GENG4412 Engineering Research Project Part 2

Final Report

Development and Evaluation of Local Path Planning Controllers for Autonomous Shuttle Bus

Jessica Spreadborough
23091438
School of Engineering, University of Western Australia

Supervisor: Thomas Braunl
School of Engineering, University of Western Australia

Word count: 6,964

School of Engineering
University of Western Australia

Submitted: 15 October 2025

DECLARATION OF CONTRIBUTION

You must provide the required information below and sign this declaration page. When you submit your report, you are confirming that the information provided is correct. See Section 2.3 of this guide for more information.

My contribution

- SLAM map recording
- Nav2 MPPI controller parameter tuning
- Nav2 RPP controller parameter tuning
- Experimental testing and data collection
- Python scripts written for data processing and analysis

Previous work used as foundation

- Previous shuttle setup (docker, ROS2, sensor setup, can bridge setup)
- Previous SLAM implementation on the older generation shuttle
- Wheel encoder odometry node

Collaborative work with other students

- Extension of previous SLAM implementation onto the new generation shuttle

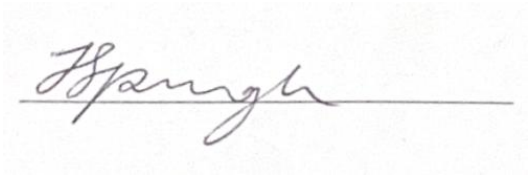
Use of AI tools

I have used AI tools in the preparation of my report: **Yes/**No

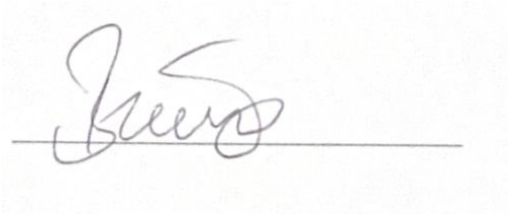
Details of how AI tools were used:
AI tools have been used in this report during editing to improve the quality of writing.

In accordance with University Policy, I certify that:

The above information is correct, and the attached work submitted for assessment is my own work and that all material drawn from other sources has been fully acknowledged and referenced.

Student signature  Date ___13/10/25_____

Supervisor confirmation
To the best of my knowledge, the student's contribution outlined above is correct.

Supervisor signature  Date ____13/10/25_____

Project Summary

Autonomous vehicles represent an emerging technology with potential to improve safety, accessibility, and efficiency in transportation. The University of Western Australia (UWA) aims to integrate autonomous shuttle buses across campus to enhance mobility while establishing the university as a leader in adopting new technology. However, campus environments present unique challenges including dynamic obstacles such as pedestrians, cyclists, and service vehicles with unpredictable behaviour, making real-time perception and local path planning essential for safe operation.

This project developed and evaluated an autonomous navigation system for UWA's nUWAy shuttle buses, focusing on reliable navigation and dynamic obstacle avoidance. The primary objectives were to implement a functional localisation system using Simultaneous Localisation and Mapping (SLAM), configure the Nav2 navigation framework, and compare the performance of two local controllers: Regulated Pure Pursuit (RPP) and Model Predictive Path Integral (MPPI). It was hypothesised that MPPI would deliver smoother navigation and superior obstacle avoidance compared to RPP.

The methodology involved implementing 2D SLAM using SLAM-Toolbox with wheel encoder odometry and lidar-based scan matching. Keepout zones were manually defined to supplement the 2D maps and clearly define path boundaries. The Nav2 framework was configured with a Smac Hybrid-A* global planner and separate RPP and MPPI local controllers. Both controllers were systematically tuned through iterative testing. Performance evaluation consisted of static and dynamic obstacle avoidance tests, as well as general driving assessments along a designated test route outside the Indian Ocean Marine Research Centre (IOMRC).

Results strongly supported the hypothesis. The MPPI controller achieved an 80% success rate in static obstacle avoidance compared to RPP's 10%, and 60% versus 0% for dynamic obstacles. MPPI also demonstrated smoother trajectories, indicating improved passenger comfort. However, MPPI consumed more CPU resources and was limited by its time-based prediction horizon that reduced lookahead distance at slow velocities, contributing to failure rates of 20% and 40% in static and dynamic tests respectively.

Future work should investigate advanced MPPI variants such as Log-MPPI or U-MPPI, which offer improved performance in cluttered environments and risk-sensitive planning. Implementation on GPU hardware, such as the shuttle's onboard NVIDIA Orin, could enable larger batch sizes and prediction horizons, potentially improving obstacle avoidance reliability. This research provides UWA with a functional autonomous navigation platform and contributes valuable insights into local motion planning for safety-critical applications in dynamic, pedestrian-shared environments.

Acknowledgements
I would like to acknowledge my project supervisor, Thomas Braunl for giving me the opportunity to work on such an interesting and challenging project. I have learnt so much throughout the last two semesters.

I would also like to thank Lee Le for assisting me with the initial setup of the software stack and for all his continuous support throughout the project.

I would also like to acknowledge Hana Allan for working with me to implement SLAM onto the new generation shuttles.

Lastly, I would like to thank Kieran Quirke-Brown, Erik Lai and the rest of the REV team for sharing their knowledge and advice.

Contents

List of Figures

List of Tables

Nomenclature

SLAM	Simultaneous Localisation and Mapping
MPPI	Model Predictive Path Integral
ROS	Robot Operating System
UWA	University of Western Australia
RPP	Regulated Pure Pursuit
IOMRC	Indian Ocean Marine Research Center
IMU	Inertial Measurement Unit
CPU	Central Processing Unit
GPU	Graphics Processing Unit
PC	Personal Computer

1. Introduction

Autonomous driving is a rapidly advancing field, set to transform the future of transportation. Reliable self-driving vehicles promise to improve the safety, accessibility, convenience and efficiency of travelling. The University of Western Australia (UWA) has goals to integrate autonomous driving across campus, using the nUWAy shuttle buses to improve campus mobility while positioning the university as a leader in adopting cutting edge technology.

Driving on the UWA campus poses a unique set of challenges as compared to driving on a conventional road. The shuttle must navigate while avoiding dynamic obstacles with unpredictable behaviour such as pedestrians, cyclists and service vehicles. These conditions make campus driving a safety-critical task that demands robust real-time perception and local path planning for effective obstacle avoidance.

This project focuses on the development and evaluation of a reliable autonomous driving system for the nUWAy shuttles. Specifically, this project first aims to design and implement a functional autonomous navigation stack combining Simultaneous Localisation and Mapping (SLAM) with the Nav2 navigation framework. The project secondly aims to compare the performance of a Regulated Pure Pursuit (RPP) and a Model Predictive Path Integral (MPPI) local controller in terms of their obstacle avoidance, reliability and smoothness in a campus environment.

It is hypothesised that the use of an MPPI controller will result in smoother, more reliable navigation and improved obstacle avoidance compared to a RPP controller. The outcomes of this research will contribute to UWA's ongoing autonomous shuttle research and to the broader understanding of local motion planning for autonomous vehicles operating in dynamic settings.

1.1 Background

1.1.1 nUWAy Shuttle Bus

UWA owns four EasyMile autonomous shuttles: two first-generation and two second-generation vehicles. All results presented in this report were collected using a second-generation shuttle, specifically nUWAy3, pictured in figure 1.

Figure 1 Image of nUWAy3

UWA intends to deploy a shuttle route from Reid Library to the Business School [1]. However, software created for this project will only be tested on a short path outside the UWA Indian Ocean Marine Research Centre (IOMRC). This path is much less busy than the proposed operational route, allowing obstacle avoidance to be tested in a controlled environment without random influence from pedestrians.

nUWAy3 is equipped with an Intel Core i7-6700TE CPU. Due to the limited computational resources of this decade-old processor, the full software stack cannot run entirely on the shuttle PC. Nodes have therefore been split between the shuttle PC and a laptop.

For safety, nUWAy3 has four SICK LMS-151 safety lidars at each bottom corner of the shuttle [2]. If an obstacle is detected within a distance determined by the shuttle's velocity, an emergency stop (E-Stop) is triggered, immediately halting the vehicle.

For navigation and localisation, the shuttle is equipped with Velodyne VLP-16 lidars on the front and rear of the vehicle [2]. These lidars have a 360° horizontal field of view and a 30° vertical field of view as well as a range of 100 m [3]. The vehicle has a footprint of 3.75 m x 1.825 m.

nUWAy3 has an XSENS MTI-10AHRS inertial measurement unit (IMU) used for tracking the motion of the shuttle. The shuttle also has wheel encoders which can be used to calculate the distance that the robot has travelled. Either or both sensors can be used to estimate the odometry of the vehicle for SLAM.

Although the shuttles can operate in both forward and reverse directions with a maximum speed of 7 m/s, this project focuses exclusively on forward navigation. All test runs are limited to a speed of 1 m/s to ensure safe operation on campus.

1.1.2 Robot Operating System 2 (ROS2)

The Robot Operating System 2 (ROS2) is the second generation of the Robot Operating System, an open source software platform for developing robotics applications [4]. All software developed in this project uses the Humble distribution of ROS2 which provides long-term support.

ROS2 provides a communication framework in which different software components, called nodes, can asynchronously exchange information by publishing or subscribing to topics. [5]. Additionally, ROS2 supports services, which implement a synchronous call-and-response pattern, and actions, which allow nodes to send a goal, receive feedback and obtain a final result [6], [7]. This architecture of ROS2 can be visualised in figure 2.

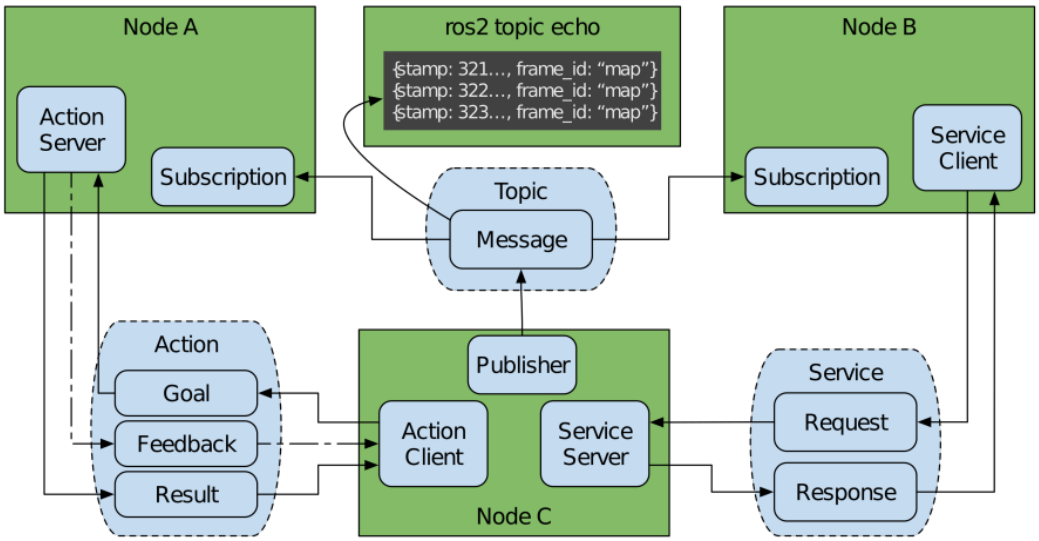


Figure 2 Communication Architecture of ROS2 [4]

ROS2 also supports a transform library (tf2) for maintaining the relationships between different coordinate frames on the robot [8]. This feature underpins localisation and navigation in this project.

Rosbags are another tool that ROS2 provides, allowing messages transmitted over topics to be recorded. In this project, rosbags were used to capture data during testing. These rosbags were later processed by a series of Python scripts to calculate key metrics used to evaluate the performance of the shuttle.

1.1.3 SLAM and SLAM-Toolbox

In autonomous driving, the vehicle must have a precise estimate of its position to ensure successful navigation to a goal pose and obstacle avoidance. This is critical for safe navigation on a university campus where the vehicle must share a path with pedestrians. Using GPS alone, often yields inaccurate positions due to interference from tall buildings and trees. Odometry sensors such as wheel encoders provide more accurate position estimates, however they are prone to the accumulation of errors over time.

SLAM is a process that allows a robot to create a map of its environment while, simultaneously, using its map to determine its location [9]. SLAM combines sensor measurements with odometry to continuously correct the vehicle's estimated position using a map as a reference, leading to more accurate positioning than GPS.

SLAM-Toolbox is an open-source package for ROS2 that includes tools for mapping and localisation [10]. SLAM-Toolbox is integrated with Nav2, allowing maps built using SLAM-Toolbox to be used for navigation.

1.1.4 Nav2

Nav2 is an open-source navigation framework for ROS2, providing tools for perception, path planning and control.

Costmaps

The Nav2 stack uses a map generated by SLAM-Toolbox and live sensor data to generate local and global costmaps. Costmaps are used by the planner and controller to indicate areas that are occupied by obstacles, which are given a lethal cost. The costmap can also be configured to give a specific radius around an obstacle a high cost.

Planners

In Nav2, a global planner aims to compute a valid and optimal path between a robot's current pose to a goal pose. The planner only has access to the global costmap, meaning it is only initially aware of obstacles that were present when the map was recorded [11].

1.1.5 Controllers

The purpose of a controller, also known as a local planner, is to guide the vehicle along the path generated by the global planner. The controller has access to the local costmap which is updated more frequently than the global costmap, allowing the controller to consider obstacles that were not present when the map was recorded. In this project, two different controllers were evaluated: RPP and MPPI

Regulated Pure Pursuit

Pure Pursuit is a path planning algorithm that calculates the curvature required to steer a robot to a lookahead point along a path. Pure Pursuit does not intrinsically consider obstacles, however, Regulated Pure Pursuit, a variation of Pure Pursuit that is integrated into the Nav2 framework, uses a proximity heuristic which causes a robot to decrease its velocity when it drives near an obstacle. However, RPP cannot replan its path around an obstacle which may result in less smooth motion or failures in path planning [12].

MPPI

MPPI works by randomly sampling a set of control inputs from a gaussian distribution at each time step and predicting how these inputs will affect the trajectory of the vehicle using a dynamics model. For each sampled trajectory, a cost function is evaluated based on several factors such as distance from the goal pose, obstacles avoidance and vehicle direction. Once the cost for each control sequence is calculated, a weighted average of all trajectories is then calculated to generate the next control sequence [13].

The benefit of using MPPI is that it can deviate from the global path if a better trajectory exists, allowing it to handle dynamic obstacles in real-time. However, the main disadvantage of MPPI is that it is far more computationally demanding than Pure Pursuit.

1.3 Project Objectives

This project aims to develop an autonomous driving system, capable of reliable navigation and dynamic obstacle avoidance, for the nUWAy shuttle buses. The main objectives of this project are:

- Implement reliable SLAM system
- Setup the Nav2 framework, carefully selecting planner and costmap parameters
- Tune the Nav2 MPPI controller to perform dynamic obstacle avoidance and driving
- Tune the Nav2 Pure Pursuit controller to perform obstacle avoidance and driving
- Perform tests to quantifiably compare the controllers with each other

By achieving the objectives of this project, the implementation of autonomous driving on a university campus will be better understood. This could lead UWA and other universities to improve safety and accessibility on campus by implementing autonomous shuttle buses. Implementing autonomous shuttles on campus could also attract students, staff and visitors to the university which has the potential to financially and reputationally benefit UWA.

Successfully developing a reliable autonomous control system with dynamic obstacle avoidance will improve safety and decrease the frequency of stops and manual takeovers. This will improve the user experience of using the shuttles and increase adoptability of the new technology.

2. Methodology

2.1 SLAM Implementation

The first step in achieving an autonomous driving system for the shuttle bus was to build a reliable SLAM implementation to allow the shuttle to map its environment and determine its position in relation to its surroundings.

Several SLAM methods were considered including SLAM-Toolbox, LIO-SAM (Lidar Inertial Odometry via Smoothing and Mapping) and ORB-SLAM. Ultimately, a 2D SLAM method using SLAM-Toolbox was chosen due to its simplicity and compatibility with the Nav2 framework.

A disadvantage of using 2D SLAM is that it lacks the necessary detail to accurately describe the boundaries of a path on a map, especially in outdoor environments. To supplement this, keepout zones were added to the costmap in Nav2. These keepout zones were areas on the costmap with a lethal cost, preventing the planner and controller from choosing paths and trajectories that overlapped these areas. A buffer zone was also added around the keepout zones with a lower cost which allowed paths to be planned in these areas but discouraged the global planner from doing so.

Initially, the XSENS IMU was planned to be used for odometry. However, initial tests showed that this produced inaccurate positioning and failed to generate useful maps. To solve this issue, wheel encoders were used for odometry instead. To facilitate this, a supplementary node was created to read speed and steering feedback from the shuttle and convert it into an odometry which served as an estimate of the position of the vehicle. This odometry estimate was combined with scan matching using lidar data from the two Velodyne lidars which improved the accuracy of the robot's position.

SLAM-Toolbox is a 2D SLAM method and uses LaserScan data from one topic for scan matching. The Velodyne lidars each output PointCloud data to separate topics. To format the lidar data to be used with SLAM-Toolbox, the PointClouds from the front and rear lidar had to be fused together and published onto a single topic. Then, the resulting PointCloud had to be flattened using the *pointcloud_to_laserscan* node which then output a single LaserScan message. This can be visualised in figure 3.

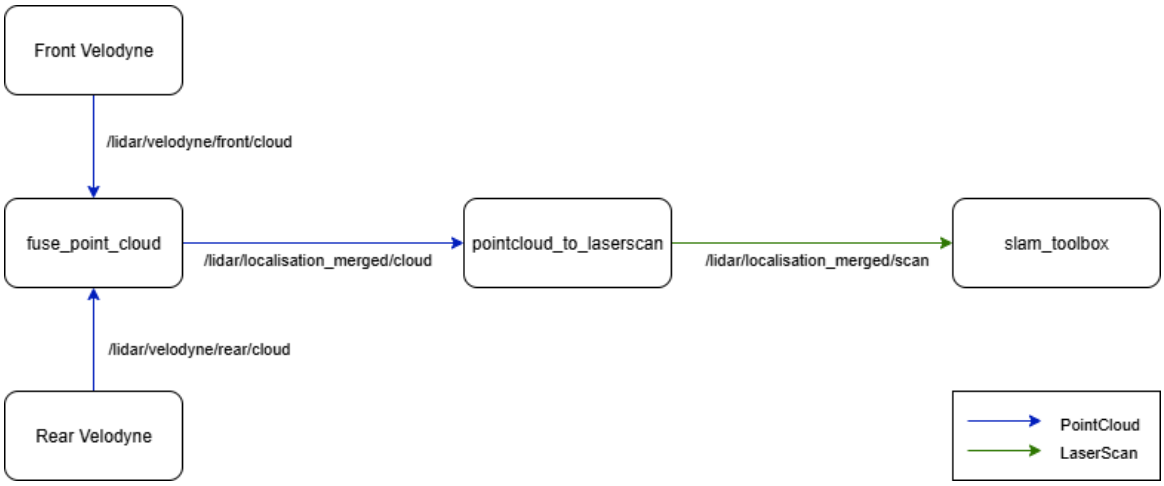


Figure 3 Lidar node architecture

Maps were created of the path outside the IOMRC, which served as the testing area, by driving the shuttle along the path and running the SLAM-Toolbox node in mapping mode. Once the desired map had been put together, the map was then saved.

During testing the saved maps were used in localisation mode, which allowed the shuttle to find its position on the map accurately without updating the map.

2.2 Nav2 Implementation

2.2.1 Transforms

To In the Nav2 navigation framework, coordinate frame transformations (TFs) are essential for maintaining a consistent spatial relationship between the robot, its sensors, and the environment. These transforms enable Nav2 to interpret sensor data, localize the robot, and plan paths in the correct reference frame. The required transform tree is illustrated in Figure 4.

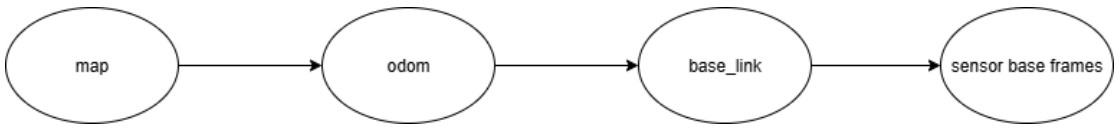


Figure 4 Required transforms for Nav2

The map to odom transform is continuously published by SLAM-Toolbox, which provides localization by aligning live sensor data with the stored map. The odom to base_link transform is produced by the wheel encoder node, which tracks the robot's relative motion over time. Finally, static transforms from base_link to all sensor frames (e.g., lidar, IMU, and camera) are defined in the robot's Universal Robot Description Format (URDF) file [14].

2.2.2 Global and Local Costmaps

Nav2 uses global and local costmaps to represent obstacles in the environment and to assist both the planner and controller in avoiding collisions. The global costmap provides a long-range view of the environment, aiding in global path planning, while the local costmap focuses on the immediate surroundings of the robot, influencing the velocities output by the controller.

Each costmap applies an inflation radius, which defines a buffer zone around detected obstacles. Cells within this radius are assigned higher costs to discourage the planner and controller from navigating too close to them. The inflation radius must be at least equal to the robot's physical radius (1.875 m), to ensure safe clearance. For this project, an inflation radius of 2.3 m was selected. This relatively high value improved obstacle awareness for the MPPI controller, allowing it to react to obstacles from a greater distance. However, a larger inflation radius also reduced navigable space, particularly in narrow corridors, limiting the robot's ability to plan paths through confined areas.

2.2.3 Global Planner

The Smac Hybrid-A* planner was chosen as the global planner due to its suitability for Ackermann-steered vehicles such as the nUWAy autonomous shuttle. Unlike purely grid-based planners, Smac Hybrid-A* incorporates the robot's kinematic constraints, enabling it to generate smooth and feasible paths that the vehicle can realistically follow. It achieves this by performing a heuristic search over a grid while considering the robot's turning radius and motion model [15], [16].

The Smac planner supports two motion models: Dubins and Reeds-Shepp curves. Both models constrain the robot's motion based on a minimum turning radius, but Dubins paths allow only forward motion, whereas Reeds-Shepp paths permit both forward and reverse motion [17]. Because the nUWAy shuttle can move in both directions, the Reeds-Shepp model was selected to provide more flexible and realistic path planning behaviour.

2.3 Controller Development

In Nav2, the controller generates velocity commands that guide the robot from its starting position to the goal pose, following the global path as a reference. In this project, an additional velocity smoother was implemented to reduce abrupt movements and minimize wear on the robot's hardware.

2.3.1 RPP Controller

A Regulated Pure Pursuit controller was developed as a baseline geometric path tracking approach for comparison against the optimization-based MPPI methods. This section describes the key parameters that were selected for this controller.

Lookahead distance and time

The lookahead distance determines the lookahead point that the controller will plan a path towards using simple geometry. Shorter lookahead distances will cause more oscillations centred around the path, however longer distances make sharp turns difficult [12]. On a university campus, there are many sharp turns that the shuttle must perform reliably. Therefore, the lookahead distance of the RPP controller was configured to be relatively short and scaled according to velocity with a minimum value of 1 m and a maximum value of 3 m.

The lookahead time is the time to project that the robot projects its position based on its current velocity. Lookahead time acts as a scaling factor for lookahead distance, meaning that small values will result in smaller lookahead distances and higher values will result in larger lookahead distances. For this project, where performing sharp turns is important, the lookahead time was tuned to 0.5 seconds.

2.3.2 MPPI Controller

This section describes the key configuration parameters and final parameter selection for the Nav2 MPPI controller. The initial configuration began with default parameters provided in the Nav2 documentation, which were then systematically tuned for the shuttle over a series of preliminary tests.

Horizon

The prediction horizon in MPPI defines how far into the future the controller will simulate and optimise trajectories. In the Nav2 MPPI controller, the horizon is a product of the number of samples per trajectory (*time_steps*) and the duration between samples (*model_dt)*.

For this project, a horizon of 5 seconds was chosen, comprised of 50 timesteps with a duration of 0.1 seconds between timesteps. With a maximum forward velocity of 1 m/s, the vehicle could therefore traverse up to 5 m within this horizon. Shorter and longer horizons were considered in initial testing, however, shorter horizons failed to detect obstacles with enough distance in front to perform a turn and longer horizons increased computational load.

The prune distance, which defines the length of the global path available to the controller, was set to 5 m to match the prediction horizon. If the prune distance were smaller than the horizon, sampled trajectories would be artificially limited to that range. This can also lead the controller to favour lower forward velocities since it cannot evaluate longer lookahead trajectories.

The local costmap size was also configured according to the maximum spatial reach of the controller to ensure that generated trajectories were not constrained by the costmap boundaries. In this project, the local costmap was set to 15 m × 15 m, providing approximately 7.5 m of coverage on either side of the robot. This distance exceeded the controller's 5 m prediction horizon plus the shuttle's radius of approximately 1.875 m, ensuring full visibility of the evaluated trajectories.

Constraint Critic

This critic ensures that only feasible controls are selected by penalising trajectories that exceed the kinematic constraints of the vehicle [18]. This critic is important to set at a reasonably high-cost weight to avoid placing unnecessary strain on the shuttle which could cause breakdowns. In the project, the cost weight was set to 4.

