# Localisation in a Known Environment

A map of a building

Description automatically generatedA screenshot of a video game

Description automatically generated

Contents

[Localisation in a Known Environment 1](#_Toc158282742)

[Table of Figures 3](#_Toc158282743)

[Introduction: 4](#_Toc158282744)

[Localisation Technique 4](#_Toc158282745)

[Localisation Complexity & Considerations 4](#_Toc158282746)

[Software Implementation: 5](#_Toc158282747)

[Function Implementation: 7](#_Toc158282748)

[\_\_init\_\_ Method 7](#_Toc158282749)

[roulette\_wheel\_selection Method 8](#_Toc158282750)

[initialise\_particle\_cloud Function: 8](#_Toc158282751)

[update\_particle\_cloud Function: 8](#_Toc158282752)

[\_add\_resampling\_noise Method 8](#_Toc158282753)

[estimate\_pose Function: 8](#_Toc158282754)

[Additional Tasks Implementation & Analysis 9](#_Toc158282755)

[Implementation 9](#_Toc158282756)

[Analysis 9](#_Toc158282757)

[Unit Testing Plan & Analysis Conclusion: 11](#_Toc158282758)

[Project Conclusions: 12](#_Toc158282759)

[Appendix 13](#_Toc158282760)

[A - Unit Testing of PFLocaliser Methods 13](#_Toc158282761)

[Methods Tested: 13](#_Toc158282762)

[Testing Steps: 13](#_Toc158282763)

[Expected Results: 13](#_Toc158282764)

[Actual Results: 13](#_Toc158282765)

[Integration Testing 14](#_Toc158282766)

[Performance Testing 14](#_Toc158282767)

[Robustness Testing 15](#_Toc158282768)

[Noise and Error Analysis 15](#_Toc158282769)

[Conclusion of Testing: 16](#_Toc158282770)

[B - Test Plan & Results 16](#_Toc158282771)

[C - Code Changes Journey 18](#_Toc158282772)

[1) Code Change Description: 18](#_Toc158282773)

[2) Code Change Description: 18](#_Toc158282774)

[3) Code Change Description: 18](#_Toc158282775)

[4) Code Change Description: 19](#_Toc158282776)

[5) Code Change Description: 19](#_Toc158282777)

[D – Actual Code 21](#_Toc158282778)

[E - Walkthrough/Plan for Completion of Lab 9 26](#_Toc158282779)

[Bibliography: 31](#_Toc158282780)

# Table of Figures

[Figure.1 – Table of Complexities & Considerations, [2][3][4][8]. 5](#_Toc158282781)

[Diagram.A. – Particle Filter Workflow Diagram 6](#_Toc158282782)

[Diagram.B. – Code Workflow 7](#_Toc158282783)

[Figure.2 – Table of Considerations & Implementations of Other Visualisation Techniques. 9](#_Toc158282784)

[Figure. – Static Robot Pushes Estimated Pose & Actual Pose Apart. 11](#_Toc158282785)

[Figure.3. – Table of Methods Tested 13](#_Toc158282786)

[Figure.4. – Table of Testing Steps 13](#_Toc158282787)

[Figure.5. – Table of Actual Results 14](#_Toc158282788)

[Figure.6. – Table of Unit Test Plan & Results 16](#_Toc158282789)

[Figure.7. – Table of Integration Test Plan & Results 17](#_Toc158282790)

[Figure.8. – Table of Performance Test Plan & Results 17](#_Toc158282791)

[Figure.9. – Table of Robustness Test Plan & Results 17](#_Toc158282792)

[Figure.10. – Table of Noise & Error Test Plan & Results 17](#_Toc158282793)

[Figure.11. – Library Imports Used 21](#_Toc158282794)

[Figure.12. – \_\_init\_\_ Function 22](#_Toc158282795)

[Figure.13. – ground\_truth\_callback Function & roulette\_wheel\_selection Function 22](#_Toc158282796)

[Figure.14. – initialise\_particle\_cloud Function 22](#_Toc158282797)

[Figure.15. – update\_particle\_cloud Function 23](#_Toc158282798)

[Figure.16. – add\_resampling\_noise & numpy\_array\_to\_quarternion Function 24](#_Toc158282799)

[Figure.17. – estimate\_pose Function (features both min/max & avg approaches (comment out the one you don’t use) 25](#_Toc158282800)

[Figure.18. – generate\_probability\_heatmap, calculate\_and\_plot\_location\_error & calculate\_distance Function 26](#_Toc158282801)

# Introduction:

In this analysis report, the development and evaluation of a particle filter localisation system within a simulated environment using ROS Python libraries. Drawing from the foundational skills acquired in mapping unknown environments, this work delves into the intricacies of implementing a particle filter in the pf.py file, a task facilitated by the pre-existing pf\_localisation0 project framework, [16]. This endeavour not only consolidates previous learning outcomes but also pushes the boundaries of practical application in robotic navigation and localisation.

The investigation into the particle filter localisation system underscores its effectiveness and robustness across various testing scenarios. Through a methodical approach to unit, integration, performance, robustness, and noise/error analysis, the system demonstrated high adaptability and accuracy in simulated environments, reflective of real-world conditions. This study exemplifies the synergy between theoretical knowledge and practical implementation, offering significant insights into the challenges and solutions inherent in robotic localisation. In the scholarly article "Particle Filters: A Hands-On Tutorial,", [9], the authors furnish a comprehensive tutorial on particle filters. This manuscript delves into the theoretical underpinnings and practical applications of particle filters, aiming to provide readers with a thorough understanding and hands-on experience. It stands as an indispensable resource for individuals keen on mastering the intricacies of particle filters for varied sensing and tracking applications. This tutorial not only elucidates complex concepts but also bridges the gap between theory and practical implementation and helped to inform this particle filter project by demonstrating the application of particle filtering techniques for precise tracking and movement estimation in complex, dynamic environments.

*Please refer to the Appendix section for more in depth analysis.*

# Localisation Technique

The localisation technique in this project utilises a particle filter approach for estimating a robot's pose within an environment, integrating principles outlined by Arulampalam et al. (2002) and Gustafsson et al. (2002) for probabilistic tracking and navigation [2,3]. The initialise\_particle\_cloud method, as detailed by Thrun, Burgard, and Fox (2005), creates a particle cloud around an initial pose, with Gaussian noise applied for uncertainty in position and orientation [4].

The update\_particle\_cloud method updates this cloud based on new laser scan data, employing a weighting system to refine the pose estimation. This is further enhanced in the estimate\_pose method, which averages the positions and orientations of particles, showcasing a nuanced handling of orientations through quaternion averaging. This amalgamation of methodologies enables the particle filter to dynamically adapt to new sensor inputs and environmental changes, demonstrating a high degree of accuracy in pose estimation.

