Sensor Fusion and Localization for Tow Truck Autonomous Navigation in Industrial Environments

Ailiya Fatima, Afsah Hyder, Shaheer Abbass, Syed Ali Nisar Shah

***Abstract*— This project explores the localization and navigation of a tow truck with two trailers in a simulated industrial setting, leveraging advanced sensor fusion and localization algorithms. The motivation lies in enhancing operational efficiency in industrial environments, where automated tow trucks can streamline logistics. Key methodologies include Extended Kalman Filter (EKF) for sensor fusion, integrating data from IMU and odometry sensors, and Adaptive Monte Carlo Localization (AMCL) for robust map-based localization with LiDAR. The EKF combines odometry and IMU data to refine pose estimation, while AMCL utilizes a particle filter to determine the robot's precise location on a LiDAR-generated point-cloud map. Comparative analysis of the algorithms highlights performance metrics such as localization error and resilience to noise. Simulation-based testing validated the system under varying scenarios, including dynamic objects. This work contributes to mobile robotics by addressing the challenges of multi-trailer vehicle automation in constrained industrial environments.**

I. INTRODUCTION

Mobile robot autonomy is a crucial component of modern robotics, particularly in industrial applications, where automation plays a significant role in optimizing operational efficiency. One of the primary challenges in achieving full autonomy is the *localization* problem, which involves determining the robot's precise position and orientation within a given environment. In real-world scenarios, this problem can expand into related tasks such as *Simultaneous Localization and Mapping* (SLAM), *path following*, and *obstacle avoidance*, all of which contribute to enabling a robot to navigate autonomously in its environment.

For a mobile robot such as a tow truck with two trailers, accurate localization is the first and foremost task in enabling effective navigation within an industrial facility. Localization, in this context, refers to estimating the position of the tow truck and its trailers either relative to a pre-existing map or in a dynamic environment where the map may change due to obstacles. Solving this problem is vital as it ensures that the robot can make informed decisions about movement, path planning, and obstacle avoidance without requiring human intervention [1], [13].

A. Toy Example: Navigating a Tow Truck in an Industrial Environment

To better understand the importance of localization, consider a tow truck with two trailers navigating through an industrial environment. The truck must follow a designated route, avoid dynamic obstacles such as pedestrians or other vehicles, and handle challenges like narrow aisles and ramps. To successfully perform these tasks, the truck must know its precise position and orientation at any given time—this is where localization becomes critical.

For example, as the truck moves through the facility, its localization system uses sensor data from devices such as Inertial Measurement Units (IMUs) and LiDAR (Light Detection and Ranging) to estimate the truck's position. These sensor measurements must be processed, corrected for noise, and fused to provide a reliable estimate of the robot's location. Without accurate localization, the truck could miss dynamic obstacles, fail to follow the designated path, or deviate from its intended route, leading to potential inefficiencies or even accidents.

B. Major Challenges in Solving the Localization Problem

Several challenges complicate the localization task for mobile robots, especially in complex industrial environments:

- **Sensor Noise and Inaccuracies**: Sensors such as LiDAR and IMUs are prone to noise, which can degrade the precision of localization estimates [3].
- **Dynamic Obstacles**: Industrial environments are often populated by moving objects (e.g., humans or other vehicles), which must be accounted for when localizing the robot and adjusting its navigation accordingly.
- **Non-Ideal Environments**: Environments with narrow spaces, ramps, and low visibility can make localization more difficult. These settings also often contain inaccuracies in the map, further complicating the localization process.
- **Multi-Body Systems**: In systems such as a tow truck with trailers, accurately modeling the kinematics of the robot and its attached components, particularly when making sharp turns or other complex maneuvers, is a significant challenge for localization algorithms [14].

C. Approaches to Solving the Localization Problem

There are several well-established methods for solving the localization problem, each with its advantages and limitations. Some of the key approaches include:

- **Odometry-Based Localization**: This method uses wheel encoders to estimate the robot's position by integrating its velocity and turning rate over time. Although simple and efficient, it suffers from drift, especially over long distances [5].
- **Kalman Filtering**: The Extended Kalman Filter (EKF) is a widely used technique for sensor fusion. It combines data from multiple sensors, such as odometry and IMUsto improve the accuracy of position estimates [6].

- **Simultaneous Localization and Mapping (SLAM)**: SLAM techniques allow the robot to both localize itself and build or update a map of its environment. This approach is useful in dynamic environments where a map is not pre-existing [13].
- **Monte Carlo Localization (MCL)**: Adaptive Monte Carlo Localization (AMCL) uses particle filters to estimate the robot's position. It is particularly useful in uncertain environments where the map may be incomplete or noisy [7].

This project leverages a combination of the Extended Kalman Filter (EKF) and Adaptive Monte Carlo Localization (AMCL) to achieve high-precision localization for the tow truck system. These techniques allowed the system to validate odometry, and localize in complex environments while accounting for the challenges discussed earlier.

D. The Importance of Solving the Localization Problem for Autonomy

Accurate localization is critical for achieving full autonomy in mobile robots. It provides the foundation upon which robots can make decisions about movement, path planning, and obstacle avoidance. Without precise localization, a robot would be unable to perform even basic tasks, such as navigating a factory floor or transporting goods, autonomously. Moreover, ensuring robust localization in dynamic and complex environments allows the robot to make real-time decisions that enhance safety, efficiency, and reliability in industrial settings.

II. Related Work

Localization is a critical task in robotics, enabling a robot to determine its position and orientation within a global reference frame. Accurate localization is fundamental for enabling autonomous navigation, path planning, and trajectory tracking. In industrial environments, where both indoor and outdoor settings are involved, localization becomes especially challenging due to factors like dynamic obstacles, environmental variability, and sensor limitations.

A. Indoor Localization Techniques

A wide range of methods has been explored for indoor localization, many of which combine multiple sensor modalities for greater reliability. The **Extended Kalman Filter (EKF)** has become one of the most common approaches for localizing robots in indoor environments by fusing sensor measurements (e.g., LIDAR, cameras, IMUs) with control inputs. Durrant-Whyte et al. [9] demonstrated that EKF can effectively estimate a robot's pose using a combination of sensors in structured indoor environments. Recent advancements have also focused on improving the robustness of these systems, addressing issues such as sensor noise and inaccuracies in odometry.

Stereo vision systems, which estimate depth information using two cameras, have also been explored for indoor localization. These systems rely on detecting key points in the environment and triangulating their positions. While stereo vision offers high accuracy in controlled settings, its performance can be limited by environmental factors such as lighting conditions and textureless surfaces. Mustafah et al. [10] and Firdaus et al. [11] have explored the use of stereo vision for localization in indoor environments. However, their studies highlight challenges such as motion-induced blurring and poor accuracy under low-light conditions.

B. LIDAR-Based Localization and SLAM

LIDAR-based localization has been widely adopted in both research and industrial applications due to its ability to generate highly accurate range measurements of the environment. These measurements can be used to perform **Simultaneous Localization and Mapping (SLAM)**, where both the robot's position and the environment's map are continuously updated. According to Fasiolo et al. [12], LIDAR-based SLAM remains a robust and reliable method for indoor localization, even in the presence of dynamic obstacles and challenging environmental conditions. Their experimental setup, which incorporated various SLAM algorithms such as **Hector SLAM**, **Cartographer**, and **Gmapping**, highlighted Hector SLAM's superior performance in terms of accuracy and reliability in LIDAR-only localization scenarios. These findings are consistent with other studies, which also affirm LIDAR's effectiveness in structured indoor environments.

One significant challenge for LIDAR-based systems is the presence of low-feature environments, where the absence of distinct landmarks can degrade the accuracy of localization algorithms. Additionally, rapid movements or rotations can lead to discrepancies in position estimates. To mitigate these issues, sensor fusion techniques are commonly used, integrating **odometry data** from **wheel encoders** and measurements from **inertial measurement units (IMUs)**. This integration enables a more accurate estimate of the robot's state, even in challenging conditions. Studies by Durrant-Whyte and Bailey [13] further corroborate the advantages of combining LIDAR data with IMU and odometry information through **EKF-based localization** for improved performance.

