



Final Year Project Report

Motion Planning of a Mobile Robot Using Model Predictive Control

Aray Zhaisanbek, Sanzhar Kuandyk

A thesis submitted in part fulfilment of the degree of

BSc in Robotics and Mechatronics

Supervisor: Dr. Matteo Rubagotti

Co-supervisors: Anuar Nurlybayev, Aruzhan Karekeshova

Department of Robotics and Mechatronics

Nazarbayev University

May 4, 2026

Table of Contents

Abstract	3
1 Introduction	5
2 Background Research	6
2.1 Motion Planning for Mobile Robots	6
2.2 Model Predictive Control (MPC)	6
2.3 SLAM and Navigation Systems in ROS 2	6
3 Literature Review	7
3.1 MPC	7
3.2 DRL	9
3.3 Discussion	10
4 Methodology	12
4.1 Phase 1: Knowledge Acquisition	12
4.2 Phase 2: Code Review and Refactoring	12
4.3 Phase 3: Simulation Environment Development	12
4.4 Phase 4: Real Robot Setup	13
4.5 Ethics Portfolio	14
5 Learning Portfolio	15
5.1 Key learning experiences	15
5.2 Skill acquisition	16
5.3 Future Learning goals	17
6 Simulation Environment and Hardware Setup	18
6.1 Simulation of the C4 Hall	18
6.2 Robot Operating System Reset	19
7 System Design and Implementation	22
7.1 Software Architecture	22
7.2 MPC Integration and Enhancements	22

8	Testing and Evaluation	24
8.1	Simulation Testing	24
8.2	Hardware Testing	25
9	Experimental Validation of Speed and Separation Monitoring (SSM)	27
9.1	Introduction to SSM	27
9.2	Experimental Methodology	27
9.3	Results and Statistical Analysis	27
9.4	Discussion of Safety Parameters	27
9.5	Implementation Recommendations	28
10	Limitations	31
11	Conclusions and Future Work	32

Abstract

Autonomous navigation in indoor environments requires robots to plan safe and efficient trajectories while sharing space with humans and obstacles. Model Predictive Control (MPC) has emerged as a powerful approach due to its ability to optimize motion over a prediction horizon while incorporating system constraints. This project focuses on developing an MPC-based navigation framework for Clearpath's Jackal robot and preparing the necessary simulation and hardware infrastructure for real-world deployment.

At the start of the project, the inherited system from a previous student group suffered from severe limitations, including hardcoded paths, outdated dependencies, and malfunctioning ROS 2 configurations. Furthermore, the robot's operating system was outdated and incompatible with required drivers. To address these issues, we performed a full OS reset, installed ROS 2 Humble, and configured LiDAR, SLAM Toolbox, and Nav2 packages. In parallel, we built a high-fidelity Gazebo simulation of the C4 block hall and refactored the inherited codebase to meet ROS 2 best practices.

The MPC controller now is fully functional and actively used during navigation. It generates smooth control commands and successfully handles dynamic constraints. Additionally, the perception system detects human positions and propagates them to the MPC controller, enabling real-time trajectory adaptation.

With The MPC ready to be implemented on a real robot and maps of experimentation environment available, experimentation can be carried out the next year.

Acknowledgments

We would like to express our sincere gratitude to our supervisor, Dr. Matteo Rubagotti, whose continuous guidance and expertise helped shape this project. We also thank our co-supervisors, Anuar Nurlybayev and Aruzhan Karekeshova, for their technical support and constructive feedback throughout the semester.

We extend our appreciation to 3rd year Robotics Engineering student Yermek Zhakupov, who assisted us with the setup of the Jackal robot. Finally, we thank our classmates and friends for their encouragement and valuable discussions.

Chapter 1: Introduction

Autonomous mobile robots are increasingly deployed in environments such as hospitals, universities, warehouses, and industrial facilities. In such environments, motion planning plays a critical role in ensuring safe, predictable, and efficient navigation among static infrastructure and moving people. Traditional navigation approaches often struggle with dynamic obstacles, system constraints, and the need for smooth, optimized trajectories.

Model Predictive Control (MPC) offers a powerful solution because it predicts future robot states, optimizes motion over a finite horizon, and enforces constraints such as velocity limits and collision avoidance. This makes MPC highly suitable for robots operating in crowded indoor spaces where safety and smooth motion are essential.

This project aims to implement and evaluate an MPC-based motion-planning system for the Clearpath Jackal robot within a realistic indoor environment. The project builds upon the partially completed work of a previous student team. However, their system suffered from significant issues: hardcoded file paths, outdated ROS 2 packages, broken launch files, and a malfunctioning robot OS. These problems prevented meaningful progress toward MPC implementation.