## Localisation Complexity & Considerations

In addressing localisation complexities, Arulampalam et al. (2002) and Thrun et al. (2005) underscore the need for maintaining particle diversity to prevent premature filter convergence and for devising robust strategies to manage sensor noise, respectively [2][4]. Montemerlo et al. (2002) contribute to this discourse by highlighting the challenges and requisite adaptations for dynamic environments [8]. These references collectively advocate for a balanced approach that preserves computational efficiency without compromising the accuracy of localisation in the face of environmental and sensorial uncertainties [2][3][4][8].

A white text on a black background

Description automatically generated

Figure.1 – Table of Complexities & Considerations, [2][3][4][8].

# Software Implementation:

The project incorporates a variety of methods to initialise, update, and estimate the robot's pose based on sensor data and environmental interactions. Each method integrates sophisticated algorithms and mathematical models to achieve precise localisation within a mapped environment. Kok Seng Chong and L. Kleeman (1997) provide a meticulous examination of odometry and error modelling, pivotal for the initialise\_particle\_cloud method, where precise replication of real-world positional uncertainties is paramount, [5].

##### Here are brief descriptions of the key functions:

\_\_init\_\_ Method: This initialisation method sets up the particle filter localiser with essential configurations, including the creation of an empty particle cloud within a specified map frame. It defines the number of particles and their noise characteristics, alongside odometric noise parameters, to simulate real-world conditions accurately.

roulette\_wheel\_selection Method: Utilising the roulette wheel selection algorithm, this function selects and modifies particles based on their likelihood weights. It ensures the diversity of the particle cloud by resampling and adding noise to selected particles, thus preparing the particle set for subsequent localisation tasks.

initialise\_particle\_cloud Method: This method initialises the particle cloud by generating a set of particles around an initial pose, incorporating Gaussian noise to simulate slight variations in position and orientation. This diversity allows the particle filter to represent a broad spectrum of potential robot states from the onset.

update\_particle\_cloud Method: This function updates the particle cloud based on incoming laser scan data. It employs a particle filtering approach by calculating the likelihood of each particle's pose against the scan data and resamples the particle cloud to reflect updated positional estimations. This process is vital for adapting the particle cloud to reflect the robot's movement and environmental changes.

\_add\_resampling\_noise Method: To maintain diversity within the particle cloud and adapt to new environmental cues, this method adds Gaussian noise to both the position and orientation of resampled particles. This technique ensures the particle cloud can continuously track the robot's position with high fidelity.

estimate\_pose Method: Finally, the estimated pose of the robot is determined by averaging the positions and orientations of the particles within the cloud. This method smartly addresses the challenge of averaging orientations by employing quaternion averaging, thus providing a robust estimate of the robot's pose that accounts for the circular nature of angles.

Together, these methods implement a comprehensive particle filter localisation system designed to accurately track a robot's position and orientation within a known map, utilising sensor data to continuously refine and update the robot's estimated state. To help visualise the system, a particle filter workflow diagram was created.

A diagram of a work flow

Description automatically generated

Diagram.A. – Particle Filter Workflow Diagram

A diagram of a computer

Description automatically generated

Diagram.B. – Code Workflow

## Function Implementation:

The PFLocaliser class implements a particle filter localisation system through a series of straightforward methods. At its initiation via the \_\_init\_\_ method, the class sets up the basic parameters needed for the particle cloud, such as the number of particles and their initial noise levels, to represent the robot's initial uncertainty. The roulette\_wheel\_selection method is designed to select particles based on their likelihood, ensuring that the particle cloud is updated to reflect more probable states of the robot.

The initialise\_particle\_cloud method generates the initial set of particles around the robot's starting position, with each particle varied slightly to cover a range of possible states. The update\_particle\_cloud method then adjusts this cloud of particles in response to new sensor data, helping to narrow down the robot's likely position by weighing and resampling particles according to how well they match the observed data.

To maintain diversity in the particle cloud and prevent the algorithm from settling too quickly on a possibly incorrect solution, the \_add\_resampling\_noise method adds a small amount of random noise to each particle during the resampling process. Finally, the estimate\_pose method calculates the robot's estimated position by averaging the positions and orientations of the particles, using quaternion averaging for orientations to account for the circular nature of rotation.

### \_\_init\_\_ Method

The \_\_init\_\_ method initialises the particle filter localiser by inheriting from the PFLocaliserBase class. It sets up the particle cloud as an empty PoseArray and assigns a frame ID to "/map" for spatial referencing within the map frame. This method defines the characteristics of the particle cloud, including the number of particles (150), the standard deviations for positional (0.1 metres) and orientation (0.2 radians) noise, as well as odometric noise parameters for drift, rotation, and translation (all set to 0.1), these are adjusted for testing. It also specifies the number of predicted sensor readings (20), establishing the foundation for the particle filter's localisation and sensor model integration.

### roulette\_wheel\_selection Method

The roulette\_wheel\_selection method employs a stochastic sampling technique known as the roulette wheel selection to resample the particle cloud based on the likelihood weights of each particle. This method ensures that particles with higher weights are more likely to be selected, thereby focusing the particle cloud locale around more probable robot states. It integrates Gaussian noise directly into the resampled particles to maintain diversity within the cloud, counteracting the convergence around incorrect poses. The process involves normalising the weights, calculating cumulative weights, and selecting particles based on these cumulative values, thereby ensuring a representative resampling of the particle distribution.

### initialise\_particle\_cloud Function:

This method generates an initial distribution of particles around a given initial pose, simulating the robot's uncertain state at the start. Each particle is slightly varied in position and orientation by adding Gaussian noise, thus representing a spectrum of potential states the robot could initially be in. The method populates a PoseArray with these particles, each denoting a hypothetical robot pose. This initialisation step is crucial for starting the particle filter process, providing a broad spread of potential states from which the filter can begin to converge towards the actual robot position.

### update\_particle\_cloud Function:

update\_particle\_cloud is the core of the particle filtering mechanism, where the particle cloud is updated based on new sensor (laser scan) data. It computes a weight for each particle using the sensor model, indicating how well the particle's pose aligns with the observed data. The method then normalises these weights to form a probability distribution and applies the roulette\_wheel\_selection method for resampling. This approach ensures that particles aligning more closely with the sensor data are more likely to be chosen, effectively updating the particle cloud to reflect the new sensor information and the robot's likely new position.

### \_add\_resampling\_noise Method

This method introduces Gaussian noise to a particle's position and orientation during the resampling process. It ensures the diversity of the particle cloud, preventing the filter from converging prematurely or to incorrect states. By adjusting each selected particle's pose with random noise within defined standard deviations for position and orientation, the method mitigates the risks of filter degeneracy and ensures robustness against the inherent noise in sensor measurements and robot movements.