Goal Critic

The goal critic activates when the vehicle is within the distance specified by the *threshold_to_consider* parameter from the goal pose. This critic encourages the controller to choose velocity commands that steer the vehicle towards the goal [18]. However, it has the undesirable behaviour of sharply accelerating the vehicle towards the end of the run. For this project, the cost weight of the goal critic was set to 10.

Goal Angle Critic

Like the goal critic, the goal angle critic only takes over when the vehicle is at a specified proximity from the goal pose. This critic penalises trajectories with orientations that differ from the orientation of the goal pose [18]. In this project, the cost weight of this critic was set to 1.

Prefer Forward Critic

The prefer forward critic encourages controls that move the vehicle forward and penalises controls that would cause the vehicle to reverse [18]. Setting the cost weight of this critic high also encourages the controller to choose high velocities. For this project, reversing is not desired behaviour since one of the goals of this project is to enhance user experience by creating a controller that avoids jerky motion. For obstacle avoidance, trajectories with higher velocities will cover more distance in front of the vehicle, allowing obstacles to be detected further away. Therefore, for this project the cost weight of this critic was set high at 20.

Obstacle Critic

The obstacle critic is responsible for choosing controls that navigate the vehicle away from obstacles. The obstacle critic can be configured to consider the whole footprint of the vehicle or just a centre point when checking for collisions [18]. For this project, the whole footprint was considered since the vehicle is large. In the obstacle critic, there are two cost weights: critical weight and repulsion weight. The critical weight is given to sampled trajectories that fall within the collision margin distance, while the repulsion weight is given to trajectories that enter the inflation radius of an obstacle. For this project, where obstacle avoidance was the focus, both cost weights were set high with the critical cost at 30 and the repulsion cost at 25.

Path Follow Critic

The path follow critic encourages the controller to choose trajectories that advance further along the global path [18]. Setting the cost weight of this critic high encourages higher velocities and results in the controller choosing trajectories that are close to the global path. For the purposes of obstacle avoidance, this critic should not be set too high as that would discourage it from deviating from the global path when it perceives an obstacle. However, setting this critic high means that trajectories will generally have higher velocities and therefore reach further in space, allowing obstacles to be perceived sooner. In this project, the cost weight of this critic was set to 2, which slightly increased the average velocity of the shuttle and ensured that after an obstacle was avoided, the shuttle returned to the global path while still allowing the vehicle to diverge from the global path when an obstacle was detected.

Path Angle Critic

The path angle critic evaluates the difference between the angle of trajectories and the angle of the global path and penalises trajectories with a large difference [18]. To avoid an obstacle, the controller must choose trajectories that deviate from the global path, therefore the cost weight of this critic must be tuned low. For the controller configured in this project, this critic was not used.

Path Align Critic

This critic penalises trajectories that have a large lateral distance from the global path [18]. For obstacle avoidance, the vehicle must diverge from the global path to navigate around an obstacle, therefore, this critic should be set to a low cost weight to avoid penalising safer trajectories. In this project, the path align critic was disabled.

Velocity Deadband Critic

This critic discourages the controller from selecting velocities that fall below a specified deadband threshold [18]. The deadband threshold is defined by the *deadband_velocites* parameter in the configuration file. In this project, low forward velocities were undesirable as they limited the distance that obstacles could be perceived and avoided. Therefore, this critic was set to have a forward deadband velocity of 0.15 m/s and a cost weight of 8.

2.4 Architecture of Software Stack

Figure 5 shows a diagram displaying the architecture of the deployed software stack developed for this project.

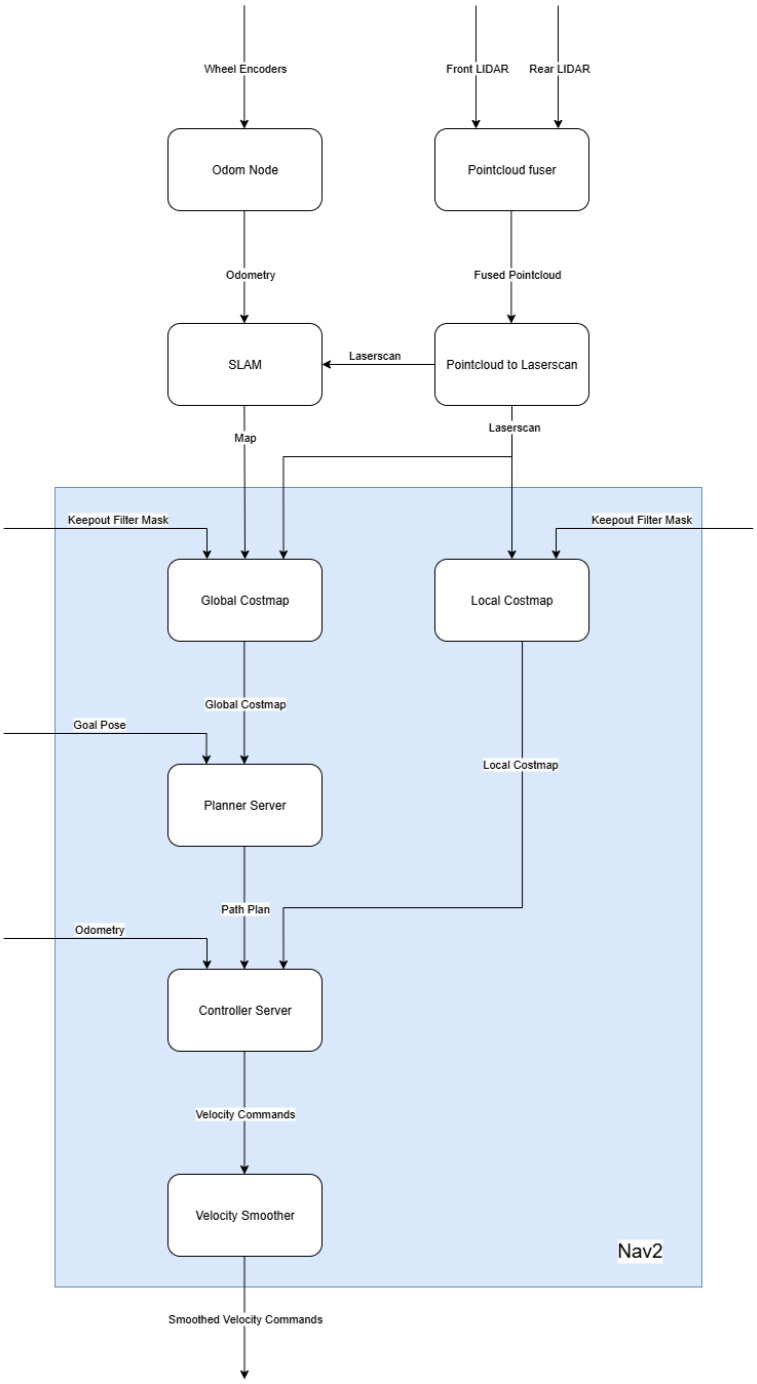


Figure 5 Architecture of final software stack

2.5 Obstacle testing

Obstacle tests were conducted to evaluate the ability of each controller to avoid static and dynamic obstacles. These tests were also used to further improve the controllers especially looking at the critic weightings for MPPI. All obstacle tests were performed on a wide section of the path outside the IOMRC.

2.5.1 Static Obstacle Testing

Experimental Setup

The purpose of this test was to evaluate obstacle avoidance for a stationary obstacle that was not present on the path when the map was recorded, for example, a stationary pedestrian. To setup this experiment, a cone was placed 5 m directly in front of the shuttle and the shuttle was sent a goal pose 30 m forwards. The experimental setup of this test can be visualised on a map in figure 6.

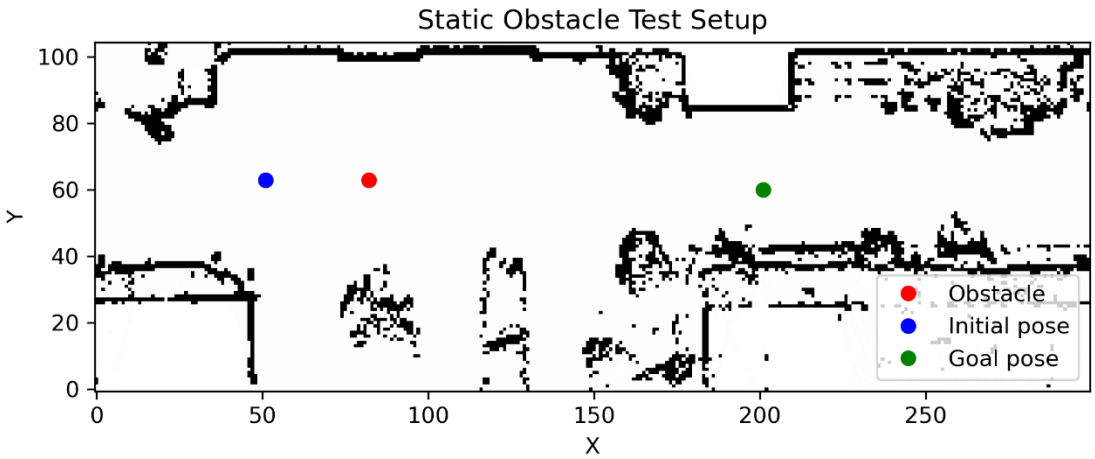


Figure 6 Static obstacle test experimental setup

For each controller, 10 consecutive runs of this test were performed to determine a success rate. The starting position was marked on the ground with tape and care was taken to drive the robot back to the same position at the beginning of each run.

The cone was placed onto the path after the map of the area had been recorded. This meant that the cone was not initially integrated into the global costmap, only the local costmap, meaning the global path planner perceived the cone as open space. This left obstacle avoidance solely to the local controller rather than being influenced by the planner.

Performance Metrics

For each run, a rosbag was recorded for later processing. The success rate, average speed and turning distance from the cone were determined from the data recorded during the tests.

A successful run is defined as a run where the shuttle successfully navigates from its starting position to the goal position without triggering an E-Stop. The success rate is the percentage of successful runs out of 10 for each controller.

The turning distance represents how far the cone was from the vehicle at the moment the vehicle began to turn to avoid it. Turning distance was only calculated for runs that successfully reached the

goal pose. The average turning distance was only calculated for controllers with a success rate of more than 50%.

To detect the start of a turn, the angular velocity of the vehicle was extracted from the rosbag. This data was analysed to find the time that the absolute value of angular velocity first exceeded 0.3 rad/s for at least 1 second which was defined as the turning time.

At this time, the lidar measurement corresponding to the forward direction was obtained to represent the distance from the obstacle when the vehicle began turning. To account for the vehicle's heading changing over time, the lidar data was processed by taking the minimum range value within the front 90° field of view. This ensured that the closest obstacle in front of the vehicle was captured even as it started to rotate.

Average speed was calculated by taking a non-zero average of the forward velocities of each run. These values were then averaged over all 10 runs for each controller.

2.5.2 Dynamic Obstacle Testing

Experimental Setup

To test dynamic obstacle avoidance, the shuttle remained stationary while a pedestrian walked directly towards the shuttle from about 10 m away. A goal pose 30 m forwards was sent to the shuttle and the velocity controls were analysed. This setup is shown in figure 7.

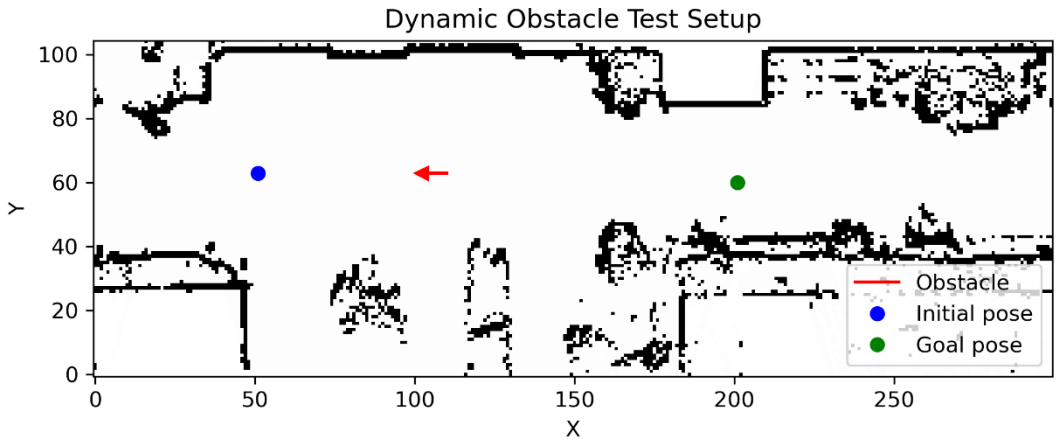


Figure 7 Dynamic obstacle test experimental setup

Performance Metrics

The angular velocity control outputs recorded for each run were graphed against the distance between the shuttle and the pedestrian to evaluate if the controllers would react to the pedestrian early enough to avoid collision.

A key measurement obtained from this test was turning time, which was evaluated similarly to the static obstacle tests except using angular velocity control data rather than actual angular velocity.

The success rate, defined as the percentage of runs that successfully sent high angular velocity commands before the obstacle was within 2 m from the shuttle.

2.6 Driving tests

Experimental Setup

After the controllers were tuned to perform obstacle avoidance well, they were then assessed in their general driving ability. Tests were conducted along a path with turning and straight driving areas. Five runs of each controller were performed from the same initial and goal pose each time. The experimental setup can be visualised on the map in figure 8.

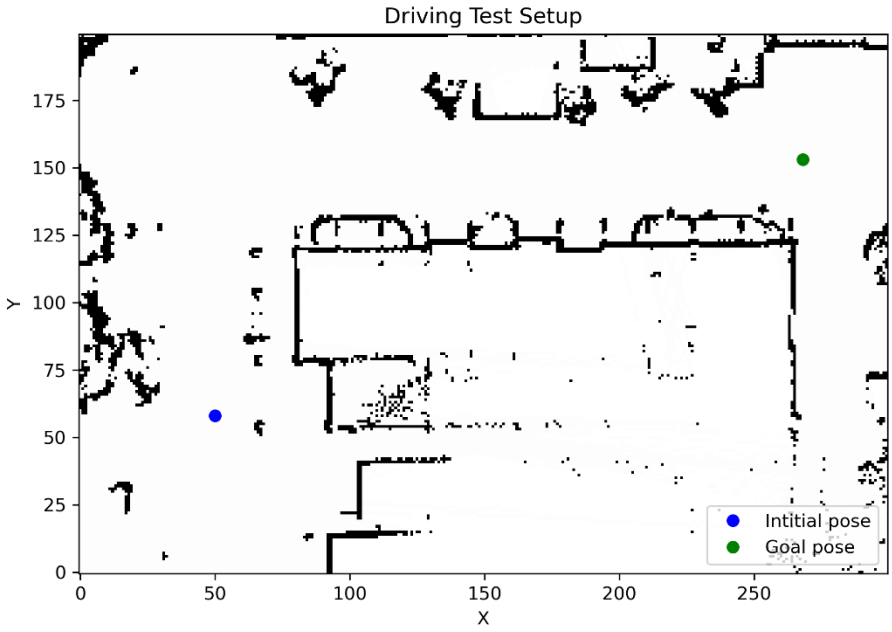


Figure 8 Driving test experimental setup

Performance Metrics

For each controller, the average speed, average CPU usage, path length and path smoothness were calculated from the rosbag recordings of each run.

Path smoothness is calculated as the sum of the squared changes in heading. To calculate the heading at each position, poses were sampled at equal spatial intervals of 1 m and the heading was calculated as the arctangent of the displacement over the interval. The change in heading is then calculated between consecutive samples. Paths with gradual direction changes will yield small path smoothness values, while paths that involve frequent, sharp turns will have higher scores. Path smoothness was only calculated for straight path segments. To determine the average path smoothness for each controller, an average was taken across the path smoothness values of the five runs.

2.7 Safety Requirements

The main safety risk when conducting tests with the shuttle is collision with pedestrians. To avoid this, the safety system must always be activated while the shuttle is in motion. The vehicle must also move at a slow speed to allow for more reaction time. For this project, the speed has been capped at 1 m/s for all tests.

3. Results & Discussion

3.1 SLAM Results

Figures 9 and 10 show the generated occupancy grid maps used for obstacle tests and driving tests. These results can be compared to satellite imagery of the area as seen in figure 11.

Figure 9 Map created using SLAM-Toolbox for obstacle tests

Figure 10 Map created using SLAM-Toolbox for driving tests

Figure 11 Satellite image of the testing path

A qualitative comparison of the SLAM mapping results with the satellite imagery demonstrates strong spatial accuracy. Path boundaries are clearly defined along the route and building structures appear in the correct positions.

During autonomous operation, accurate localisation was observed while running SLAM-Toolbox. This allowed consistent path following performance when navigating to a goal pose. Localisation accuracy can be demonstrated by plotting recorded pose messages on the map. For example, figure 12 shows the recorded poses for one of the obstacle tests. These results align with visual observations of the shuttle's position while conducting the test.

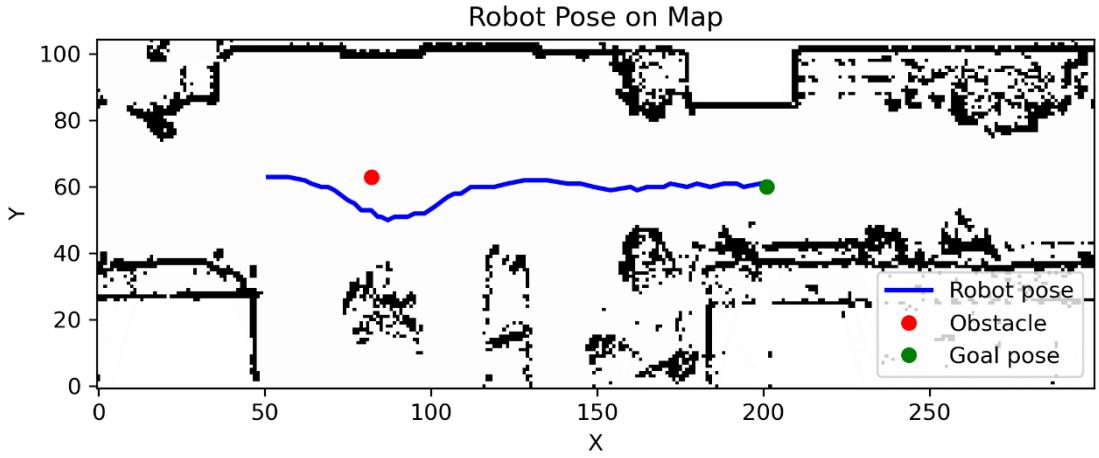


Figure 12 Recorded pose of the vehicle on the map generated by SLAM-Toolbox from static obstacle testing

Despite overall accuracy, there were some limitations to the SLAM implementation. 2D lidar-based SLAM cannot distinguish between traversable and non-traversable surfaces at the same height. For example, there was a flat garden bed to the side of the path which appeared as an open space on the map, allowing the path planner to generate trajectories through it. To mitigate this, keepout zones were introduced to define hard boundaries at the edges of paths.

Keepout zones were manually defined and overlaid on the map as a costmap layer. Originally, keepout zones were defined as entirely having a critical cost. However this caused an issue where the path planner, which generates trajectories based on the centre of the robot, planned paths that caused part of the robot footprint to intersect with the critical cost area. This caused the controller, which considers the total footprint of the robot, to fail to generate controls. Therefore, a greyscale cost scaling approach was introduced where dark regions were given a critical cost, medium grey regions were given a moderate cost and light grey regions were given a low cost. Figure 13 shows the final map with keepout zones applied. In the figure, the high cost areas are coloured magenta, while the moderate cost areas are red and the low cost areas are blue.

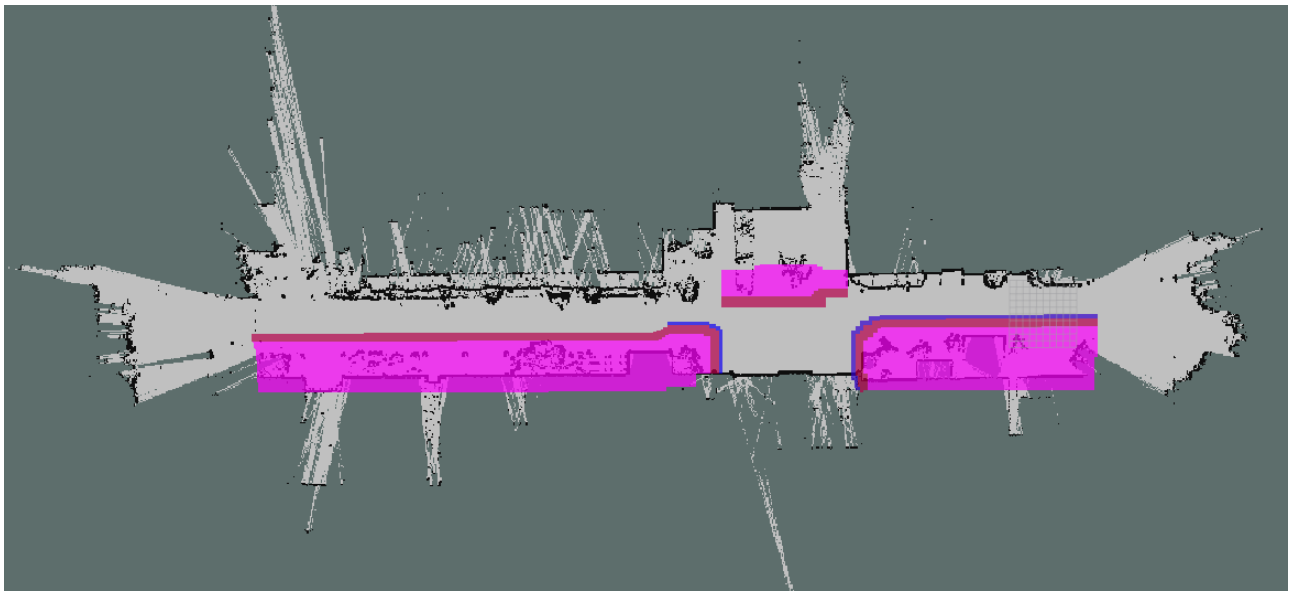


Figure 13 Keepout zones on the SLAM-Toolbox map

Additional limitations were observed when mapping areas with trees or windows. Windows in adjacent buildings caused obstacles to appear beyond their actual positions due to their reflective surface which caused the lidar to scatter. This effect can be visualised in figure 14. Trees also produced inconsistent mapping results due to moving branches. These limitations were documented but did not significantly impact controller testing.

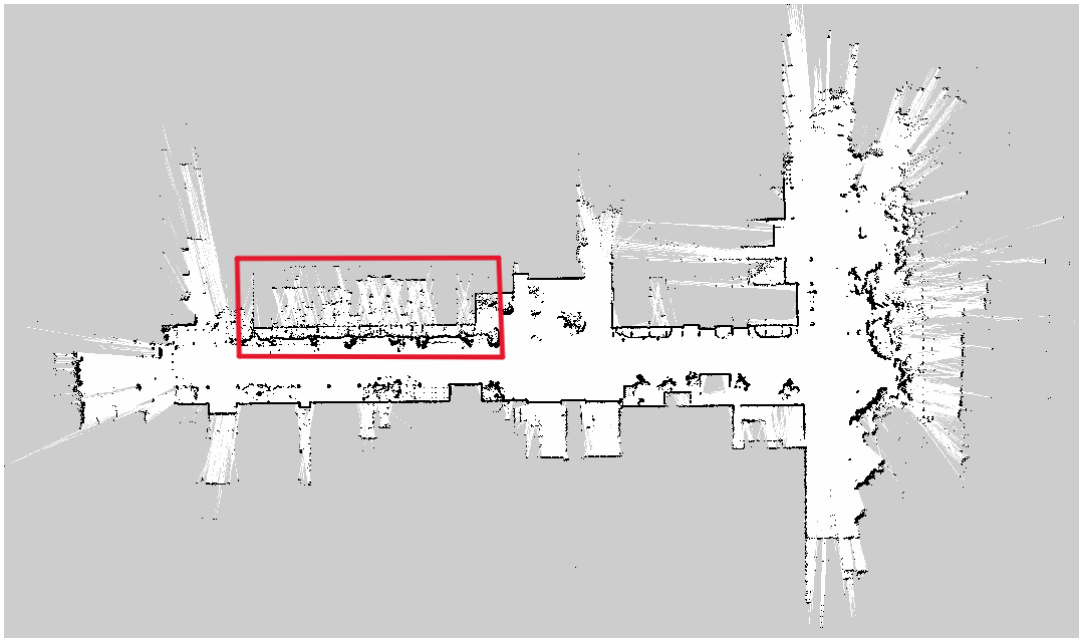


Figure 14 Errors caused by reflective surfaces on the SLAM-Toolbox map

3.2 Static Obstacle Test Results

Table 1 summarises the performance metrics for both controllers across 10 test runs.

Table 1 Static Obstacle Test Results Summary

Controller	Number of successful runs out of 10	Success rate	Average speed (m/s)	Average turning distance from obstacle (m)
MPPI	8	80%	0.516	2.52
RPP	1	10%	0.583	Insufficient number of successful runs

The MPPI controller successfully navigated around the obstacle in 8 out of 10 runs, while the RPP controller succeeded in only one run. This substantial difference demonstrates that the MPPI controller can perform more reliable obstacle avoidance in this scenario.