To overcome these limitations, we focused on:

- resetting and reinstalling the robot's operating system with ROS 2 Humble,
- developing Gazebo simulation of the C4 corridor,
- refactoring the inherited simulation and control code,
- installing and configuring essential navigation packages.

The remainder of this report is organized as follows: Chapter 2 reviews relevant research on motion planning and MPC. Chapter 3 presents the methodology adopted throughout the semester. Chapter 4 describes the simulation environment and hardware setup. Chapter 5 details the system design and implementation. Chapter 6 reports testing and evaluation results. Chapter 7 concludes the report and outlines future work.

Chapter 2: **Background Research**

2.1 Motion Planning for Mobile Robots

Mobile robot navigation involves determining a collision-free path from the robot's current position to a goal position. Traditional approaches include graph-based planners (A^* , D^*) and sampling-based planners (RRT, PRM). However, these methods often assume static environments or do not fully incorporate robot dynamics.

2.2 Model Predictive Control (MPC)

Model Predictive Control is an optimization-based method widely used in robotics due to its ability to predict future system behavior and incorporate constraints directly into the control formulation. At each control step, MPC solves an optimization problem that minimizes a cost function penalizing deviation from the desired trajectory and control effort. Constraints on velocity, acceleration, and obstacle avoidance can be integrated explicitly.

Recent research highlights its success in mobile robot navigation, especially when combined with collision avoidance frameworks such as the Velocity Obstacle paradigm. MPC is computationally intensive, but advances in numerical optimization and reduced prediction horizons enable real-time performance on embedded systems.

2.3 SLAM and Navigation Systems in ROS 2

Simultaneous Localization and Mapping (SLAM) enables robots to construct an environment map while estimating their pose. Tools such as SLAM Toolbox and Nav2 provide robust implementations for 2D LiDAR-based mapping and navigation. These form the backbone of modern mobile robot systems and serve as the baseline framework for integrating MPC.

Chapter 3: Literature Review

3.1 MPC

According to paper [10] Model Predictive Control is an advanced control strategy used in various applications, including robotics, to manage the behavior of dynamic systems. MPC operates by predicting the future behavior of a system over a defined prediction horizon. It uses a model of the system to forecast future states based on current inputs and states. The goal is to determine the optimal control inputs that will drive the system towards a desired state or trajectory while adhering to constraints. In MPC, the state of the system is represented by a state vector $x \in \mathbb{R}^n$, and the control inputs are denoted by $u(t)$. The control inputs are chosen from a convex set of input constraints U . A crucial aspect of MPC is the formulation of a cost function that quantifies the performance of the control inputs. This cost function typically includes terms that penalize deviations from desired states and excessive control efforts. For instance, a common form is $L(x,u)=x^T Q u + u^T R u$, where Q and R are diagonal matrices that weigh the importance of state performance and actuator effects, respectively. At each sampling time, MPC solves an optimization problem to find the control inputs that minimize the cost function while satisfying the system's constraints. This process is repeated at each time step, allowing the controller to adapt to changes in the system or environment.

To ensure stability, MPC often incorporates a terminal cost function $F(x(T))$ that penalizes the terminal state, helping to keep the system within a stable region. In the context of autonomous mobile robots, MPC can be combined with other techniques, such as artificial potential fields, to handle trajectory tracking and obstacle avoidance. The robot can be guided towards a virtual goal while avoiding collisions, demonstrating the versatility of MPC in complex environments.

MPC offers several compelling advantages that make it particularly suitable for motion planning in mobile robots. One of its most significant strengths lies in its predictive capability: by utilizing a system model to forecast future states, MPC can anticipate and respond to changes in system dynamics or external disturbances, thereby optimizing performance over a specified time horizon [10]. This forward-looking nature allows MPC to calculate control actions that are not only responsive but also proactive, enabling more efficient motion planning even in cluttered or densely populated environments [4]. Furthermore, MPC is highly flexible—it can be tailored to a wide array of systems and applications by customizing cost functions to meet specific performance goals, whether for trajectory tracking or obstacle avoidance [10]. In addition, MPC supports real-time control, making it ideal for dynamic scenarios where rapid and reliable decision-making is critical [14]. Unlike more simplistic planning approaches, MPC can seamlessly integrate both the dynamic and kinematic constraints of mobile robots into its computations, resulting in more accurate and reliable motion trajectories [14]. In addition to this MPC approach enables simultaneous control of state variables and trajectory planning between obstacles. This integration allows the robot to navigate efficiently while maintaining stability in the presence of both static and moving obstacles[4].