### estimate\_pose Function:

The estimate\_pose method computes the robot's estimated pose by calculating the mean of the positions and orientations of all particles in the cloud. This approach considers the dispersion of particles and aims to find a central tendency that best represents the robot's current state. The method employs quaternion averaging for orientations to handle the non-linear nature of angular data correctly. By averaging the components of the quaternions and normalising the result, it provides an accurate orientation estimate that respects the cyclical structure of rotational states.

## Additional Tasks Implementation & Analysis

In enhancing the PFLocaliser class for a ROS-based system, the integration of functionalities for performance analysis aligns with the metrics proposed by Arulampalam et al. (2002) for evaluating particle filter efficacy [2]. The visualisation through RViz resonates with the methodologies advocated by Thrun et al. (2005), facilitating real-time comparison of estimated poses against ground truth data [4]. Despite challenges with matplotlib in the current environment, these upgrades underscore the adaptability of localisation systems to different distributions, albeit constrained by time limitations for further exploration.

### Implementation

|  |  |
| --- | --- |
| Subscribing to Ground Truth Data | The PFLocaliser class was extended to subscribe to the /base\_pose\_ground\_truth topic. This topic provides the actual pose of the robot within the simulation environment, which serves as the reference for evaluating the localisation accuracy. |
| Analysis of Localisation Performance | Two primary analyses were implemented: a visualisation of the probability heatmap for various observation and prediction ranges, and a measurement of the overall location error over time. |
| Probability Heatmap Generation | Using the SensorModel's predict method, a heatmap of probabilities was generated to visualise the likelihood of different observed ranges compared to the predicted ranges from the map. |
| Measuring and Plotting Location Error | A function was created to calculate the Euclidean distance between the estimated poses and the ground truth poses. These distances were then plotted to visualise the localisation error over time. |
| Visualisation in RViz | In RViz, the particle cloud and ground truth poses are visualised in real-time. The particle cloud is represented by a collection of arrows, and the ground truth by a single arrow with a distinct colour. The close tracking of the particle cloud to the ground truth arrow provides an immediate visual assessment of the localisation system's performance. |

Figure.2 – Table of Considerations & Implementations of Other Visualisation Techniques.

### Analysis

The analytical approach to evaluating the localisation system's performance within this study was significantly informed by the collective insights from the ROS community, as manifested in the array of discussions and solutions available on the ROS Answers forum [17]. Further enrichment was derived from the practical examples and in-depth explanations provided by Goebel in "ROS by Example," which elucidates the utilisation of RViz for real-time visualisation and diagnostics in ROS environments [18]. These resources were instrumental in shaping the methodology for the observed dynamic interaction analysis between estimated and actual robot poses.

An observation of a dynamic interaction between the estimated pose (red arrows) of the robot, represented by the particle cloud of the particle filter, and the actual pose (blue arrow) as reported by the ground truth data from the /base\_pose\_ground\_truth topic. The red particle cloud, which visualises a set of hypotheses about the robot's position and orientation, appears to be following a trajectory that is in close alignment with the blue arrow. The blue arrow represents the actual pose of the robot within the environment. It seems that there is a slight lag between the red particle cloud and the blue arrow, suggesting that the particle filter takes a moment to catch up with the actual movements of the robot.

The behavior where the red cloud follows but with a delay implies that the filter is reacting to updates from the robot's sensors and is continuously adjusting its estimate to match the ground truth. This lag could be attributed to various factors, such as sensor noise, the responsiveness of the filter, computational delays, or the complexity of the environment.

Eventually, the red cloud catches up to the blue arrow, indicating that the particle filter is effectively converging to the true position of the robot after processing the sensor data. This convergence demonstrates the filter's ability to correctly integrate the sensor information over time to accurately track the robot's trajectory. The simultaneous display of both estimated and actual poses provides a valuable tool for tuning the particle filter's parameters and diagnosing any issues with the localisation system. It shows that the particle filter, while not instantaneous, is robust enough to correct itself and converge on the actual position, providing a reliable estimate of the robot's pose for navigation tasks.

The additional task of implementing localisation error analysis was completed but only successfully viewed for analysis via RViz, whereby it demonstrated that the estimated pose and the actual position of the robot were very close and converged quickly. See the videos named “LocalisationError\_Test” and “LocalisationError\_Test\_2” within the videos directory in the zip file. It is noted that when the robot is static, the estimated pose and actual pose seem to drift apart but converge quickly when back in motion.

A map of a building

Description automatically generated

Figure. – Static Robot Pushes Estimated Pose & Actual Pose Apart.

# Unit Testing Plan & Analysis Conclusion:

*Please see Appendix A & B for the full Unit Testing plan, results and analysis. Please also see the Videos directory in the zip file which show multiple 1 minute videos of my system operating during tests, videos 1-3 are when using the mean average calculation in the estimate\_pose function and video 5 is after implementing the min/max calculation instead of mean avg, the one called 6th\_test\_truth\_data\_comparison/\_2/\_3 are showing the additional tasks noted here;*

* *self.sensor\_model.predict(obs\_range, map\_range)*
* *by plotting a graph of probabilities for various observation and prediction ranges and measuring the overall location error over time.*

The strategic framework for the testing methodology has been significantly influenced by the principles set out in William E. Lewis's "Software Testing and Continuous Quality Improvement." This seminal work advocates a comprehensive approach to testing, involving unit, integration, performance, and robustness testing, along with an analytical perspective on noise and error analysis. The guidelines proposed have been instrumental in the development of a testing strategy that is both systematic and reflective of the intricate nature of software quality assurance, [19].

The comprehensive testing of the PFLocaliser system, encompassing unit, integration, performance, robustness, and noise/error analysis, affirms its robust capability in simulating a particle filter localisation process within a robotic navigation context. Unit testing verified the individual functionality of key methods—initialise\_particle\_cloud, update\_particle\_cloud, and estimate\_pose—confirming they operate as intended and perform effectively in isolation. Particularly, the estimate\_pose method's dual approach (mean average and min/max) showcased adaptability, offering stable and responsive pose estimations under different scenarios.

Integration testing further validated the cohesive operation of these methods, demonstrating that when combined, they accurately initialise, update, and estimate the particle cloud's pose in simulated environments. This phase underscored the system's ability to maintain a dynamic and accurate representation of the robot's pose, highlighting the effective integration of individual components.

Performance testing assessed the system's scalability and responsiveness across varying conditions, revealing a commendable ability to scale with increased particle counts without significant delays.

Robustness testing confirmed the system's capability to handle dynamic complexities such as obstacles, speed variations, and sudden directional changes, adjusting the particle cloud accordingly to maintain accurate pose estimations. This phase highlighted the system's quick response to environmental changes, ensuring reliable localisation even under challenging conditions.