For successful MPPI runs, the average turning distance was 2.52m. This gave sufficient clearance to the static obstacle when performing a turn.

The MPPI controller maintained a lower average speed compared to RPP. This difference reflects MPPI's ability to reduce speed as part of obstacle avoidance. The RPP controller maintained higher speeds as it relies primarily on global path replanning rather than velocity adjustment for obstacle response.

Figures 15-22 present representative trajectories and velocities from successful and failed runs for both controllers.

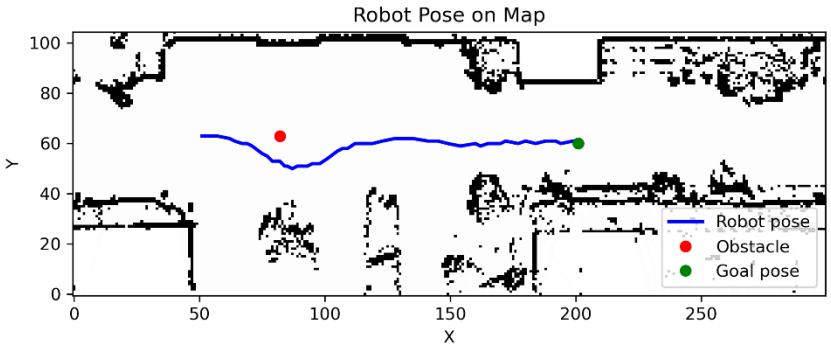


Figure 15 MPPI successful run trajectory

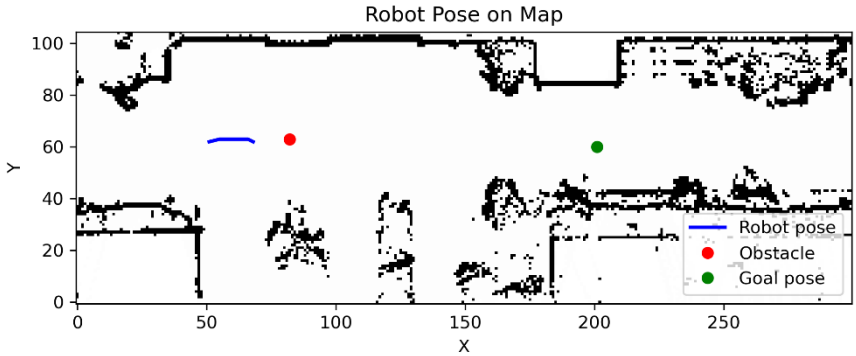


Figure 16 MPPI unsuccessful run trajectory

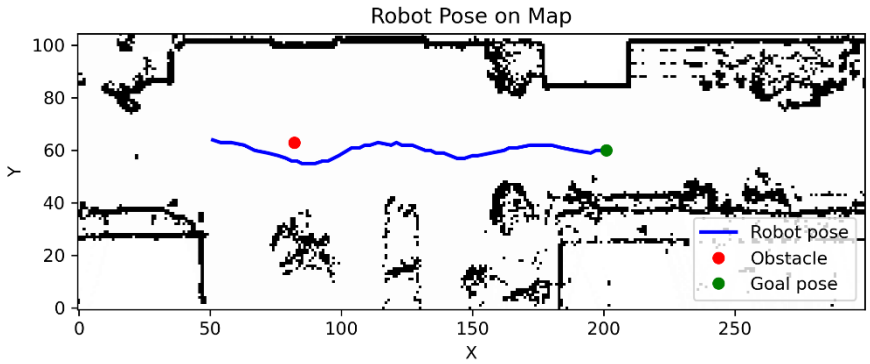


Figure 17 RPP successful run trajectory

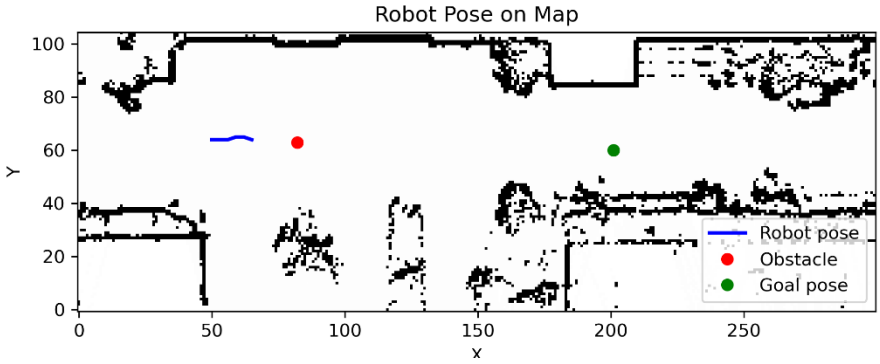


Figure 18 RPP unsuccessful run trajectory

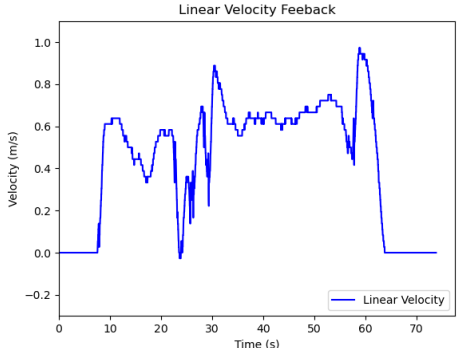


Figure 19 MPPI successful run linear velocity

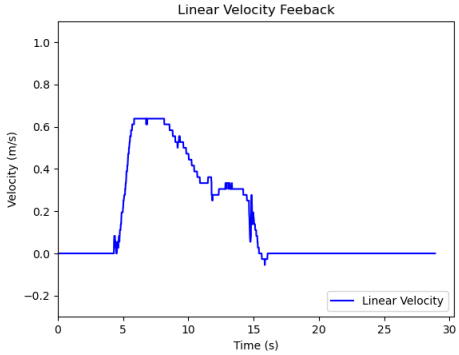


Figure 20 MPPI unsuccessful run linear velocity

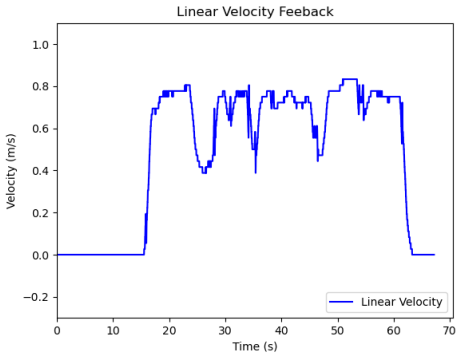


Figure 21 RPP successful run linear velocity

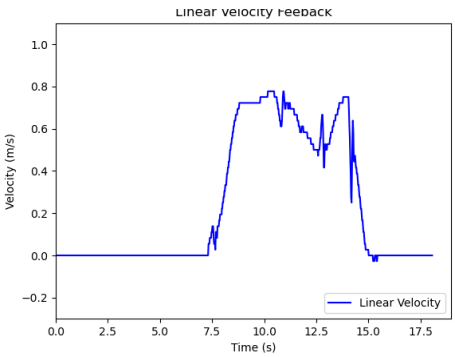


Figure 22 RPP unsuccessful run linear velocity

Successful MPPI behaviour displayed early obstacle detection, gradual deviation from the global path, velocity reduction and finally a smooth return to the global path once the robot cleared the obstacle.

There were two main issues that lead to unsuccessful runs for the MPPI controller. The first issue occurred when the controller failed to select trajectories with large angular velocities early enough. In these cases, the vehicle attempted to turn around the obstacle, however there was not enough space for the vehicle to complete the turn. The second type of failure occurred when the controller successfully began navigating around the obstacle with sufficient clearance but failed to select trajectories that returned to the global path. The sampled trajectories continued deviating laterally until the vehicle footprint intersected a keepout zone, causing the controller to fail to generate the next controls.

The RPP controller exhibited consistent failures across 9 of 10 runs. In all failed cases, the vehicle followed the initial global path directly toward the obstacle without attempting obstacle avoidance. Collisions were prevented only by the emergency stop.

The RPP controller had one successful run, however this run showed no evidence that the controller chose velocity commands based on distance from obstacle since the controller did not deviate from the global path.

These results support the hypothesis and illustrate that implementing an MPPI controller will result in more reliable obstacle avoidance than an RPP controller.

3.2.1 MPPI Limitations

Despite the strong overall performance, the 20% failure rate of the MPPI indicates important limitations. Firstly, the MPPI controller uses a time-based horizon rather than a distance-based horizon, meaning that the distance covered by the horizon is dependent on the velocity. During the obstacle approach, the vehicle decreases velocity thus reducing the lookahead distance at the time when a large lookahead would be desirable.

Increasing the horizon could mitigate this limitation, however doing so comes at a computational cost. Observations made while tuning the controller suggested that increasing the horizon to values greater than 5 seconds caused the controller to fail to meet its desired frequency.

Another potential solution to this limitation is to encourage the controller to choose higher velocity commands to increase the lookahead distance by increasing the cost weight of the path follow critic. Increasing this critic could also reduce the second type of MPPI failure by more aggressively encouraging the vehicle to return to the path after clearing an obstacle. However, this comes with the trade-off of penalising the controller for choosing trajectories that deviate from the global path which may lead to less reliable obstacle avoidance.

3.2.2 Test Limitations

Several limitations of this experiment should be acknowledged. Despite efforts to mark and align the vehicle to the same starting position for each run, there was some variability in the start positions across runs. This variability may have contributed to inconsistent timing of obstacle detection and potentially influenced some failure cases.

Another testing limitation is that the global path was not the same in each test, even though all runs used the same global path planner with identical parameters. Ideally, defining a fixed path would eliminate this variable. However, this is unrealistic for operational autonomous vehicles which must follow paths based on environmental perception. The variability observed here reflects realistic operational conditions where planning and control must function together.

3.3 Dynamic Obstacle Test Results

Table 2 displays the results for each controller from 5 consecutive runs of the dynamic obstacle test.

Table 2 Dynamic Obstacle Test Results Summary

Controller	Number of successful runs out of 5	Success rate	Average turning distance from obstacle (m)
MPPI	3	60%	2.46
RPP	0	0%	Insufficient number of successful runs

The MPPI controller achieved 60% success rate while RPP failed in all attempts. For successful MPPI runs, the vehicle began turning around the obstacle at an average distance of 2.46 m. This distance is relatively short, and which is likely insufficient to avoid a moving pedestrian if they continue walking towards the shuttle.

Successful MPPI runs output high angular velocities, exceeding 0.3 rad/s for at least 1 second before the dynamic obstacle reached the safety zone of the vehicle. Unsuccessful MPPI runs did manage to output high angular velocities but not until after the pedestrian had already crossed into the safety zone.

For the entirely unsuccessful RPP runs, no attempt was made to output high angular velocities regardless of the distance between the obstacle and the shuttle.

Test Limitations

The 40% failure rate and small avoidance distances reveal some limitations in the MPPI controller and the test. For one, the shuttle remained stationary throughout the test which limited the distance covered by rollouts. This also explains the why the average turning distance in this test was less than that for the static obstacle tests.

Similarly to the static obstacle tests, the global path was not the same for all tests, however all tests used the same path planner. This setup caused some inconsistencies in results but reflected realistic operation of autonomous vehicles.

3.4 Driving Test Results

Table 3 summarises the performance metrics for both controllers across 5 runs.

Table 3 Driving Test Results Summary

Controller	Average Speed (m/s)	Average CPU Usage (%)	Average Path Length (m)	Average Path Smoothness
MPPI	0.637	20.2	58.3	1.31
RPP	0.698	16.6	59.5	1.58

Both controllers achieved a 100% success rate in navigating from the start pose to the goal pose, demonstrating that both controllers are capable of reliable basic navigation in obstacle-free environments.

Figures 23-26 show the trajectories and velocities of the first run of the test for both controllers.

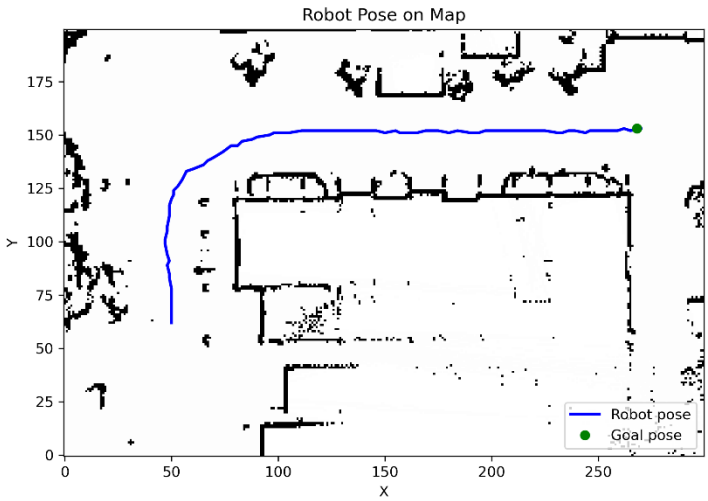


Figure 23 MPPI run 1 trajectory

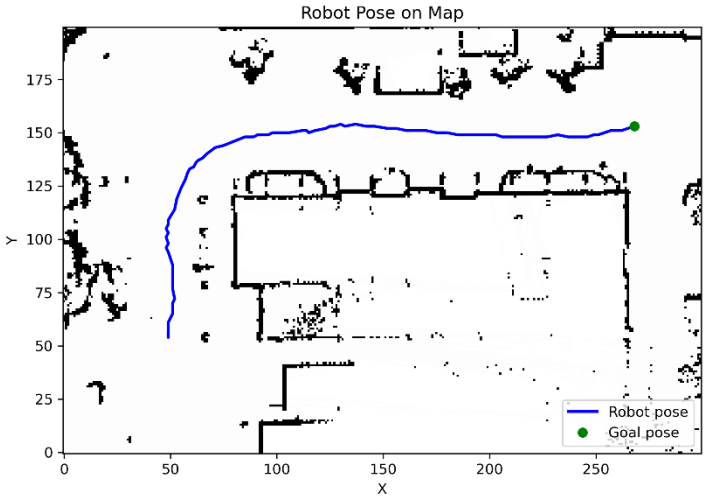


Figure 24 RPP run 1 trajectory

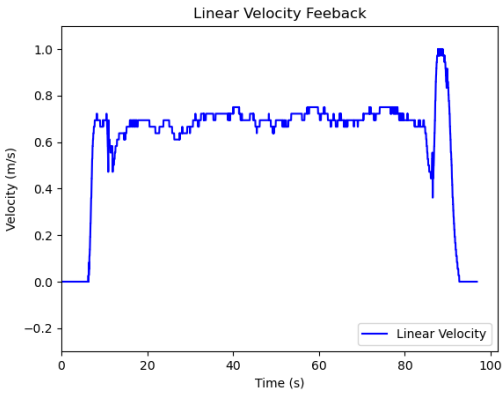


Figure 25 MPPI run 1 linear velocity

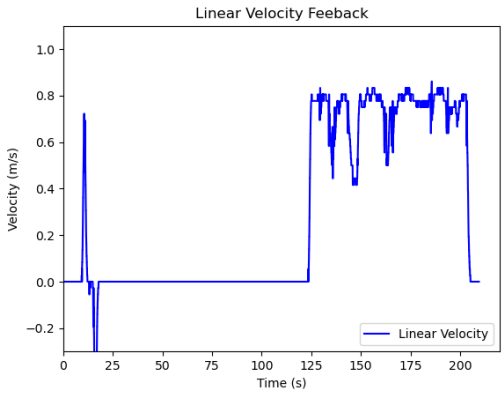


Figure 26 RPP run 1 linear velocity

RPP maintained a higher average velocity compared to MPPI. However, MPPI demonstrated significantly smoother trajectories. Lower smoothness values indicate less variation in angular acceleration, corresponding to more comfortable passenger experience. The average path length travelled using the MPPI controller was slightly shorter than the RPP controller, although this could be attributed to variations in the starting position.

The MPPI controller consumed significantly more computational resources than RPP. This aligns with expectations given MPPI's sampling-based optimisation approach. However, neither controller approached CPU saturation, confirming both will perform reliably during normal driving within the shuttle's computational capabilities.

4. Conclusions & Future Work

4.1 Conclusion

This project successfully developed and evaluated an autonomous navigation system for the nUWAy shuttle buses at UWA. The implementation integrated SLAM-Toolbox for localisation and mapping with the Nav2 navigation framework, enabling reliable autonomous operation in a campus environment.

The comparative evaluation between MPPI and RPP controllers demonstrated clear performance differences. The MPPI controller achieved an 80% success rate in static obstacle avoidance compared to RPP's 10%, and 60% versus 0% for dynamic obstacles. These results strongly support the hypothesis that MPPI provides superior obstacle avoidance capabilities. Furthermore, MPPI produced smoother trajectories, indicating improved passenger comfort despite operating at slightly lower average speeds.

However, the MPPI controller's limitations were also evident. The 20% failure rate in static obstacle tests and 40% in dynamic tests, combined with relatively short turning distances averaging 2.52 m and 2.46 m respectively, indicate that further optimisation is needed before deployment in high-traffic campus areas. The time-based prediction horizon proved problematic as velocity reductions during obstacle approach diminished lookahead distance at critical moments. Additionally, MPPI's higher CPU usage represents a constraint for future development.

The successful implementation of 2D SLAM with keepout zones effectively addressed outdoor mapping challenges, though limitations with reflective surfaces and vegetation were noted. The system demonstrated consistent localisation accuracy during operation, providing a robust foundation for autonomous navigation.

This research contributes valuable insights into local motion planning for autonomous vehicles in dynamic, pedestrian-shared environments. The findings demonstrate that an MPPI controller offers significant advantages over an RPP controller for safety-critical campus applications, though computational constraints and tuning complexity remain important considerations. The developed system provides UWA with a platform for continued autonomous shuttle research and brings the university closer to its goal of implementing autonomous driving on campus.

4.2 Future work

Different MPPI algorithms could be investigated such as Log-MPPI or U-MPPI. In Log-MPPI, instead of sampling from a gaussian distribution, control inputs are sampled from a product of a normal distribution and log-normal distribution. This control algorithm reduces the risk of getting stuck in local minima, thereby improving collision avoidance in dynamic cluttered environments such as a university campus. Log-MPPI also runs reliably with a smaller batch size, reducing the computational cost and allowing for larger horizons to be configured which could improve obstacle avoidance [19].

U-MPPI uses the unscented transform when sampling trajectories to propagate a set of sigma points through system dynamics, approximating the probability distribution of the next state and allowing the covariance of each state to be tracked. This algorithm then calculates a risk-sensitive cost-function which penalises trajectories with large covariance, discouraging uncertain paths [20]. This algorithm could prove to be useful for safety critical applications such as campus driving.

There is also the potential to implement an MPPI controller on a GPU rather than a CPU for example MPPI-Generic [21]. This utilises the NVIDIA Orin installed on each of the shuttles which has superior compute power compared to the onboard PC. Using a GPU allows for parallel processing which could allow the sampled trajectory rollouts to be computed more efficiently, allowing for larger batch sizes or horizons to be configured which could improve obstacle avoidance capabilities.

References

[1] "UWA students first in Australia to build 'brains' of autonomous bus." Accessed: Oct. 08, 2025. [Online]. Available: https://www.uwa.edu.au/news/article/2021/june/uwa-students-first-in-australia-to-build-brains-of-autonomous-bus

[2] "Command an EZ10 Gen2 for R&D purposes." EasyMile, Sept. 09, 2021.

[3] "VLP-16 datasheet." Velodyne Lidar, 2019. Accessed: Oct. 11, 2025. [Online]. Available: https://ouster.com/downloads/velodyne-downloads

[4] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, Architecture, and Uses In The Wild," *Sci. Robot.*, vol. 7, no. 66, p. eabm6074, May 2022, doi: 10.1126/scirobotics.abm6074.

[5] "Understanding nodes — ROS 2 Documentation: Humble documentation." Accessed: Oct. 09, 2025. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html

[6] "Understanding actions — ROS 2 Documentation: Humble documentation." Accessed: Oct. 09, 2025. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html

[7] "Understanding services — ROS 2 Documentation: Humble documentation." Accessed: Oct. 09, 2025. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

[8] "Tf2 — ROS 2 Documentation: Humble documentation." Accessed: Oct. 11, 2025. [Online]. Available: https://docs.ros.org/en/humble/Concepts/Intermediate/About-Tf2.html

[9] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, June 2006, doi: 10.1109/MRA.2006.1638022.

[10] S. Macenski and I. Jambrecic, "SLAM Toolbox: SLAM for the dynamic world," *J. Open Source Softw.*, vol. 6, no. 61, p. 2783, May 2021, doi: 10.21105/joss.02783.

[11] "Navigation Concepts — Nav2 1.0.0 documentation." Accessed: Oct. 10, 2025. [Online]. Available: https://docs.nav2.org/concepts/index.html#planners

[12] S. Macenski, S. Singh, F. Martin, and J. Gines, "Regulated Pure Pursuit for Robot Path Tracking," May 31, 2023, *arXiv*: arXiv:2305.20026. doi: 10.48550/arXiv.2305.20026.

[13] H. Guo, "Application of Model Predictive Path Integral Controller in Autonomous Driving: A Simulation Study," in *Proceedings of the 2024 2nd International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2024)*, vol. 115, Y. Wang, Ed., in Advances in Computer Science Research, vol. 115. , Dordrecht: Atlantis Press International BV, 2024, pp. 463–471. doi: 10.2991/978-94-6463-540-9_46.

[14] "Setting Up Transformations — Nav2 1.0.0 documentation." Accessed: Oct. 12, 2025. [Online]. Available: https://docs.nav2.org/setup_guides/transformation/setup_transforms.html

[15] "Tuning Guide — Nav2 1.0.0 documentation." Accessed: Sept. 07, 2025. [Online]. Available: https://docs.nav2.org/tuning/index.html

[16] S. Macenski, M. Booker, J. Wallace, and T. Fischer, “Cost-Aware Kinematically Feasible Planning for Mobile and Surface Robotics,” May 08, 2025, *arXiv*: arXiv:2401.13078. doi: 10.48550/arXiv.2401.13078.

[17] “15.3.2 Reeds-Shepp Curves.” Accessed: Oct. 13, 2025. [Online]. Available: https://lavalle.pl/planning/node822.html

[18] “Model Predictive Path Integral Controller — Nav2 1.0.0 documentation.” Accessed: Sept. 07, 2025. [Online]. Available: https://docs.nav2.org/configuration/packages/configuring-mppic.html

[19] I. S. Mohamed, K. Yin, and L. Liu, “Autonomous Navigation of AGVs in Unknown Cluttered Environments: log-MPPI Control Strategy,” July 18, 2022, *arXiv*: arXiv:2203.16599. doi: 10.48550/arXiv.2203.16599.

[20] I. S. Mohamed, J. Xu, G. S. Sukhatme, and L. Liu, “Towards Efficient MPPI Trajectory Generation with Unscented Guidance: U-MPPI Control Strategy,” Dec. 21, 2024, *arXiv*: arXiv:2306.12369. doi: 10.48550/arXiv.2306.12369.

[21] B. Vlahov, J. Gibson, M. Gandhi, and E. A. Theodorou, “MPPI-Generic: A CUDA Library for Stochastic Trajectory Optimization,” Aug. 13, 2025, *arXiv*: arXiv:2409.07563. doi: 10.48550/arXiv.2409.07563.