Despite its many advantages, Model Predictive Control (MPC) presents several challenges that have been actively addressed in recent research. One of the primary concerns is its computational complexity. Since MPC requires solving an optimization problem at every

control step, it can be demanding in terms of processing power—especially in systems with many states or when real-time decision-making is required, such as in autonomous mobile robots [7][10]. To mitigate this, several studies propose innovative approaches. For instance, [7] introduces a zero-order robust optimization (zoRO) scheme that avoids the need for ellipsoidal sets in the optimization process, thus reducing computational demands. Similarly, the use of Nonlinear MPC (NMPC) in [4] reformulates the problem within a nonlinear framework, which allows for better handling of system dynamics without increasing computational complexity. Paper [10] further improves efficiency by proposing an explicit MPC solution in the form of a look-up table. This method allows pre-computed optimal control actions to be stored and accessed in real-time, significantly reducing the processing load. Additionally, the use of a discrete MPC framework with a structured prediction horizon enables efficient updating of state measurements and control inputs during each sampling interval [10].

Another challenge of MPC is its sensitivity to disturbances. Since optimal control inputs often lie near the boundary of constraints, even minor disturbances can cause constraint violations, jeopardizing the robot's safety. Paper [7] addresses this by proposing a robust MPC formulation that models and propagates bounded process disturbances using ellipsoidal sets. This enhances safety by ensuring that control actions remain valid even under uncertain operating conditions.

MPC may also yield suboptimal control actions in the presence of disturbances. Although the zero-order optimization scheme proposed in [7] is inherently suboptimal, the authors demonstrate that it can still maintain convergence to the reference trajectory when obstacles are absent. This shows that, for many practical applications, maintaining smooth and safe trajectories is achievable even with simplified optimization schemes. Another limitation lies in the dependence of MPC on an accurate system model. Inaccurate models can lead to poor control performance, especially in tasks such as obstacle avoidance and path stabilization [4]. Paper [4] emphasizes this issue and proposes integrating stability and performance analyses into the control system to determine appropriate parameter values, thus improving robustness despite potential model inaccuracies.

Tuning the parameters of an MPC controller is also a nontrivial task. Incorrect tuning can cause performance degradation or even instability. The authors of [4] provide a systematic method for parameter tuning by analyzing the system's stability and performance. They establish mathematical criteria to guide the selection of optimal parameter values, which enhances both the reliability and efficiency of the control system.

Finally, handling constraints and potential deadlocks in real-time is particularly difficult for MPC in cluttered environments. In response to this, paper [14] offers an MPC-based motion planning strategy that uses the Velocity Obstacle (VO) paradigm. It detects deadlock situations by evaluating the waypoint's feasibility and adjusts the target location based on proximity to the robot, ensuring continuous progress without violating safety constraints. This approach does not require modifying the VO formulation, which maintains computational simplicity while improving trajectory feasibility and reducing conservativeness [14].

Through these varied strategies—ranging from novel optimization techniques to robust modeling and intelligent constraint handling—recent research has significantly advanced the applicability of MPC for motion planning in dynamic and complex robotic environments.

	Aspect	Type	Details	Proposed Solutions	Citation
1	Predictive Capability	Advantage	Predicts future states for proactive control	-	[8]
2	Constraint Handling	Advantage	Handles dynamic and kinematic constraints	-	[1], [8]
3	Flexibility	Advantage	Can be adapted for different robot tasks	-	[8]
4	Real-Time Applicability	Advantage	Good for fast-changing environments	-	[18]
5	Obstacle Avoidance	Advantage	Enables safe pathing with collision avoidance	Combines with potential fields or VO	[19], [4]
6	Computational Complexity	Disadvantage	Optimization at every step is demanding	zBRO [2], NMPC [4], Explicit MPC [8]	[2], [4], [8]
7	Sensitivity to Disturbances	Disadvantage	Risk of constraint violations near boundaries	Robust MPC with ellipsoids [2]	[2]
8	Suboptimal Control (Robust)	Disadvantage	Some robust MPCs are suboptimal	Still converge in ideal conditions [2]	[2]
9	Model Dependence	Disadvantage	Inaccurate models reduce control performance	Tuning with stability performance analysis [4]	[4]
10	Tuning Complexity	Disadvantage	Hard to select optimal cost weights	Systematic tuning with math tools [4]	[4]

Figure 3.1: Advantages and Disadvantages of MPC methods with proposed solutions

3.2 DRL

What is DRL?

