Politecnico di Torino

Department of Electronics and Telecommunications

Master’s degree in Mechatronic Engineering

2024/2025

Sensors, Embedded Systems and Algorithms for Service Robotics

Report 1

02/12/2024

Group n° 1

Gabriel Cunha – S335404

Baptiste Hooghe – S334817

Thiago Levin – S334838

Paul Novoa – S339335

1 Objective

The aim of this laboratory activity is to develop practical skills and deepen theoretical understanding of robot localization using ROS 2, a modern middleware framework for robotics. Localization is a fundamental task in robotics, that involves estimating a robot's position and orientation in its environment. This is critical for enabling autonomous navigation and decision-making. The difficulty depends on environment complexity and this report might show several differences between simulation and real testing.

Specifically, the activity focuses on implementing an Extended Kalman Filter (EKF) which is a powerful algorithm for estimation of the robot's position and orientation (pose) within its environment. The Extended Kalman Filter is a mathematical framework notably useful in the presence of noise (gaussian distribution) and uncertainty. It extends the classic Kalman Filter to handle non-linear systems, making it particularly suited for robotic applications where motion and sensor models are often non-linear. The EKF operates in two stages:

Prediction: In this stage, the EKF uses a motion model based on odometry or IMU data to predict the robot's next state. This step incorporates process noise to account for uncertainties in the motion model, such as wheel slippage or imperfect sensors.

Correction (Update): Using measurements from external sensors, such as landmarks or range sensors, the EKF corrects its predicted state. The measurement model, which relates the observed data to the robot's state, is used to compute a correction term. The filter adjusts the predicted state using this term to produce a more accurate estimate.

The EKF combines information from odometry, IMU, and landmark observations, by weighing them according to their respective uncertainties. This process enhances the accuracy and robustness of localization, even in noisy or dynamic environments.

The laboratory activity also provides an opportunity to study the ROS 2 framework in depth, including its node-based architecture, inter-process communication, and tools for sensor data processing and visualization. By completing this lab session, we will develop the skills required to design, implement, and evaluate a modular localization system using ROS 2 and apply these skills to both simulation and real-world robotic platforms.

2 Design

The code is structured in 3 main python scripts, namely "EKF_Node.py", "ekf.py", and "probabilistic_models.py". The last two scripts simply provide functions that are imported at the beginning of the first script.

2.1 "ekf.py" Description

In this python script, the python class "RobotEKF" is defined, it is an implementation of the extended Kalman filter (EKF). The class is initialized by giving the functions "eval_gux", "eval_Gt", and "eval_Vt" which are, respectively, the system dynamics function $g(u_t, x_{t-1})$, the Jacobian of the system dynamics function with respect to the state $G_t = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}}$, and the Jacobian of the system dynamics function with respect to the control input $V_t = \frac{\partial g(u_t, x_{t-1})}{\partial u_t}$. Additionally, the state estimate μ_{t-1}, the covariance estimate Σ_{t-1}, and the noise matrix M_t are initialized. The class has two defined functions, "predict" and "update".

The "predict" function takes as input u_t, μ_{t-1}, Σ_{t-1}, and M_{t-1}. It then uses "eval_gux" to make the current state prediction μ_t and uses "eval_Gt" and "eval_Vt" to make the current covariance prediction Σ_t. This corresponds to evaluating the following equations:

$$\mu_t = g(u_t, \mu_{t-1})$$

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + V_t M_t V_t^T$$

In addition to μ_t and Σ_t, the "update" function takes as input z, Q_t, "eval_hx", and "eval_Ht" which are, respectively, the current measurement, the measurement noise matrix, the sensor output function $h(x_t)$, and the derivative of the sensor output function with respect to the state $H_t = \frac{\partial h(x_t)}{\partial x_t}$. First, the innovation covariance and expected sensor output, S_t and $\hat{z}$, are calculated. This corresponds to evaluating the following equations:

$$S_t = H_t \Sigma_t H_t^T + Q_t$$

$$\hat{z} = h(\mu_t)$$

Then, the Kalman gain, K_t, is calculated as follows.

$$K_t = \Sigma_t H^T S_t^{-1}$$

Finally, the predictions μ_t and Σ_t are updated.

$$\mu_t = \mu_t + K_t(z - \hat{z})$$

$$\Sigma_t = (I - K_t H_t)\Sigma_t$$

2.2 "PROBABILISTIC_MODELS.PY" DESCRIPTION

In this python script, two python functions are defined, "velocity_mm_simpy" and "landmark_sm_simpy". The first function outputs "eval_gux", "eval_Gt", and "eval_Vt", which correspond to $g(u_t, x_{t-1})$, G_t, and V_t from the previous section, respectively. The second function outputs "eval_hx" and "eval_Ht", which correspond to $h(x_t)$ and H_t from the previous section, respectively.