Appendices

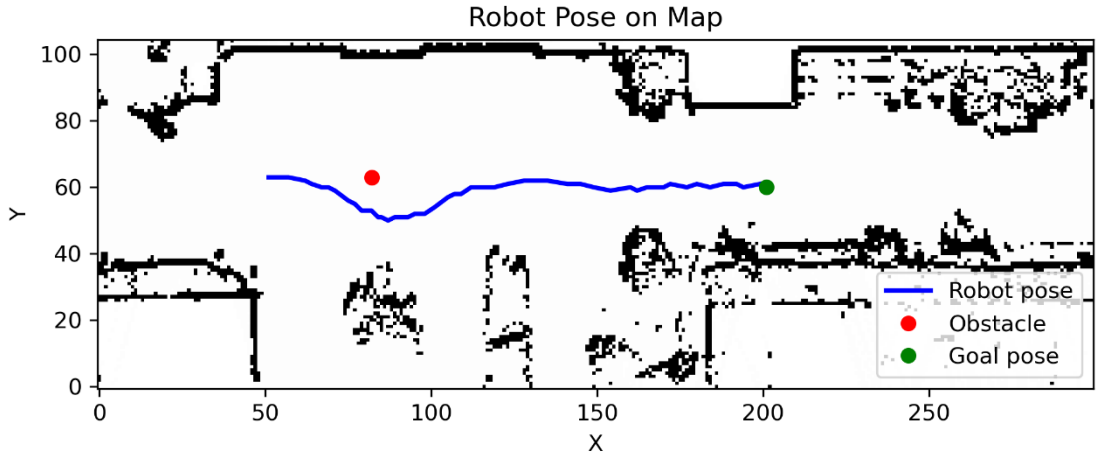
Static Obstacle Test Data

MPPI Performance Metrics

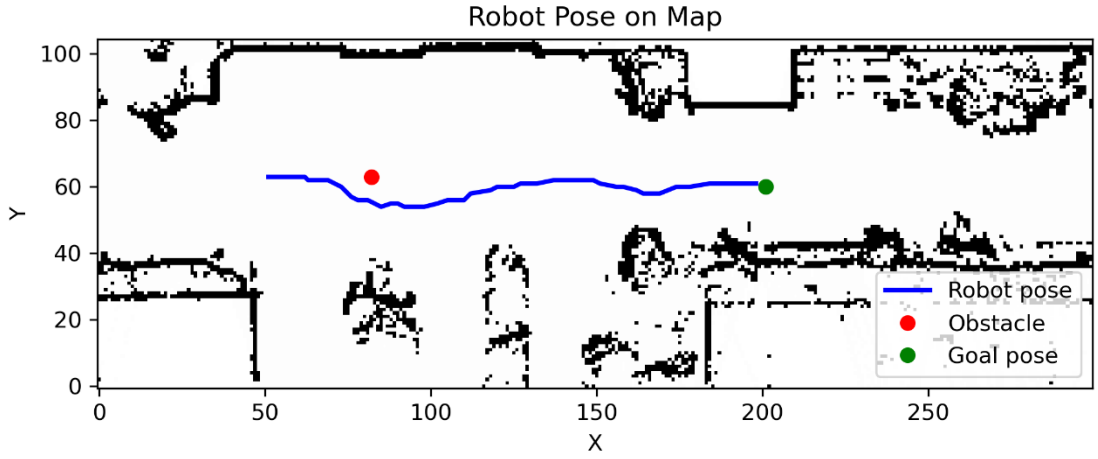
Run number	Average Speed (m/s)	Path Length (m)	Turning Distance from Obstacle (m)
1	0.576681	31.05301	2.600534
2	0.502691	30.2895	2.371051
3	0.583504	30.26709	2.721554
4	0.480066	30.42317	2.452031
5	0.60061	31.11483	2.434096
6	0.548508	31.02143	2.342834
7	0.588851	30.84828	3.307006
8	0.588768	30.62544	1.893202
9	0.393488	3.414177	1.697354
10	0.299183	7.200349	2.391317

MPPI Trajectories

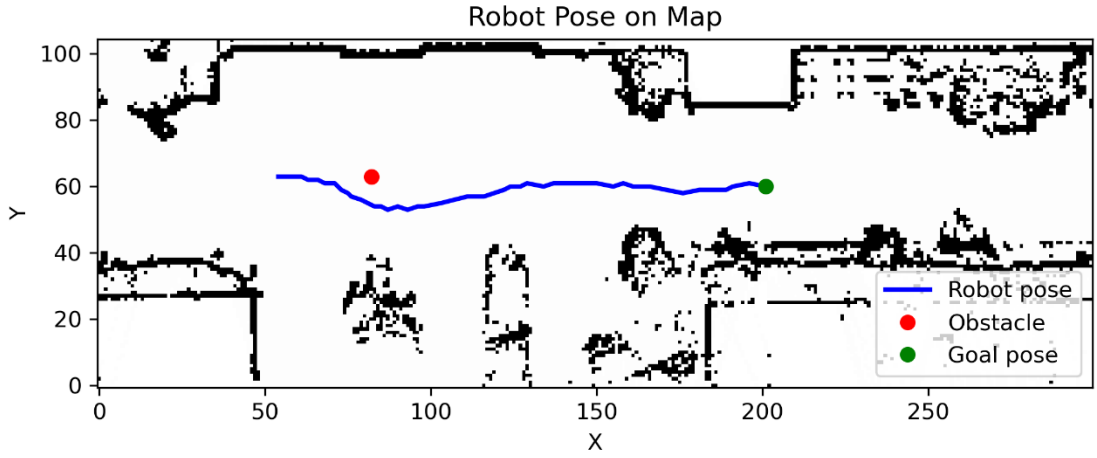
Run 1



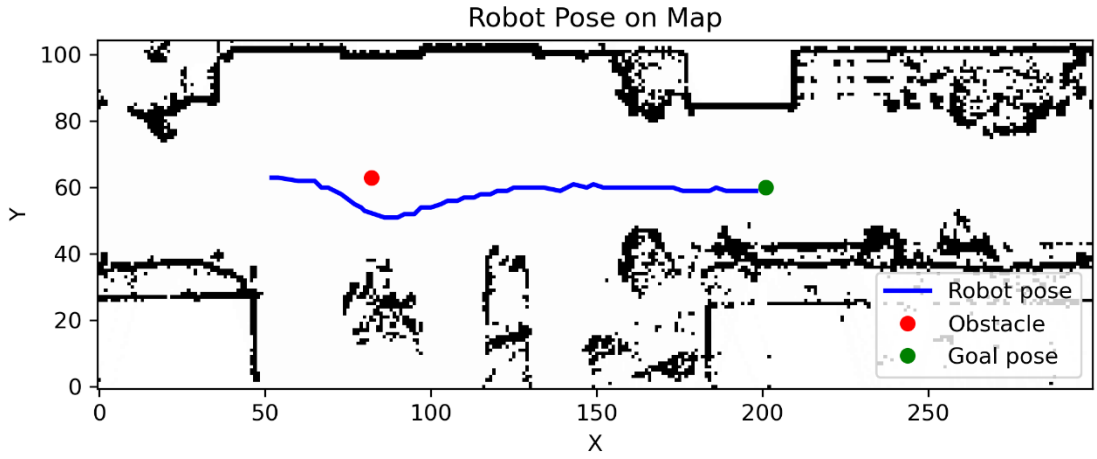
Run 2



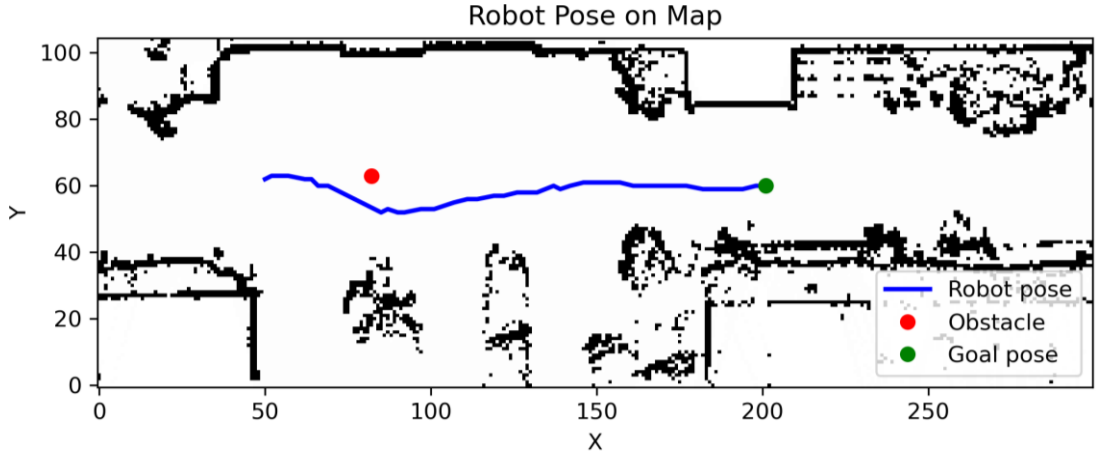
Run 3



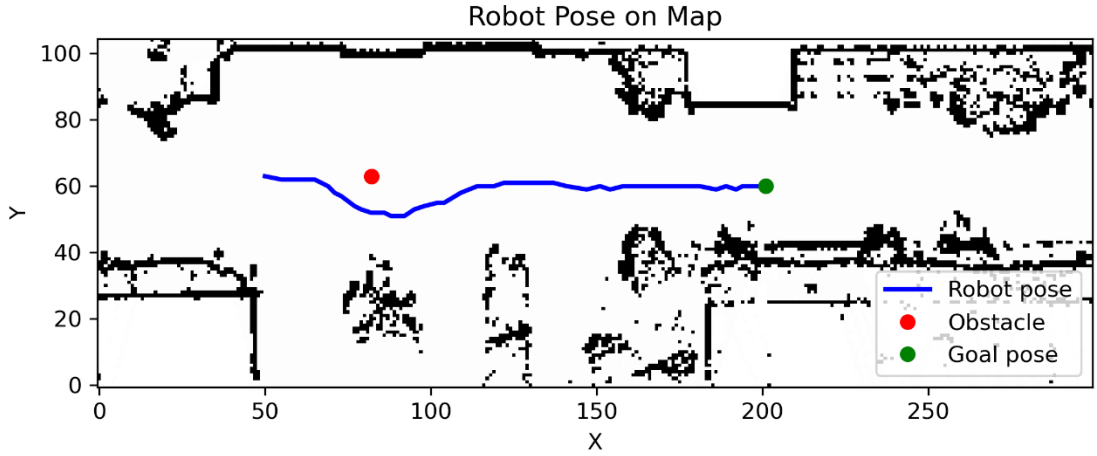
Run 4



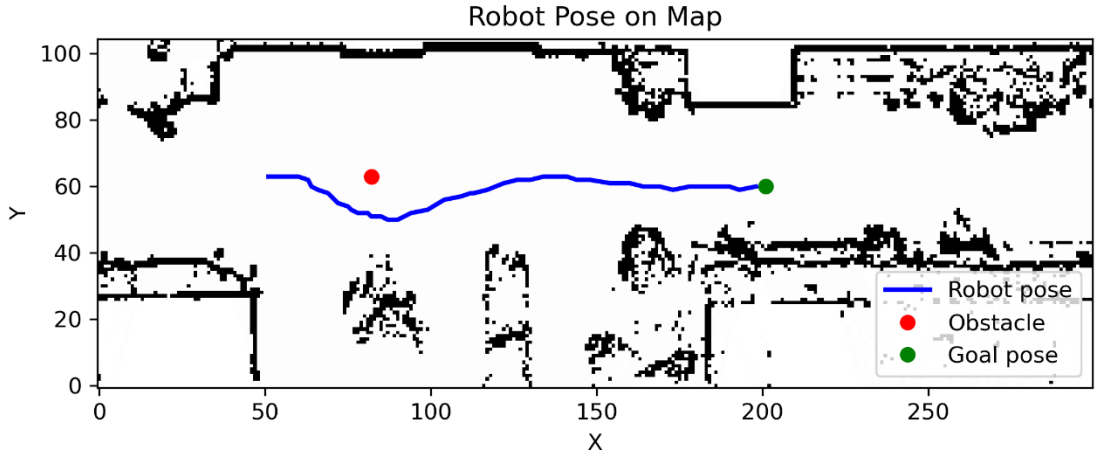
Run 5



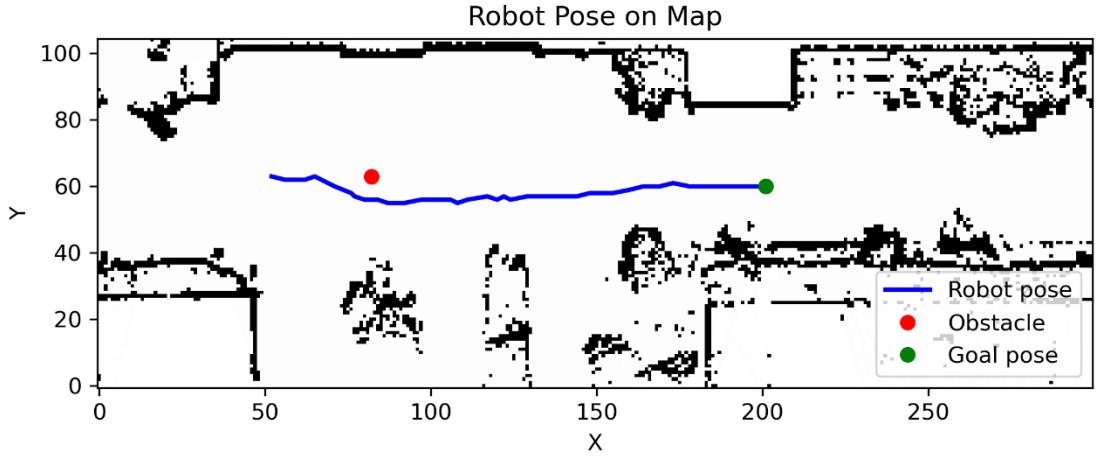
Run 6



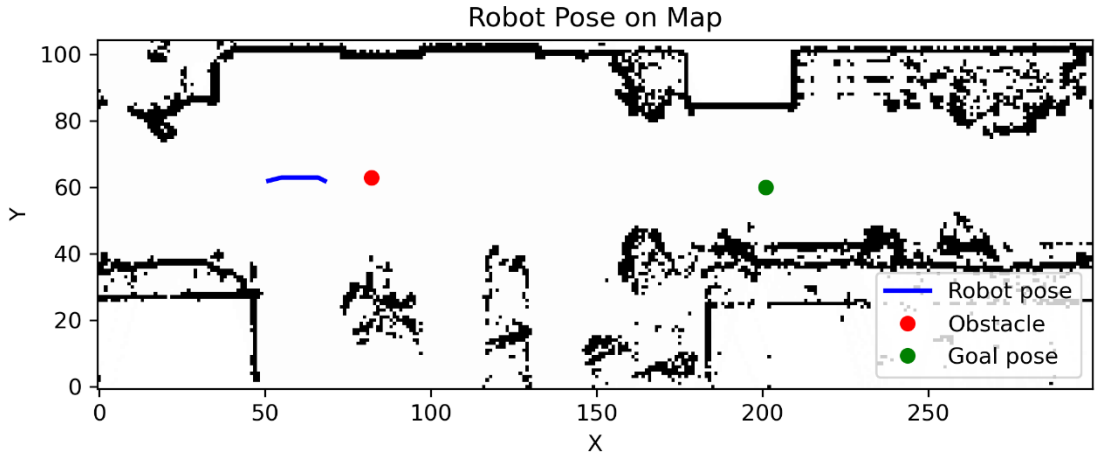
Run 7



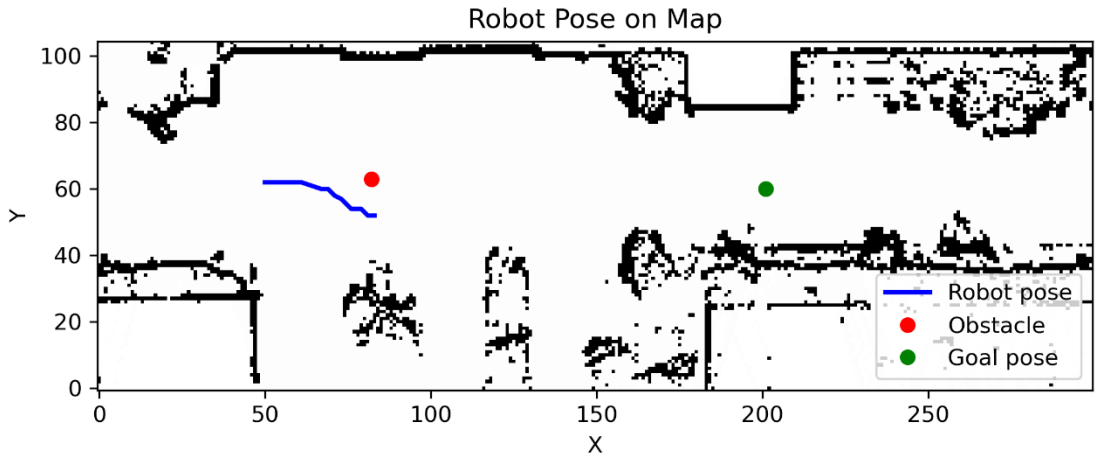
Run 8



Run 9

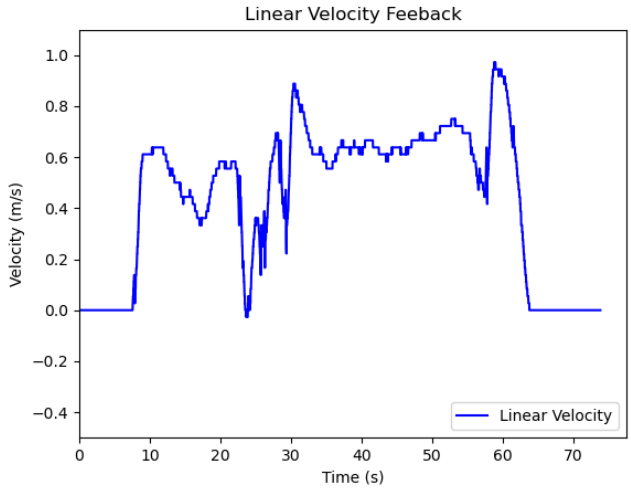

Robot Pose on Map
Robot pose
Obstacle
Goal pose
X
Y
0
50
100
150
200
250
0
20
40
60
80
100

Run 10

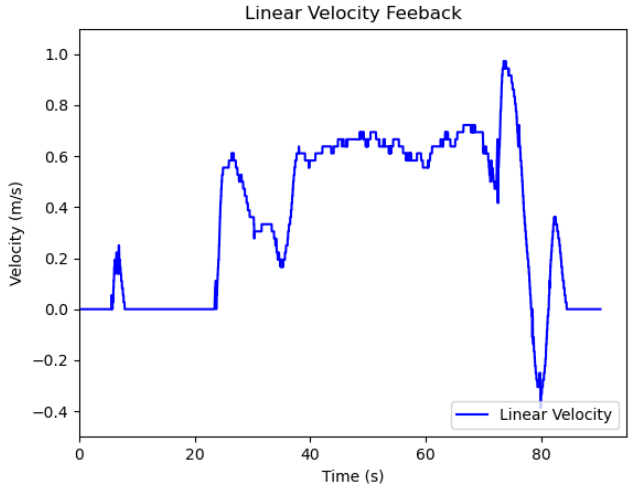

Robot Pose on Map
Robot pose
Obstacle
Goal pose
X
Y
0
50
100
150
200
250
0
20
40
60
80
100