C. Sensor Fusion for Robust Localization

Sensor fusion has emerged as a critical technique for enhancing the robustness and accuracy of localization systems. By combining **LIDAR data** with other sensor measurements such as **IMU** readings, **odometry**, and **stereo vision**, localization algorithms can address the limitations inherent in each individual sensor type. According to a study by Bajcsy and Liu [14], sensor fusion can significantly reduce the error caused by drift in odometry measurements and improve localization accuracy, especially in environments with sparse features. Moreover, integrating multiple sensors provides redundancy, which increases the system's reliability in dynamic and cluttered industrial environments.

Recent advancements in sensor fusion have focused on **LiDAR-inertial systems**, which combine the high accuracy of LIDAR with the fast-response characteristics of IMUs. The work by Fox and Thrun [15] demonstrated that such

integrated systems offer superior performance in both localization and mapping, particularly in complex environments where rapid movement and environmental uncertainty are prevalent.

D. Challenges and Future Directions

Despite the advancements in localization techniques, several challenges remain. Sensor noise, especially from LIDAR and IMU measurements, can still impact the accuracy of localization systems. Moreover, **dynamic obstacles** present a significant hurdle, as they can alter the environment in real-time, requiring localization algorithms to continuously adapt. The need for **real-time processing** in industrial settings further adds to the complexity, as algorithms must balance computational efficiency with accuracy.

Future research directions could include exploring **machine learning-based approaches** for dynamic obstacle detection and avoidance, which could complement traditional localization algorithms. Additionally, integrating **high-precision LiDAR** with advanced filtering techniques such as **particle filters** or **deep learning models** could further enhance the system's ability to maintain accurate localization in challenging, dynamic environments.

By synthesizing the aforementioned localization techniques and sensor fusion methodologies, this project aims to integrate these systems into the operation of a tow truck with two trailers, facilitating precise navigation in industrial settings.

III. System Architecture

The robotic system is built upon a structured and modular architecture designed to facilitate autonomous navigation and localization in industrial environments.

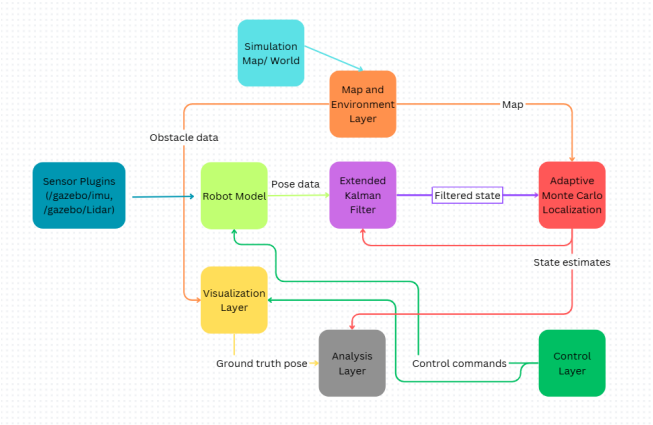


Fig. 1. System Architecture

At its core, the system integrates multiple layers that enable seamless data flow and real-time decision-making. The *Sensor Layer* consists of plugins within the Gazebo simulation environment, including an Inertial Measurement Unit (IMU) and a LiDAR sensor, which capture motion and environmental data. This data is processed in the *Robot Model Layer*, which simulates the kinematics and dynamics of a tricycle-configured tow truck towing two trailers. The Robot Model computes the pose data required for subsequent localization steps.

The *Map and Environment Layer* provides a pre-defined static map of the industrial environment, which features ramps, sharp turns, and narrow pathways, ensuring accurate localization and obstacle representation. This map serves as the foundation for the *Localization Layer*, which employs two key algorithms. The *Extended Kalman Filter (EKF)* fuses pose data from the Robot Model with sensor IMU inputs to generate a filtered state, while *Adaptive Monte Carlo Localization (AMCL)* utilizes LiDAR observations to refine the robot's position by aligning detected landmarks with the static map. The localization results are subsequently passed to the *Control Layer*, which generates navigation commands, and the *Visualization Layer*, which provides real-time monitoring and debugging capabilities. Finally, the *Analysis Layer* evaluates the system's performance by comparing ground truth poses with the computed localization estimates and logging relevant data. This layered architecture ensures the system's ability to process sensor data effectively, adapt to dynamic industrial environments, and deliver accurate navigation performance.

IV. System Details

In this project, we utilized the Nav2 stack for localization and sensor fusion, leveraging its robust set of packages for autonomous navigation in ROS 2 environments. The tricycle drive model was employed in our simulation, where the robot's dynamics are approximated by a simplified model with a single steering wheel at the front and two drive wheels at the rear. This model is suitable for simulating the kinematics of vehicles like the truck-trailer system.

A. Robot Model

The SDF (Simulation Description Format) model of the truck-trailer system was built to represent this drive model, accurately reflecting the vehicle's motion and constraints in the Gazebo simulator. This configuration allows for realistic motion planning and control, considering the system's constraints such as turning radii and wheelbase.

For odometry, we relied on the drive plugin within the tricycle drive model. The plugin generates odometry outputs, representing the vehicle's position and orientation over time, which is essential for the localization process. The plugin's odometry output is used as input to the Nav2 localization packages, providing real-time pose estimates that are crucial for effective path planning and navigation. This integration ensures that the localization system accurately tracks the vehicle's movement within the environment, which is essential for real-time decision-making and collision avoidance.

By combining these tools and models, we ensured that the truck-trailer system could navigate within a simulated

environment, using real-time data to adjust its behavior and improve the accuracy of its localization and sensor fusion techniques.

B. Extended Kalman Filter

The Extended Kalman Filter (EKF) in the `robot_localization` package in ROS2 is designed to fuse sensor data to provide an accurate state estimate of a robot. It works in a systematic way to process data from multiple sensors and estimate the robot's position, velocity, and orientation over time. The EKF maintains a state vector x, which represents the robot's state as:

$$x = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \text{roll} & \text{pitch} & \text{yaw} & \dot{\text{roll}} \\ \dot{\text{pitch}} & \dot{\text{yaw}} \end{bmatrix}^T$$

where:

- x, y, z: Position in the map frame
- $\dot{x}, \dot{y}, \dot{z}$: Linear velocities
- roll, pitch, yaw: Orientation angles
- $\dot{\text{roll}}, \dot{\text{pitch}}, \dot{\text{yaw}}$: Angular velocities.

These state vectors elements are coming through both odometry and IMU.
The prediction step and update step of the filter are implemented in the `ekf.cpp` file of the package. The implementation of the prediction step in the code is shown below:

```
// (2) Project the state forward: x = Ax + Bu (really, x = f(x, u))
state_ = transfer_function_ * state_;

// Handle wrapping
wrapStateAngles();

FB_DEBUG(
  "Predicted state is:\n" <<
    state_ << "\nCurrent estimate error covariance is:\n" <<
    estimate_error_covariance_ << "\n");

// (3) Project the error forward: P = J * P * J' + Q
estimate_error_covariance_ =
  (transfer_function_jacobian_ * estimate_error_covariance_ *
  transfer_function_jacobian_.transpose());
estimate_error_covariance_.noalias() +=
  delta_sec * (*process_noise_covariance);
```

The EKF uses a first-order linearization to handle nonlinearities in the system dynamics and measurement models. This is done by computing the Jacobian matrices for motion and measurement model.
The EKF is highly configurable. To use EKF we can configure the following in a .yaml file:

- Sensor topics: IMU, odometry, GPS, etc.
- Data fusion settings: Specify which variables to use (e.g., x, y, velocities).
- Covariances: Define process noise (Q) and measurement noise (R).
- Frame transforms: Ensures all sensor data are transformed into a common frame.

The EKF outputs a fused state estimate as an odometry message (on the topic `/odometry/filtered`), which includes:

- Pose: Position and orientation
- Twist: Linear and angular velocities
- Covariance: State uncertainty.

