for the SLAM implementation. The SLAM system will be built based on a 3 Degree of Freedom (DoF) robot, namely the Kobuki robot.

A. SLAM State Vector

The proposed Feature-based Extended Kalman Filter (EKF) SLAM method models the map M of the environment as a Gaussian Random Vector (GRV) of robot and feature poses represented in the frame N:

$$M = ^N x_k$$

$$^N x_k \sim N(^N x_k, ^N P_k)$$

Where :

$$^N x_k = \begin{bmatrix} ^N x_{B_k} \\ ^N x_{F_1} \\ . \\ . \\ ^N x_{F_n} \end{bmatrix}$$

$$^N P_k = \begin{bmatrix} ^N P_{Bk} & ^N P_{B_k F_1} & . & . & ^N P_{B_k F_n} \\ ^N P_{F_1 B_k} & ^N P_{F_1 F_2} & . & . & ^N P_{F_1 F_n} \\ & & & & \\ . & . & . & . & . \\ . & . & . & . & . \\ ^N P_{F_n B_k} & ^N P_{F_n F_2} & . & . & ^N P_{F_n F_n} \end{bmatrix}$$

The state vector $^N x_k$ represents the poses of the robot B_k and the n features F_i in frame N, while the covariance matrix $^N P_k$ captures the uncertainty associated with these poses and their inter-relationships.

B. robot state

In the proposed method, the robot state contains the pose in x,y and the orientation:

$$^N x_{B_k} = \begin{bmatrix} ^n x_b \\ ^n y_b \\ ^n \theta_b \end{bmatrix}$$

$$^N x_{B_k} \sim N(^N x_{B_k}, ^N P_{B_k})$$

The robot state vector $^N x_{B_k}$ follows a Gaussian distribution with mean $^N x_{B_k}$ and covariance matrix $^N P_{B_k}$.

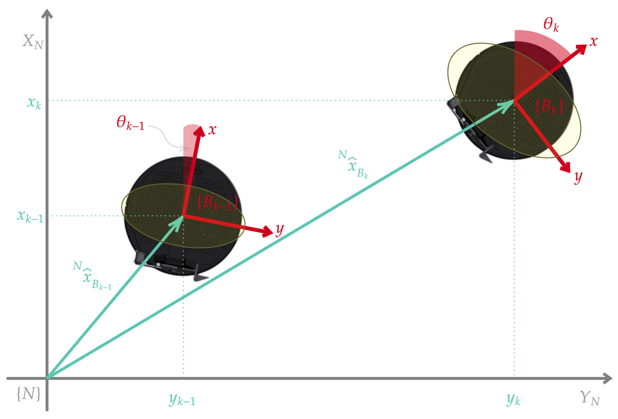

Fig. 1. Pose estimation using dead reckoning

Figure 1 illustrates the robot poses at times $k-1$ and k in frame N, showcasing the movement of the robot within the environment. The pose estimation is performed using dead reckoning, where the robot's motion and control inputs are used to estimate its subsequent position and orientation.

C. Feature state

In this system, the already mapped features are encoded as 2D points represented in Cartesian coordinates referenced to the frame N:

$$^N x_{F_N} = \begin{bmatrix} ^n x_F \\ ^n y_F \end{bmatrix}$$

$$^N x_{F_n} \sim N(^N x_{F_n}, ^N P_{B_{F_n}})$$

The feature state vector $^N x_{F_n}$ follows a Gaussian distribution with mean $^N x_{F_n}$ and covariance matrix $^N P_{B_{F_n}}$. The feature state represents the location of the nth feature in the environment with respect to the frame N.

III. Robot motion model

In robotics, a motion model describes how a robot's position and orientation change over time. Accurately modeling this motion is important for localization and mapping tasks. The Jacobian of the motion model is a matrix that describes how uncertainty in the robot's current pose affects the predicted pose. This matrix is used in the Kalman filter prediction step to estimate the robot's future pose with uncertainty.