2.3 "EKF_NODE.PY" DESCRIPTION

For task 1 and task 2, this script is slightly different because of the addition of 2 states, the linear and angular velocities. However, the main ideas are the same. The following is a description of task 1

In this python script, the ROS2 Node "EKF" is defined. The node subscribes to two topics, /odom and /landmarks. The node also publishes Odometry messages to one node /ekf. The node is initialized by defining "self.ekf", which is done by using the "RobotEKF" class described in Section 2.1. The "RobotEKF" class requires the functions that were described in Section 2.2. The node has 4 main functions.

The first function is “timer_callback”, which is called at a frequency of 20 Hz. This function performs the “predict” step of the EKF.

The second function is “landmark_callback”, which is called every time a message is published to the /landmarks topic. This function performs the “update” step of the EKF by using the landmark sensor output model described in “landmark_sm_simpy”. Finally, the state estimate μ_t is published to the /ekf topic as an Odometry message.

The third function is “odom_callback”, which is called every time a message is published to the /odom topic. This function updates the u_t vector with the current linear and angular velocities for the prediction step of the EKF.

The fourth and final function is “load_landmarks” which is called once at the initialization of the EKF node to load the file that contains the landmark locations. This is used in the “landmark_callback during the EKF update step.

The only difference in task 2 is a change to the velocity motion model to include the linear and angular velocity. In addition, we also add a callback function for the wheel encoder data and the IMU data with the corresponding sensor functions and Jacobians to do the update part of the EKF.

3 Experiments

3.1 Simulation

In the first stage, the package will be designed, developed, and validated in a simulated environment using Gazebo. Gazebo provides a realistic physics simulation where the robot can interact with various sensors and landmarks, allowing for iterative testing and refinement of the localization algorithm.

The robot operated in a predefined map featuring static landmarks at known locations, which provided relative positional measurements. The odometry system supplied linear and angular velocities, while the IMU and wheel encoders measured orientation and motion dynamics. These sensors were used to implement an EKF for fusing data and improving localization accuracy. The static landmarks served as external references to correct drift from odometry and IMU, enabling the evaluation of EKF performance under controlled conditions in simulation and real-world tests.

After debugging, about five experimental runs were conducted using teleop_key function as a controller to move the robot. It enabled to get useful data and fine-tune the noise parameters Q_t (process noise) and M_t (measurement noise). The experiments aimed to evaluate and optimize the Extended Kalman Filter's performance. Key metrics recorded included localization error, measured as the difference between the estimated pose and the ground truth, focusing on the EKF’s runtime and processing delays. These metrics ensured the system's accuracy and efficiency under varying noise conditions.

3.2 Real Robot

After successful simulation validation, the EKF package was deployed on a physical robot in a controlled lab environment. In real-world tests, a camera replaced the simulated /landmarks topic, providing landmark data. The robot was manually controlled using the teleop_key function.

A mismatch in the initial pose occurred due to null values in the position initialization, requiring a near-zero value (e.g., 0.0001) for numerical stability.

Finally, real-world conditions introduced differences compared to the simulation, such as sensor noise and environmental variability, which affected system performance. These discrepancies and their impact on localization are analyzed in the following section.

4 Results and Discussion

In each task, the results of this implementation of the EKF to estimate the states was compared to the results in the /odom topic for both the real robot and the simulation. The results are presented and discussed in this section.

4.1 Task 1

In Task 1, the EKF was implemented to estimate the x-y position as well as the yaw angle of the robot in a Gazebo simulation.

As you can see in Figure 1, at the beginning of the simulation, the state estimation of the EKF is relatively good. However, as the movement of the robot becomes more dynamic later in the simulation, the state estimation deteriorates. For example, around 130 seconds, there is a big jump in the yaw angle which heavily perturbs the state estimation of the EKF. After the jump though, we can see that the EKF is starting to converge back on to the true state before the simulation is cut off.