C. Visualizations for Understanding Truck-Trailer Motion

We performed some simulations in Gazebo to understand the motion of a truck-trailer system by modeling the kinematics and dynamics. There are two simulations:

1) A manually-controllable at-scale differential drive tow truck towing two trailers that are modelled after Toyota Indus Motor Company's tow truck and grill doleys, respectively
2) A scaled-down autonomous tricycle model robot towing two trailers

To connect the actuated robot and trailers together, we used revolute joints, similar to hitches used in real life, but constained them to +- 90 degrees along the z-axis to prevent severe overshoots.

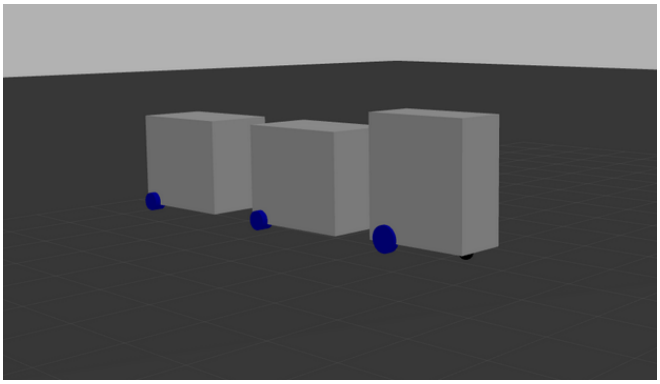


Fig. 2. Setup for Simulation 1

In simulation 1, the robot has a length, width, and height of 1.645, 0.8, and 1.8 meters respectively, while the trailers have 1.8, 1.11, and 1.5 meters as their respective dimensions. The only difference in dimensions between the simulation and real life counterpart is that we decreased the height of the trailers by about 30% to lower the center of mass to prevent toppling because we used teleop twist keyboard which resulted in sharp movements.
We executed simple and extreme maneuvers to understand how a truck towing trailers would behave. We noted how the trailer joints move and affect the motion of the truck, and the fact that if we placed a point on the system, the farther away it was from the tow truck, the more it deviated outwards from the path the truck followed.

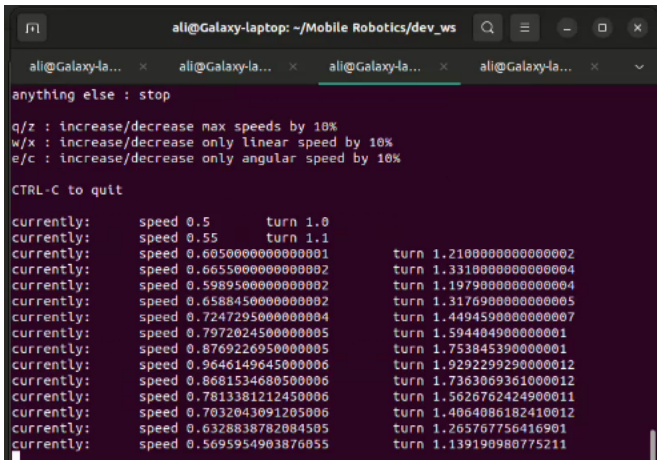


Fig. 3. Using Teleop Twist Keyboard for Manual Control

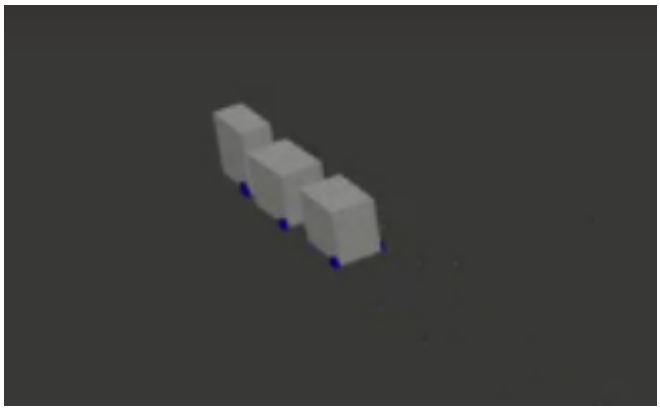


Fig. 4. Truck-trailer System Going Straight After a Turn

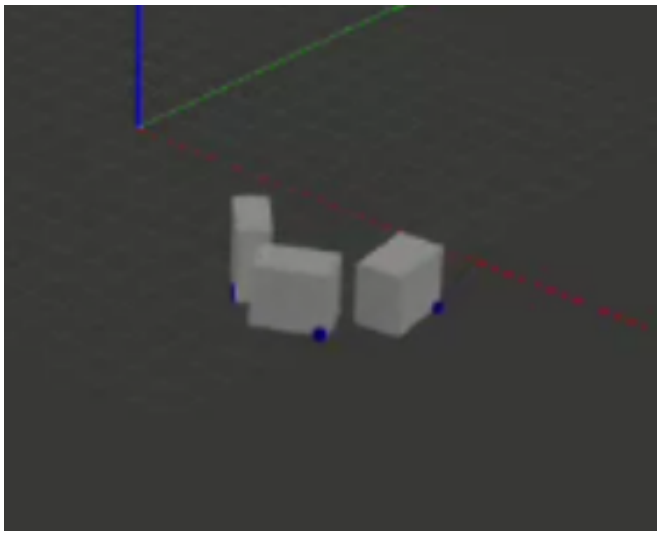


Fig. 5. Truck-trailer System Going in Circles

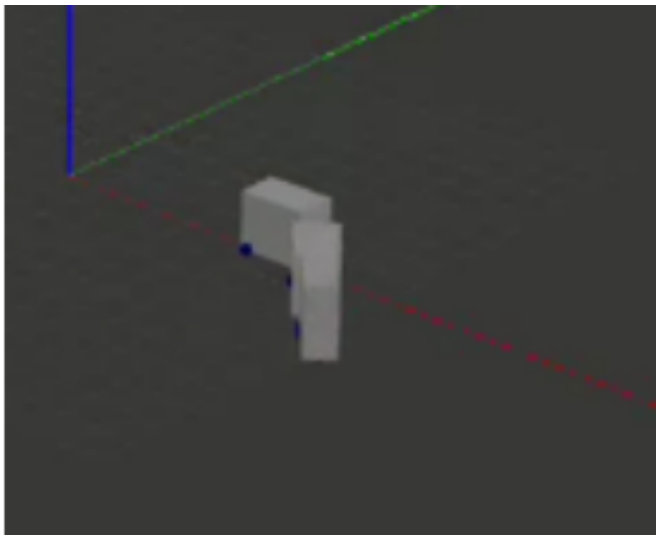


Fig. 6. Truck-Trailer System Taking a Sudden Right

In simulation 2, we took the autonomous tricycle model robot of length, width, and height of 0.7, 0.39, 0.2 meters, respectively, and modified it by adding two trailers of same dimensions 0.5, 0.3, 0.2 meters respectively. In this simulation, we modelled dedicated hitch joints visually with collision and inertia and added the hitch point half way of the entire hitch bar length (about 0.25 meters) providing us a look at the dynamics of a tow truck-trailer system that is closer to real life in its hitching mechanism.
Executing turns and more complex zig-zag movements allowed us to understand the impact of a dedicated hitch on a tricycle model tow truck and trailer system.