MPPI Velocities

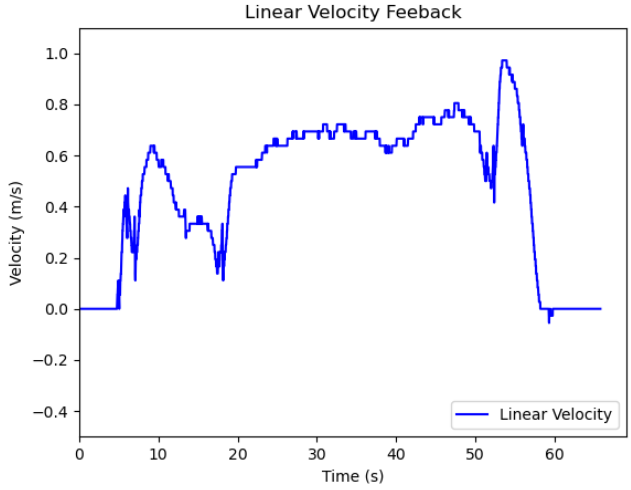
Run 1



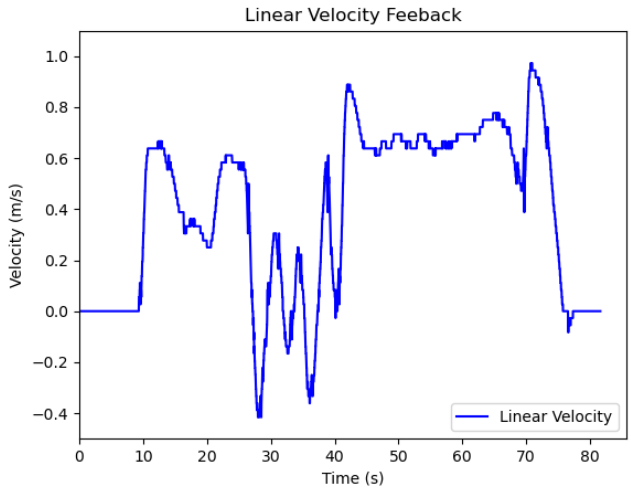
Run 2



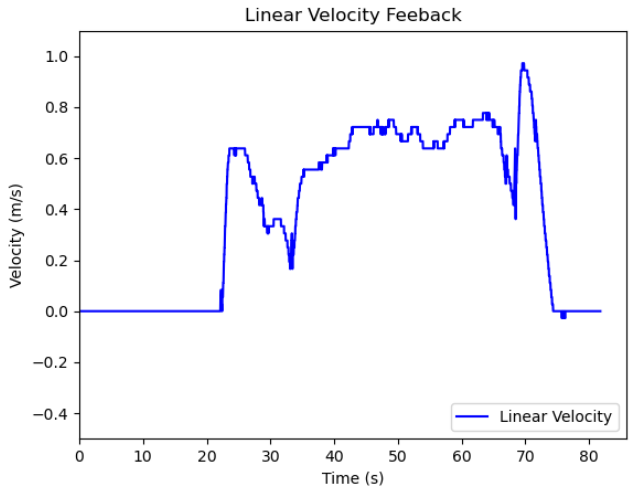
Run 3



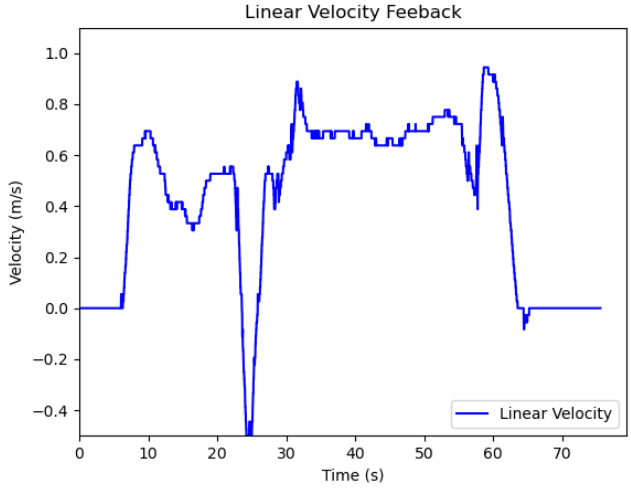
Run 4



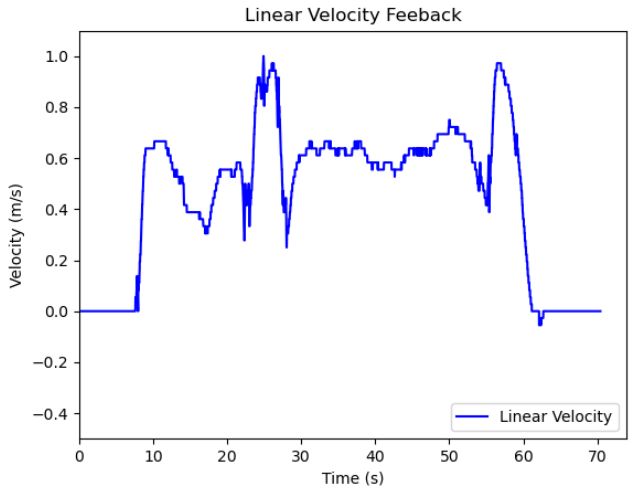
Run 5



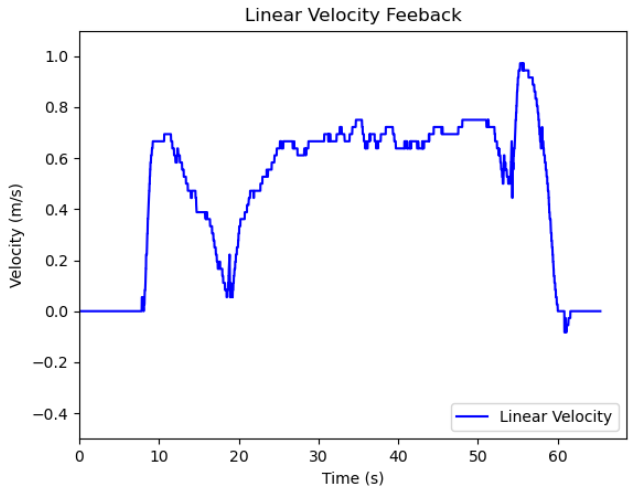
Run 6



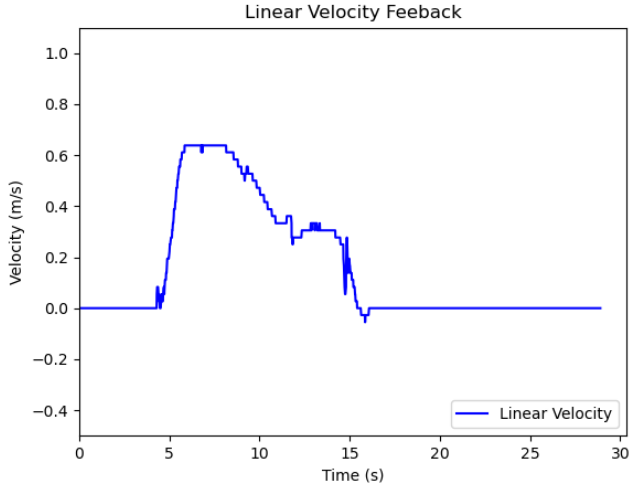
Run 7



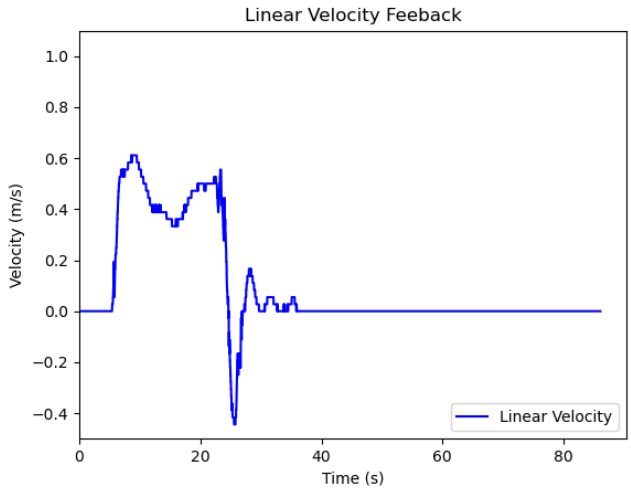
Run 8



Run 9



Run 10

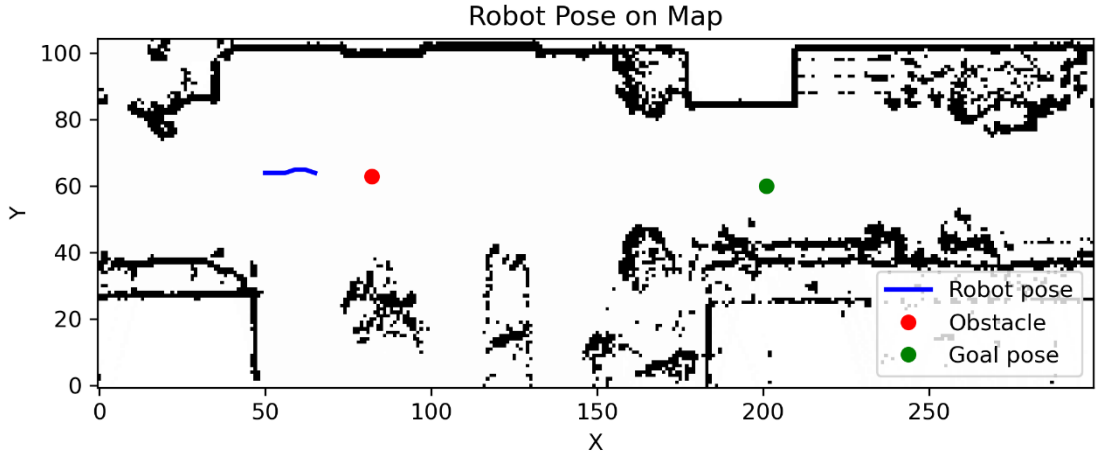


RPP Performance Metrics

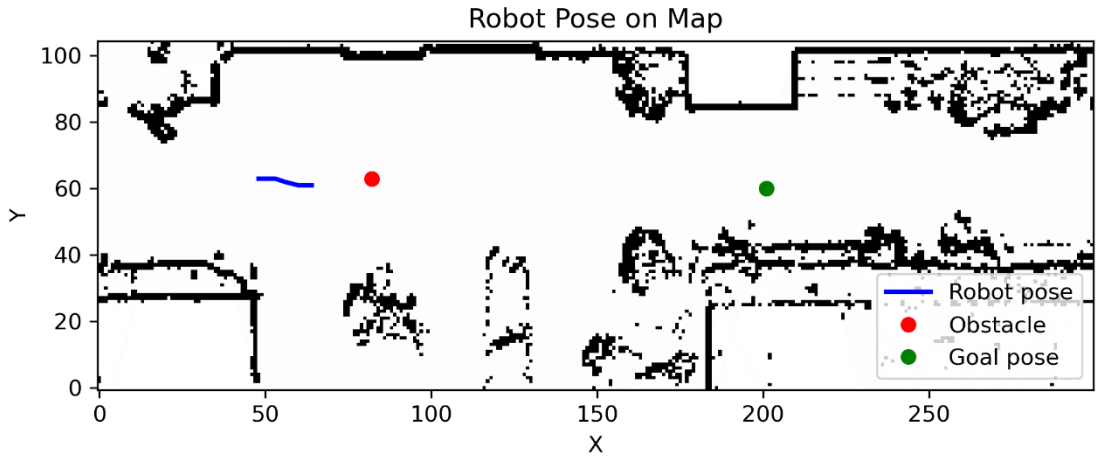
Run number	Average Speed (m/s)	Path Length (m)	Turning Distance from Obstacle (m)
1	0.543051	3.146245	N/A
2	0.565573	3.248261	N/A
3	0.568994	2.982608	N/A
4	0.559283	3.260198	N/A
5	0.584452	3.001602	N/A
6	0.675322	31.22796	2.352962
7	0.587419	2.500488	N/A
8	0.530936	2.938294	N/A
9	0.595305	2.92881	N/A
10	0.618005	4.070639	N/A

RPP Trajectories

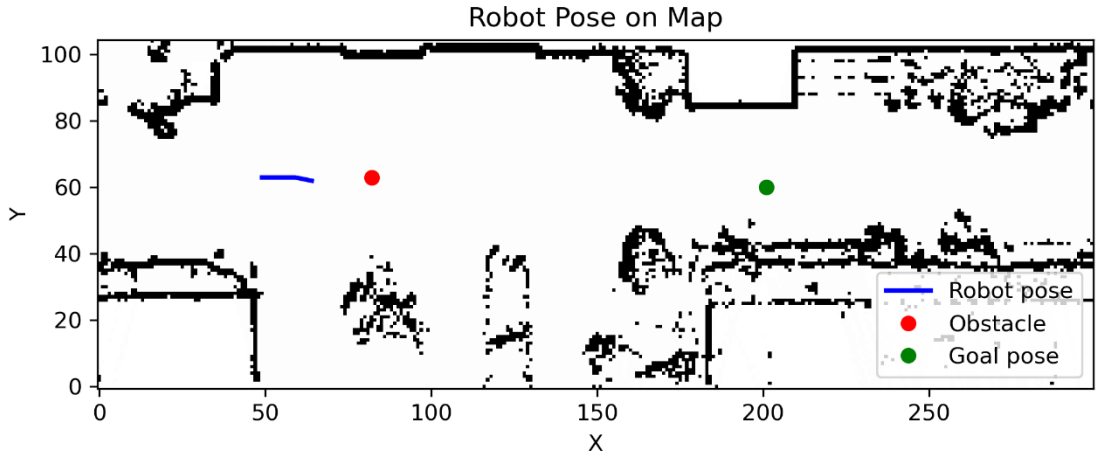
Run 1



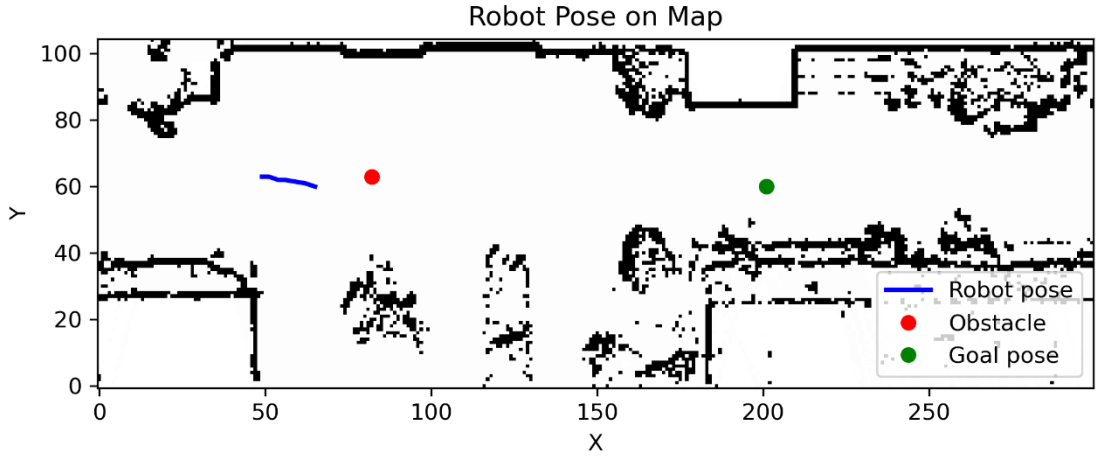
Run 2



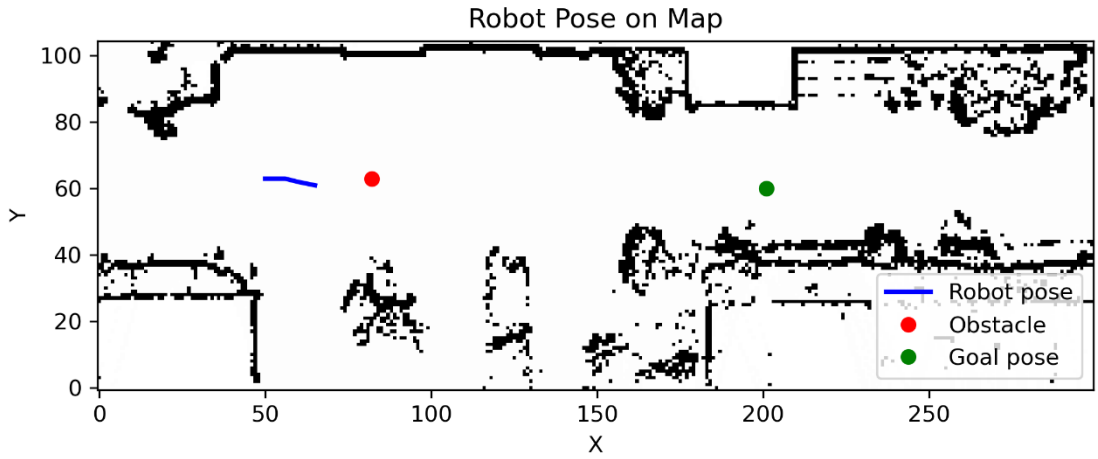
Run 3



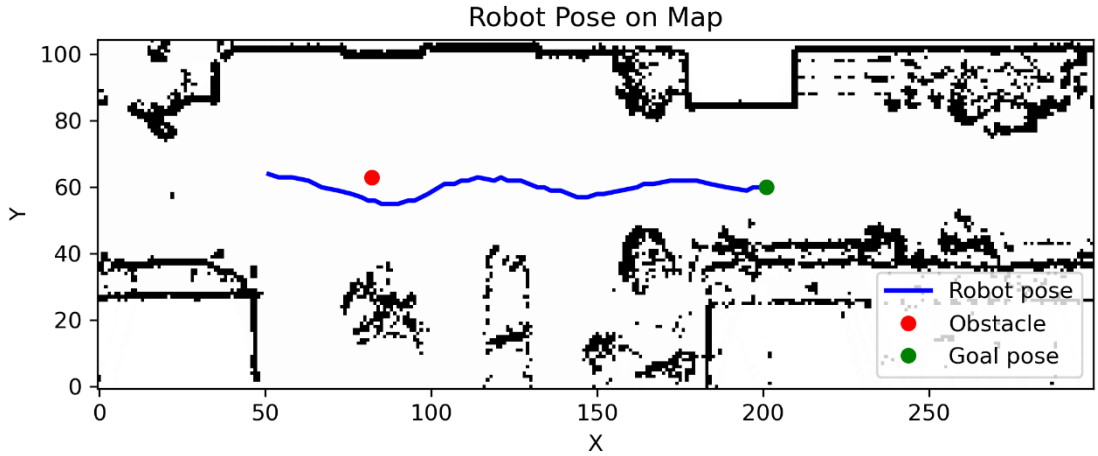
Run 4



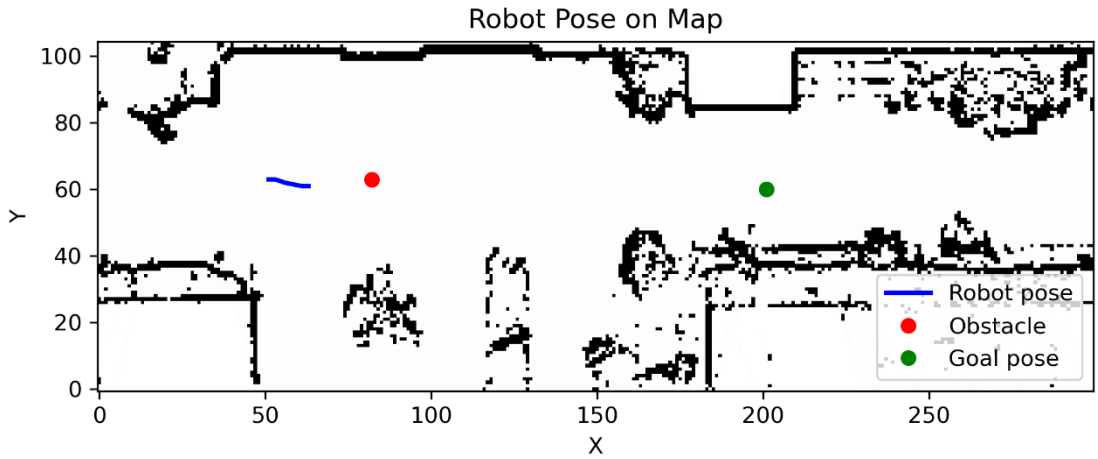
Run 5



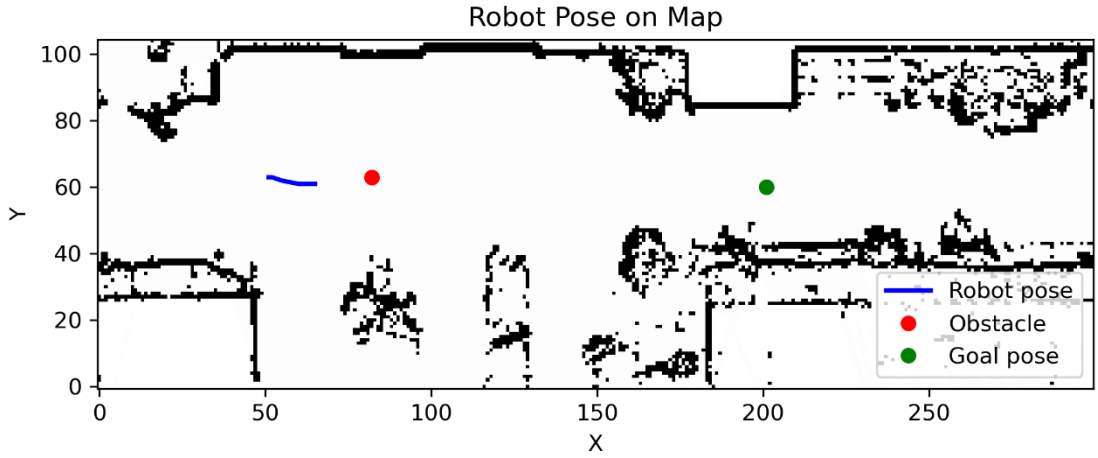
Run 6



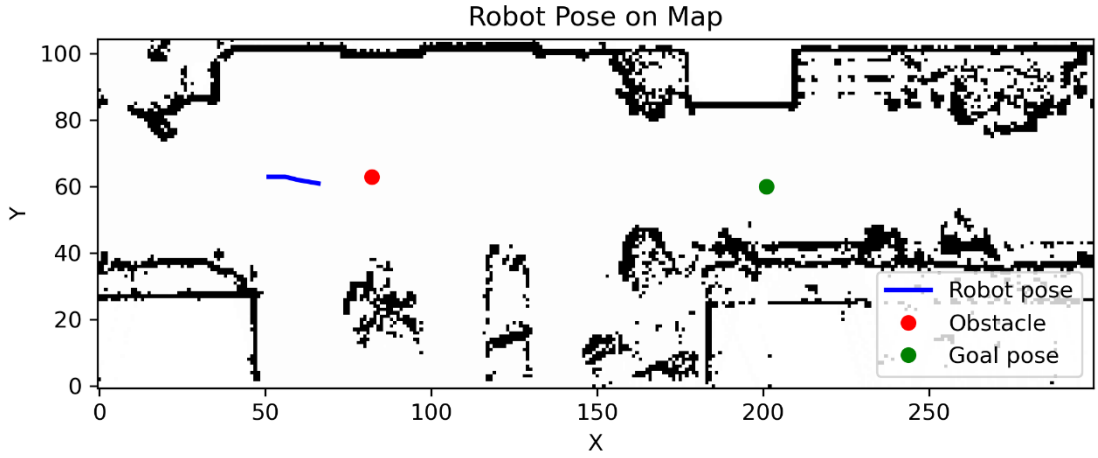
Run 7



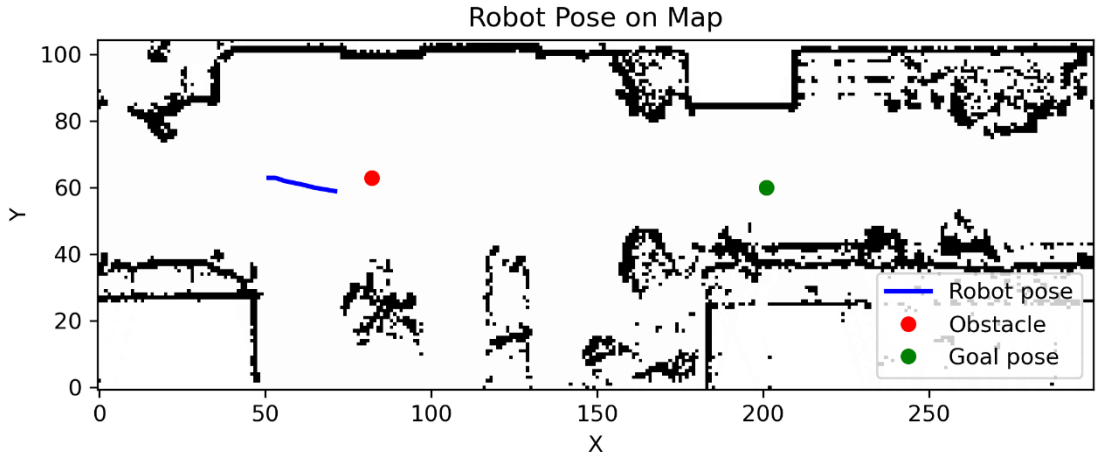
Run 8



Run 9

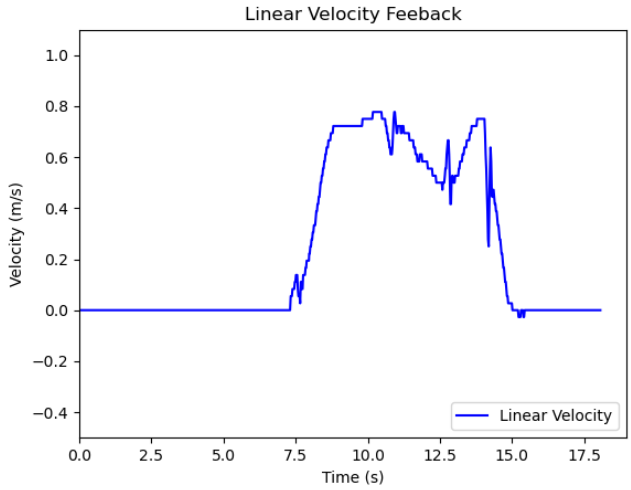


Run 10

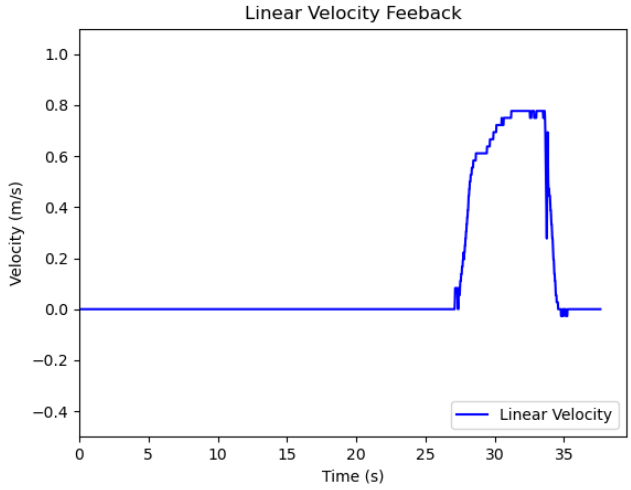


RPP Velocities

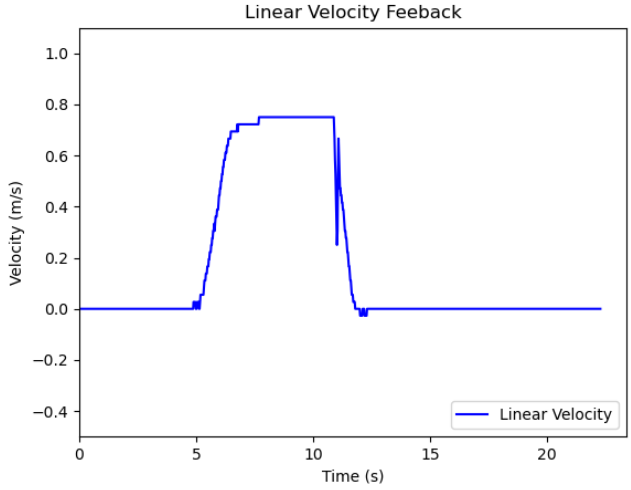
Run 1



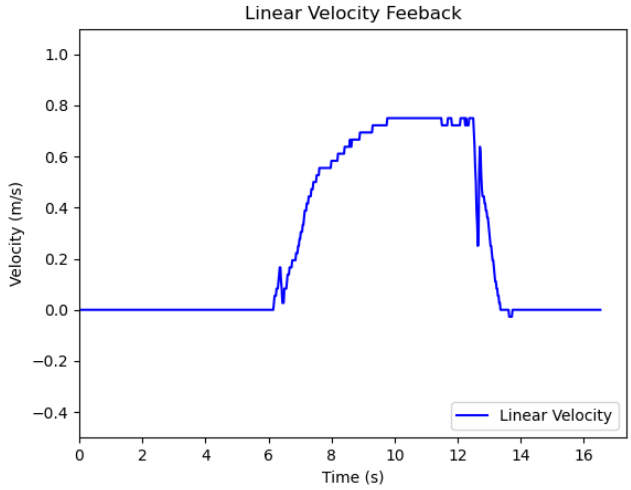
Run 2



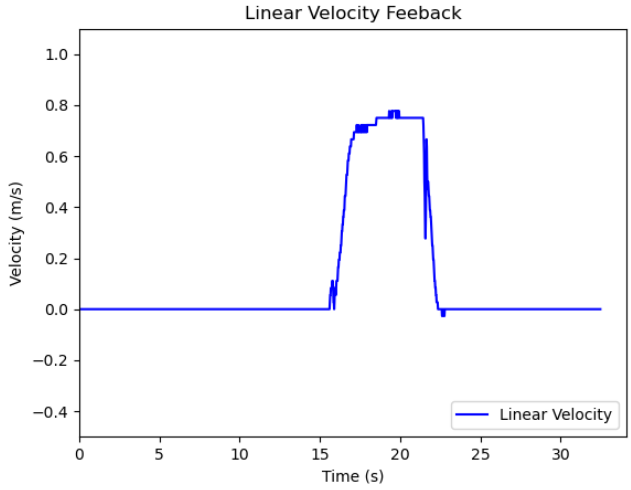
Run 3



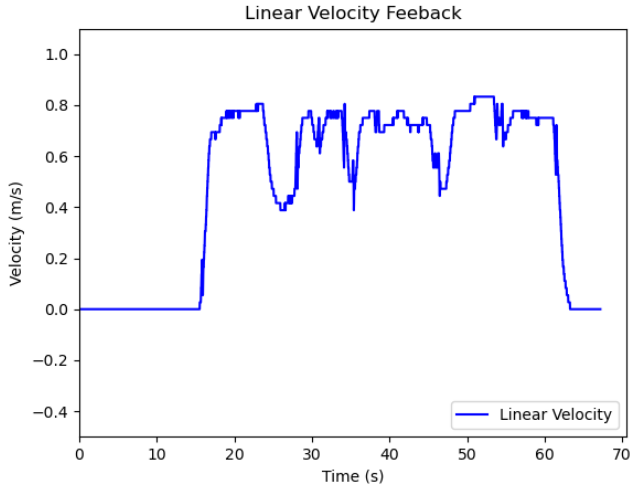
Run 4



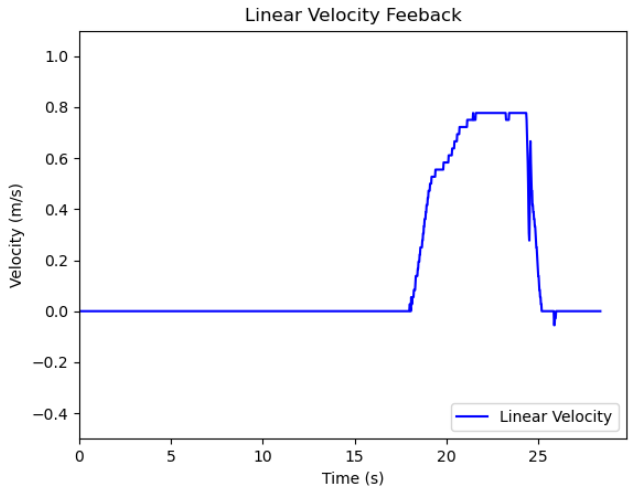
Run 5



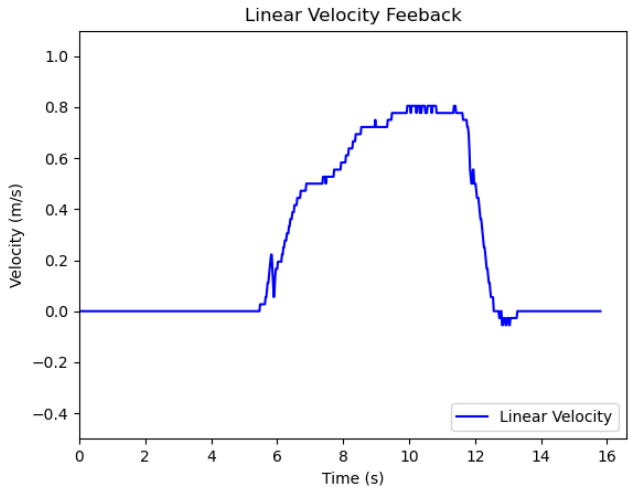
Run 6



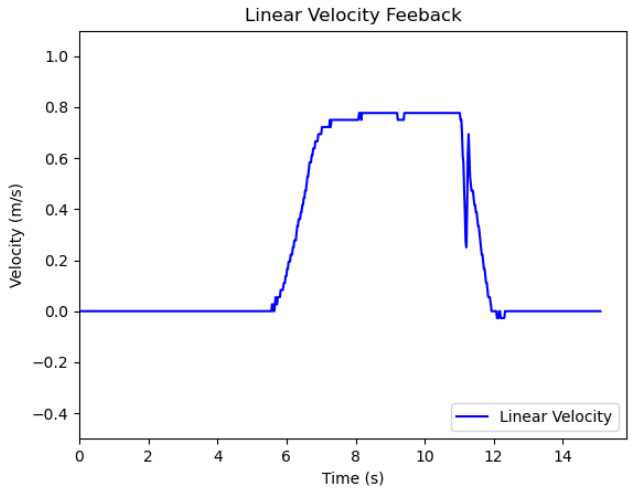
Run 7



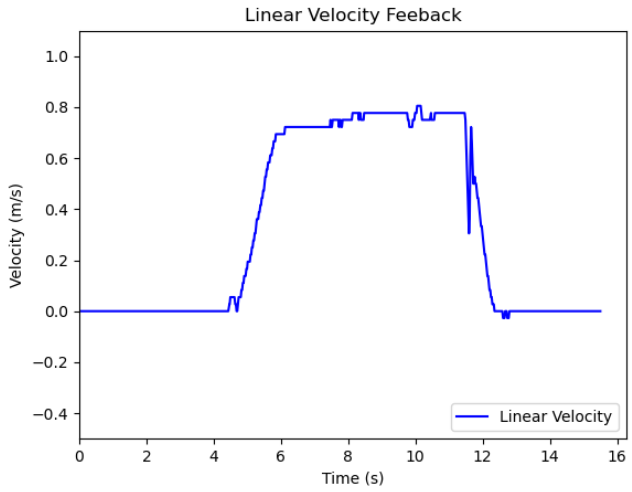
Run 8



Run 9



Run 10



Dynamic Obstacle Test Data

MPPI Performance Metrics

Run Number	Turning Distance from Obstacle (m)
1	No turn
2	No turn
3	2.250743
4	2.201266
5	2.937031