A. Robot Parameters

The Turtlebot robot has the following physical properties:

- Distance of the base: $b = 0.23$ m
- Radius of the wheels: $r = 0.035$ m

B. Velocity Calculation

We can obtain the left and right velocities of the robot from the encoders. Since the encoders measure angular velocities, we have to convert them to linear velocities:

$$V_L = \omega_L \cdot r \qquad V_R = \omega_R \cdot r$$

Once we have the left and right linear velocities, we can calculate the linear and angular velocity of the Turtlebot robot:

$$V = \frac{V_L + V_R}{2} \qquad \omega = \frac{V_L - V_R}{b}$$

C. Motion Model Derivation

The motion model describes how the robot's pose changes over time. We can express it as follows:

$$\hat{x}_k = f(x_{k-1}, u_k, w_k)$$

where

$$\hat{x}_k = \begin{bmatrix} x_k \; y_k \; \theta_k \end{bmatrix}^T \quad \hat{x}_{k-1} = \begin{bmatrix} x_{k-1} \; y_{k-1} \; \theta_{k-1} \end{bmatrix}^T$$

$$u_k = \begin{bmatrix} V_R \; V_L \end{bmatrix}^T \;\; w_k \sim N(0, Q_k); \quad Q_k = \begin{bmatrix} \sigma_{v_R}^2 & 0 \\ 0 & \sigma_{v_L}^2 \end{bmatrix}$$

The motion model can be derived by considering the kinematics of the robot. The robot's pose is determined by its position and orientation, which can be updated using the velocity and angular velocity. Specifically, the new pose is obtained by integrating the linear and angular velocity over time:

$$x_k = x_{k-1} + V \cdot cos\theta_{k-1} \cdot dt$$

$$y_k = y_{k-1} + V \cdot sin\theta_{k-1} \cdot dt$$

$$\theta_k = \theta_{k-1} + \omega \cdot dt$$

By combining all the elements, we obtain:

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \frac{(V_{L_{noise}}+VL)+(V_{R_{noise}}+VR)}{2} \cdot cos\theta_{k-1} \cdot dt \\ y_{k-1} + \frac{(V_{L_{noise}}+VL)+(V_{R_{noise}}+VR)}{2} \cdot sin\theta_{k-1} \cdot dt \\ \theta_{k-1} + \frac{(V_{L_{noise}}+VL)-(V_{R_{noise}}+VR)}{b} \cdot dt \end{bmatrix}$$

where x_k, y_k, and θ_k are the new pose variables, V_L and V_R are the left and right wheel velocities, $V_{L_{noise}}$ and $V_{R_{noise}}$ are their corresponding noise, b is the distance between the wheels, and dt is the time step.

In these equations, A_k and W_k represent Jacobians with respect to state vector in our case [x y θ]^T and noise w_k = $[V_{Rnoise} \; V_{Lnoise}]^T$, respectively. The values of θ_k, v, dt, and b are taken from the corresponding variables in the code.

$$A_k = \frac{\partial f(\hat{x}_{k-1}, u_k, w_k)}{\partial \hat{x}_k} = \begin{bmatrix} 1 & 0 & -V \cdot \sin(\theta_{k-1}) \cdot dt \\ 0 & 1 & V \cdot \cos(\theta_{k-1}) \cdot dt \\ 0 & 0 & 1 \end{bmatrix}$$

$$W_k = \frac{\partial f(\hat{x}_{k-1}, u_k, w_k)}{\partial w_k} = \begin{bmatrix} \frac{1}{2} \cdot \cos(\theta_{k-1}) \cdot dt & \frac{1}{2} \cdot \cos(\theta_{k-1}) \cdot dt \\ \frac{1}{2} \cdot \sin(\theta_{k-1}) \cdot dt & \frac{1}{2} \cdot \sin(\theta_{k-1}) \cdot dt \\ -\frac{dt}{b} & \frac{dt}{b} \end{bmatrix}$$

Once the Jacobian of the motion model and the Jacobian of the noise are determined, the prediction uncertainty can be calculated. The prediction can be computed using the following formula.

$$P_k = A_k \cdot P_{k-1} \cdot A_k^T + W_k \cdot Q_k \cdot W_k^T$$

IV. Detection part

This section discusses the detection techniques employed to obtain the projected range and azimuth of the feature from the robot. Initially, the OpenCV library is utilized for ArUco detection, resulting in the detection of ArUco markers in the camera frame, denoted as $^C p = [^C x \; ^C y \; ^C z]^T$. These detected markers are then transformed into the robot frame using the transformation matrix $^B T_C$, yielding the coordinates $^B p =^B T_C \cdot^C p$ in the robot frame. With the point transferred to the robot frame, the projected range from the center of the robot to the marker can be calculated as $^B \rho = \sqrt{^B x^2 +^B y^2}$, while the bearing can be determined as $^B \theta = atan2(^B y,^B x)$. These calculations enable the estimation of the range and azimuth of the detected feature relative to the robot's position. The transformation process from the camera frame to the robot frame is illustrated in Figure 2.