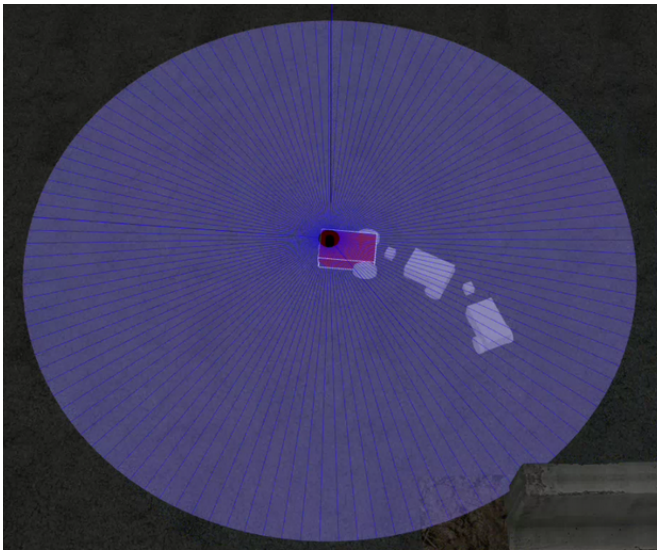


Fig. 7. Truck-Trailer System Taking a Left Turn

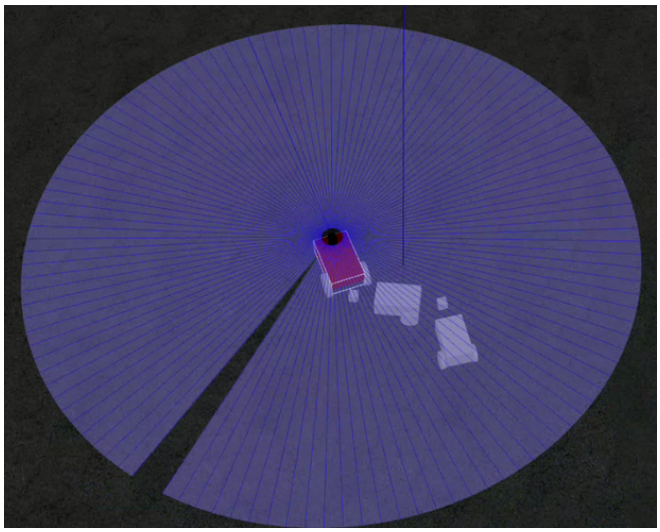


Fig. 8. Truck-Trailer System Executing a Zig-Zag Motion

The simulations provided us with a ballpark of what to expect with respect to the dynamics of the tow-truck trailer system like the hitches angles during motion and allowed us to quickly perform sanity checks on our results.

D. Challenges

A major thing that we were unable to implement was to include the trailer angles for sensor fusion and localization. The reason for this is that for this project we have been utilizing Nav2 stack packages including EKF and AMCL. The pre-written .yaml and config files for these algorithms work for the state vector

$$\begin{bmatrix} x\_pos & y\_pos & z\_pos \\ \text{roll} & \text{pitch} & \text{yaw} \\ x\_vel & y\_vel & z\_vel \\ \text{roll\_vel} & \text{pitch\_vel} & \text{yaw\_vel} \\ x\_accel & y\_accel & z\_accel \end{bmatrix}$$

To add the trailer angles to our state vector and modify our kinematic model accordingly, we cannot directly configure robot_localization's YAML file to process this new state. This is because the robot_localization package is hardcoded to handle a 15-element state vector and editing that would mean editing the source code and all other node dependancies of the packages. So we realized that it was a great deal of work within the scope of this project.

V. Experimental Results

A. EKF Parameter Tuning

The performance of the algorithms (for EKF and for AMCL in the next section) is evaluated through a process of finding error. A python script namely "error.py" takes in the topics of ground truth and estimated pose for each algorithm separately and then calculates the difference in x,y and θ between the two to understand how accurate the estimate is. Since the orientation is given in quaternion angles, it is converted to radians using:

$$\theta = 2\cos^{-1}(w)$$

Where:

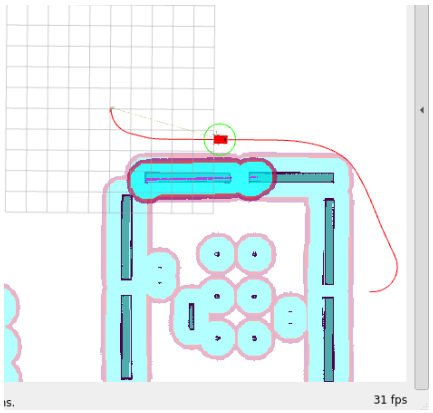
1) w is the scalar part of the quaternion.
2) θ is the rotation angle in radians.

The error plots contain error in x,y and theta for a case of fixed path given to the robot. To analyze the performance of the Extended Kalman Filter(EKF) for a given fixed path in a world, we varied its different parameters and observed their imapcts on the error.
The parameters for EKF include Table I:

TABLE I
Parameter Effects for System Behavior

Parameter	Effect
`frequency`	Increasing this value leads to faster updates, but higher computational load. Lower values result in slower updates and delayed convergence.
`sensor_timeout`	Higher values allow for more sensor delay tolerance but risk stale data; lower values ensure fresher data but may discard delayed sensor inputs.
`process_noise_covariance`	Higher values make the filter trust the process model more, leading to slower convergence but smoother estimates. Lower values make it trust sensor measurements more.
`initial_covariance`	Higher values result in greater initial uncertainty, leading to slower filter convergence. Lower values speed up convergence but may assume incorrect confidence.
`transform_timeout`	Increasing this value allows delayed transformations but risks outdated transforms. Lower values enforce stricter timings for fresher data.
`two_d_mode`	Enabling this mode restricts the filter to two dimensions, simplifying computations but ignoring vertical motion.
`map_frame`	Defines the reference frame for state estimation. Incorrect values will result in inconsistent or incorrect pose estimates.
`odomX_pose_rejection_threshold`	Sets the Mahalanobis distance threshold for rejecting outlier poses
`odomX_twist_rejection_threshold`	Sets threshold for rejecting outlier velocity data.

The fixed path we considered for this algorithm was



For our analysis, we considered the parameters with their normal setting as:

1) Frequency = 30Hz
2) Sensor Timeout= 0.1s
3) odom0_pose_rejection_threshold= 5.0
4) odom0_twist_rejection_threshold= 1.0
5) process_noise_covariance = an array that contains all zero off-diagonals and covariances around the value of 0.01õ.03

The error profile for the normal condition looks like:

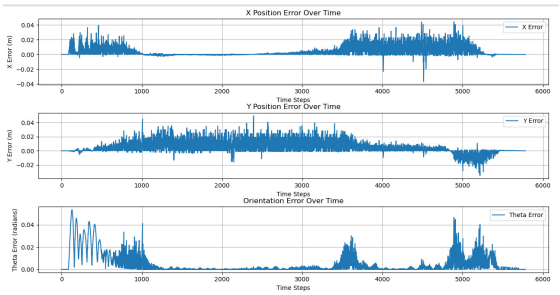


Fig. 9. At default conditions

1) Frequency: The frequency (Hz) at which the filter will output a position estimate. The filter will not begin computation until it receives at least one message from one of the inputs. It will then run continuously at the frequency specified, regardless of whether it receives more measurements. Defaults to 30 if unspecified.
A higher frequency results in more frequent updates, which improves responsiveness but also increase computational load. A lower frequency reduces the system's computational demands but leads to less frequent corrections, affecting the filter's accuracy and responsiveness.

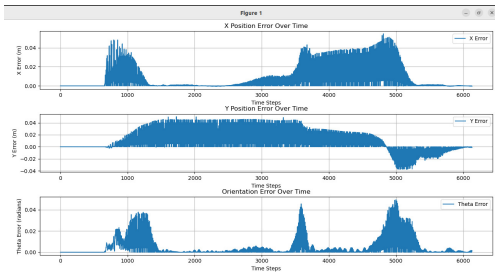


Fig. 10. At frequency=50Hz the error is less

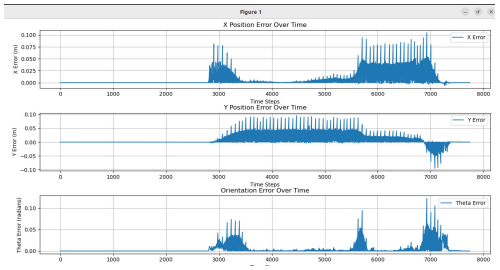


Fig. 11. At frequency=10Hz the error is more

2) Sensor Timeout: The sensor timeout defines the time (in seconds) after which a sensor reading is considered outdated if no new data arrives. If this timeout is too short, valid data is discarded, leading to the filter predicting the state without corrections. If it's too long, stale data incorrectly influences the state estimate, causing inaccuracies or delays in corrections.