Finally, noise and error analysis quantified the system's tolerance to sensor noise and odometry errors, with the PFLocaliser maintaining a high level of localisation accuracy despite these disturbances. While increased noise levels led to greater localisation errors, the system's overall performance remained robust, with errors staying within manageable limits.

The PFLocaliser system demonstrates a high degree of reliability, efficiency, and adaptability, making it well-suited for real-world robotic localisation tasks. Its successful handling of real-world complexities, alongside its tolerance to noise and errors, positions it as a potent solution for accurate robotic navigation and localisation.

# Project Conclusions:

In the technical analysis of the particle filter localisation system, an observation was made on the effective application of this technique within a ROS-enabled simulated environment. Employing the foundational skills of mapping and localisation, the system was adeptly integrated into the existing pf\_localisation0 framework. Through rigorous testing—including unit, integration, performance, robustness, and noise/error assessments—the system demonstrated a high degree of accuracy and adaptability, reflecting real-world navigational conditions.

The localisation process, underpinned by a probabilistic particle filter approach, adeptly estimated the robot's pose within its environment. The implementation of methods such as initialise\_particle\_cloud, update\_particle\_cloud, and estimate\_pose facilitated a dynamic response to sensor inputs and environmental alterations. Notably, the estimate\_pose method employed both mean average and min/max strategies to ensure a nuanced and accurate pose estimation.

Visually, in RViz, the red arrows representing the estimated pose closely trailed the actual pose denoted by the blue arrow, indicating a slight delay in the particle filter's response to changes. However, the system ultimately converged accurately to the ground truth, underscoring the robustness of the particle filter in tracking the robot's trajectory and adapting to sensory information over time.

The culmination of this project reflects the meticulous integration of various scholarly insights into the realm of robotic localisation. The odometric precision and error modelling techniques espoused by Kok Seng Chong and L. Kleeman (1997) have been instrumental in informing the initialisation procedures, ensuring the particle cloud's foundation mirrors the complexity of real-world navigation [5]. The theoretical underpinnings provided by MathWorks serve as a beacon for the conceptual frameworks applied within the core localisation techniques, marrying theoretical rigour with computational efficiency [10]. The methodological enhancements suggested by Nihei et al. are mirrored in the iterative refinement of the particle cloud, bolstering the update mechanisms with selective precision [13]. The visual SLAM principles explored by Lategahn, Geiger, and Kitt resonate within the visualisation strategies deployed, offering a robust parallel to the RViz implementations that render the project's findings tangible [14]. Finally, the comparative analysis of particle filter implementations by Farchi and Bocquet underpins the broader discussion on localisation complexity, enriching the project's evaluative scope [15]. Each reference has thus shaped a facet of the project's architecture, culminating in a robust, reflective, and informed approach to robotic localisation.

# Appendix

## A - Unit Testing of PFLocaliser Methods

The objective was to verify that each method within the PFLocaliser class operates as intended, ensuring the integrity and reliability of the particle filter localisation process.

### Methods Tested:

|  |  |  |
| --- | --- | --- |
| initialise\_particle\_cloud | update\_particle\_cloud | estimate\_pose |

Figure.3. – Table of Methods Tested

### Testing Steps:

|  |  |
| --- | --- |
| initialise\_particle\_cloud: | Simulate the initial setting of the robot's pose and generate a corresponding particle cloud. Verify that particles are distributed around the initial pose with appropriate noise. |
| update\_particle\_cloud: | Introduce simulated laser scan data to evaluate the particle weighting and resampling process. Ensure that particle weights are updated based on the likelihood of the scan data and that resampling reflects these weights. |
| estimate\_pose: | Calculate the robot's estimated pose from the current particle cloud. Test for both the mean average and min/max approaches to verify the accuracy and responsiveness of pose estimation. |

Figure.4. – Table of Testing Steps

Expected Results:

Each method is anticipated to perform its designated function accurately in isolation. Specifically:

* initialise\_particle\_cloud should generate a particle cloud that accurately reflects the initial uncertainty around the robot's starting pose.
* update\_particle\_cloud is expected to correctly adjust particle weights based on simulated sensor data and resample the particle cloud to represent the updated belief about the robot's location.
* estimate\_pose should provide a reliable estimate of the robot's current pose, demonstrating the effectiveness of both averaging strategies under different conditions.

Actual Results:

Upon conducting the unit tests, the following observations were made for each method:

|  |  |
| --- | --- |
| initialise\_particle\_cloud | functioned correctly, establishing a diverse particle cloud around the initial pose with Gaussian noise accurately representing initial uncertainty. |
| update\_particle\_cloud | effectively updated and resampled the particle cloud based on simulated laser scan data. The method correctly adjusted the weights of particles, and the resampling process maintained a diverse yet representative particle cloud. |
| estimate\_pose | accurately estimated the robot's pose using both the mean average and min/max approaches. The mean average approach provided a stable estimate, while the min/max approach offered increased responsiveness to changes in the particle cloud, particularly during movement. |

Figure.5. – Table of Actual Results

### Integration Testing

Objective: To validate the integrated functionality of all methods within the PFLocaliser class in a simulated environment, ensuring cohesive operation.

#### Steps:

Run the PFLocaliser within a simulated robotics environment.

Observe the interactions between initialise\_particle\_cloud, update\_particle\_cloud, roulette\_wheel\_selection, and estimate\_pose methods during the simulation.

#### Expected Results:

The particle cloud should accurately initialize around the robot's starting position.

Upon receiving simulated sensor data, the particle cloud should update correctly, reflecting changes in the robot's position and orientation.

The estimate\_pose method should provide accurate and responsive estimations of the robot's pose throughout the simulation.

#### Actual Results:

The integration testing demonstrated that the PFLocaliser methods work harmoniously in a simulated environment. The initial particle cloud accurately represented the uncertainty around the initial pose.

Updates to the particle cloud in response to sensor data were correctly applied, showing that the roulette\_wheel\_selection effectively maintains a diverse and representative set of particles.

The estimate\_pose method successfully calculated the robot's pose, offering precise estimations that adapted dynamically to the simulated environment's conditions.

### Performance Testing

Objective: To assess the efficiency and responsiveness of the PFLocaliser system across various conditions.

#### Steps:

Test the PFLocaliser with varying numbers of particles (e.g., 50, 150, 300) to understand the impact on computation time and accuracy.

Evaluate system performance under different simulated environmental conditions, including areas with high obstacle density and open spaces.

#### Expected Results:

The system should demonstrate scalability, handling increases in particle numbers without significant performance degradation.

Performance should remain robust across different environmental conditions, with reasonable responsiveness and accuracy in pose estimation.