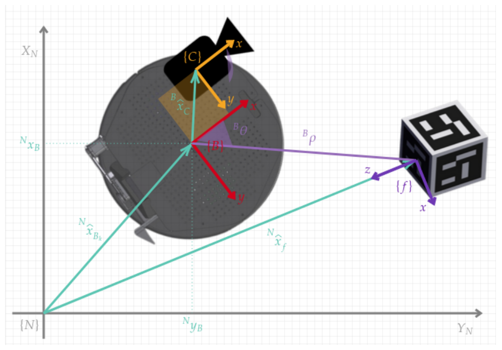

Fig. 2. transformation from camera frame to robot

V. Observation model

The observation model in SLAM establishes the connection between the robot's sensor measurements and its estimated map of the environment. By considering the robot's pose and the map, the model predicts the expected sensor measurements, which are then compared to the actual measurements obtained from the sensors. The difference between predicted and actual measurements is used to update the robot's estimate of its pose and the map. The observation model incorporates sensor properties, such as range, bearing, or camera information, to accurately predict the expected measurements and refine the estimation process. It serves as a crucial component in the iterative SLAM framework, integrating sensor data to improve the accuracy of the robot's pose and map estimation.

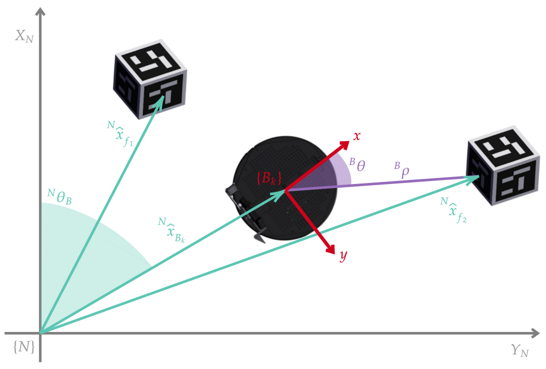

Fig. 3. Feature Detection

A. Direct Observation

In our approach, the observation model is denoted by $h(\hat{x}_k, v_K)$, where $\hat{x}_k$ is the state vector at time k. The Jacobian matrix of the observation model is denoted by H, and is made up of the partial derivatives of the observation function with respect to the state variables.

1) Observation variables: Let us first define the variables used in the observation function. The pose of the robot B is defined as $^N\hat{x}_B = [^Nx_B \; ^Ny_B \; ^N\theta_B]^T$, where ^Nx_B, ^Ny_B, and $^N\theta_B$ are the robot's position and orientation in the world frame N. The position of a feature f in the environment is defined as $^N\hat{x}_f = [^Nx_f \; ^Ny_f]^T$, where ^Nx_f and ^Ny_f are the feature's position in the world frame N.

The observation z_k is defined as the vector $[^B\rho \; ^B\theta]^T$, where $^B\rho$, $^B\theta$, and $^B\varphi$ are the range, bearing, and elevation angle of the feature with respect to the robot's frame B.

2) Observation function: Given the robot's pose $^N\hat{x}_B$ and the position of a feature $^N\hat{x}_f$, we can compute the predicted observation z_k using the following function:

$$z_k = h_{xf}(^N\hat{x}_B, ^N\hat{x}_f, v_k)$$

where v_k is the measurement noise at time k. Now, let's break down the variables involved in the observation function:

- Pose Definition:
$$^N\hat{x}_{B_k} = [^Nx_B \; ^Ny_B \; ^N\theta_B]^T$$
Here, ^Nx_B, ^Ny_B, and $^N\theta_B$ are the robot's position and orientation in the world frame N.
- Feature Definition:
$$^N\hat{x}_f = [^Nx_f \; ^Ny_f]^T$$
Here, ^Nx_f, ^Ny_f, and ^Nz_f are the feature's position in the world frame N.
- Observation:
$$z_k = [^B\rho \; ^B\theta]^T$$
Here, $^B\rho$ and $^B\theta$ are the range and azimuth angle of the feature with respect to the robot's frame B.