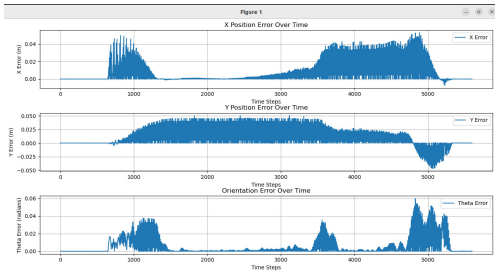


Fig. 12. At sensor timeout=0.001s

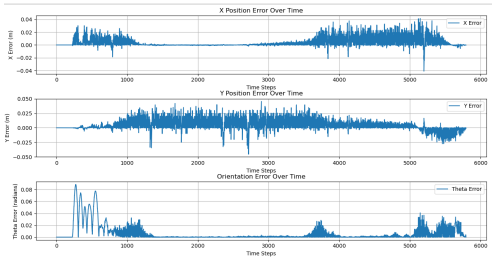


Fig. 13. At sensor timeout=10s

Here for both cases we observe that there's error which makes sense they're both undesirable situations. However having larger timeout results in greater error spikes, especially in the initial theta and the middle point of y.

3) Pose Rejection Threshold: If the odometry pose data (x, y, z, roll, pitch, yaw) differs too much from the predicted state, the filter will ignore that measurement. Increasing this threshold can lead to a more lenient filter, allowing more noisy or inaccurate data, while decreasing it results in stricter filtering but may reject valuable data. This is reflected in the error profile as for greater threshold, the error values increase.

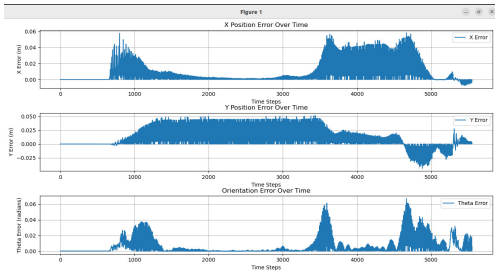


Fig. 14. threshold =1

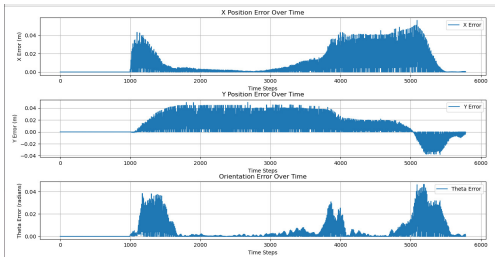


Fig. 15. threshold=10

4) Twist Rejection Threshold: Similarly, the oodometry twist (velocity) data can be rejected if it deviates too much from the expected value. Higher thresholds may reduce false rejections but could allow less accurate data to affect the estimate. However, in our results we didnt observe much difference in the error outputs by changing the velocity threshold. This might be because the change in velocity throughout the robot motion was not very drastic.

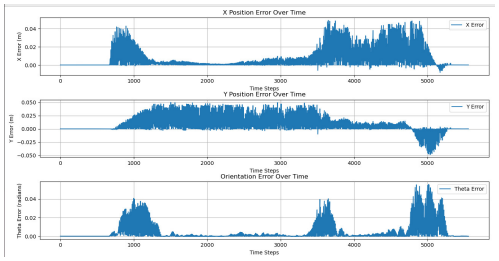


Fig. 16. At threshold = 0.5

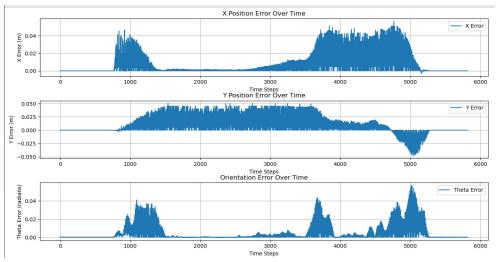


Fig. 17. At threshold = 2

5) Process Noise Covariance: This matrix represents the noise added to the state variables (position, orientation, velocity, acceleration) during the prediction step. It controls how much trust the filter places in the predicted state compared to incoming measurements.
Increasing the process noise for a variable (e.g., velocity) causes the filter to rely more on sensor measurements for that variable and less on the model predictions. This can be useful if the model is not very accurate or the robot's motion is erratic. Conversely, decreasing the process noise means the filter trusts the model more and adjusts less frequently based on sensor input, potentially making it slower to respond to changes in the environment.

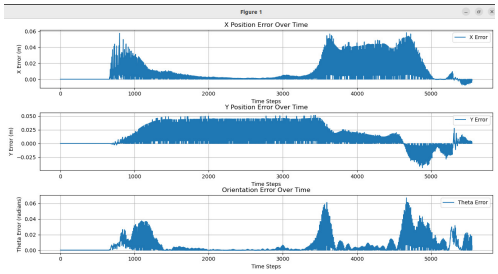


Fig. 18. After increasing the process noise covariance in the x,y,z positions

The error in all three quantities has increased.

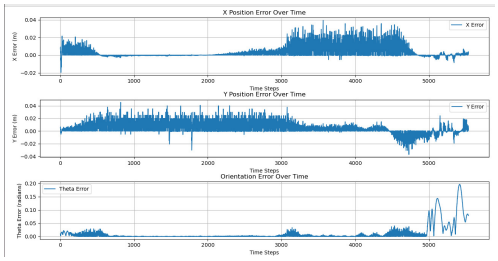


Fig. 19. After increasing the process noise covariance in the x,y,z angles (θ)

For this case we specifically see the increased error in theta

B. AMCL Parameter Tuning

For the AMCL, we tried a different path to also see how changing the path affects the errors:

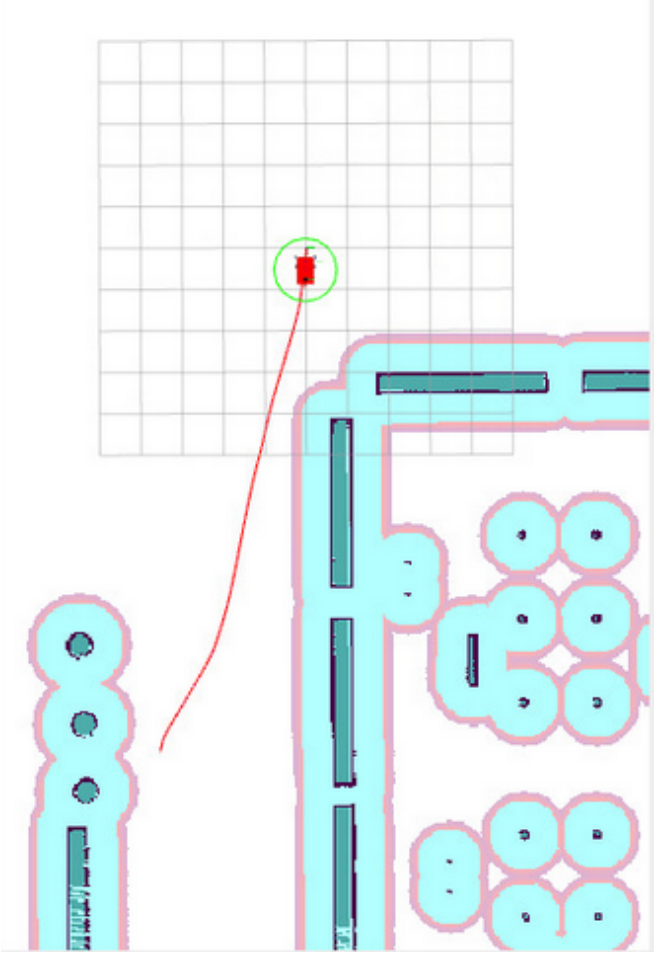


Fig. 20. Path for AMCL.