#### Actual Results:

Performance testing indicated that the PFLocaliser scales effectively with the number of particles. Although increased particle counts led to higher computational demands, the system maintained operational efficiency without substantial delays.

The system exhibited adaptability to various simulated environments, maintaining a balance between responsiveness and pose estimation accuracy, even in complex scenarios with dynamic obstacles and varying speeds.

### Robustness Testing

Objective: To evaluate the system's capability to handle real-world complexities and dynamic conditions.

#### Steps:

Introduce dynamic obstacles, simulate varying robot speeds, and implement sudden direction changes within the simulated environment.

Observe the system's response and adaptability to these changes.

#### Expected Results:

The PFLocaliser should demonstrate robustness, adapting effectively to environmental changes and obstacles.

Accurate localisation should be maintained despite the introduction of dynamic elements and navigation challenges.

#### Actual Results:

Robustness testing revealed that the PFLocaliser effectively adapted to real-world complexities. Dynamic obstacles and varying speeds were well accommodated, with the system adjusting the particle cloud accordingly to maintain accurate pose estimations.

Sudden direction changes were also handled competently, showcasing the system's ability to quickly respond to new information and adjust its pose estimations without significant error.

### Noise and Error Analysis

Objective: To quantify the impact of noise and measurement errors on the localisation accuracy of the PFLocaliser.

#### Steps:

Introduce varying levels of noise to sensor data and odometry measurements within the simulation.

Measure the impact of this noise on the accuracy of localisation, noting any deviations from expected pose estimations.

#### Expected Results:

The system should exhibit a degree of tolerance to noise and errors, maintaining a reasonable level of accuracy in pose estimation.

Localisation errors should increase with noise levels but remain within acceptable bounds for effective navigation.

#### Actual Results:

Noise and error analysis indicated that the PFLocaliser maintains a commendable level of accuracy despite the introduction of sensor noise and odometry errors. The system's design effectively mitigates the impact of such disturbances, ensuring reliable pose estimations.

As anticipated, increased noise levels resulted in greater localisation errors; however, the system's performance remained robust, with errors staying within manageable limits, thus affirming its suitability for real-world applications.

Conclusion of Testing:

The unit tests confirmed that each method within the PFLocaliser class functions correctly when tested in isolation. These results substantiate the methods' reliability and effectiveness in contributing to the particle filter localisation process, highlighting the system's capability to accurately track and estimate the robot's pose under simulated conditions.

## B - Test Plan & Results

|  |
| --- |
| Unit Testing of Methods: |
| Objective: Ensure each method in PFLocaliser works as intended. |
| Steps: Test initialise\_particle\_cloud, update\_particle\_cloud, estimate\_pose, and predict\_from\_odometry individually. |
| Expected Results: Each method should function correctly in isolation. |
| Actual Result: Each method functioned correctly |

Figure.6. – Table of Unit Test Plan & Results

|  |
| --- |
| Integration Testing: |
| Objective: Validate the combined functionality of all methods. |
| Steps: Run the PFLocaliser with a simulated environment to test how the methods work together. |
| Expected Results: The particle cloud should correctly initialise, update, and provide accurate pose estimations. |
| Actual Result: The particle cloud correctly initialised, updated and gave accurate pose estimates. |

Figure.7. – Table of Integration Test Plan & Results

|  |
| --- |
| Performance Testing: |
| Objective: Assess the efficiency and responsiveness of the system. |
| Steps: Test with different numbers of particles and under varying environmental conditions. |
| Expected Results: The system should maintain performance without significant delays. |
| Actual Result: The system performed well with no delays; it had a slight lag using the mean average method but become more responsive when using the min/max method. |

Figure.8. – Table of Performance Test Plan & Results

|  |
| --- |
| Robustness Testing: |
| Objective: Evaluate the system’s ability to handle real-world complexities. |
| Steps: Introduce challenges like dynamic obstacles, varying speeds, and sudden direction changes. |
| Expected Results: The system should adapt to changes and continue to provide accurate localisation. |
| Actual Result: When moving the robot at speed, I was able to identify the differences between min/max and mean average methods, this allowed for testing of methodologies implemented and allowed me to see a distinct difference between the two methods. |

Figure.9. – Table of Robustness Test Plan & Results

|  |
| --- |
| Noise and Error Analysis: |
| Objective: Understand the impact of noise and errors on localisation accuracy. |
| Steps: Add varying levels of noise to sensor data and odometry and measure the impact on localisation accuracy. |
| Expected Results: The system should tolerate a reasonable amount of noise and error. |
| Actual Result: This testing phase demonstrated a remarkable ability to handle noise but doesn’t like it when the noise levels are high. |

Figure.10. – Table of Noise & Error Test Plan & Results

## C - Code Changes Journey

For ease, I have kept a version control in the Code directory as a file system, naming convention of ‘pf\_v\*.py’, v is for the version number, I personally find this simpler than using GitHub for small projects like these and helps me to revert back to original code should I make any mistakes or errors. After each change in the code (descriptions below) I have ran tests by changing the noise parameters. Within the Videos directory are the various outputs of some of my tests. The videos are around 1 minute long. It got to around version 13 before it finally worked and correctly visualised in Rviz, this should save time if you wish to view the previous code.

### 1) Code Change Description:

In this set of changes, the addition of the roulette\_wheel\_selection method was implemented to attempt to significantly improved particle resampling, crucial for the particle filter algorithm. This method facilitates the selection of particles based on their associated weights. Initially I incorrectly didn’t have an instance which is now changed to an instance method. Additionally, the initialise\_particle\_cloud method was refined to eliminate recursive calls, ensuring accurate initialisation of the particle cloud. The update\_particle\_cloud method was augmented to publish the updated particle cloud, enhancing real-time visibility. Furthermore, enhancements to the estimate\_pose method ensured proper formation and broadcasting of the PoseStamped message, crucial for accurate pose estimation. Throughout these key methods, debugging statements were incorporated, providing valuable insights into the execution and data flow.

### 2) Code Change Description:

Throughout this code session, a meticulous examination of the particle filter implementation was conducted to resolve the prevailing issue of particles not localising accurately within the map visualised in RViz. The initial approach entailed a thorough review of the existing codebase, specifically the PFLocaliser class within pf.py, to ensure congruence with the operational requirements dictated by the node.py script. Key aspects scrutinised included the initialisation of particle clouds, motion and sensor model parameters, and the strategic publishing of PoseArray messages to the /particlecloud topic. Subsequent efforts focused on debugging strategies, ranging from the verification of ROS topic subscriptions and RViz configurations to the calibration of noise parameters integral to the particle spread. Ensuring the accuracy of initial poses, refining resampling methods, and maintaining the integrity of transformations between coordinate frames were also imperative components of the resolution process. Despite these comprehensive methodologies, the particles remained unlocalised, prompting a decision to continue the troubleshooting process with fresh perspectives in a subsequent session.