As the gap between the simplified simulation, based on which robot operates, and actual surrounding conditions continue to widen, the working environment is becoming increasingly complex for robots. As a result, traditional path planning methods, such as Probabilistic Roadmap Method or Dynamic Window Approach, are no longer sufficient for motion planning of mobile robots. [17]

Traditional path planning methods generally focus on static objects or oversimplify the model of moving objects by considering them as stationary at particular moment of time [9]. However, the Deep Reinforcement learning (DRL) can learn to make optimal decisions without having an environmental model, thus, making it highly adaptable to different environments. [17]. Because of this, the DRL method is becoming more common for path planning of mobile robots. Through DRL methods, mobile robots have an extensive interaction with the surrounding environment. Due to this, robots, trained by DRL, do not need a map of the environment in advance, they have robust learning capabilities and lower reliance on sensor accuracy. [9].

Reinforcement Learning (RL) helps systems to learn from an interaction with a real world and then find an optimal decision-making strategy. In this method ‘agent’ is a decision-maker, while ‘environment’ is the surroundings of the mobile robot. [9]. The robot then receives a feedback signal, which comes as a reward. This process trains the algorithm, which robot then uses while operating. A mathematical framework is used (Markov Decision process), where elements are S, A, R, and P: environments State, Action taken by the agent, Reward value received by agent and state transition Probability.[9]. The state transition probability is the probability of the system to state s_{t+1} from s_t when action a_t is taken by the agent when reward r_t was obtained. Despite receiving instantaneous feedback at each step, the main goal of DRL is to optimize the overall cumulative reward over the long period of time, instead of immediate rewards.[9].

While path planning of mobile robots, devices such as sensors, cameras etc. may be utilized to establish a connection with the surrounding environment. There are key elements in DRL method: actions space, which consists of three types: discrete movement, which includes basic maneuvers such as simple turns; continuous velocity, which includes angular and linear velocities of the mobile robots; and motor speed; reward function, which trains the robot to avoid obstacles(i.e., an RL agent can obtain a positive reward when the robot is getting closer to a target or, an agent can obtain a negative reward when the robot collides with the obstacle or gets too close to it); state space, which focuses on the position of the robot in surrounding space, as well as its starting point, target and obstacles. When planning the motion of the robot, it is necessary to define the above elements. [13].

Advantages of DRL

The DRL-based learning involves real-time interaction between the agent and environment. Unlike the traditional motion planning methods, robots are not reliant on a simplified simu-

lation, which often fails to consider working conditions. Also, RL agents can be trained with a minimum amount of datasets. [11]. Since the DRL method focuses on feedback reward learning, the requirement for a simulation etc. is minimal. Moreover, DRL-based training is robust. For example, Soft Actor-Critic (SAC) algorithm has strong robustness, as it uses stochastic sampling, due to which it can easily adjust to surrounding disturbances. [15] in addition to this, SAC has strong exploration ability, as it can control the ratio of the maximum entropy to the reward, which helps an agent find an optimal solution with multiple reward functions. As a result, a mobile robot can reach the target while avoiding obstacles at the same time.

Disadvantages of DRL Traditional DRL algorithms have several issues. They can be improved by integrating multiple algorithms.

DRL-based learning can face scalability issues. For Q-Learning (RL algorithm), Q-value is calculated based on the Bellman equation. [11]. Then it is used by an agent for path planning. However, as the size of the environment increases, the memory complexity also increases. However, this issue is not the case in Deep Q-Learning, as the Q-value is predicted with a neural network, which can also be trained [5], however it also adds to the cost of DRL-based training.

Another drawback of DRL-based training is slow convergence. The convergence time is high for DRL-based path planning, as it takes a significant amount of time for the agent to learn the task. This issue is especially critical when the working space increases. To reduce it, agent can be pre-trained by a 2D simulator environment and use this training later when it is interacting with a real life environment. [11]. Moreover, Intrinsic Curiosity Module (ICM) can be used to accelerate the learning and allowing to quickly adapt to environment. [17].

Moreover, DRL method has low sample efficiency. DRL algorithms require a significant amount of training samples before learning to complete task. Usage of ICM in the LI-PPO algorithm can provide intrinsic reward, thus improving the efficiency.[17]

DRL-based training often has stability issues. Training process is often unstable, which makes it prone to policy collapse or getting stuck in local optima. [17]. The issue can be resolved by a 3D path planning in a limited observation space (LOS-PPO), however this solution is costly. Also, Long-Short Term Memory (LSTM) networks can be used to address long-term dependencies which leads to a more stable training[17].

In addition to this, there is a high complexity of training. Training the agent to avoid obstacles can be very complex, especially in a dynamic environment, where obstacle locations are not known in advance.

And finally, the cost of training, based on DRL, is usually high and requires a significant amount of computational resources. In addition to this, solving the issues mentioned above also come at high costs. However, it is addressed in [17], the proposed LI-PPO algorithm can enhance the efficiency to reduce the computational costs of DRL based training.