RPP Performance Metrics

Run Number	Turning Distance from Obstacle (m)
1	No turn
2	No turn
3	No turn
4	No turn
5	No turn

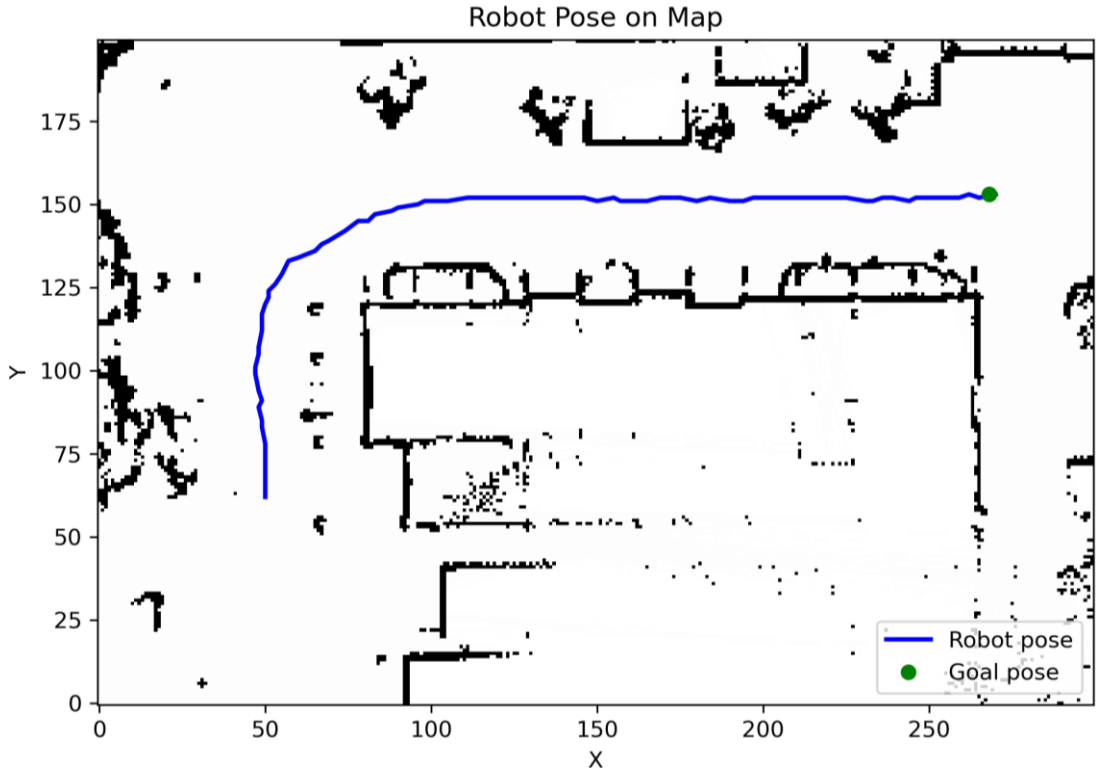
Driving Test Data

MPPI Performance Metrics

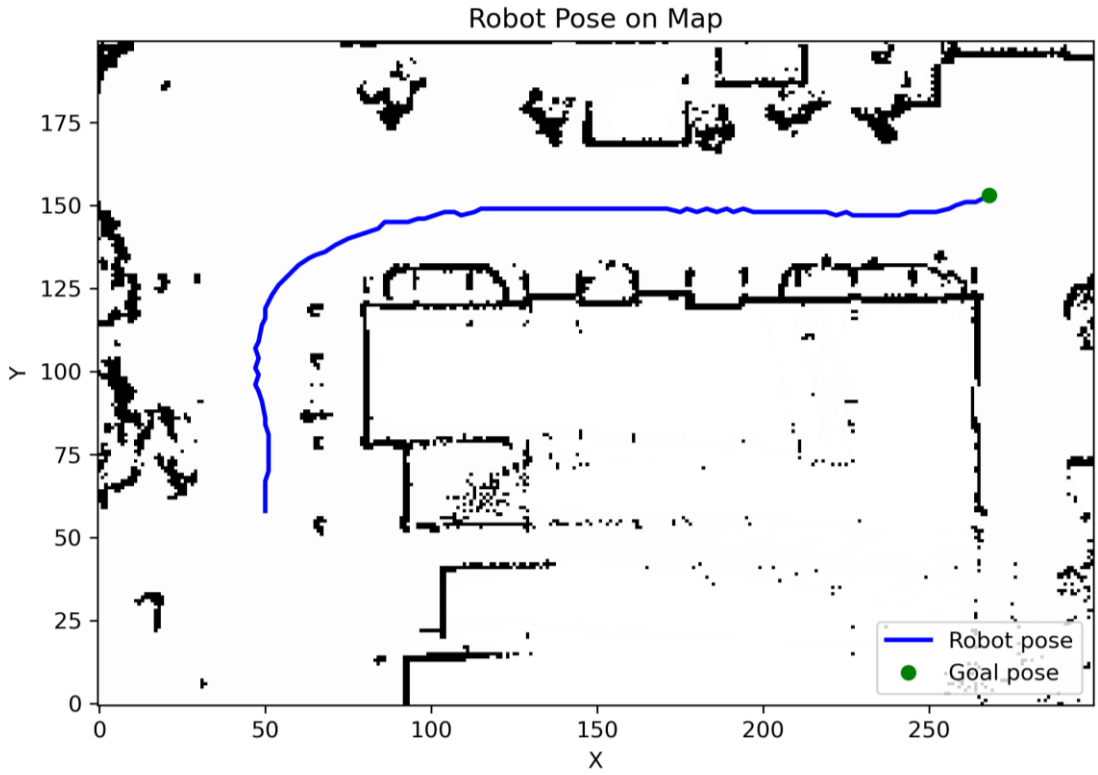
Run Number	Average Speed (m/s)	Average CPU Usage (%)	Path Length (m)	Path Smoothness
1	0.680722	19.93711	58.61317	1.245792
2	0.596154	20.05195	58.98969	1.42527
3	0.687279	20.39082	57.37519	1.376209
4	0.60664	19.87757	58.28398	1.547808
5	0.612855	20.65532	58.08344	0.939129

MPPI Trajectories

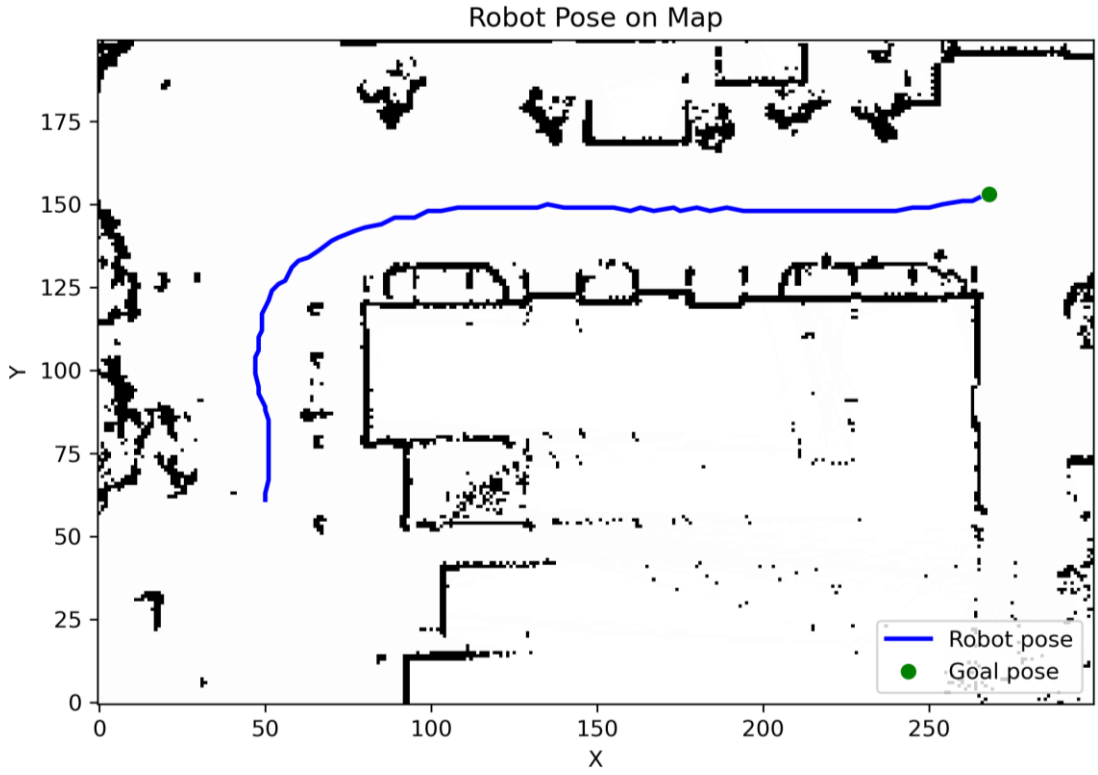
Run 1



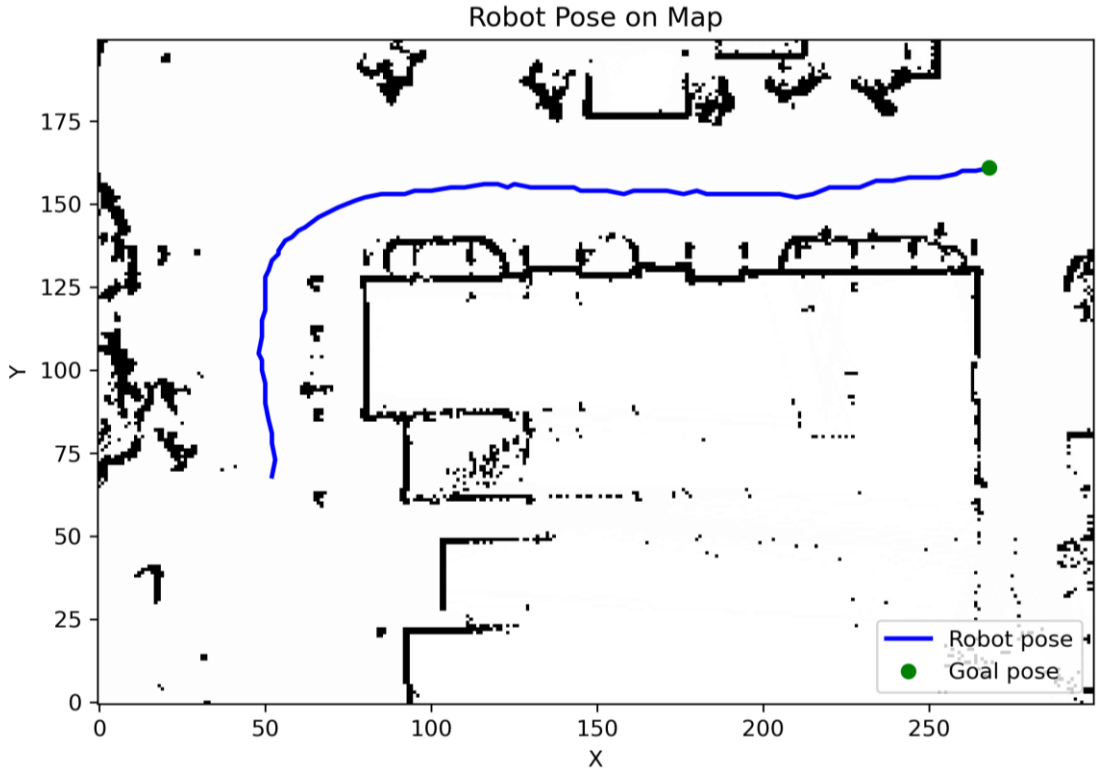
Run 2



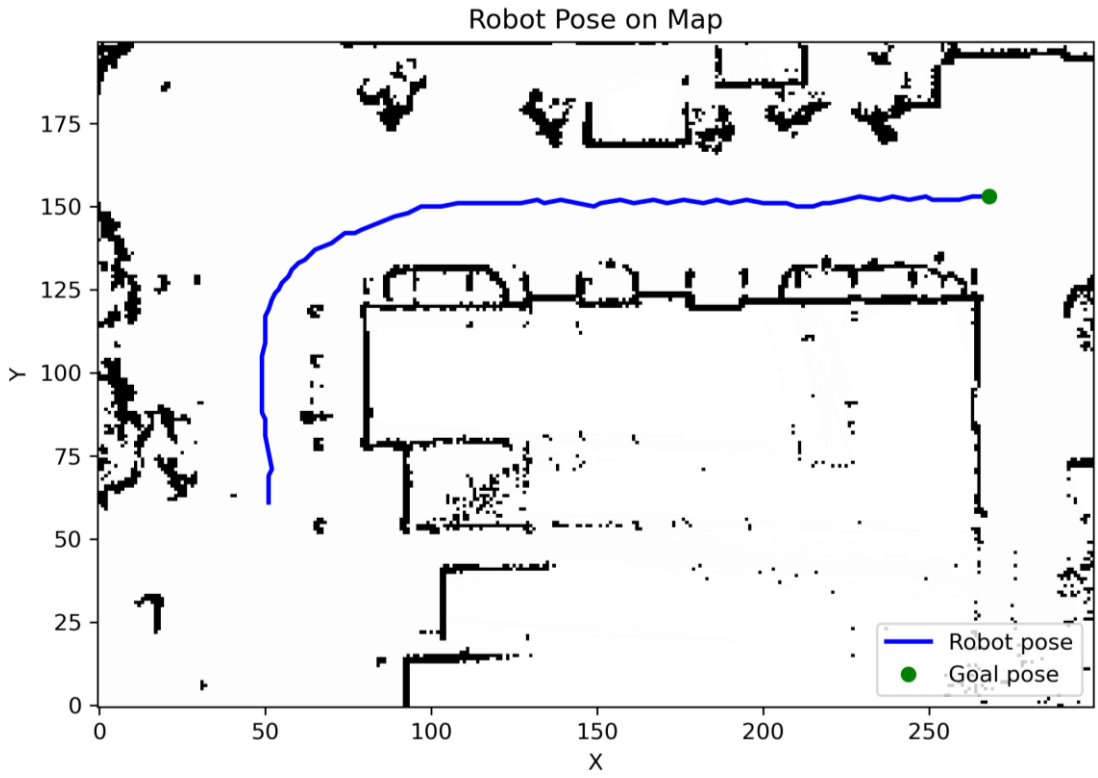
Run 3



Run 4

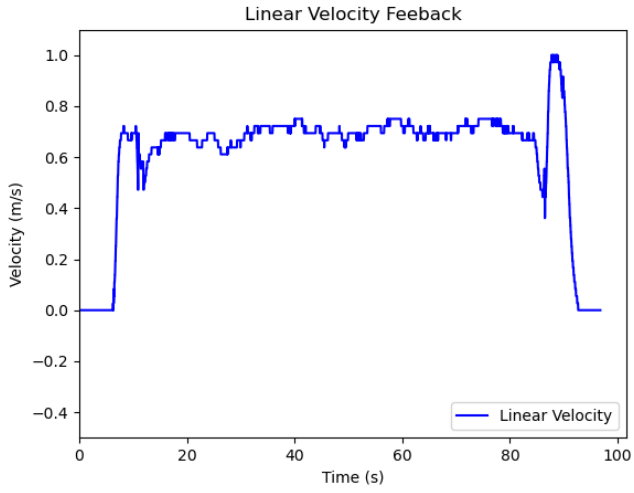


Run 5

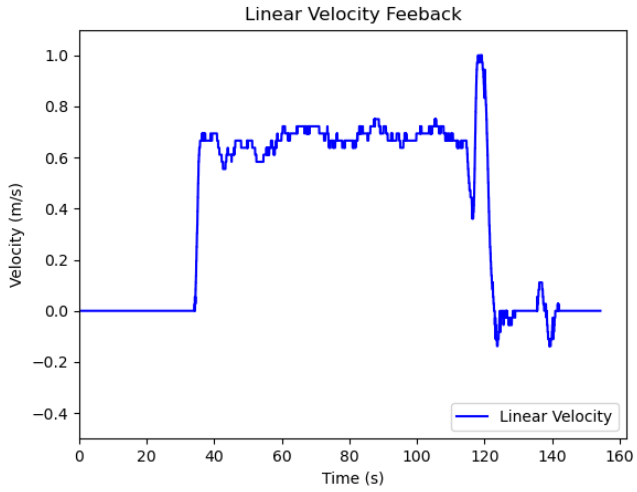


MPPI Velocities

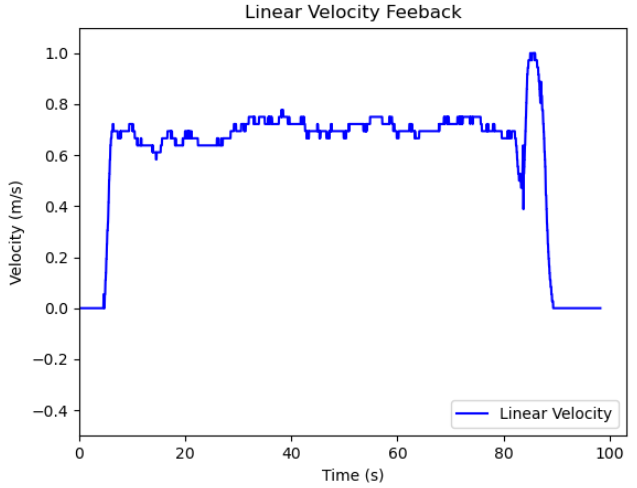
Run 1



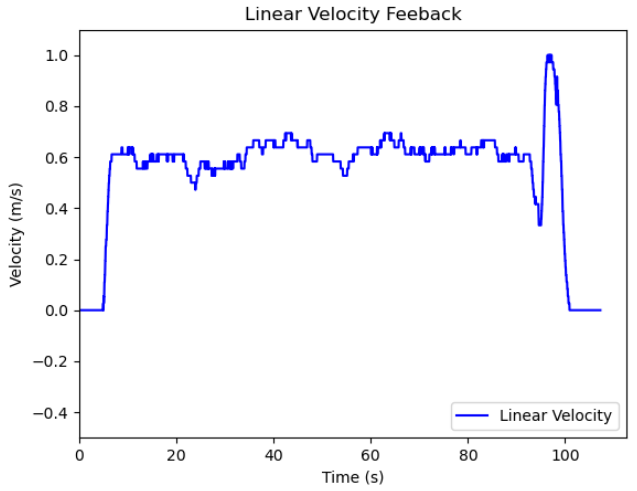
Run 2


Linear Velocity Feeback
Velocity (m/s)
Time (s)
Linear Velocity

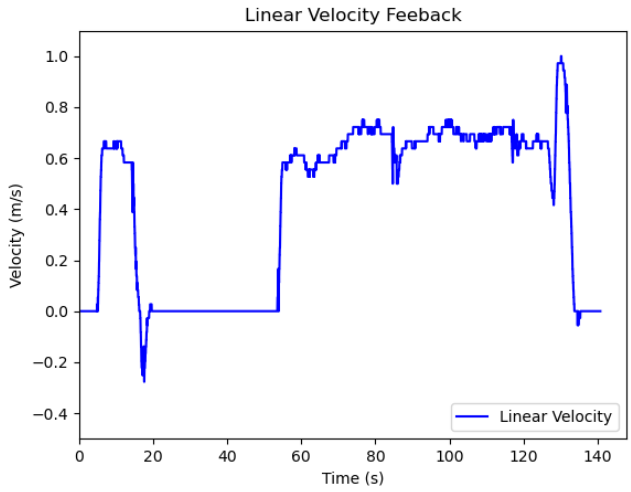
Run 3


Linear Velocity Feeback
Velocity (m/s)
Time (s)
Linear Velocity

Run 4


Linear Velocity Feeback
Velocity (m/s)
Time (s)
Linear Velocity

Run 5

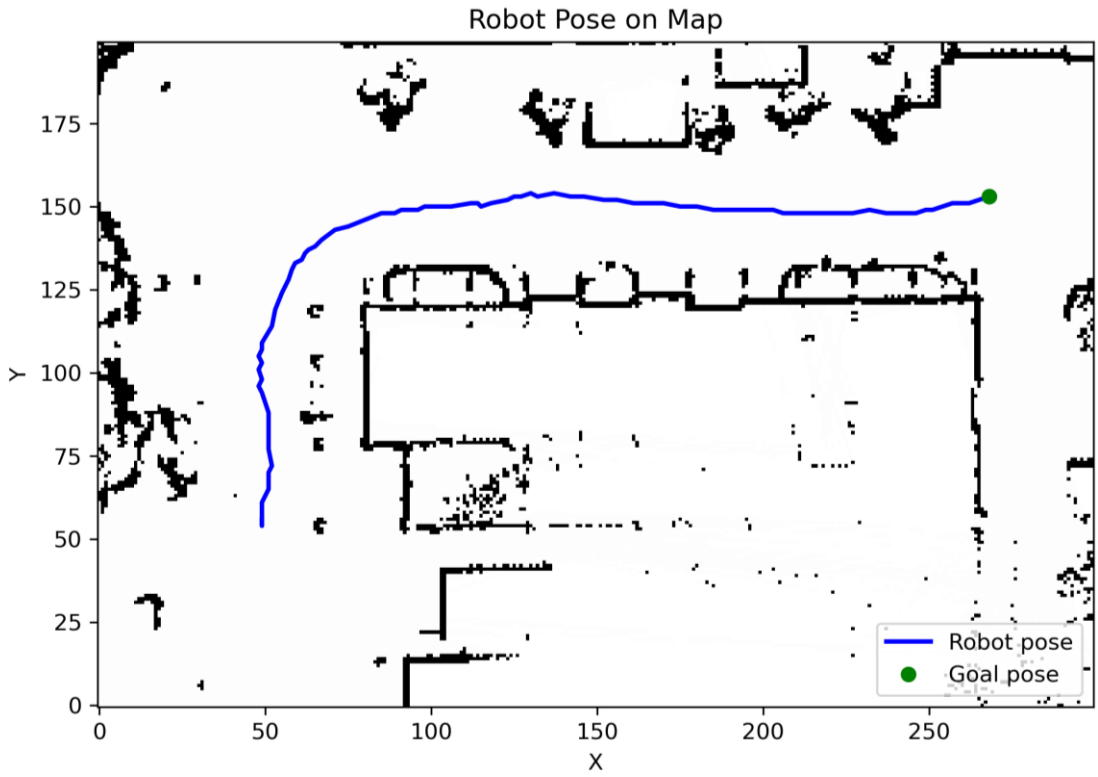


RPP Performance Metrics

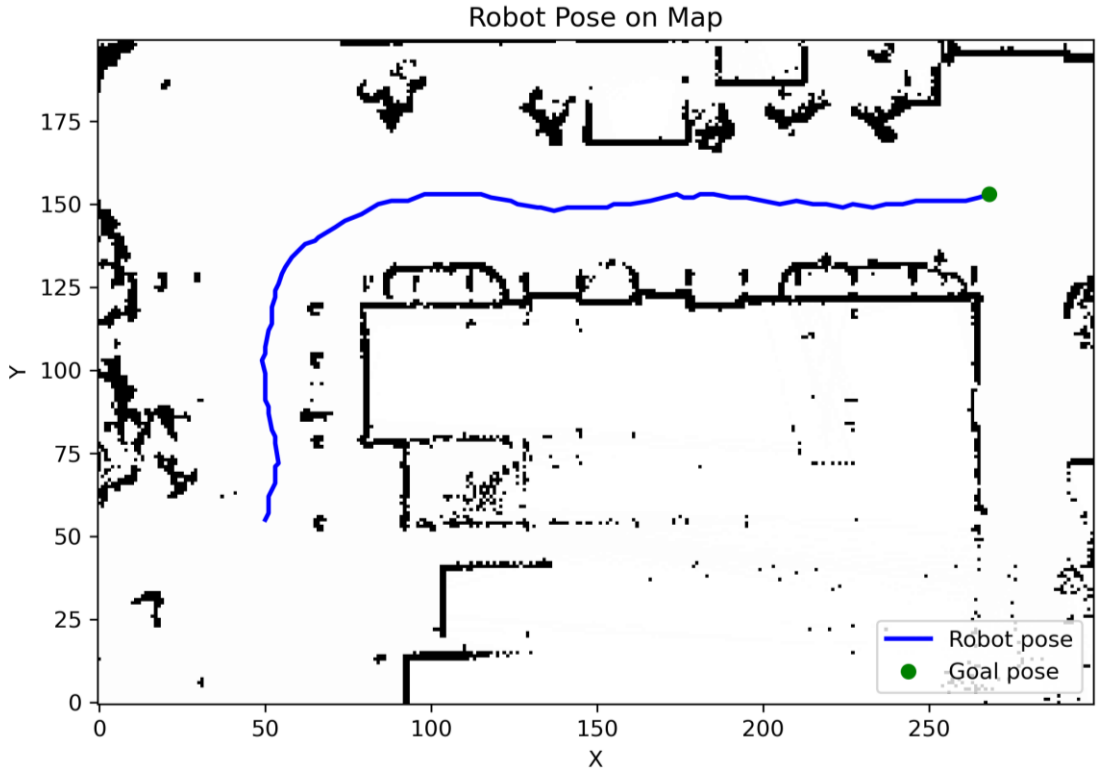
Run Number	Average Speed (m/s)	Average CPU Usage (%)	Path Length (m)	Path Smoothness
1	0.677306	16.96617	60.07378	1.353878
2	0.722576	16.40102	59.96809	2.524827
3	0.687354	16.23711	59.52243	1.160518
4	0.703621	16.57959	58.58302	1.375463
5	0.701288	16.75773	59.25867	1.499537