The observation function h_{xf} is defined as follows:

$$^N\Delta x = ^N x_f - ^N x_B$$
$$^N\Delta y = ^N y_f - ^N y_B$$

$$z_k = h = \begin{bmatrix} ^B\rho \\ ^B\theta \end{bmatrix} = \begin{bmatrix} \sqrt{\Delta x^2 + \Delta y^2} \\ atan2(\Delta y, \Delta x) - ^N\theta_B \end{bmatrix} + \begin{bmatrix} W_\rho \\ W_\theta \end{bmatrix}$$

The Jacobian matrix H of the observation function with respect to the state vector x_k is:

$$H = \frac{\partial h}{\partial \hat{x}_{Bk}} = \begin{bmatrix} \frac{-\Delta x}{\sqrt{\Delta x^2+\Delta y^2}} & \frac{-\Delta y}{\sqrt{\Delta x^2+\Delta y^2}} & 0 \\ \frac{\Delta y}{\Delta x^2+\Delta y^2} & \frac{\Delta x}{\Delta x^2+\Delta y^2} & -1 \end{bmatrix}$$

$$V_k = \frac{\partial h}{\partial v_k} = I$$

for slam

$$H_k = \begin{bmatrix} \frac{\partial h}{\partial x_{Bk}} & \frac{\partial h}{\partial x_{marker_1}} & \frac{\partial h}{\partial x_{marker_2}} & \cdots & \frac{\partial h}{\partial x_{marker_n}} \end{bmatrix}$$

$$\frac{\partial h}{\partial x_{marker_n}} = \begin{bmatrix} \frac{\Delta x}{\sqrt{\Delta x^2+\Delta y^2}} & \frac{\Delta y}{\sqrt{\Delta x^2+\Delta y^2}} \\ \frac{\Delta y}{\Delta x^2+\Delta y^2} & \frac{\Delta x}{\Delta x^2+\Delta y^2} \end{bmatrix}$$

$$\frac{\partial h}{\partial \hat{x}_{Bk}} = \begin{bmatrix} \frac{-\Delta x}{\sqrt{\Delta x^2+\Delta y^2}} & \frac{-\Delta y}{\sqrt{\Delta x^2+\Delta y^2}} & 0 \\ \frac{\Delta y}{\Delta x^2+\Delta y^2} & \frac{\Delta x}{\Delta x^2+\Delta y^2} & -1 \end{bmatrix}$$

B. Inverse Observation

We will utilize the OpenCV library to employ camera detection and obtain the following information in relation to the robot frame: the projected distance from the robot, denoted by ρ, and the horizontal angle, denoted by θ. It is important to note that this section will only focus on a single feature.

The robot's state can be updated using the inverse observation model as follows:

$${}^N\hat{x}_k^+ = g({}^N\hat{x}_k, z_k, v_k) = [{}^N\hat{x}_{B_k} \quad ({}^N\hat{x}_{B_k} \boxplus (z_f + v_k)]$$

Here, ${}^N\hat{x}_k$ is the robot's state at time k, z_k is the camera measurement, and v_k is the measurement noise. ${}^N\hat{x}_{B_k}$ is the state of the robot's body frame at time k, and z_f is the feature position in the robot frame.