3.3 Discussion

The comparison between Model Predictive Control (MPC) and Deep Reinforcement Learning (DRL) shows strengths and limitations for motion planning in mobile robotics, emphasizing the importance of selection based on context in which robot is being used.

MPC demonstrates strong performance in structured and moderately dynamic environments, where an accurate system models are available. Its predictive nature and ability to handle constraints makes it useful for tracking trajectory and avoiding obstacles when system dynamics is known. Moreover, MPC's real-time decision-making and optimization-based control allow for precise and interpretable adjustments of robots behaviour during operation [10][14]. However, its reliance on model accuracy and high computational demand restricts its applications in highly unpredictable or large-scale environments. Recent studies propose several solutions to address these issues, such as explicit MPC with precomputed control laws [10], robust MPC using ellipsoidal sets [7], and nonlinear MPC formulations that capture real-world dynamics better [4]. Nevertheless, fine-tuning MPC parameters and maintaining stability under disturbances remain open challenges that require further automation and adaptation strategies.

In contrast, DRL is great in environments, where dynamics are challenging to model or always changing. Its ability to learn directly from interactions with the environment gives it superior adaptability, especially in unstructured and uncertain domains [9][17]. DRL eliminates the need for a predefined model, allowing robots to learn robust policies through trial-and-error. Algorithms like Soft Actor-Critic (SAC) have strong robustness and exploration capabilities, allowing robots to operate effectively in a surroundings with moving obstacles and noisy sensor data [15]. However, DRL faces significant challenges, including high sample inefficiency, slow convergence, and unstable training processes. These issues become more evident as the complexity of the task increases. To address them, approaches such as Intrinsic Curiosity Module (ICM), LI-PPO, and LSTM-based architectures have been proposed to improve stability and efficiency while training [17]. However, still, the training cost and resource intensity of DRL remain a critical issue to its real-time deployment. When comparing both approaches, it becomes evident that MPC offers interpretability, constraint satisfaction, and safety guarantees, making it suitable for applications requiring formal performance guarantees and deterministic behavior. DRL, on the other hand, provides flexibility and generalization in environments where model-based approaches would not be applicable. While MPC is more mature and stable, DRL is more useful in dealing with unexpected scenarios.. A promising research direction is the integration of the two: hybrid frameworks that use MPC for short-term planning and DRL for long-term decision-making could potentially deploy the strengths of both methods. For instance, DRL could be used to adaptively tune MPC parameters online or predict optimal trajectories that MPC can follow in real-time.

Further improvements in both control methods are necessary to make them more suitable for real-world mobile robots. For MPC, these include enhanced real-time solvers, adaptive model identification, and better robustness to disturbances. For DRL, future work should focus on improving sample efficiency and stability during training.

Overall, while neither approach is universally superior, understanding their complementary characteristics enables more informed selection and innovative hybrid strategies for advanced motion planning systems.

Chapter 4: Methodology

The project was completed in four major phases, combining software reconstruction, simulation development, and hardware preparation.

4.1 Phase 1: Knowledge Acquisition

We began by studying ROS 2 Humble, Nav2, TF2, RViz, and Gazebo. We also reviewed MPC literature, explored its mathematical foundation, and gained basic proficiency in C/C++ to understand and modify ROS packages.

4.2 Phase 2: Code Review and Refactoring

The inherited simulation framework contained extensive hardcoding of paths, robot namespaces, and configuration parameters. These caused build failures and prevented portability. We addressed these issues by:

- replacing hardcoded values with parameter files,
- reorganizing the workspace into a clean ROS 2 structure,
- updating outdated ROS APIs,
- modularizing launch files to improve readability and maintenance.

4.3 Phase 3: Simulation Environment Development

Using floor measurements of the C4 corridor, we created a Gazebo world that reflects real hall dimensions. The simulation integrates:

- Jackal robot model with LiDAR, IMU, and odometry plugins,
- physical properties such as lighting and friction,
- textures and geometry matching the real space.

This environment serves as a safe platform for validating navigation algorithms before deploying them to hardware. During this phase, the primary objective was to establish a stable

simulation environment using ROS 2 Humble, Gazebo, and the Clearpath simulator packages. However, multiple integration issues were encountered.

Initially, conflicts arose due to mixing apt-installed Clearpath packages located in `/opt/ros/humble` with source-built versions in the overlay workspace. This resulted in build failures and inconsistent package resolution. The issue was resolved by removing conflicting apt packages and rebuilding the workspace from source to ensure version consistency.

Further complications occurred during the `colcon build` process due to missing dependencies (e.g., `yolo_msgs`), which required restructuring the workspace and ensuring proper build order.