This may be due to the linearization that is required by the EKF to work. If the state changes too rapidly, the linearization becomes a worse approximation of the dynamics and thus the EKF is less effective. This could be mitigated by tuning the Q_t and M_t matrices. This deterioration of the state estimation is can also be seen in Figure 2, where the X-Y position estimation by /ekf and /odom is plotted against the true position in /ground_truth. Clearly, /odom provides a much better estimation of the state of the robot than the presented EKF.

The Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE) metrics were computed for the /ekf and /odom topics relative to the true position given in the /ground_truth topic. The results are presented in Table 1. Again, it is clear that /odom gives a better estimation of the state.

Metric	/ekf	/odom
RMSE	51.35	1.19
MAE	19.17	0.03

Table 1. Error Metric Results for /ekf and /odom in Task 1

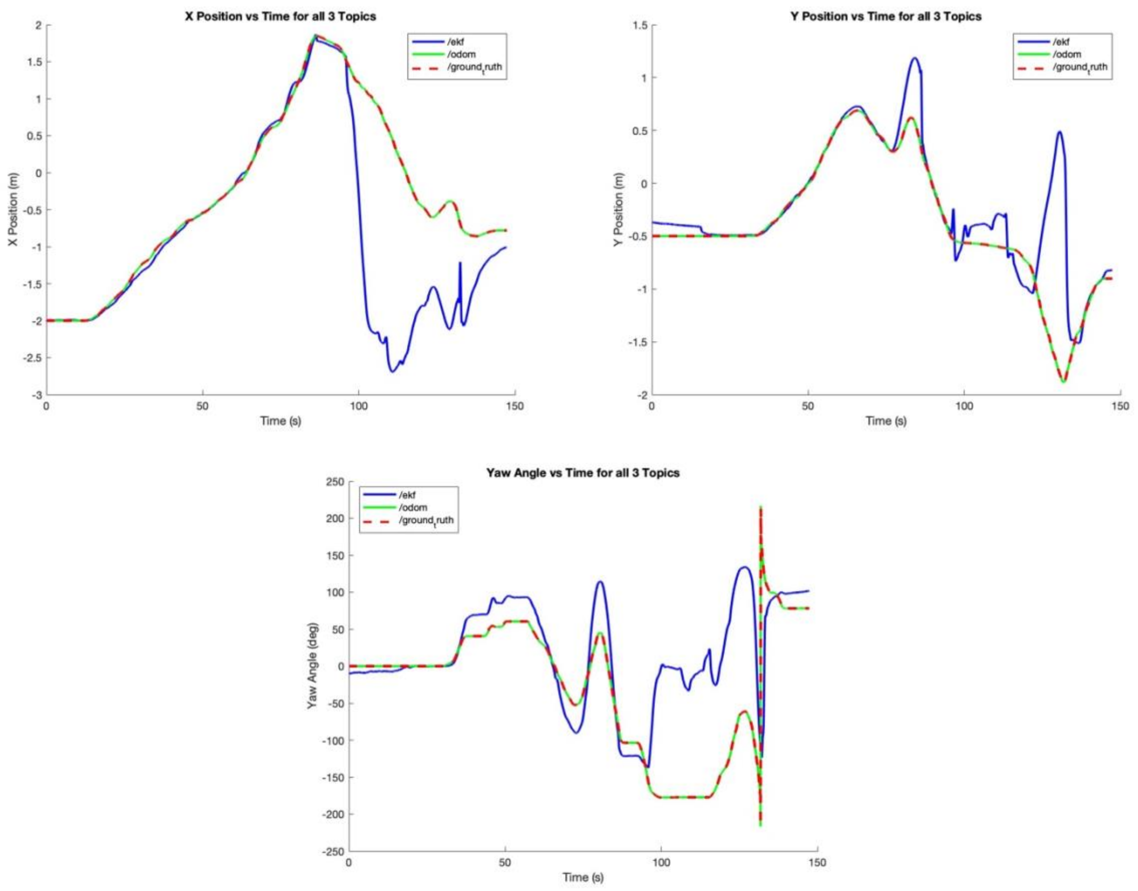

Figure 1. X, Y, and Yaw vs Time for all 3 Topics Task 1

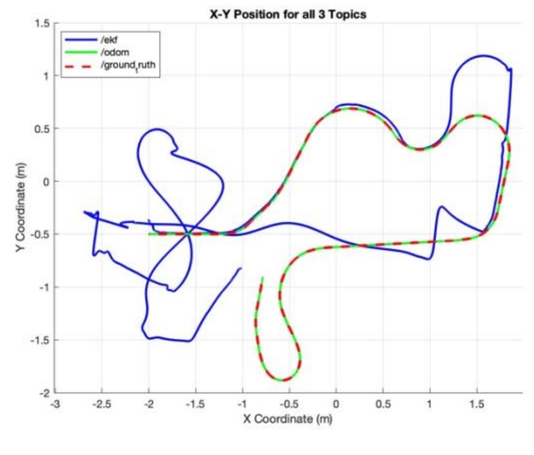

Figure 2. X-Y Position Estimation of the Robot vs /ground_truth in Task 1

4.2 Task 2

In Task 2, the EKF was implemented to estimate the x-y position, the yaw angle, as well as the linear and angular velocities of the robot in a Gazebo simulation.

As you can see in Figure 3, at the beginning of the simulation, the state estimation of the EKF is relatively good for all 5 states. However, around 60 seconds in, the state estimation deteriorates severely for all 5 states. Eventually, around time 70 seconds the estimation of the x-y position, yaw angle as well as the angular velocity goes back to being relatively good. However, the estimation of the linear velocity remains quite poor, though it does improve.

The severe drop in accuracy for that short period may be due to the linearization that is required by the EKF to work. If the state changes too rapidly, the linearization becomes a worse approximation of the dynamics and thus the EKF is less effective. As can be seen in the plot of the ground truth, at around 60 seconds the yaw angle drops almost 360 degrees. This is a sudden change in the state that can impact the estimation. In addition, as can be seen in the angular velocity, the robot was rapidly changing directions during that time, which would add to the error due to linearization. This effect could be mitigated by tuning the Q_t and M_t matrices.

This deterioration of the state estimation is can also be seen in Figure 4, where the X-Y position estimation by /ekf and /odom is plotted against the true position in /ground_truth. Clearly, /odom provides a much better estimation of the state of the robot than the presented EKF. We can also observe after 70 seconds, when the estimated state converges back onto the true state.

The Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE) metrics were computed for the /ekf and /odom topics relative to the true position given in the /ground_truth topic. The results are presented in Table 2. Error Metric Results for /ekf and /odom in Task 2. Again, it is clear that /odom gives a better estimation of the state. However, there is an improvement relative to the results in task 1.