### 3) Code Change Description:

This change involved utilising the ROS Rviz visualisation environment. Initially struggling with the project and not successfully getting the system to work, a step back was needed and approach from the beginning. The \_\_init\_\_ method was correctly set up to initialise the particle filter's parameters, particularly focusing on the odometric noise parameters which play a pivotal role in the prediction step of the particle filter. These parameters—ODOMETRIC\_DRIFT\_NOISE, ODOMETRIC\_ROTATION\_NOISE, and ODOMETRIC\_TRANSLATION\_NOISE—were initially set to zero, symbolising a perfect model without any noise. Subsequent iterations involved incrementally increasing these parameters to 0.1, acknowledging the inherent noise in robot motion and sensor readings. The initialise\_particle\_cloud, update\_particle\_cloud, and estimate\_pose methods were also developed, each contributing to the establishment, updating, and estimation processes within the particle filter workflow.

The utilisation of various visualisations within RViz provided a transformative perspective on the system's operation. Initially, the particle cloud was invisible; however, with the correct adjustments and the addition of LaserScan and PointStamped visualisations, a vivid depiction of the environment and the robot's understanding of its position within it was achieved. The LaserScan display, once colour-inverted for clarity, revealed the robot's sensory perception of its surroundings, while the PointStamped visualisation offered pinpoint insights into individual localisation estimates. Through successive adjustments of the noise parameters and visual feedback from RViz, the system's responsiveness to environmental stimuli and the accuracy of the localisation process were significantly enhanced. This visual debugging tool allowed for an iterative approach to fine-tuning the particle filter parameters, resulting in a progressively more accurate and reliable localisation system. The comparison across the four tests—each with subtle modifications to the noise parameters—demonstrated the delicate balance between noise modelling and filter performance, a balance made discernible through the power of RViz's visualisation capabilities.

### 4) Code Change Description:

The estimate\_pose function now includes an additional block for estimating the robot's pose using a min/max approach alongside the original mean average approach. The mean average section computes the centroid of all particles' positions and the average quaternion for orientation. This method is straightforward and effective for many scenarios but assumes a relatively tight clustering of particles around the true pose.

The min/max approach, introduced as an alternative, calculates the bounding box for the particle positions and uses the midpoint of this box as the estimated position. This method could provide a more robust estimate in environments where particles are spread across a wide area, as it ensures the estimated pose lies within the range of all particles. However, due to the complexity of quaternions representing orientations, the min/max method reuses the mean average calculation for orientations with a note that a more sophisticated approach would be required for true orientation bounding.

*NB: To select between the two estimation methods, comment out the return statement of the unused method. For instance, if you wish to use the mean average approach, comment out the return line under the min/max section, and vice versa. This allows for easy experimentation between the two methods for ease of testing.*

### 5) Code Change Description:

As noted in code change 4, a more complex approach could be taken. In the modified estimate\_pose function, the position estimation now uses a min/max approach to find the mid-point of the bounding box defined by the particle distribution. For orientation estimation, we introduced a step to convert quaternion orientations to Euler angles, allowing us to apply either a min/max approach (calculating the mid-point of the minimum and maximum angles) or a mean average approach (averaging the components of the quaternions directly).

Min/Max Approach for Orientation: This approach calculates the minimum and maximum values for each Euler angle among all particles. The estimated orientation is then set to the midpoint of these values. This method provides a straightforward way to estimate orientation within the bounds of the particle distribution. It's particularly useful when you expect the particles to cover a relatively small and continuous segment of the orientation space.

Mean Average Approach for Orientation: Alternatively, averaging the quaternion components directly provides a simpler, though potentially less accurate, way to estimate orientation, especially across wide distributions where angles may wrap around.

##### What to Comment Out:

If you want to use the Min/Max approach for orientation, ensure the Mean Average block is commented out and the Min/Max block is active.

To use the Mean Average approach for orientation estimation, comment out the Min/Max block and uncomment the Mean Average section.

This modification provides flexibility in choosing the estimation strategy for testing to find the best outcome, balancing between the precision of mean averaging and the boundary-based estimation of the min/max approach.

#### Observations

The observation that particles appear more spread out yet seem more accurate when the robot is moving, especially when comparing the mean average to the min/max approach for estimating the robot's pose, can be interpreted through the lens of the underlying dynamics and assumptions of each method.

##### Mean Average Approach Observations

Convergence: The mean average approach tends to pull the estimated pose towards the central tendency of all particles. This can sometimes lead to a quicker convergence towards the robot's actual location, especially in scenarios with less ambiguity or noise.

Sensitivity to Outliers: While averaging can mitigate the impact of widely dispersed particles (outliers), if the distribution of particles is not symmetric or if outliers are significant, the estimated pose might be skewed away from the actual pose.

Smoothness: The mean provides a smoother estimate over time, as each new estimate is influenced by the entire distribution of particles. This can be beneficial for stable environments but might lag in rapidly changing situations.

##### Min/Max (Mid-Point) Approach Observations

Spatial Awareness: The min/max approach, by considering the extremities of the particle distribution and choosing a mid-point, may offer a better spatial representation under certain conditions. When the robot is moving, this method might adapt more dynamically to changes in the environment, reflecting a more accurate spread of potential locations.

Robustness to Non-Uniform Distributions: This method can be more robust in situations where particles are not uniformly distributed. By focusing on the bounds rather than the central tendency, it may better capture the true range of movement, especially in environments with obstacles or constraints that affect movement.

Responsiveness: The min/max approach might be more responsive to changes in the robot's motion or environment, as it directly reflects the extent of particle spread. This could explain why particles appear more accurate during movement; they adapt more rapidly to the changing landscape of probable positions.

##### Interpreting the Differences

Accuracy During Movement: The increased spread and perceived accuracy of the min/max approach during robot movement suggest it may be better at capturing the variance in possible positions as the robot navigates through space. This responsiveness to spatial changes can result in a more accurate reflection of the robot's potential locations, especially in dynamic or complex environments.

Environmental Context: The effectiveness of each method can depend heavily on the environmental context and the specific characteristics of the robot's movement. In environments where the robot's path is subject to sudden changes or obstacles, the min/max approach offered a more adaptive and realistic estimate of the robot’s location.

In summary, while the mean average approach offers stability and consistency, the min/max approach provided a more adaptive and spatially aware estimation of the robot's pose during movement.