After resolving the simulation errors encountered during earlier stages of development, the focus of the work shifted toward enhancing the capabilities of the MPC framework. Specifically, the MPC formulation was extended to handle a greater number of dynamic constraints simultaneously, enabling more robust interaction with moving agents in the environment. In parallel, an updated formulation of the SSM parameters was incorporated into the controller, improving the system's ability to account for human-robot safety distances in dynamic scenarios.

Given these substantial modifications to both the environment and the control architecture, a comprehensive retuning of the MPC's constant parameters was performed.

As a result, the simulation environment can now be launched reliably, and full integration between the perception modules, navigation stack, and MPC controller has been achieved. The system operates in a stable manner, providing a consistent foundation for further evaluation and experimentation.

4.4 Phase 4: Real Robot Setup

Initially, the robot was fully non-operational due to outdated libraries, incompatible ROS 2 installations, broken TF frames, and missing device drivers. RViz would not run, and core sensor topics were unavailable.

To resolve this, we:

- reset the robot's Ubuntu OS,
- installed ROS 2 and Nav2,
- configured SLAM Toolbox and LiDAR drivers,
- set up DDS networking between laptop and robot.

As a result, robot can now be operated using joystick and a laptop as its hardware. In addition to this, robot can navigate in an environment with Nav2 navigation framework. Moreover, Jackal is able to draw an accurate map of a surrounding environment using SLAM (Simultaneous Localization and Mapping).

4.5 Ethics Portfolio

The goal of this project is to implement the simulated MPC in the real world, tune it by taking into consideration the real environment's conditions (specifically constraints), so that the JACKAL robot will avoid any obstacles (including dynamic ones such as humans) during its operation. Of course, therefore, the experimentation phase will involve robot-human interaction. Involving humans in an experiment raises ethical and legal considerations about physical safety, human-subject participation, and data protection. Being intended for warehouse usage, the JACKAL weighs 17 kg and can carry 20 kg, and capable of moving at 2m/s, its collision with humans might result in an injury. Resolving this issue comes at a cost of time: robot will be set not to move at its max speed near humans. The experiment will involve only people who consented to take part, and the experimentation phase will start with testing with fixed obstacles, then obstacles resembling human presence and only then with real humans. A minimum-distance abort rule will be defined before the beginning of human trials, and breach of that threshold will immediately halt experiments and trigger MPC revision. Experiments will always be conducted in presence of a supervisor who will perform an emergency stop if needed. This is consistent with the international guidelines such as ISO 13482 and ISO 15066, which are specified on reducing risk in human-robot interaction. Experiments will involve interaction with people from the lab, i.e. our capstone team members and possibly other students/Ras who work with the Jackal.

Video or motion data from the experiment may be recorded. These recordings may identify participants; therefore they should be treated as protected research data. The access to this data will be available only to the research team, and the data itself will only be used for research purposes. Storing raw video is preferred for technical completeness even though anonymized trajectories would reduce privacy risk; this trade-off is justified by restricting access and clearly informing participants of retention and purpose.

Besides ethics and legality, the project has sustainability and social responsibility implications. Development of warehouse obstacle avoiding robots may take over certain jobs but also decrease the risk of human injury or even death in a warehouse environment. From a sustainability perspective, improving robot navigation in a warehouse may improve efficiency in logistics in the long term. Conducting our research with consideration of all ethical and legal issues can help to achieve this without unnecessary harmful costs.

From an ethical point of view, the goal of the project is to avoid causing harm and consider the social implications of deploying such systems in the real world. Though these considerations do not limit results a project can achieve, they come at a cost: reducing speed of operation at the cost of time. Including ethical consideration into design methodology, we make sure that the project maintains alignment with both ethical principles and accepted safety practices in human-robot experimentation.

Chapter 5: Learning Portfolio

5.1 Key learning experiences

5.1.1 Technical challenges

During this phase of the Capstone project, we encountered several technical issues. The most challenging one was the reconstruction and stabilization of a broken robotics pipeline. The system that we received from last year's capstone project was not fully functional as it contained hardcoded file paths, outdated ROS2 packages, broken launch files and incompatible dependencies. This made us rebuild the entire ROS2 environment from scratch.

Another issue that we encountered was the instability of the ROS2 environment. Conflicts between apt installed packages and source-built workspaces, missing dependencies, package detection errors and broken environment sources caused failures during build. In addition to this, the simulation in Gazebo was faulted, as we encountered failures during controller activation (robot did not accept velocity commands), timing inconsistencies in control loops, transform synchronization errors and the most significant one: robot ignoring obstacles (passing through walls, moving unrealistically and etc.).

Issues were encountered with hardware as well. Our Jackal robot was initially unusable, as it had outdated drivers, broken TF tree, missing sensor topics and Rviz was not launching. This required us to perform full reset and reinstallation, which was a completely new experience for our team.