To transfer from polar to Cartesian will follow these formula:

$${}^Bx = {}^B\rho \cdot cos({}^B\theta)$$
$${}^By = {}^B\rho \cdot sin({}^B\theta)$$

In order to transfer the feature's position from the robot frame to the world frame, we will do compunding of the feature with the robot pose:

$${}^N\hat{x}_{B_k} \boxplus (z_f + v_k) =$$
$$\left[\begin{bmatrix} {}^Nx_B \\ {}^Ny_B \\ 1 \end{bmatrix} + \begin{bmatrix} cos({}^N\theta_B) & -sin({}^N\theta_B) & 0 \\ sin({}^N\theta_B) & cos({}^N\theta_B) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^Bx \\ {}^By \\ 1 \end{bmatrix}\right]$$

$${}^N\hat{x}_{B_k} \boxplus (z_f + v_k) =$$
$$\begin{bmatrix} {}^Bx \cdot cos({}^N\theta_B) - {}^By \cdot sin({}^N\theta_B) + {}^Nx_B \\ {}^Bx \cdot sin({}^N\theta_B) + {}^By \cdot cos({}^N\theta_B) + {}^Ny_B \end{bmatrix}$$

$${}^N\hat{x}_{B_k} \boxplus (z_f + v_k) =$$
$$\begin{bmatrix} {}^B\rho \cdot cos({}^B\theta) \cdot cos({}^N\theta_B) - {}^B\rho \cdot sin({}^B\theta) \cdot sin({}^N\theta_B) + {}^Nx_B \\ {}^B\rho \cdot cos({}^B\theta) \cdot sin({}^N\theta_B) + {}^B\rho \cdot sin({}^B\theta) \cdot cos({}^N\theta_B) + {}^Ny_B \end{bmatrix}$$

The Jacobian of the state vector represents the partial derivatives of the robot's state with respect to its own state. The Jacobian of the inverse observation model represents the partial derivatives of the feature's position in the world frame with respect to the camera measurements ρ, θ:

$$G_{1_K} = \frac{\partial g({}^N\hat{x}_k, z_f)}{\partial {}^N\hat{x}_k} = \begin{bmatrix} I \\ J_{1\boxplus} \end{bmatrix}$$

$$G_{2_K} = \frac{\partial g({}^N\hat{x}_k, z_f)}{\partial v_k} = \begin{bmatrix} 0 \\ J_{2\boxplus} \end{bmatrix}$$

where $J_{1\boxplus}$ is partial derivative of ${}^N\hat{x}_{B_k} \boxplus (z_f + v_k)$ with respect ${}^N\hat{x}_{B_k}$:

$$J_{1\boxplus} = \frac{\partial {}^N\hat{x}_{B_k} \boxplus (z_f + v_k)}{\partial {}^N\hat{x}_B}$$

$$J_{1\boxplus} == \begin{bmatrix} 1 & 0 & -{}^Bx \cdot sin({}^N\theta) - {}^By \cdot cos({}^N\theta) \\ 0 & 1 & {}^Nx \cdot cos({}^N\theta) - {}^By \cdot sin({}^N\theta) \end{bmatrix}$$

And where $J_{2\boxplus}$ is partial derivative of ${}^N\hat{x}_{B_k}$ with respect camera measurements ρ, θ :

$$J_{2\boxplus} = \frac{\partial {}^N\hat{x}_{B_k} \boxplus (z_f + v_k)}{\partial z_k}$$

$$J_{2\boxplus} = \begin{bmatrix} \frac{\partial {}^N\hat{x}_{B_k} \boxplus (z_f + v_k)}{\partial \rho} & \frac{\partial {}^N\hat{x}_{B_k} \boxplus (z_f + v_k)}{\partial \theta} \end{bmatrix}$$

$$\frac{\partial {}^N\hat{x}_{B_k} \boxplus (z_f + v_k)}{\partial \rho} = \begin{bmatrix} cos({}^B\theta + {}^N\theta) \\ sin({}^B\theta + {}^N\theta) \end{bmatrix}$$
$$\frac{\partial {}^N\hat{x}_{B_k} \boxplus (z_f + v_k)}{\partial \theta} =$$
$$\begin{bmatrix} -\rho \cdot sin({}^B\theta + {}^N\theta) \\ -\rho \cdot sin({}^B\theta) \cdot sin({}^N\theta) + \rho \cdot cos({}^B\theta) \cdot cos({}^N\theta) \end{bmatrix}$$

C. Update Step

The update step in the Extended Kalman Filter (EKF) is a crucial stage that determines whether an update will occur or not. It relies on data association techniques, which involve checking the ArUco marker IDs and performing compatibility tests based on the Mahalanobis distance and chi-square statistics.