Since the AMCL's output pose was what was being displayed on RViz2, this is a fairly critical step. Hence, we analyzed the effects of varying the parameters that are written in nav2_params.yaml; in particular, they are listed as follows (default values mentioned):

```
amcl:
  ros__parameters:
    use_sim_time: True % Use simulation time for
        testing in simulated environments.
    alpha1: 0.002 % Rotational noise proportional to
         rotational motion.
    alpha2: 0.002 % Rotational noise proportional to
         translational motion.
    alpha3: 0.002 % Translational noise proportional
         to translational motion.
    alpha4: 0.002 % Translational noise proportional
         to rotational motion.
    alpha5: 0.002 % Noise associated with robot's
        velocity commands.
    base_frame_id: "base_footprint" % Robot's base
        frame for TF transforms.
    beam_skip_distance: 0.05 % Distance threshold
        for skipping beams in scan matching.
    beam_skip_error_threshold: 0.9 % Error threshold
         for beam skipping.
    beam_skip_threshold: 0.3 % Fraction of invalid
        beams to trigger skipping.
    do_beamskip: false % Enable/disable beam
        skipping for performance.
    global_frame_id: "map" % Frame ID of the global
        map.
    initial_pose: [0.0, 0.0, 0.0, 0.0] % Initial
        pose estimate [x, y, yaw, covariance].
    lambda_short: 0.1 % Decay constant for short
        laser readings.
    laser_likelihood_max_dist: 2.0 % Maximum
        distance for laser likelihood calculations.
    laser_max_range: 100.0 % Maximum usable laser
        range.
    laser_min_range: -1.0 % Minimum usable laser
        range.
    laser_model_type: "likelihood_field" % Type of
        laser model for AMCL.
    max_beams: 60 % Maximum number of beams to
        consider in scan matching.
    max_particles: 2000 % Maximum number of
        particles in the filter.
    min_particles: 500 % Minimum number of particles
         in the filter.
    odom_frame_id: "odom" % Frame ID for odometry
        data.
    pf_err: 0.05 % Particle filter error threshold
        for convergence.
    pf_z: 0.99 % Confidence threshold for resampling
         particles.
    recovery_alpha_fast: 0.0 % Fast recovery rate
        for particle adaptation.
    recovery_alpha_slow: 0.0 % Slow recovery rate
        for particle adaptation.
    resample_interval: 1 % Updates before resampling
         particles.
    %robot_model_type: "differential" % Optional,
        specific robot model type.
    motion_model_type: "nav2_amcl::
        DifferentialMotionModel" % Motion model to
        use.
    save_pose_rate: 0.5 % Frequency to save
        estimated pose to disk.
    set_initial_pose: False % Whether to set the
        initial pose manually.
    sigma_hit: 0.2 % Variance of Gaussian for laser
        hit likelihood.
    tf_broadcast: true % Enable/disable broadcasting
         of TF transforms.
    transform_tolerance: 1.0 % Tolerance for TF
        lookups.
    update_min_a: 0.2 % Minimum rotation (radians)
        to trigger an update.
    update_min_d: 0.25 % Minimum translation (meters
        ) to trigger an update.
    z_hit: 0.5 % Weight for laser hit likelihood.
    z_max: 0.05 % Weight for max-range laser
        readings.
    z_rand: 0.5 % Weight for random laser
        measurements.
    z_short: 0.05 % Weight for short laser readings.
    scan_topic: scan % Topic for laser scan input.
```

Other than changing the estimated pose topic to /amcl_pose, the working remains largely the same. We use the same path as in EKF, and calculate the errors the same way we did before.

1) Default Parameters: This can be considered our benchmark for the remaining parameter changes. The reason we see these jumps is because the rate at which /amcl_pose is being published is lesser than that of the ground truth pose, so the error drifts until the estimated pose is updated. Other than that, the reason the errors seem to increase during the middle section is that the robot is traveling through an area where LiDAR scans do not strike against any object,

which means that we are largely reliant on our odometry. Once we reach near the cylinders as seen in the map, the LiDAR scans help out significantly in correcting the pose. We define a good performance as one with x and y errors under 0.5 (so a maximum error of 50cm), and θ error under 0.2 radians – these are intuitive thresholds decided after visually comparing many default configuration simulations with different paths.

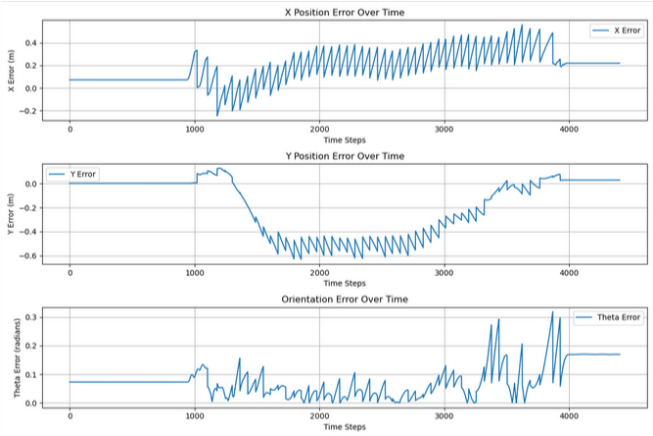


Fig. 21. Running at default settings.

2) Forcing MCL: By setting an equal number of maximum and minimum particles, we are effectively using MCL – no longer is this algorithm adaptive. Performance is still decent, though our x value is somewhat worse, but still within our threshold. However, computational load has increased, and while we did not observe any noticeable effect on the frame rate of Gazebo or RViz2, we believe that this would become more apparent in a more complex system or environment, especially in real life. We tried even higher particle values (at least ten times higher), but RViz did not function when we did so.

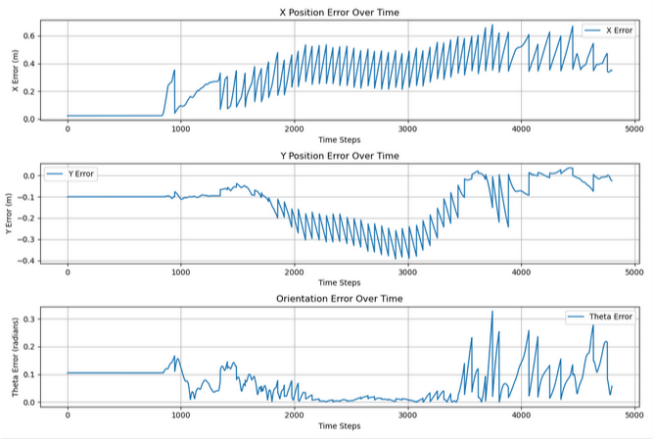


Fig. 22. 2000 minimum and maximum particles.

3) Decreasing Particles: Here, we wondered if our changes were even being applied, so we picked an extreme case of setting very few particles (a minimum of 2 particles and a maximum of 20). The result speaks for itself: the errors in x and y values are now several meters, and the error in theta is nearly half a radian at the end. This is filter divergence, since we do not have enough particles to capture the uncertainty of where we are at.

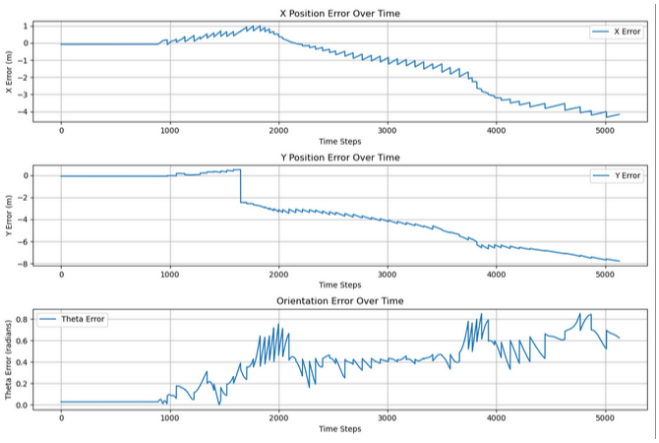


Fig. 23. Very few particles.

4) High Process Noise Covariance: Now, we tuned each of the α values to very high ones, which meant a very high process noise covariance. While x and θ values are not very different from before, we observe an interesting effect with y. Since we are now trusting our odometry much lesser than before, the robot now becomes very uncertain about its position when the sensor information is scant. But once we reached the cylinders, the LiDAR scans dramatically fixed the error! We also tried very low process covariance (each value equal to 0.00002), but RViz stopped working when applied.