Metric	**/ekf**	**/odom**
RMSE	39.54	1.41
MAE	7.11	0.061

Table 2. Error Metric Results for /ekf and /odom in Task 2

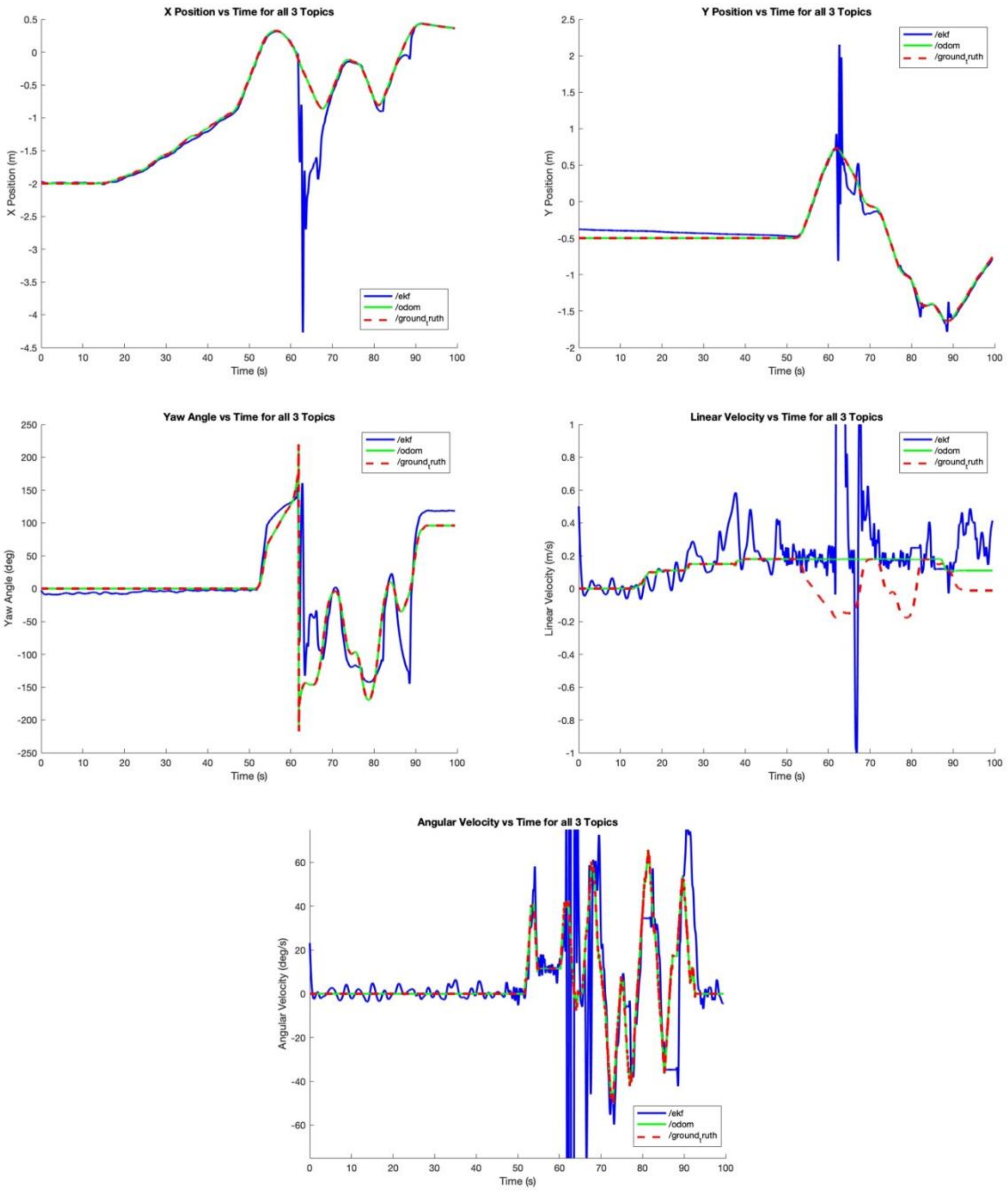

Figure 3. X, Y, Yaw, Linear Velocity, and Angular Velocity vs Time for all 3 Topics Task 2

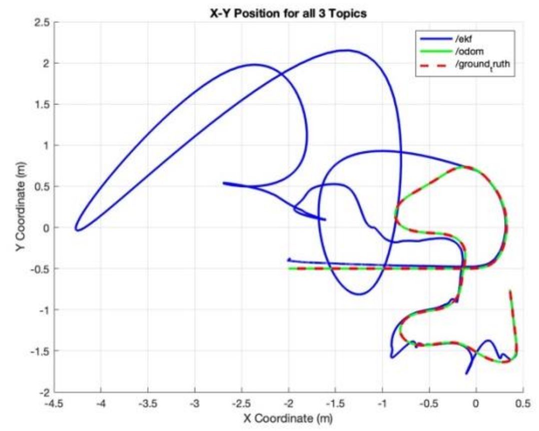

Figure 4. X-Y Position Estimation of the Robot vs /ground_truth in Task 2

4.3 Task 3

In Task 3, the EKF was implemented to estimate the x-y position, as well as the yaw angle of the robot, just as in task 1, but on a real robot. We assume that /odom is the more accurate estimation of the state.

As can be seen in Figure 5, at the beginning of the simulation, when the robot is static, the estimation is good. As the robot starts moving however, the estimation of the EKF starts to deteriorate a bit, but still tracks well with /odom. One can note some big spikes in the estimation of the EKF at certain times, but the EKF quickly recovers. In general, the state estimation of the EKF is quite satisfying. The x-y position graph is shown in Figure 6. The big spikes in the data are very apparent, which makes it seem like the estimation is very poor.

The discrepancy in the accuracy of the EKF may be due to multiple reasons. One reason might be slippage of the wheels. During the experiment, it was clear that at some points the wheels of the robot were slipping on the ground. This could make the prediction step of the EKF inaccurate as there is a difference between the model it has and the real world. Another aspect might be the inaccurate placement of the landmarks. Since the landmarks were placed by hand based on given coordinates, there is no guarantee that the landmarks were exactly where the EKF thought they were. This could severely impact the landmark sensor update step of the EKF, as it heavily relies on the location of the landmarks to update the state. Basically, the combination of differences between the EKF internal model and real life, as well as the differences between the real-life landmark locations and the data the EKF was given could explain the inaccuracies in the EKF state estimation.