To perform the update, the EKF utilizes the following equations:

$$y_k = z_k - h(\hat{x}_{k|k-1}) \tag{1}$$

where y_k represents the innovation or measurement residual, z_k is the observed measurement, and $h(\hat{x}k|k-1)$ is the predicted measurement based on the estimated state $\hat{x}k|k-1$ at the previous time step.

The innovation covariance matrix, S_k, is computed as:

$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{2}$$

where H_k is the measurement Jacobian matrix evaluated at the predicted state, $P_{k|k-1}$ is the error covariance matrix of the predicted state, and R_k represents the measurement noise covariance matrix.

The Kalman gain, K_k, is then calculated as:

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{3}$$

Using the Kalman gain, the state estimate update equation is given by:

$$\hat{x}k = \hat{x}k|k-1 + K_k y_k \tag{4}$$

Finally, the error covariance matrix update equation is:

$$P_k = (I - K_k H_k) P_{k|k-1} \tag{5}$$

where I represents the identity matrix.

To assess the compatibility between the observed measurement and the predicted measurement, the Mahalanobis distance is computed. It is given by:

$$D = \sqrt{(y_k - h(\hat{x}k|k-1))^T S_k^{-1} (y_k - h(\hat{x}k|k-1))} \tag{6}$$

If $D^2 < X_{ij}^2$, the EKF accepts the reading and performs the update. Otherwise, it will discard the measurement.

By performing these calculations and evaluating the compatibility of the observed measurements with the predicted ones, the EKF determines whether to incorporate the measurements into the state estimate and update the covariance matrix accordingly.

VI. Results

This section presents the results of the project, divided into different stages. Firstly, the motion model's performance is evaluated by showcasing the prediction results. Following that, the results of direct observations on the map based ekf. Both of these evaluations are conducted in a simulation environment. Finally, the entire system is tested as a SLAM implementation, using both simulated and real robot experiments.

A. Simulation Results

In the simulation phase, the performance of the system is assessed using simulated data. The accuracy of the prediction model, the effectiveness of the direct observations on the map based EKF, and the performance of the SLAM implementation are evaluated. The simulation is conducted using the Stonfish simulator.

Initially, only the prediction model is used to estimate the robot's trajectory. Figure 4 shows the ground truth trajectory in red and the predicted trajectory in blue. It can be observed that the robot's estimated trajectory deviates significantly from the ground truth, indicating considerable drift. The yellow uncertainty region represents the growing uncertainty as the robot moves.

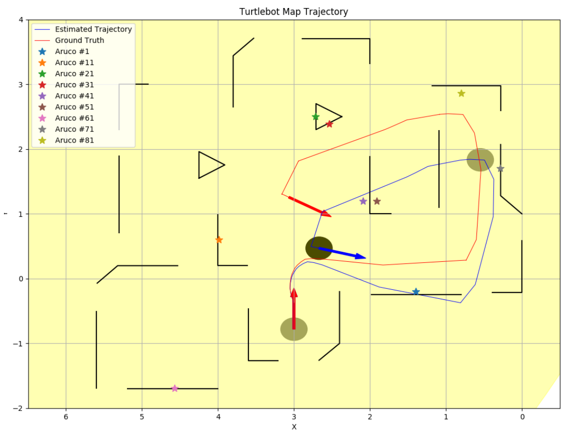

Fig. 4. Prediction-only trajectory

Next, the direct observations are incorporated into the system, and a map-based EKF is applied. The system is tested on a known map, and the results are encouraging. The blue trajectory in Figure 5 represents the robot's estimated trajectory based on the map-based EKF, while the red trajectory represents the ground truth. The two trajectories closely align with each other, indicating accurate estimation. The yellow uncertainty ellipses become smaller as the system updates its estimate with the direct observations, and they are significantly smaller compared to the prediction-only scenario. Additionally, the error graphs in Figure 6 demonstrate that the difference between the ground truth and predicted pose remains within the bounds of 2 sigma.