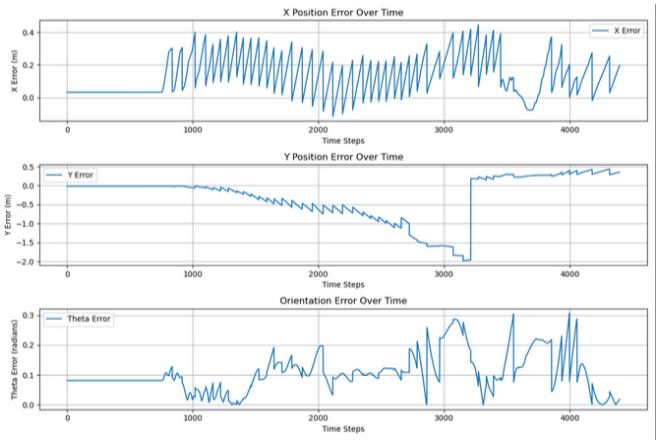


Fig. 24. Setting alphas to very high values.

5) 5 Beams: Here, we are now considering only 5 evenly spaced beams at each iteration of the AMCL algorithm. As expected, the localization has worsened noticeably, even when we approached the cylinders, because we have effectively reduced our sensor data by a factor of ten times. However, this is not without a silver lining: the computational

load is significantly reduced, now that we have lesser data to deal with.

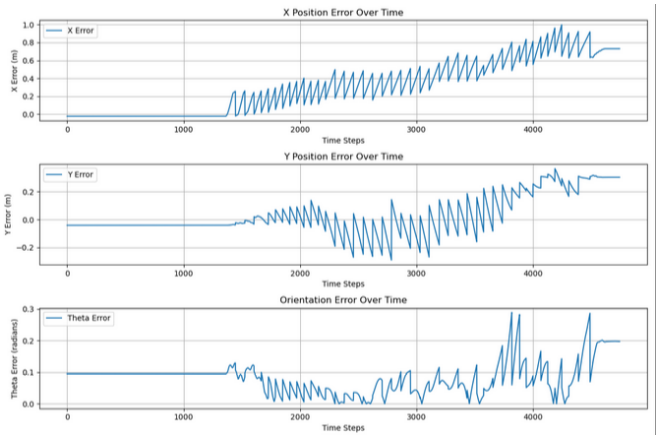


Fig. 25. Using only 5 beams.

6) Dynamic Particle Reduction: Now, we are reducing the particles very quickly here whenever we reach some confidence regarding our current state. This does not seem to have any noticeable effect, except worsening our x value. We believe that this would require a specific situation to lead to filter divergence: when we are somewhat confident about our position, but instead are completely wrong, because there is not much chance of detecting the error now that we have reduced the particles so significantly at just an inkling of confidence.

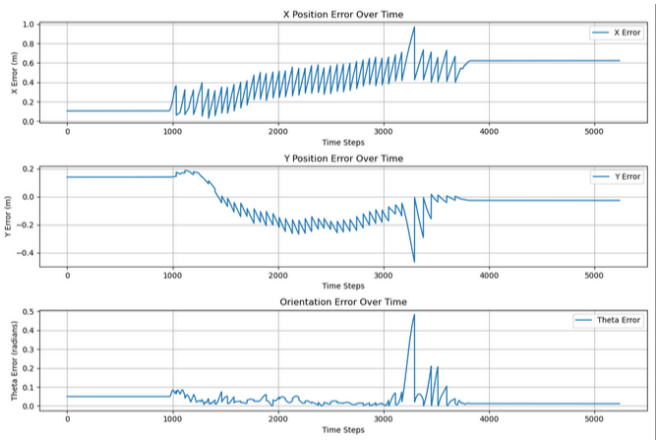


Fig. 26. Aggressive reduction of particles.

7) Sampling Interval: Increasing the resampling interval effectively means that the AMCL runs fewer times, since we now only implement results when the interval has passed. While the final errors are not so different from those of the previous results, we now notice that we have periods where the error keeps increasing during the sampling interval, then it suddenly corrects with the introduction of a new AMCL pose. Note that keeping a sampling interval below 1 (meaning that we now resample more frequently than the AMCL runs, which is senseless) results in RViz not running properly – we are presented with a blank screen.

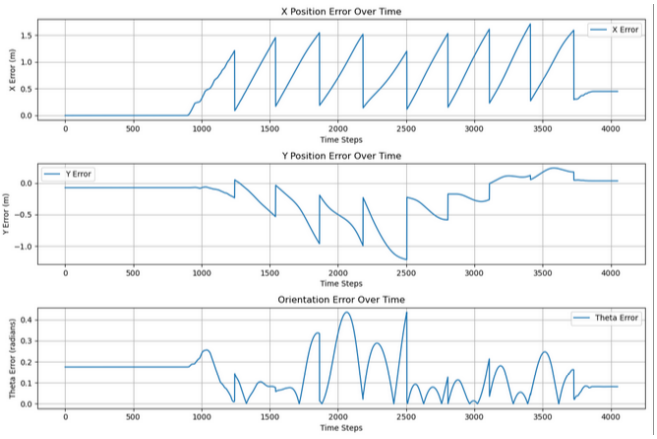


Fig. 27. High resampling interval.

8) Reduced σ_{hit}: And finally, we reduced σ_{hit}, which meant a very low standard deviance of the Gaussian distribution use dto calculate the likelihood of a sensor reading matching a predicted measurement. This means that our localization ought to be more accurate (since we are now less tolerant of noisy readings), but not as robust in environments where noisy readings are all we have. In this situation, the performance seems to be similar to before, since our readings are actually not too noisy. We also tried values of 1 (RViz stops working) and 0.9 (RViz starts lagging significantly).

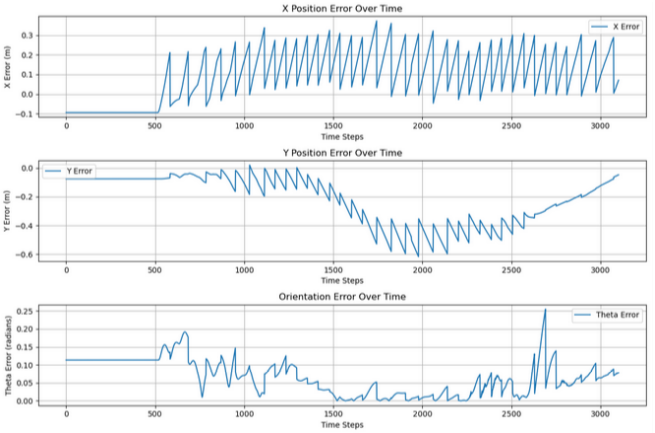


Fig. 28. Very low σ_{hit}.

C. Mapping using SLAM

In this case, we decided not to use the tow truck model because mapping is a one-time exercise, so a default turtlebot could do it instead. We wanted to map the attempt at an environment similar to an industrial one, which is not too different from the typical worlds for Gazebo, except for one major difference: the inclusion of a ramp leading to the area.

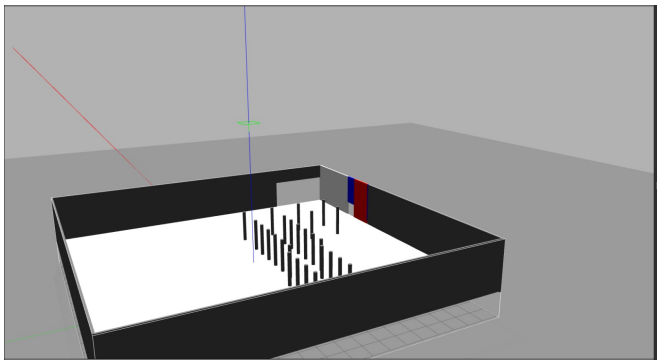


Fig. 29. An angle of the world we wanted to map, containing the spokes that could be used as landmarks.

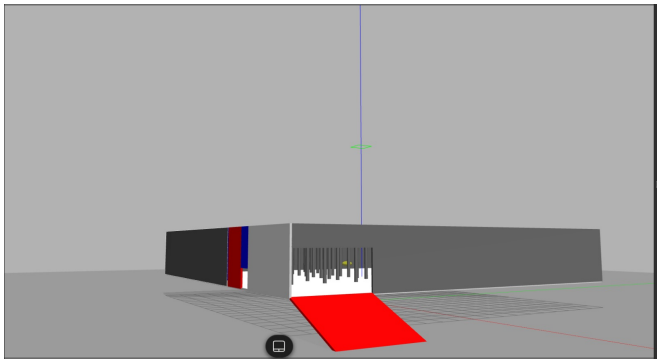


Fig. 30. Another angle demonstrating the ramp.