RPP Trajectories

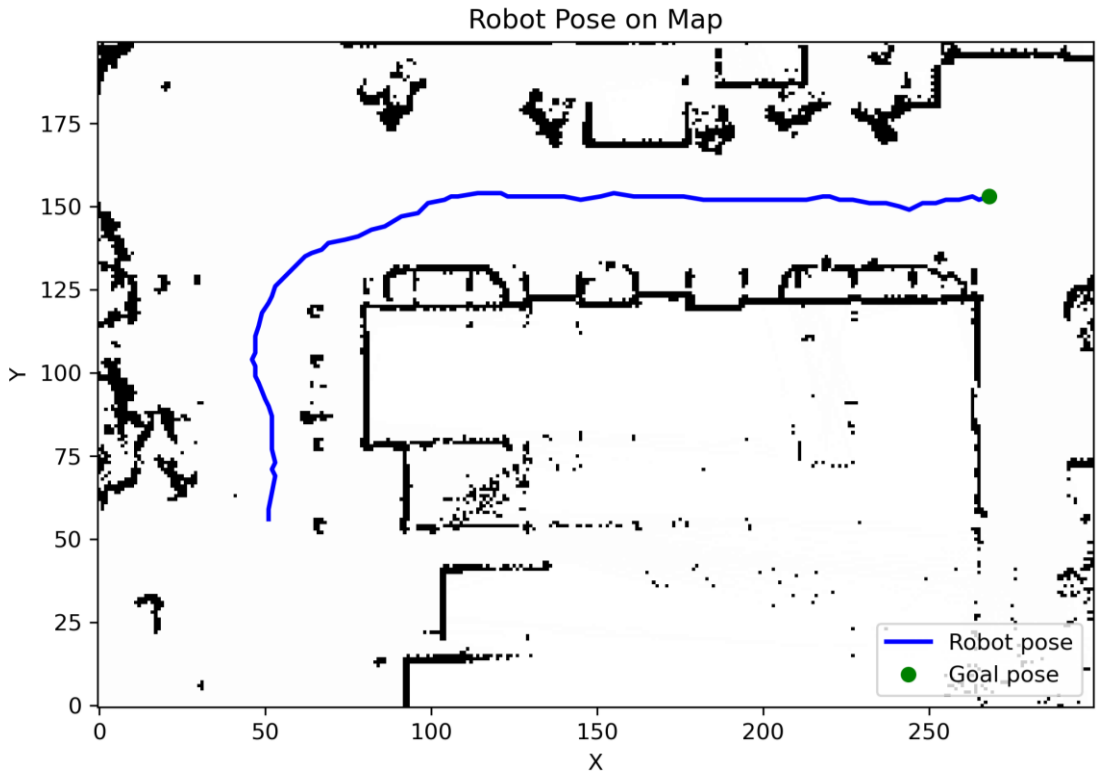
Run 1


Robot Pose on Map
Robot pose
Goal pose
X
Y
0
50
100
150
200
250
0
25
50
75
100
125
150
175

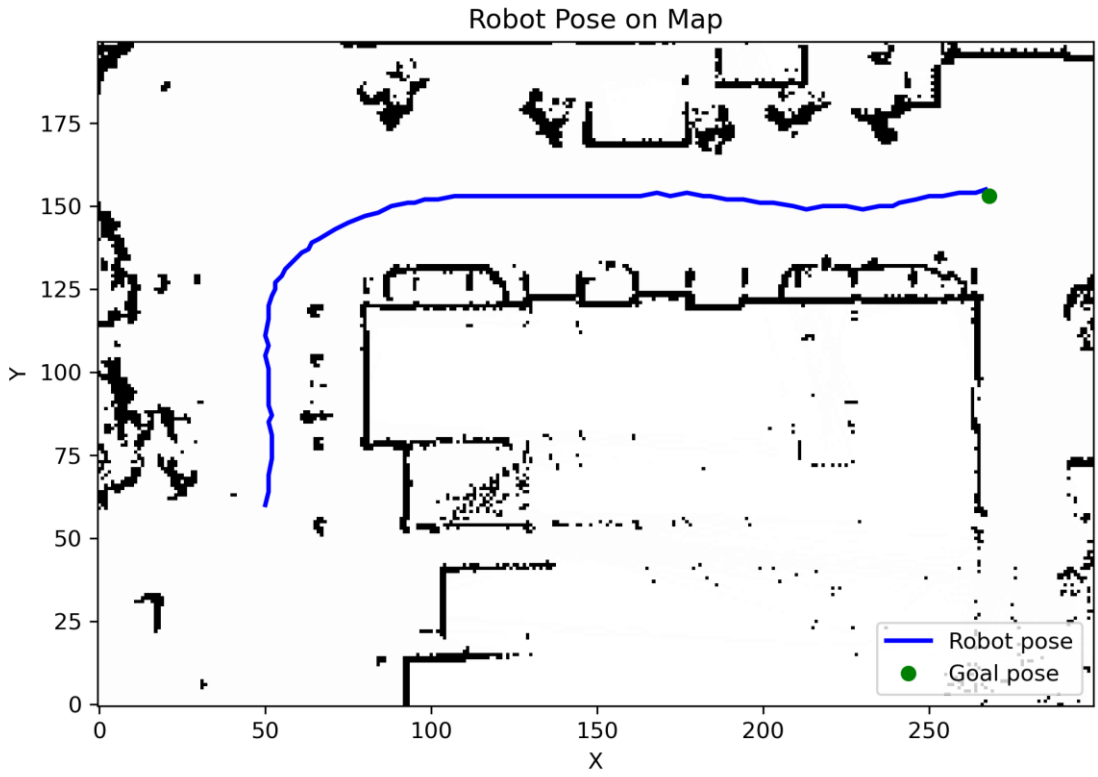
Run 2


Robot Pose on Map
Robot pose
Goal pose
X
Y
0
50
100
150
200
250
0
25
50
75
100
125
150
175

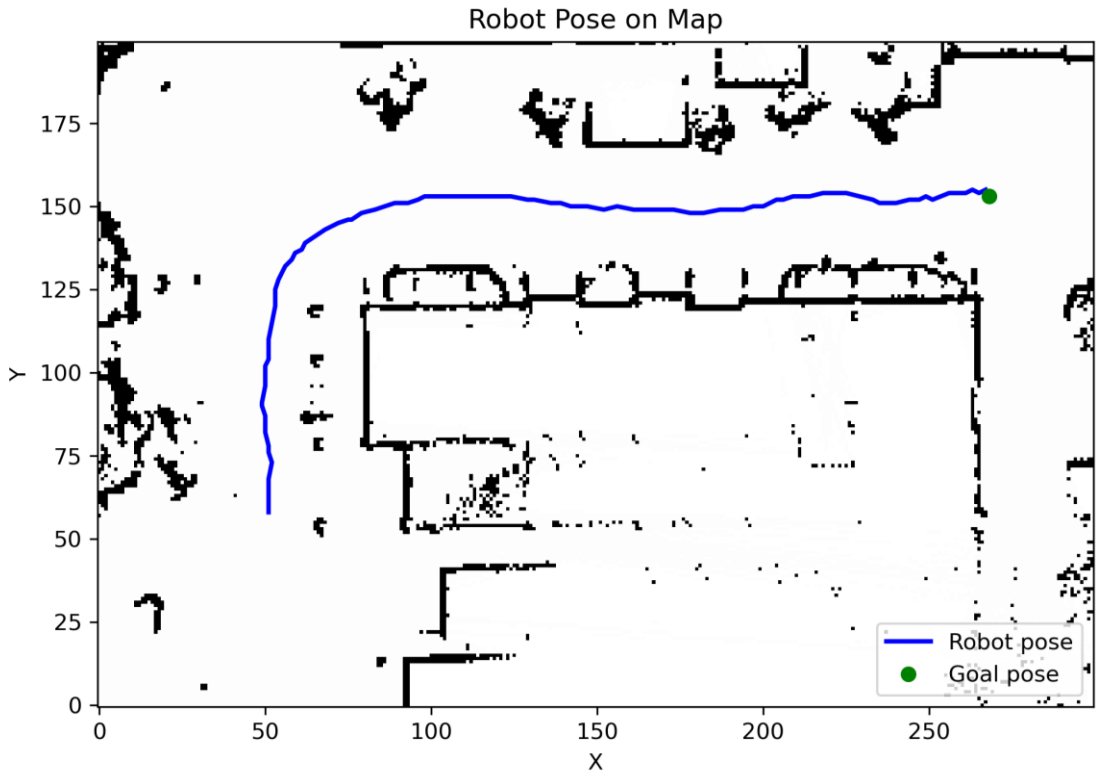
Run 3


Robot Pose on Map
Robot pose
Goal pose
X
Y
0
25
50
75
100
125
150
175
0
50
100
150
200
250

Run 4

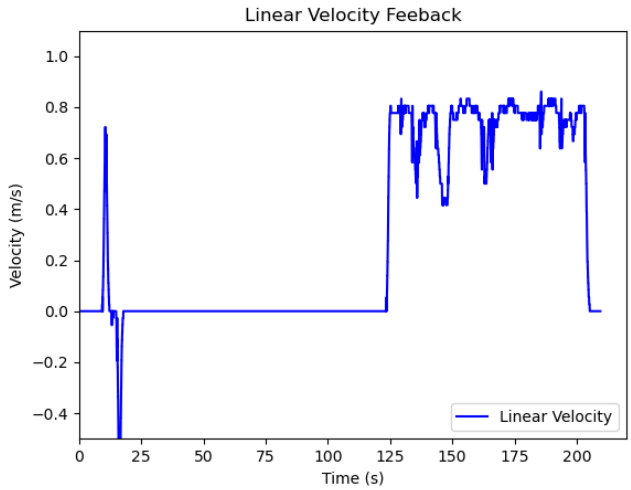

Robot Pose on Map
Robot pose
Goal pose
X
Y
0
25
50
75
100
125
150
175
0
50
100
150
200
250

Run 5

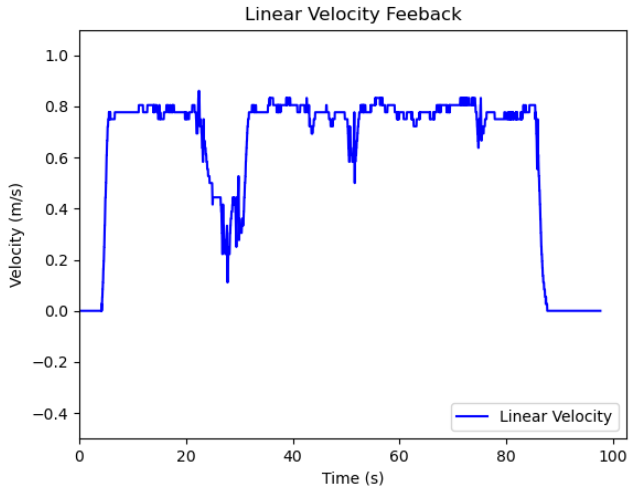


RPP Velocities

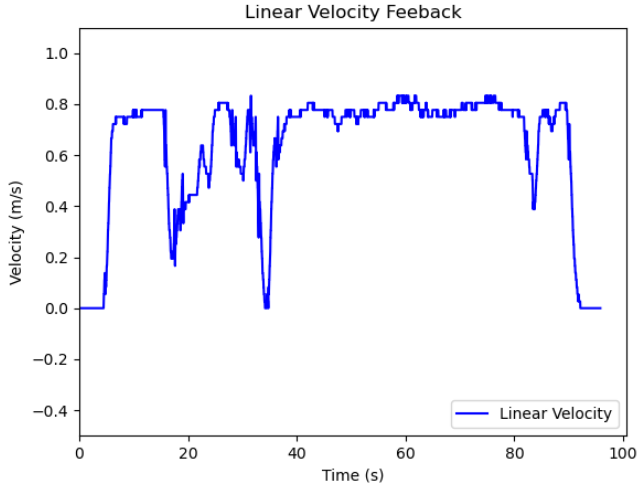
Run 1



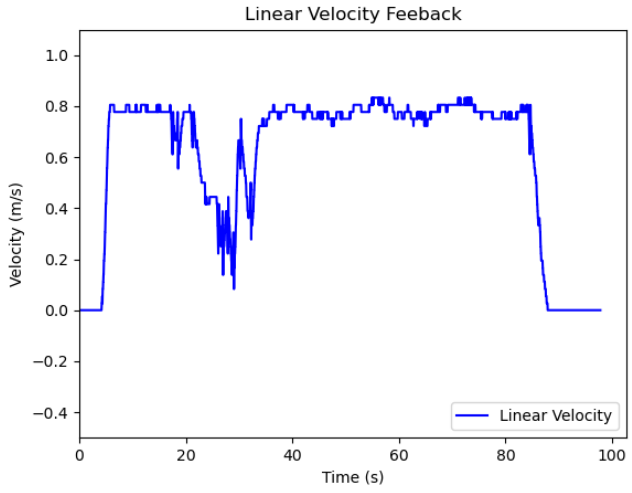
Run 2



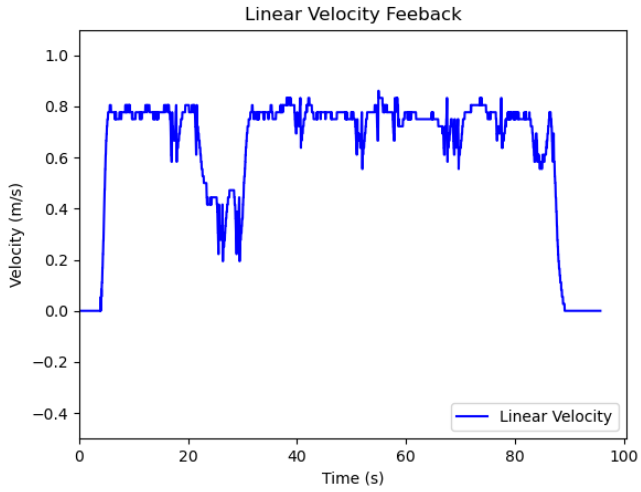
Run 3



Run 4



Run 5



Final Controller Parameters

MPPI Controller

Parameter	Value
odom_topic	/wheel_odom
GoalChecker.plugin	nav2_controller::SimpleGoalChecker
GoalChecker.xy_goal_tolerance	0.5
GoalChecker.yaw_goal_tolerance	0.5
GoalChecker.stateful	TRUE
controller_frequency	20
FollowPath.plugin	nav2_mppi_controller::MPPIController
FollowPath.time_steps	50
FollowPath.model_dt	0.1
FollowPath.batch_size	3000
FollowPath.vx_std	0.2
FollowPath.vy_std	0
FollowPath.wz_std	0.3
FollowPath.vx_max	1
FollowPath.vx_min	-1
FollowPath.vy_max	0
FollowPath.wz_max	0.7
FollowPath.wz_min	-0.7
FollowPath.ax_max	1
FollowPath.ax_min	-1
FollowPath.iteration_count	1
FollowPath.prune_distance	5
FollowPath.transform_tolerance	0.2
FollowPath.temperature	0.3
FollowPath.gamma	0.01
FollowPath.motion_model	Ackermann
FollowPath.visualize	TRUE
FollowPath.TrajectoryVisualizer.trajectory_step	10
FollowPath.TrajectoryVisualizer.time_step	49
FollowPath.AckermannConstraints.min_turning_r	0.2

FollowPath.critics	['ConstraintCritic', 'ObstaclesCritic', 'GoalCritic', 'GoalAngleCritic', 'PreferForwardCritic', 'VelocityDeadbandCritic', 'PathFollowCritic']
FollowPath.ConstraintCritic.enabled	TRUE
FollowPath.ConstraintCritic.cost_power	1
FollowPath.ConstraintCritic.cost_weight	4
FollowPath.GoalCritic.enabled	TRUE
FollowPath.GoalCritic.cost_power	1
FollowPath.GoalCritic.cost_weight	10
FollowPath.GoalCritic.threshold_to_consider	4
FollowPath.GoalAngleCritic.enabled	TRUE
FollowPath.GoalAngleCritic.cost_power	1
FollowPath.GoalAngleCritic.cost_weight	1
FollowPath.GoalAngleCritic.threshold_to_consider	0.5
FollowPath.PreferForwardCritic.enabled	TRUE
FollowPath.PreferForwardCritic.cost_power	1
FollowPath.PreferForwardCritic.cost_weight	20
FollowPath.PreferForwardCritic.threshold_to_consider	0.5
FollowPath.ObstaclesCritic.enabled	TRUE
FollowPath.ObstaclesCritic.cost_power	1
FollowPath.ObstaclesCritic.repulsion_weight	25
FollowPath.ObstaclesCritic.critical_weight	30
FollowPath.ObstaclesCritic.consider_footprint	TRUE
FollowPath.ObstaclesCritic.collision_cost	10000
FollowPath.ObstaclesCritic.collision_margin_distance	1
FollowPath.ObstaclesCritic.near_goal_distance	0
FollowPath.PathAlignCritic.enabled	FALSE
FollowPath.PathAlignCritic.cost_power	1
FollowPath.PathAlignCritic.cost_weight	2
FollowPath.PathAlignCritic.max_path_occupancy_ratio	0.05
FollowPath.PathAlignCritic.trajectory_point_step	1
FollowPath.PathAlignCritic.threshold_to_consider	0.5
FollowPath.PathAlignCritic.offset_from_furthest	6
FollowPath.PathAlignCritic.use_path_orientations	FALSE
FollowPath.PathFollowCritic.enabled	TRUE
FollowPath.PathFollowCritic.cost_power	1
FollowPath.PathFollowCritic.cost_weight	2
FollowPath.PathFollowCritic.offset_from_furthest	6
FollowPath.PathFollowCritic.threshold_to_consider	5
FollowPath.PathAngleCritic.enabled	FALSE
FollowPath.PathAngleCritic.cost_power	1
FollowPath.PathAngleCritic.cost_weight	1
FollowPath.PathAngleCritic.offset_from_furthest	4
FollowPath.PathAngleCritic.threshold_to_consider	0.5
FollowPath.PathAngleCritic.max_angle_to_furthest	1
FollowPath.PathAngleCritic.forward_preference	TRUE
FollowPath.VelocityDeadbandCritic.enabled	TRUE

FollowPath.VelocityDeadbandCritic.cost_power	1
FollowPath.VelocityDeadbandCritic.cost_weight	8
FollowPath.VelocityDeadbandCritic.deadband_velocities	[0.15, 0.0, 0.0]

RPP

Parameter	Value
use_sim_time	FALSE
controller_frequency	20
min_x_velocity_threshold	0.0001
min_y_velocity_threshold	0
min_theta_velocity_threshold	0.001
failure_tolerance	0.3
progress_checker_plugin	progress_checker
goal_checker_plugins	['general_goal_checker']
controller_plugins	['FollowPath']
progress_checker.plugin	nav2_controller::SimpleProgressChecker
progress_checker.required_movement_radius	0.5
progress_checker.movement_time_allowance	10
general_goal_checker.plugin	nav2_controller::SimpleGoalChecker
general_goal_checker.xy_goal_tolerance	0.25
general_goal_checker.yaw_goal_tolerance	0.25
FollowPath.plugin	nav2_regulated_pure_pursuit_controller::RegulatedPurePursuitController
FollowPath.desired_linear_vel	0.8
FollowPath.lookahead_dist	2
FollowPath.min_lookahead_dist	1
FollowPath.max_lookahead_dist	3
FollowPath.lookahead_time	0.5
FollowPath.rotate_to_heading_angular_vel	1.8
FollowPath.transform_tolerance	0.1
FollowPath.use_velocity_scaled_lookahead_dist	TRUE
FollowPath.min_approach_linear_velocity	0.3
FollowPath.approach_velocity_scaling_dist	0.6
FollowPath.use_collision_detection	FALSE
FollowPath.max_allowed_time_to_collision_up_to_carrot	1
FollowPath.use_regulated_linear_velocity_scaling	TRUE
FollowPath.use_fixed_curvature_lookahead	TRUE
FollowPath.curvature_lookahead_dist	0.8
FollowPath.use_cost_regulated_linear_velocity_scaling	FALSE
FollowPath.cost_scaling_dist	3
FollowPath.cost_scaling_gain	0.6
FollowPath.regulated_linear_scaling_min_radius	2
FollowPath.regulated_linear_scaling_min_speed	0.5
FollowPath.use_rotate_to_heading	TRUE
FollowPath.allow_reversing	TRUE
FollowPath.rotate_to_heading_min_angle	0.785
FollowPath.stateful	TRUE