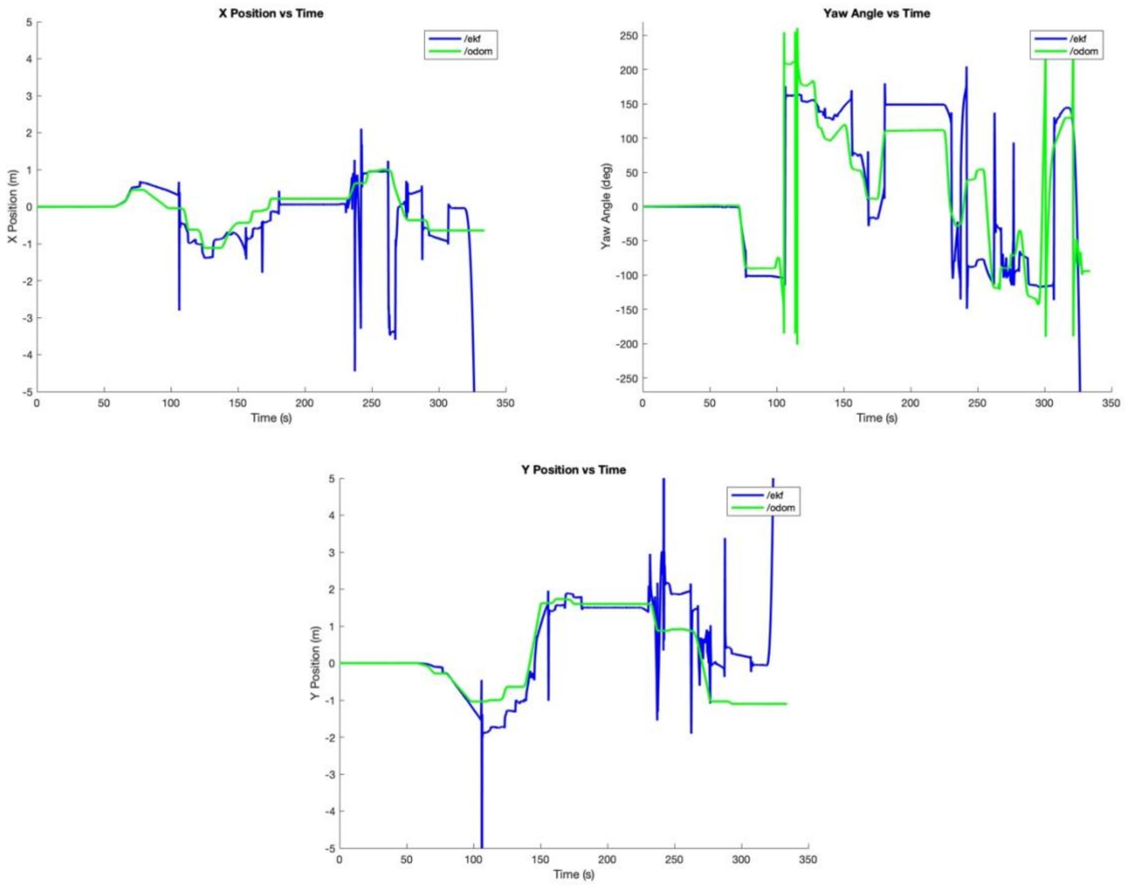

Figure 5. X, Y, Yaw vs Time for Task 3

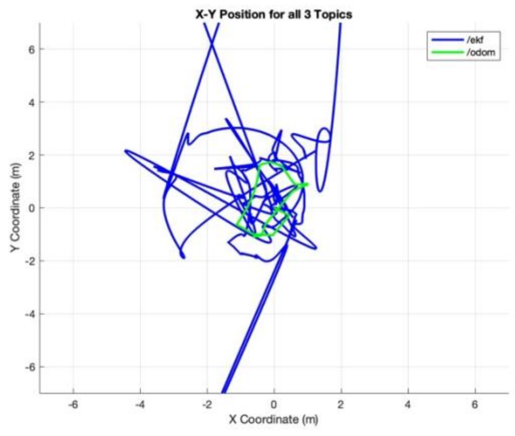

Figure 6. X-Y Position Estimation of the Robot in Task 3