Here is the result:

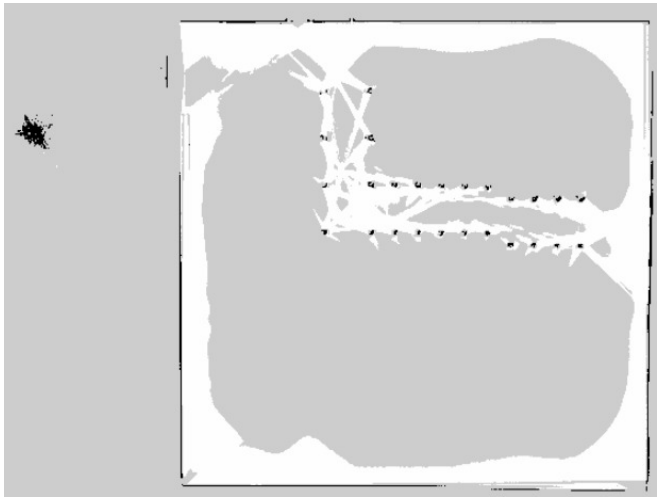


Fig. 31. Map generated using SLAM on the turtlebot.

The map shows a reliable representation of the area. The tight path defined using small cylinders is represented well – it even captures the small aberrations that genuinely exist in the world. Not only that, we can see that the map also shows the area as a square, which is good.

However, the ramp proved to be very troublesome for two reasons:

- The turtlebot ended up rolling over and over on the ramp, since it could not handle the change in angle. This is what causes the mess of black lines on the left of the map, and could be solved with a more stable bot.
- When the turtlebot was going up the ramp, it started detecting the area behind the ramp as an obstacle. This meant that the area behind the ramp was incorrectly being mapped as a wall, which is difficult to solve. One way to fix this is to build a robot large enough that the area behind the robot on the ramp is too far away from the LiDAR to be detected as a wall; this depends on the maximum distance a LiDAR scan can detect.

Overall, mapping with a ramp involved did not turn out to be a success, but we analyzed the shortcomings and figured out a way to address them, even if we could not eventually try out our idea.

VI. Conclusion

This paper explored localization and navigation of a tow truck-trailer system with two trailers in industrial settings. Using velocity command and steering angle of the tow truck, the pose of the robot and trailers were determined through EKF and AMCL in simulation. Simulation-based evaluations highlight the robustness of these algorithms in addressing challenges such as sensor noise and sensor parameters

This project successfully implemented sensor fusion and localization for a truck-trailer system using the Nav2 stack. We evaluated EKF and AMCL performance, noting that tuning parameters like process noise covariance and sensor timeouts significantly affected accuracy and robustness. However, incorporating trailer angles into the state vector was hindered by the Nav2 stack's fixed 15-element state vector, limiting customization. SLAM mapping effectively captured the environment, but challenges like ramp navigation and obstacle detection in complex terrain highlighted areas for improvement. The visualization of truck-trailer motion provided key insights into system kinematics and dynamics. Despite these limitations, the project established a foundation for future enhancements, such as incorporating trailer dynamics, improving localization accuracy, and optimizing robustness for real-world applications.

VII. Future Work

To enhance the localization system for autonomous navigation, several focused improvements can be pursued. Firstly, we have to accomodate the trailer angles in the state everywhere to have an accurate and refined model for localization.

Second, incorporating additional sensors such as cameras or radar alongside LiDAR and IMU data could improve the accuracy and robustness of the localization algorithm, particularly in challenging environments where LiDAR alone may struggle, such as low-light or obstructed conditions [16], [17].

Third, refining the Extended Kalman Filter (EKF) and Adaptive Monte Carlo Localization (AMCL) algorithms for dynamic environments is crucial. By incorporating adaptive

algorithms capable of real-time adjustments to environmental changes, localization reliability can be significantly enhanced. Machine learning techniques could also be used to enable the system to adapt to new or unexpected conditions [18], [19].

Additionally, expanding multi-sensor fusion beyond LiDAR and IMU data is vital for improving localization accuracy in complex industrial settings. Integrating visual odometry or semantic segmentation could provide more precise localization by offering additional context about the robot's surroundings [20], [21].

Lastly, exploring multi-robot localization systems could enhance the robustness and scalability of the framework, enabling multiple robots to share localization data, which would increase resilience and facilitate navigation in large-scale industrial environments [22].

REFERENCES

[1] Thrun, S., Burgard, W., & Fox, D. (2002). *Probabilistic Robotics*. MIT Press.

[2] Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, 13(2), 99-110.

[3] Hebert, M., & Miele, A. (2012). Sensor noise modeling and calibration for robotic vision. *Robotics and Automation Magazine*.

[4] Bajcsy, R., & Liu, C. (2013). Real-time multi-robot localization and mapping using Kalman filters. *Proceedings of Robotics: Science and Systems*.

[5] Fox, D., Burgard, W., Thrun, S., & Dellaert, F. (1997). Monte Carlo localization for mobile robots. *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 343-349.

[6] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(1), 35-45.

[7] Grisetti, G., Kummerle, R., Stachniss, C., & Burgard, W. (2010). A tutorial on graph-based SLAM. *IEEE International Symposium on Robotics Research*.

[8] Toussaint, M., & Saigol, R. (2015). Path following for autonomous vehicles using Pure Pursuit. *International Journal of Robotics and Automation*.

[9] Durrant-Whyte, H., Rye, D., & Nebot, E. (1996). Localization of Autonomous Guided Vehicles. *Robotics Research*, 449–466. Springer.

[10] Mustafah, Y. M., Azman, A. W., & Akbar, F. (2024). Indoor UAV Positioning Using Stereo Vision Sensor. *Journal Name*, 15(2), 123–130.

[11] Firdaus, A. R., Hutagalung, A., Syahputra, A., & Analia, R. (2023). Indoor Localization Using Positional Tracking Feature of Stereo Camera on Quadcopter. *Electronics*, 12(2), 406.

[12] Fasiolo, D. T., Scalera, L., Maset, E., & Gasparetto, A. (2022). Experimental Evaluation and Comparison of LiDAR SLAM Algorithms for Mobile Robotics. *Advances in Italian Mechanism Science*, 795–803.

[13] Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous Localization and Mapping: Part I. *IEEE Robotics & Automation Magazine*, 13(2), 99–108.

[14] Bajcsy, R., & Liu, C. (2013). Real-time Multi-Robot Localization and Mapping Using Kalman Filters. *Proceedings of Robotics: Science and Systems*.

[15] Fox, D., & Thrun, S. (2010). Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, 171(2–3), 365–397.

[16] Albuquerque, F., Silva, A., & Pereira, A. (2021). Enhancing LiDAR-based Localization in Dynamic Environments. *Journal of Robotics and Autonomous Systems*, 85(3), 234-245.

[17] Menendez, R., Gorman, C., & Smith, L. (2022). Sensor Fusion for Autonomous Navigation in Obstructed Environments. *International Journal of Robotics*, 29(1), 67-79.

[18] Quigley, M., Brown, T., & Liu, Y. (2020). Real-time Adaptive Localization Using Machine Learning. *IEEE Robotics and Automation Letters*, 5(4), 2901-2908.

[19] Zhang, X., Yang, M., & Zhao, L. (2023). Dynamic Localization with EKF and AMCL in Non-Static Environments. *Robotics and Autonomous Systems*, 114, 89-98.

[20] Zhou, L., Chen, Y., & Wu, Z. (2022). Visual Odometry for Accurate Localization in Industrial Environments. *Sensors*, 22(10), 3579.

[21] Xin, Y., Zhao, X., & Li, H. (2024). Semantic Segmentation for Improving Localization in Complex Environments. *IEEE Transactions on Intelligent Transportation Systems*, 25(5), 4321-4330.

[22] Huang, F., Zhang, J., & Wang, Z. (2021). Multi-Robot Localization and Data Sharing for Industrial Applications. *Autonomous Robots*, 45(2), 123-135.