FollowPath.max_linear_accel	1
FollowPath.max_angular_accel	1

Results Software

Static Obstacle Test Trajectory Graphing Code

```
import sys
from nav_msgs.msg import OccupancyGrid
from geometry_msgs.msg import PoseWithCovarianceStamped
from matplotlib import pyplot as plt
import numpy as np
import os
import re
from utils import extract_rosbag_data

x_min = 300
x_max = 600
y_min = 175
y_max = 280

def world_to_map(info, x, y):
    mx = int((x - info.origin.position.x) / info.resolution) - x_min
    my = int((y - info.origin.position.y) / info.resolution) - y_min
    return mx, my

def graph_trajectory(rosbag_file, graph_filename):
    costmap_times, costmap_msgs = extract_rosbag_data(rosbag_file, "/map",
OccupancyGrid)
    pose_times, pose_msgs = extract_rosbag_data(rosbag_file, "/pose",
PoseWithCovarianceStamped)

    if not costmap_msgs or not pose_msgs:
        print("Skipping file")
        return

    poses = [(i.pose.pose.position.x, i.pose.pose.position.y) for i in pose_msgs]

    first_costmap = costmap_msgs[0]
    grid = np.array(first_costmap.data,
dtype=np.int8).reshape((first_costmap.info.height, first_costmap.info.width))
    grid = grid[y_min:y_max, x_min:x_max]

    info = first_costmap.info

    plt.figure(figsize=(8, 8))
    plt.imshow(grid, cmap='binary', origin='lower')

    goal_x, goal_y = world_to_map(info, 60, 0.5)
    xs, ys = [], []
    for (x, y) in poses:
        mx, my = world_to_map(info, x, y)
        xs.append(mx)
        ys.append(my)
    plt.plot(xs, ys, 'b-', linewidth=2, label="Robot pose")
    plt.plot(82, 63, 'ro', label='Obstacle')
    plt.plot(goal_x, goal_y, 'go', label='Goal pose')

    plt.title("Robot Pose on Map")
```

```
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.legend(loc="lower right")
    plt.savefig(graph_filename, dpi=300)
    plt.close()

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 graph_obstacles_trajectory.py <rosbag_file>")
        sys.exit(1)

    dir = sys.argv[1]

    for dirpath, _, filenames in os.walk(dir):
        for filename in filenames:
            if filename.lower().endswith(".db3"):  # Check if it's a file (not a
folder)
                rosbag_file = os.path.join(dirpath, filename)
                match = re.search(r"rosbag2_(\d{4}_\d{2}_\d{2}-\d{2}_\d{2}_\d{2})",
rosbag_file)
                if match:
                    graph_filename = dirpath + "/costmap_" + match.group(1)
                    print("Processing " + rosbag_file + "...")
                    graph_trajectory(rosbag_file, graph_filename)
```

Driving Test Trajectory Graphing Code

```
import sys
from nav_msgs.msg import OccupancyGrid
from geometry_msgs.msg import PoseWithCovarianceStamped
from matplotlib import pyplot as plt
import numpy as np
import os
import re
from utils import extract_rosbag_data

x_min = 200
x_max = 500
y_min = 200
y_max = 400

def world_to_map(info, x, y):
    mx = int((x - info.origin.position.x) / info.resolution) - x_min
    my = int((y - info.origin.position.y) / info.resolution) - y_min
    return mx, my

def graph_trajectory(rosbag_file, graph_filename):
    costmap_times, costmap_msgs = extract_rosbag_data(rosbag_file, "/map",
OccupancyGrid)
    pose_times, pose_msgs = extract_rosbag_data(rosbag_file, "/pose",
PoseWithCovarianceStamped)

    if not costmap_msgs or not pose_msgs:
        print("Skipping file")
        return

    poses = [(i.pose.pose.position.x, i.pose.pose.position.y) for i in pose_msgs]

    first_costmap = costmap_msgs[0]
```

```
    grid = np.array(first_costmap.data,
dtype=np.int8).reshape((first_costmap.info.height, first_costmap.info.width))
    # grid = np.flipud(grid)
    grid = grid[y_min:y_max, x_min:x_max]

    info = first_costmap.info

    plt.figure(figsize=(8, 8))
    plt.imshow(grid, cmap='binary', origin='lower')

    xs, ys = [], []
    for (x, y) in poses:
        mx, my = world_to_map(info, x, y)
        xs.append(mx)
        ys.append(my)

    goal_x, goal_y = world_to_map(info, 30.0, 0.0)
    plt.plot(xs, ys, 'b-', linewidth=2, label="Robot pose")
    plt.plot(goal_x, goal_y, 'go', label="Goal pose")

    plt.title("Robot Pose on Map")
    plt.xlabel("X")
    plt.ylabel("Y")

    plt.legend(loc="lower right")
    plt.savefig(graph_filename, dpi=300)
    plt.close()

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 graph_driving_trajectory.py <rosbag_file>")
        sys.exit(1)

    dir = sys.argv[1]

    for dirpath, _, filenames in os.walk(dir):
        for filename in filenames:
            if filename.lower().endswith(".db3"):  # Check if it's a file (not a
folder)
                rosbag_file = os.path.join(dirpath, filename)
                match = re.search(r"rosbag2_(\d{4}_\d{2}_\d{2}-\d{2}_\d{2}_\d{2})",
rosbag_file)
                if match:
                    graph_filename = dirpath + "/costmap_" + match.group(1)
                    print("Processing " + rosbag_file + "...")
                    graph_trajectory(rosbag_file, graph_filename)
```

Velocity Graphing Code

```
import sys
from geometry_msgs.msg import TwistStamped
from matplotlib import pyplot as plt
import numpy as np
import os
import re
from utils import extract_rosbag_data, plottable_times
import yaml

def calc_average_speed(speed_fb_msgs):
```

```
    speeds = [k.twist.linear.x for k in speed_fb_msgs]
    if not 0 in speeds:
        print("Skipping file")
        return
    non_zero_speeds = [speed for speed in speeds if speed != 0]
    average_speed = np.mean(non_zero_speeds)
    return average_speed

def graph_speeds(rosbag_file, graph_filename):
    speed_fb_times, speed_fb_msgs = extract_rosbag_data(rosbag_file, "/can_twist",
TwistStamped)

    if not speed_fb_times:
        print("Skipping file")
        return

    average_speed = calc_average_speed(speed_fb_msgs)
    print("Average Speed:", average_speed)

    plt.plot(plottable_times(speed_fb_times), [i.twist.linear.x for i in
speed_fb_msgs], 'b-', label="Linear Velocity")
    plt.xlabel('Time (s)')
    plt.ylabel('Velocity (m/s)')
    plt.title(f"Linear Velocity Feeback")
    plt.legend(loc="lower right")
    plt.xlim(0)
    plt.ylim(bottom=-0.5, top=1.1)
    plt.savefig(graph_filename)
    plt.close()

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 graph_speed_fb.py <rosbag_file>")
        sys.exit(1)

    dir = sys.argv[1]

    for dirpath, _, filenames in os.walk(dir):
        for filename in filenames:
            if filename.lower().endswith(".db3"):  # Check if it's a file (not a
folder)
                rosbag_file = os.path.join(dirpath, filename)
                match = re.search(r"rosbag2_(\d{4}_\d{2}_\d{2}-\d{2}_\d{2}_\d{2})",
rosbag_file)
                if match:
                    graph_filename = dirpath + "/velocities_fb_" + match.group(1)
                    print("Processing " + rosbag_file + "...")
                    graph_speeds(rosbag_file, graph_filename)
```

Static Obstacle Test Turning Distance Calculation Code

```
import sys
from geometry_msgs.msg import TwistStamped
from sensor_msgs.msg import LaserScan
from matplotlib import pyplot as plt
import numpy as np
import os
import re
from utils import extract_rosbag_data
import yaml
import math
```

```
def calc_dist_from_obstacle_when_turning_feedback(speed_fb_times, speed_fb_msgs,
lidar_times, lidar_msgs):
    angular_velocities = get_angular_velocities(speed_fb_msgs)
    lidar_range_times, lidar_ranges = get_lidar_ranges(lidar_times, lidar_msgs)

    yaw_threshold = 0.3
    min_duration = 1.0
    dt = np.median(np.diff(speed_fb_times)/1e9)
    min_samples = int(np.ceil(min_duration/ dt))
    print(min_samples)

    above = np.abs(angular_velocities) > yaw_threshold
    onset_idx = None
    for i in range(len(above) - min_samples + 1):
        if np.all(above[i:i+min_samples]):
            onset_idx = i
            break

    if onset_idx is None:
        print("No turn detected")
        return 0, 0

    turn_time = speed_fb_times[onset_idx]
    turn_time_rel = (turn_time - lidar_range_times[0]) / 1e9
    print(f"Detected turn start at t = {turn_time_rel:.3f} s")

    closest_time = math.inf
    for i in range(len(lidar_range_times)):
        time_diff = abs(lidar_range_times[i] - turn_time)
        if time_diff < closest_time:
            closest_time = time_diff
            lidar_at_turn = lidar_ranges[i]
            time_of_lidar = lidar_range_times[i]

    print(f"Distance from obstacle before turn: {lidar_at_turn} at {(time_of_lidar -
lidar_range_times[0])/1e9}")
    return time_of_lidar, lidar_at_turn

def get_lidar_ranges(lidar_times, lidar_msgs):
    lidar_ranges = []
    lidar_range_times = []

    lidar_index_min = 270
    lidar_index_max = 450
    for i in range(len(lidar_msgs)):
        front_lidar_ranges = lidar_msgs[i].ranges[lidar_index_min:lidar_index_max]
        min_front_lidar_range = min(front_lidar_ranges)
        if min_front_lidar_range < math.inf:
            lidar_ranges.append(min_front_lidar_range - 1.875)
            lidar_range_times.append(lidar_times[i])

    print("Initial obstacle distance:", lidar_ranges[0])

    return lidar_range_times, lidar_ranges


def get_angular_velocities(speed_fb_msgs):
    return [i.twist.angular.z for i in speed_fb_msgs]

def get_turning_distance_fb(rosbag_file):
```

```
    lidar_times, lidar_msgs = extract_rosbag_data(rosbag_file,
"/lidar/localisation_merged/scan", LaserScan)
    speed_fb_times, speed_fb_msgs = extract_rosbag_data(rosbag_file, "/can_twist",
TwistStamped)

    if not lidar_times or not speed_fb_times:
        print("Skipping file")
        return

    time_of_lidar, lidar_at_turn =
calc_dist_from_obstacle_when_turning_feedback(speed_fb_times, speed_fb_msgs,
lidar_times, lidar_msgs)
    print("Turning distance from obstacle:", lidar_at_turn)

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 graph_turning_obstacle_distance_feedback.py
<rosbag_file>")
        sys.exit(1)

    dir = sys.argv[1]

    for dirpath, _, filenames in os.walk(dir):
        for filename in filenames:
            if filename.lower().endswith(".db3"):  # Check if it's a file (not a
folder)
                rosbag_file = os.path.join(dirpath, filename)
                match = re.search(r"rosbag2_(\d{4}_\d{2}_\d{2}-\d{2}_\d{2}_\d{2})",
rosbag_file)
                if match:
                    print("Processing " + rosbag_file + "...")
                    get_turning_distance_fb(rosbag_file)
```

Dynamic Obstacle Test Turning Distance Calculation Code

```
import sys
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
from matplotlib import pyplot as plt
import numpy as np
import os
import re
from utils import extract_rosbag_data
import yaml
import math

def calc_dist_from_obstacle_when_turning(cmd_vel_times, cmd_vel_msgs, lidar_times,
lidar_msgs):
    angular_velocities = get_angular_velocities(cmd_vel_msgs)
    lidar_range_times, lidar_ranges = get_lidar_ranges(lidar_times, lidar_msgs)

    yaw_threshold = 0.2
    min_duration = 0.2
    dt = np.median(np.diff(cmd_vel_times))
    min_samples = int(np.ceil(min_duration/ dt))

    above = np.abs(angular_velocities) > yaw_threshold
    onset_idx = None
    for i in range(len(above) - min_samples + 1):
```

```
        if np.all(above[i:i+min_samples]):
            onset_idx = i
            break

    if onset_idx is None:
        print("No turn detected")
        return 0, 0

    turn_time = cmd_vel_times[onset_idx]
    turn_time_rel = (turn_time - lidar_range_times[0]) / 1e9
    print(f"Detected turn start at t = {turn_time_rel:.3f} s")

    closest_time = math.inf
    for i in range(len(lidar_range_times)):
        time_diff = abs(lidar_range_times[i] - turn_time)
        if time_diff < closest_time:
            closest_time = time_diff
            lidar_at_turn = lidar_ranges[i]
            time_of_lidar = lidar_range_times[i]

    print(f"Distance from obstacle before turn: {lidar_at_turn} at {(time_of_lidar -
lidar_range_times[0])/1e9}")
    return time_of_lidar, lidar_at_turn

def get_lidar_ranges(lidar_times, lidar_msgs):
    lidar_ranges = []
    lidar_range_times = []

    lidar_index_min = 315
    lidar_index_max = 405
    for i in range(len(lidar_msgs)):
        front_lidar_ranges = lidar_msgs[i].ranges[lidar_index_min:lidar_index_max]
        min_front_lidar_range = min(front_lidar_ranges)
        if min_front_lidar_range < math.inf:
            lidar_ranges.append(min_front_lidar_range - 1.875)
            lidar_range_times.append(lidar_times[i])

    print("Initial obstacle distance:", lidar_ranges[0])

    return lidar_range_times, lidar_ranges


def get_angular_velocities(cmd_vel_msgs):
    return [i.angular.z for i in cmd_vel_msgs]

def get_turning_distance(rosbag_file):
    lidar_times, lidar_msgs = extract_rosbag_data(rosbag_file,
"/lidar/localisation_merged/scan", LaserScan)
    cmd_vel_times, cmd_vel_msgs = extract_rosbag_data(rosbag_file, "/cmd_vel_nav",
Twist)

    if not lidar_times or not cmd_vel_times:
        print("Skipping file")
        return

    time_of_lidar, lidar_at_turn = calc_dist_from_obstacle_when_turning(cmd_vel_times,
cmd_vel_msgs, lidar_times, lidar_msgs)
    print("Turning distance from obstacle: ", lidar_at_turn)

if __name__ == "__main__":
    if len(sys.argv) != 2:
```

```
        print("Usage: python3 graph_turning_obstacle_distance.py <rosbag_file>")
        sys.exit(1)

    dir = sys.argv[1]

    for dirpath, _, filenames in os.walk(dir):
        for filename in filenames:
            if filename.lower().endswith(".db3"):  # Check if it's a file (not a
folder)
                rosbag_file = os.path.join(dirpath, filename)
                match = re.search(r"rosbag2_(\d{4}_\d{2}_\d{2}-\d{2}_\d{2}_\d{2})",
rosbag_file)
                if match:
                    print("Processing " + rosbag_file + "...")
                    get_turning_distance(rosbag_file)
```

Pose Graph, Path Smoothness Calculation and Path Length Calculation Code

```
import sys
from geometry_msgs.msg import PoseWithCovarianceStamped
from matplotlib import pyplot as plt
import numpy as np
import os
import re
import math
from utils import extract_rosbag_data, plottable_times

def calc_path_length(pose_msgs):
    last_pose = None
    path_length = 0
    for msg in pose_msgs:
        current_pose = [msg.pose.pose.position.x, msg.pose.pose.position.y]
        if last_pose:
            path_length += math.dist(last_pose, current_pose)
        last_pose = current_pose
    return path_length

def calc_path_displacement(pose_msgs):
    first_pose = [pose_msgs[0].pose.pose.position.x, pose_msgs[0].pose.pose.position.y]
    last_pose = [pose_msgs[-1].pose.pose.position.x, pose_msgs[-
1].pose.pose.position.y]
    path_displacement = math.dist(first_pose, last_pose)
    return path_displacement

def calc_path_smoothness(pose_msgs):
    if len(pose_msgs) < 3:
        return 0.0

    poses = [[msg.pose.pose.position.x, msg.pose.pose.position.y] for msg in pose_msgs
if msg.pose.pose.position.x > 0.0]
    distances = [0.0]
    headings = []
    window_size = 1

    # Compute cumulative distance along path
    for i in range(1, len(poses)):
        dx = poses[i][0] - poses[i-1][0]
        dy = poses[i][1] - poses[i-1][1]
        ds = math.sqrt(dx*dx + dy*dy)
        distances.append(distances[-1] + ds)

    # Sample heading every ~window_size
    sample_points = []
    for d in range(0, int(distances[-1] // window_size)):
        target_dist = d * window_size
        # find closest pose to this distance
        idx = min(range(len(distances)), key=lambda i: abs(distances[i]-target_dist))
        if idx > 0:
            dx = poses[idx][0] - poses[idx-1][0]
            dy = poses[idx][1] - poses[idx-1][1]
            headings.append(math.atan2(dy, dx))
            sample_points.append(poses[idx])

    smoothness = 0.0
    for i in range(1, len(headings)):
        dtheta = headings[i] - headings[i-1]
        dtheta = math.atan2(math.sin(dtheta), math.cos(dtheta))
```

```
        smoothness += dtheta**2

    return smoothness

def graph_pose(pose_times, pose_msgs, graph_filename):
    if not pose_times:
        print("Skipping file")
        return

    plt.plot([i.pose.pose.position.x for i in pose_msgs], [j.pose.pose.position.y for j
in pose_msgs], 'black', label="Pose")
    plt.xlabel('x (m)')
    plt.ylabel('y (m)')
    plt.legend(loc="lower right")
    plt.ylim(bottom=-18, top=3)
    plt.savefig(graph_filename)
    plt.close()

def main():
    if len(sys.argv) != 2:
        print("Usage: python3 graph_pose.py <rosbag_file>")
        sys.exit(1)

    dir = sys.argv[1]

    for dirpath, _, filenames in os.walk(dir):
        for filename in filenames:
            if filename.lower().endswith(".db3"):  # Check if it's a file (not a
folder)
                rosbag_file = os.path.join(dirpath, filename)
                match = re.search(r"rosbag2_(\d{4}_\d{2}_\d{2}-\d{2}_\d{2}_\d{2})",
rosbag_file)
                if match:
                    graph_filename = dirpath + "/poses_" + match.group(1)
                    print("Processing " + rosbag_file + "...")
                    pose_times, pose_msgs = extract_rosbag_data(rosbag_file, "/pose",
PoseWithCovarianceStamped)
                    graph_pose(pose_times, pose_msgs, graph_filename)
                    print(f"Path length = {calc_path_length(pose_msgs)}")
                    print(f"Path smoothness = {calc_path_smoothness(pose_msgs)}")
                    print(f"Path displacement = {calc_path_displacement(pose_msgs)}")

if __name__ == "__main__":
    main()
```

Literature Review

Several studies present different formulations of the MPPI cost function. In [1], the cost function is comprised of four parts: speed, track position, sideslip angle and control cost. The speed cost penalises control inputs that cause deviations from the target velocity. The track position cost discourages trajectories that lead the vehicle off track. The sideslip angle cost improves vehicle stability by penalising excessive lateral movement. Finally, the control cost helps to smooth control actions by penalising large control inputs. These components are aggregated as a weighted sum to yield the final cost.

In [2], a similar cost function is used, but with different components: distance, target progress, yaw alignment, speed, and collision avoidance. The distance cost is the Euclidean distance between the

vehicle and the closest waypoint. The target cost “penalises trajectories that do not get closer to the target point” to prevent the vehicle from moving backwards [2]. Yaw cost, defined as the deviation between current and reference yaw, is used specifically in this paper for a lane merging scenario to keep the vehicle “sufficiently parallel to the target lane” [2]. This is likely unnecessary for campus driving applications where the target orientation may be different from the current orientation. The speed cost is calculated the same as in [1]. Finally, the safe distance cost is formulated so that “the cost is only considered if there is a collision” otherwise it is set to zero [2].

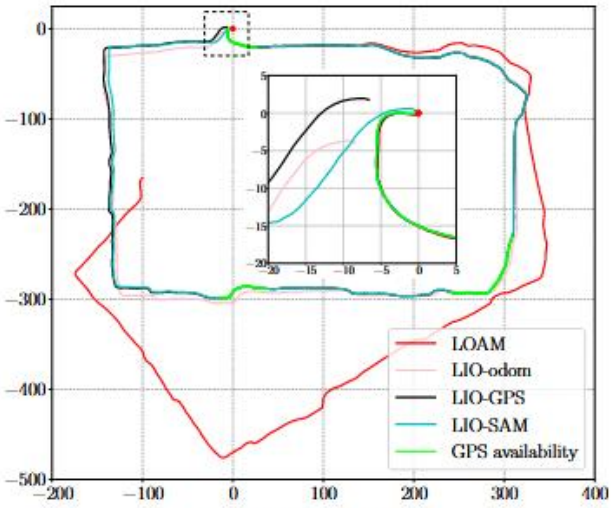
In, [3], the U-MPPI control strategy is proposed. This approach introduces a risk-sensitive cost function which incorporates uncertainty during evaluation to address the lack of reliable safety features in other MPPI models. The risk-cost is evaluated using the “log-expectation of the exponentiated quadratic cost” [3]. This cost function can be configured to increase risk-seeking behaviour which can encourage autonomous vehicle to choose trajectories with higher rewards but more uncertainty. Safety is a critical concern for autonomously driving on the UWA campus, especially since the shuttle will share a path with pedestrians. Incorporating a similar cost function could lead to increased performance.

Additionally, [2] supports the hypothesis that MPPI-based control for autonomous driving is more reliable than machine learning techniques. The study argues that “solutions based on [machine learning] methods are unlikely to meet the safety requirements of autonomous vehicles” since outputs are difficult to validate and interpret [2].

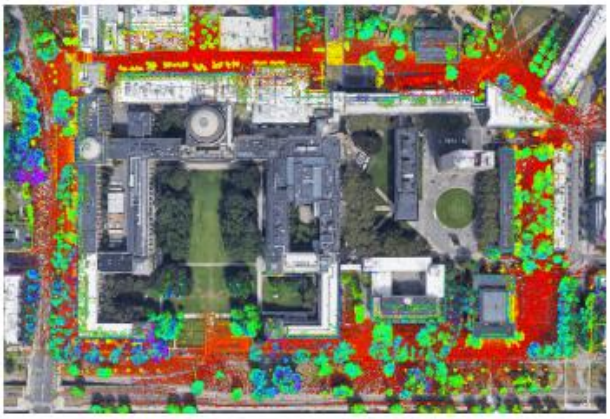
Alongside MPPI, this project aims to develop a robust SLAM system which, considering both Lidar and vision methods.

Lidar Inertial Odometry via Smoothing and Mapping (LIO-SAM) is a “tightly-coupled lidar inertial odometry framework” [4]. LIO-SAM uses “local sliding window-based scan-matching” which results in better performance as opposed to scan-matching on a global scale [4]. [4] makes reference to vision-based SLAM methods but suggests they are unreliable due to their sensitivity to ‘initialization, illumination, and range’ unlike Lidar-based methods which are ‘invariant to illumination change’ [4].

LIO-SAM's suitability is further supported by its successful deployment in a comparable university campus setting, where buildings and trees impeded GPS reception. The results of this experiment can be seen in figure 2. Given the similar environmental constraints at UWA, LIO-SAM emerges as a promising candidate for SLAM implementation.



(a) Trajectory comparison



(b) LIO-SAM map aligned with Google Earth

Figure 27 Results of LIO-SAM and other Lidar Slam methods using a Campus dataset [1]

Alternatively, ORB-SLAM proposes a monocular vision-based SLAM method. This method identifies ORB (Oriented FAST and Rotated BRIEF) features extracted from keyframes [5]. This method has demonstrated effective mapping and localization on the New College dataset, as shown

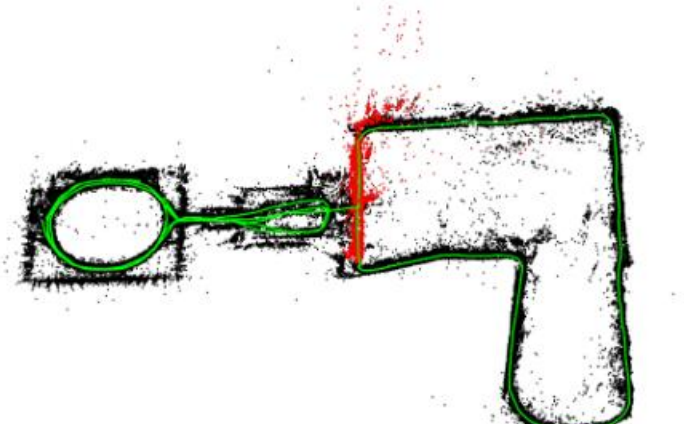


Figure 28 Map built using the New College lidar and vision data set and ORB-SLAM

in figure 3, which is similar to UWA's campus environment [5]. However, this dataset was gathered "in the afternoon" of a day in "early November", indicating that different lighting and weather conditions have not been taken into account [6]. As such, ORB-SLAM will be treated as an optional extension to this project rather than a primary SLAM solution, given its sensitivity to environmental changes.

In summary, this literature review highlights the strengths and weaknesses of existing MPPI control strategies and SLAM systems. The insights gained support the development of a risk-sensitive MPPI controller paired with a lidar-based SLAM system.

[1] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," 2016.

[2] M. Testouri, G. Elghazaly, and R. Frank, "Towards a Safe Real-Time Motion Planning Framework for Autonomous Driving Systems: A Model Predictive Path Integral Approach," 2023.

[3] I. S. Mohamed, J. Xu, G. S. Sukhatme, and L. Liu, "Toward Efficient MPPI Trajectory Generation With Unscented Guidance: U-MPPI Control Strategy," *IEEE Transactions on Robotics,* vol. 41, pp. 1172-1192, 2025, doi: 10.1109/TRO.2025.3526078.

[4] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," 2020.

[5] R. Mur-Artal, J. Montiel, and J. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics,* vol. 31, pp. 1147 - 1163, 10 2015, doi: 10.1109/TRO.2015.2463671.

[6] M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman, "The New College Vision and Laser Data Set," *I. J. Robotic Res.,* vol. 28, pp. 595-599, 05 2009, doi: 10.1177/0278364909103911.