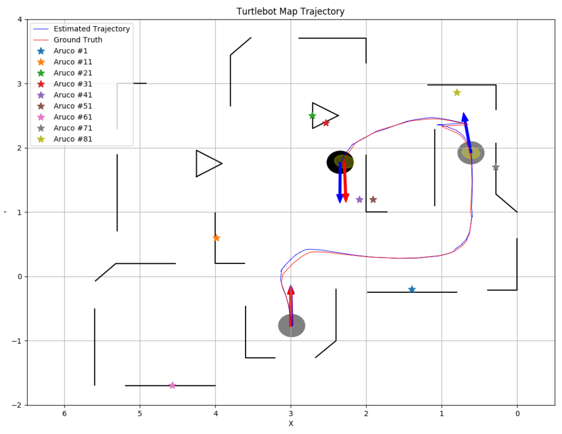

Fig. 5. Map-based EKF trajectory

Furthermore, the system was upgraded to SLAM after confirming the accuracy of the direct observations. The SLAM implementation was tested in the same environment as the previous experiments. Initially, the system encountered challenges in properly updating the robot's position, resulting in significant drift and accepting readings from any feature

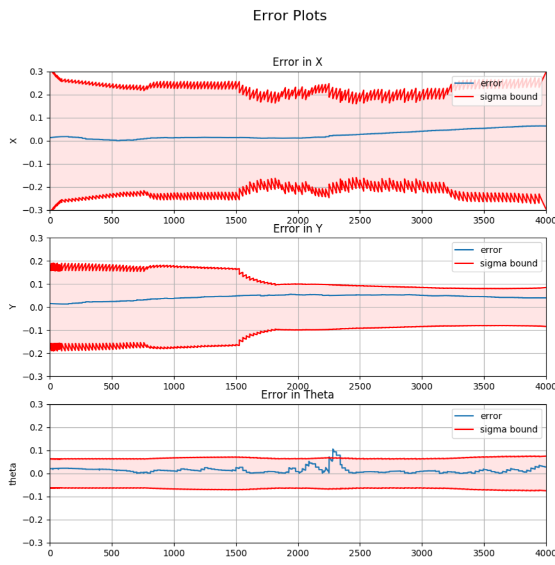

Fig. 6. Error graphs

without proper verification. To address this issue, compatibility tests were implemented using Mahalanobis distance and chi-square tests to determine the validity of the received readings for updating.

The trajectory plot in Figure 7 illustrates the estimated trajectory (blue) and the ground truth trajectory (red). It can be observed that the robot initially experienced drifting, but after closing the loop and re-observing the first feature it encountered, it corrected its position and returned closer to the ground truth. The stars represent the ground truth locations of the ArUco markers, while the plus signs denote the mapped features. Towards the end of the trajectory, the uncertainty ellipses reduced after the loop closure, indicating improved localization accuracy.

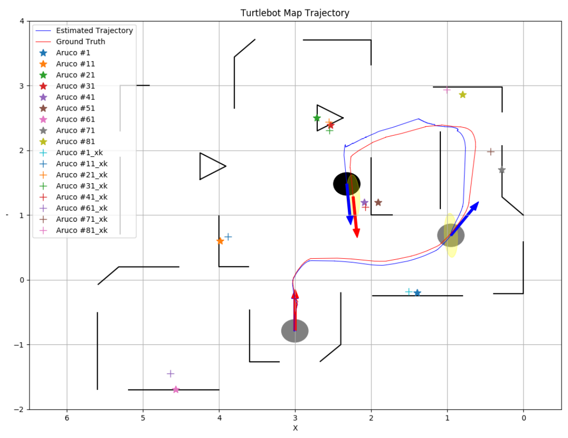

Fig. 7. SLAM trajectory

The error graphs in Figure 8 demonstrate that the difference between the estimated pose and the ground truth remains within the bounds of 2 sigma. This indicates that the SLAM implementation effectively estimates the robot's position and reduces localization errors.