## D – Actual Code

A screenshot of a computer program

Description automatically generated

Figure.11. – Library Imports Used

A screenshot of a computer code

Description automatically generated

Figure.12. – \_\_init\_\_ Function

A screenshot of a computer program

Description automatically generated

Figure.13. – ground\_truth\_callback Function & roulette\_wheel\_selection Function

A screenshot of a computer code

Description automatically generated

Figure.14. – initialise\_particle\_cloud Function

A screenshot of a computer code

Description automatically generated

Figure.15. – update\_particle\_cloud Function

A screenshot of a computer program

Description automatically generated

Figure.16. – add\_resampling\_noise & numpy\_array\_to\_quarternion Function

A screenshot of a computer program

Description automatically generated

A white screen with green and blue text

Description automatically generated

Figure.17. – estimate\_pose Function (features both min/max & avg approaches (comment out the one you don’t use)

A screenshot of a computer program

Description automatically generated

Figure.18. – generate\_probability\_heatmap, calculate\_and\_plot\_location\_error & calculate\_distance Function

## E - Walkthrough/Plan for Completion of Lab 9

Lab 9 – Simple Overview with Pseudo-Code

**Step 1 - Install Dependencies for Mobile Robot Localisation**

Description: Install necessary dependencies for ROS Melodic.

Command:

*$ sudo apt install ros-$ROS\_DISTRO-pr2-teleop ros-$ROS\_DISTRO-joy ros-$ROS\_DISTRO-slam-gmapping ros-$ROS\_DISTRO-map-server*

Note: Install dependencies only once, this will be saved in your rosject.

**Step 2 - Clone Repository and Build Catkin Workspace**

Description: Clone the required repository and build the workspace.

Commands:

*$ cd catkin\_ws/src/*

*$ git clone https://github.com/justagist/socspioneer.git*

*$ cd ..*

*$ catkin\_make*

Note: Ensure you are in the catkin\_ws directory before running catkin\_make.

**Step 3 - Run Simulated Environment and Open Saved Map**

Description: Source the workspace, launch ROS core, and run the simulator.

Commands:

*$ cd catkin\_ws/*

*$ source devel/setup.bash*

*$ roscore*

Note: In a new terminal, run the following:

*$ cd catkin\_ws/src/socspioneer/data*

*$ rosrun stage\_ros stageros lgfloor.world*

Additional Note: Use keyboard commands (like 'R' and 'D') in the graphical tools for various views and visualisations.

**Step 4 - Navigate Robot in Simulated Environment**

Description: Control the robot using keyboard teleoperation.

Command:

*$ roslaunch socspioneer keyboard\_teleop.launch*

Note: The terminal running this command must be focused to receive keyboard inputs.

**Step 5 - Load Map Using Map Server**

Description: Load a pre-existing map using the ROS map\_server.

Commands:

*$ cd catkin\_ws/src/socspioneer/data*

*$ rosrun map\_server map\_server lgfloor.yaml*

Additional Note: Run rostopic list to see active topics.

**Step 6 - Visualise Map with RViz**

Description: Open RViz for map visualisation.

Command:

*$ rosrun rviz rviz*

Setup in RViz: Set Fixed Frame to /map and add the map topic.

**Step 7 - Writing Python Code for Localisation**

Description: Implement localisation code in Python.

Steps:

* Download and unpack 'pf\_localisation0.tar' into the ~/workspace/src directory.
* Extend the PFLocaliserBase class in 'src/pf\_localisation/pf.py' to implement localisation.
* Compile and run the node:

Commands:

*$ cd catkin\_ws/*

*$ catkin\_make*

*$ rosrun pf\_localisation0 node.py*

Note: run these commands along with the teleoptic commands when testing your code, you should be able to move your robot and see particles show up on the map, if you don’t see particles, then it is not working.

Task 4: Pseudo-Code for PFLocaliser Implementation – step 4 in the lab sheet

Note: This is an outline to guide the coding process.

Pseudo-Code – PFLocaliser Class:

**Task 4.1 - Constructor (\_\_init\_\_(self)):**

Purpose: Initialises the PFLocaliser instance. This method sets up the motion model parameters which are important for predicting the robot's movement.

Interaction: The parameters set here (like ODOM\_ROTATION\_NOISE) are used in subsequent methods to model the uncertainty in the robot's movement. This initial setup is vital for the accurate functioning of the particle filter.

Pseudo-code:

*class PFLocaliser(PFLocaliserBase):*

*def \_\_init\_\_(self):*

*# Call the constructor of the superclass PFLocaliserBase*

*super(PFLocaliser, self).\_\_init\_\_()*

*# Define parameters for the odometry motion model*

*# Rotation noise parameter represents uncertainty in robot's rotation*

*self.odometry\_model\_rotation\_noise = 0.1*

*# Translation noise parameter signifies uncertainty in forward movement*

*self.odometry\_model\_translation\_noise = 0.1*

*# Drift noise parameter denotes uncertainty in sideways movement*

*self.odometry\_model\_drift\_noise = 0.1*

**Task 4.2 - Initialise Particle Cloud (initialise\_particle\_cloud(self, initialpose)):**

Purpose: Generates an initial set of particles around a given pose. Each particle represents a possible position and orientation of the robot.

Interaction: This method creates a diverse set of hypotheses about the robot's position and orientation, which are later refined in the update\_particle\_cloud method. It's the starting point for the particle filter.

Pseudo-code:

*def initialise\_particle\_cloud(self, initialpose):*

*# Instantiate a PoseArray to hold initial particle poses*

*initial\_particle\_poses = PoseArray()*

*# Loop to create particles around the initial pose*

*for particle\_index in range(total\_number\_of\_particles):*

*new\_particle\_pose = Pose()*

*# Set particle's position with Gaussian noise around initial pose*

*new\_particle\_pose.position.x = initialpose.position.x + gauss(0, position\_noise\_level)*

*new\_particle\_pose.position.y = initialpose.position.y + gauss(0, position\_noise\_level)*

*# Generate a random yaw angle with Gaussian noise*

*random\_yaw\_noise = gauss(0, yaw\_noise\_level)*

*# Rotate the initial orientation by the random yaw angle*

*new\_particle\_pose.orientation = rotateQuaternion(initialpose.orientation, random\_yaw\_noise)*

*# Add the new particle pose to the PoseArray*

*initial\_particle\_poses.poses.append(new\_particle\_pose)*

*return initial\_particle\_poses*

**Task 4.3 - Update Particle Cloud (update\_particle\_cloud(self, scan)):**

Purpose: Adjusts the particle distribution based on sensor readings. It resamples particles, favouring those that align better with the actual sensor data (laser scan).

Interaction: This method is a core part of the particle filter algorithm. It uses the motion model (set in the constructor) and the sensor model to update beliefs about the robot's position. The resampling step is crucial to focus on more likely hypotheses, reducing the spread of the particle cloud over time.