Integration components like Nav2, SLAM and MPC were also challenging due to real-time synchronization problems, nodes not communicating properly and etc.

5.1.2 What we didn't know initially

At the beginning of the project:

- We did not understand how ROS2 environments and overlays work
- We did not know how package conflicts (apt vs source) affect builds
- We had never performed a full robot OS reset and setup
- We did not understand controller lifecycle management in ROS2
- We did not know how to debug TF tree and timestamp mismatches
- We had no experience debugging distributed robotic systems
- We assumed that if nodes launch successfully, the system should work (which was incorrect)

5.1.3 How we learned

Our learning process included active use of documentation, particularly of ROS2 Humble, guides for Nav2 and SLAM Toolbox, Gazebo and Clearpath resources. We also reviewed Clearpath simulation packages and intensively study previous team's codebase.

In addition to this, practice significantly impacted our learning process. Rebuilding workspaces multiple times, removing conflicting packages and frequent testing of components helped us to understand the working process. The guidance we received from our supervisor and his RAs also helped us a lot.

5.1.4 Learning progress

After almost a year working on our project, we can now systematically debug ROS2 systems, starting from environment, then moving on to packages, nodes, topics and controllers. We understand how ROS2 communication works. While initially we could not figure why we encountered failures during running, now we can identify and fix dependence and build conflicts. During our work, we successfully managed to restore ROS2 environments, resolve package conflict issues, build an operational Gazebo simulation and bring the JACKAL to an operational state.

5.2 Skill acquisition

5.2.1 Technical Skills

Technical Skills

- ROS2 Navigation Stack (Nav2): Gained solid understanding of mapping, localization, and planning modules. Learned to configure and launch different navigation nodes. We reached confident level.
- ROS2 Architecture: we acquired deep understanding of ROS2 communication, environmental sourcing and system structure. We applied this knowledge while debugging Nav2, TF, controller communication issues. We reached confident level.
- C++ Programming: Improved proficiency by reading and modifying ROS2 packages and controller plugins written in C++. We reached working level, not of a professional developer, but enough to work on our project.
- Model Predictive Control (MPC/NMPC): Understood its mathematical foundation and how it predicts robot trajectories while respecting constraints. We reached confident level.
- ACADOS Solver: Learned how it enables real-time optimization in robotics control loops. We reached working level.
- Simulation Tools: Became proficient in Gazebo and RViz for visualization, debugging, and testing robot behavior virtually. We reached confident level.
- SLAM Mapping and Sensor integration: Learned how to use SLAM Toolbox in ROS2 to

generate maps using LiDAR data and save them for localization. We reached confident level.

- **Hardware Operation:** Learned to safely operate the Jackal UGV, test motion commands, and monitor real-time sensor feedback.

5.2.2 Soft skills

Soft Skills • **Teamwork and Collaboration:** We divided responsibilities effectively — one focusing on ROS2 integration, the other on theoretical understanding and documentation. Regular synchronization meetings helped maintain progress.

- **Independent Learning:** We often faced technical challenges without direct answers and learned to research, test, and validate solutions independently.
- **Time Management:** Balancing simulation debugging, ROS learning, and report writing required careful planning and prioritization.
- **Technical Communication:** We practiced documenting our experiments clearly so that our supervisor and future teams can easily replicate or extend our work.

5.2.3 System Integration

Our major learning outcome was an understanding of how all components interact as a single system. Sensors, SLAM, Nav2, Controller are all connected and should be operational for a robot to operate. TF connects all coordinate frames and it is important for the timing to be consistent across all nodes. Failure at even one of those layers can cause failure for entire system. System integration was the most important and the most challenging step we had to overcome.

5.3 Future Learning goals

For future learning beyond our Capstone project, we would want to achieve next goals:

- **Full MPC integration:** stabilize the control pipeline and ensure real-time performance.
- **System Optimization:** Work on reducing latency in real-time NMPC computation and improving the overall robustness of the system.
- **Broader Human-aware navigation experiments:** extend the scale of simulation to different environments and much larger presence of dynamic obstacles (humans) to further validate SSM constraints.
- **Soft Skills:** Continue improving project coordination, communication, and presentation skills, especially for the final demonstration and defense.