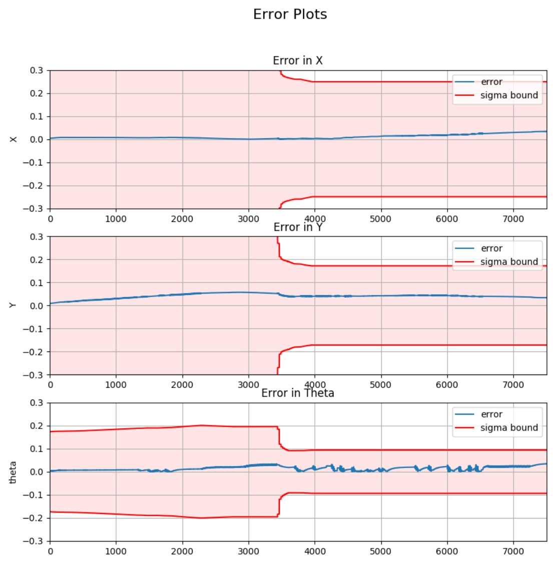

Fig. 8. Error graphs

The simulation results highlight the successful integration of SLAM techniques. By incorporating compatibility tests and loop closures, the system improved the accuracy of the estimated trajectory and reduced drift. The reduced uncertainty and error bounds indicate the efficacy of the SLAM approach in accurately mapping the environment and estimating the robot's pose.

B. *Real Robot Results*

In the real robot experiments, the SLAM system was tested on the Koubuki robot to evaluate its performance in a real-world scenario. Despite encountering a delay issue in receiving sensor messages, the mapped features remained in their correct positions, indicating the system's ability to accurately map the environment. However, the robot's motion caused a misalignment between the map and the robot's actual position, highlighting the challenges of real-time mapping and localization on a physical robot. Addressing the delay issue and improving synchronization between the robot's motion and the mapping process will be crucial for enhancing the SLAM system's accuracy and reliability in real-time applications.

VII. Discussion

Based on the obtained results of the feature-based SLAM, several key points should be addressed. Firstly, the performance of the SLAM system improves as more features are available in the environment. Insufficient features in an area can lead to mapping inaccuracies and incorrect feature placement when the robot observes a new feature while experiencing drift. However, the system tends to self-correct with loop closure. In cases where the robot has drifted significantly, returning to the first observed feature may not pass the compatibility test due to the accumulated drift.

VIII. Autonomous Task

During the autonomous task, we introduced an exploration behavior between the Hands-on-Planning and Hands-on-Localization phases. The primary objective was to empower the robot to navigate through an unfamiliar map containing randomly positioned ArUco markers. We implemented this functionality and evaluated its performance using the Stonefish simulator. To demonstrate the exploration process, we have prepared a video that can be accessed via the following link:
https://youtu.be/5NsqSn8aCA8.

In the video, you can observe the robot engaging in exploration and localization tasks using ArUco markers. Initially, the robot moves smoothly and accurately with the assistance of ground truth odometry. However, there is a particular moment where the robot encounters a turn without any ArUco markers in sight, resulting in a slight drift. Nevertheless, if the robot approaches the first ArUco marker once again, it is highly likely to correct the drift and regain its accuracy.

IX. conclusion

In conclusion, this report explored the implementation of feature-based SLAM using ArUco markers. The simulation implementation demonstrated promising results, showcasing the effectiveness of ArUco markers for localization and mapping..

Feature-based SLAM with ArUco markers offers a practical and cost-effective solution for simultaneous localization and mapping in various robotic applications. Its simplicity and versatility make it a popular choice, especially in scenarios where markers can be easily deployed and detected. However, it is important to note that the performance of ArUco-based SLAM heavily relies on the availability of distinct markers in the environment.

X. Gantt Chart

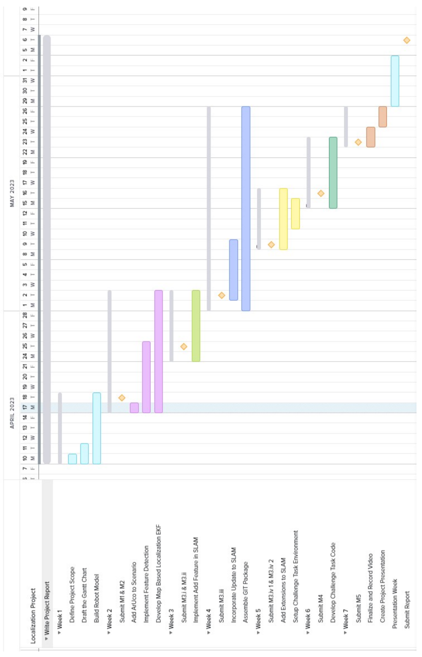

Fig. 9. Gantt Chart