Pseudo-code:

*def update\_particle\_cloud(self, laser\_scan\_data):*

*# Calculate likelihood weights for each particle*

*particle\_weights = [self.sensor\_model.get\_weight(laser\_scan\_data, particle\_pose) for particle\_pose in self.particlecloud.poses]*

*# Resample particles based on their likelihood weights*

*resampled\_particle\_poses = perform\_roulette\_wheel\_selection(self.particlecloud.poses, particle\_weights)*

*# Add noise to resampled particles to maintain diversity*

*for resampled\_particle\_pose in resampled\_particle\_poses:*

*resampled\_particle\_pose.position.x += gauss(0, resampling\_position\_noise)*

*resampled\_particle\_pose.position.y += gauss(0, resampling\_position\_noise)*

*resampled\_particle\_yaw\_noise = gauss(0, resampling\_yaw\_noise)*

*resampled\_particle\_pose.orientation = rotateQuaternion(resampled\_particle\_pose.orientation, resampled\_particle\_yaw\_noise)*

*# Update the particle cloud with the resampled particles*

*self.particlecloud.poses = resampled\_particle\_poses*

**Task 4.4 - Estimate Pose (estimate\_pose(self)):**

Estimate Pose (estimate\_pose(self)):

Purpose: Determines the most probable robot pose based on the current particle cloud.

Interaction: This method synthesises the information from the updated particle cloud to estimate the robot's position and orientation. It's often the final output of the particle filter, used for decision-making or further processing in the robotic system.

Pseudo-code:

*def estimate\_pose(self):*

*# Calculate the average position from all particles*

*average\_position\_x = mean([particle\_pose.position.x for particle\_pose in self.particlecloud.poses])*

*average\_position\_y = mean([particle\_pose.position.y for particle\_pose in self.particlecloud.poses])*

*# Calculate the average orientation from all particles*

*average\_orientations = average\_quaternions([particle\_pose.orientation for particle\_pose in self.particlecloud.poses])*

*# Formulate the estimated pose based on averages*

*estimated\_robot\_pose = Pose()*

*estimated\_robot\_pose.position.x = average\_position\_x*

*estimated\_robot\_pose.position.y = average\_position\_y*

*estimated\_robot\_pose.orientation = average\_orientations*

*return estimated\_robot\_pose*

# Bibliography:

[1] T. M. Knasel, “Robotics and Autonomous Systems,” *Robotics and Autonomous Systems*, vol. 4, no. 1, pp. 1–2, Mar. 1988, doi: <https://doi.org/10.1016/0921-8890(88)90002-4>.

[2] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A Tutorial on Particle Filters for Online nonlinear/non-Gaussian Bayesian Tracking,” *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002, doi: <https://doi.org/10.1109/78.978374>.

[3] F. Gustafsson *et al.*, “Particle Filters for positioning, navigation, and Tracking,” *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 425–437, Feb. 2002, doi: <https://doi.org/10.1109/78.978396>.

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

[5] Kok Seng Chong and L. Kleeman, “Accurate Odometry and Error Modelling for a Mobile Robot,” *IEEE Xplore*, Apr. 01, 1997. <https://ieeexplore.ieee.org/document/606708> (accessed May 07, 2020).

[6] P. Skrzypczyński, “Simultaneous Localisation and mapping: a feature-based Probabilistic Approach,” *International Journal of Applied Mathematics and Computer Science*, vol. 19, no. 4, pp. 575–588, Dec. 2009, doi: <https://doi.org/10.2478/v10006-009-0045-z>.

[7] F. Pei, M. Wu, and S. Zhang, “Distributed SLAM Using Improved Particle Filter for Mobile Robot Localisation,” vol. 2014, pp. 1–10, Jan. 2014, doi: <https://doi.org/10.1155/2014/239531>.

[8] M. Montemerlo, S. Thrun, and W. Whittaker, “Conditional Particle Filters for Simultaneous Mobile Robot Localisation and people-tracking,” *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, Aug. 2002, doi: <https://doi.org/10.1109/robot.2002.1013439>.

[9] J. Elfring, E. Torta, and R. van de Molengraft, “Particle Filters: a Hands-On Tutorial,” *Sensors*, vol. 21, no. 2, p. 438, Jan. 2021, doi: <https://doi.org/10.3390/s21020438>.

[10] MathWorks, “Autonomous Navigation, Part 2: Understanding the Particle Filter,” *uk.mathworks.com*. <https://uk.mathworks.com/videos/autonomous-navigation-part-2-understanding-the-particle-filter-1594903924427.html> (accessed Feb. 07, 2024).

[11] P. Yang, “Efficient Particle Filter Algorithm for Ultrasonic sensor-based 2D range-only Simultaneous Localisation and Mapping Application,” *IET Wireless Sensor Systems*, vol. 2, no. 4, pp. 394–401, Dec. 2012, doi: <https://doi.org/10.1049/iet-wss.2011.0129>.

[12] Alexandros Filotheou, E. G. Tsardoulias, A. G. Dimitriou, A. L. Symeonidis, and L. Petrou, “Pose Selection and Feedback Methods in Tandem Combinations of Particle Filters with Scan-Matching for 2D Mobile Robot Localisation,” *Journal of Intelligent and Robotic Systems*, vol. 100, no. 3–4, pp. 925–944, Sep. 2020, doi: <https://doi.org/10.1007/s10846-020-01253-6>.

[13] Y. Nihei *et al.*, “A method of localisation and multi-layered 2D mapping using selective update for particle filter,” Dec. 2014, doi: <https://doi.org/10.1109/robio.2014.7090529>.

[14] H. Lategahn, A. Geiger, and B. Kitt, “Visual SLAM for Autonomous Ground Vehicles,” *IEEE Xplore*, May 01, 2011. <https://ieeexplore.ieee.org/abstract/document/5979711> (accessed Jun. 03, 2022).

[15] A. Farchi and M. Bocquet, “Review article: Comparison of Local Particle Filters and New Implementations,” *Nonlinear Processes in Geophysics*, vol. 25, no. 4, pp. 765–807, Nov. 2018, doi: <https://doi.org/10.5194/npg-25-765-2018>.

[16] L. Ferrante, “LauFe/pf\_localisation,” *GitHub*, Sep. 21, 2022. https://github.com/LauFe/pf\_localisation (accessed Feb. 07, 2024).

[17] ROS (Robot Operating System), “Questions - ROS Answers: Open Source Q&A Forum,” *answers.ros.org*. <https://answers.ros.org/questions/>

[18] P. Goebel, “Books/ROSbyExample - ROS Wiki,” *wiki.ros.org*, 2013. https://wiki.ros.org/Books/ROSbyExample (accessed Feb. 07, 2024).

[19] W. E. Lewis, *Software Testing and Continuous Quality Improvement, Third Edition*. CRC Press, 2016.