Chapter 6: Simulation Environment and Hardware Setup

6.1 Simulation of the C4 Hall

We reconstructed the 4th-floor corridor of Block C4 using precise measurements. The Gazebo world includes:

- accurately scaled walls and corridors,

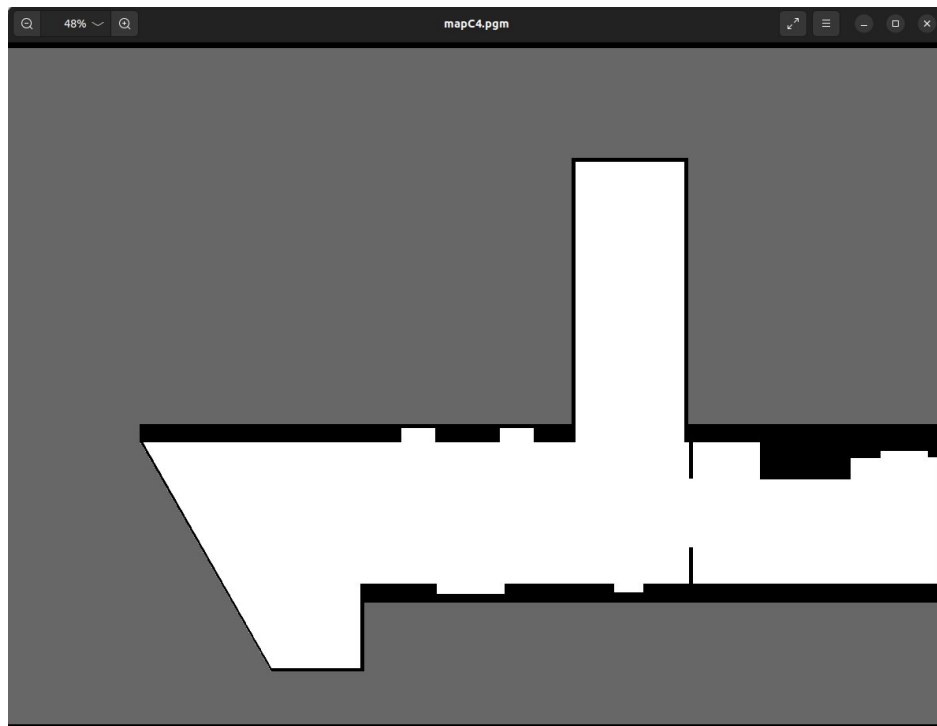


Figure 6.1: pgm file of the C4 coridor.

- realistic lighting and materials,
- a Jackal model with simulated LiDAR, IMU, and wheel encoders.

This environment allowed us to test SLAM, navigation, and controller behavior without risking hardware damage.

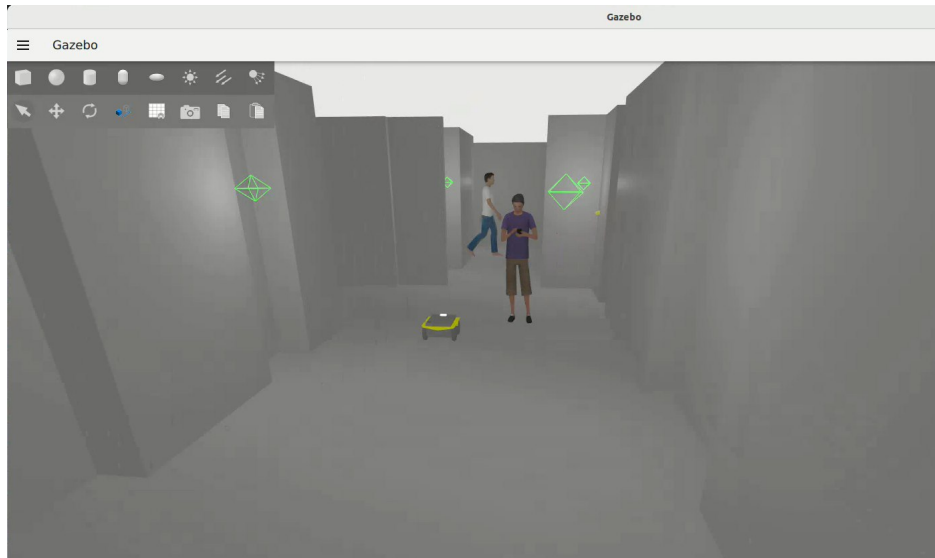


Figure 6.2: Gazebo simulation of the C4 hall.

6.2 Robot Operating System Reset

The real Jackal robot required a full OS reset due to broken dependencies. We installed:

- Ubuntu with ROS 2 Humble,
- Nav2 and SLAM Toolbox,
- LiDAR and sensor drivers.

In addition to the robot-side reset, a partial reconfiguration of the ROS 2 environment on the development laptop was necessary. This was caused by inconsistencies in environment sourcing, conflicting overlays, and incompatible Python dependencies introduced during integration of perception and MPC components. As a result, the system initially failed to correctly recognize the ROS distribution, and several core tools (e.g., `ros2 doctor`) were unavailable.

Furthermore, integration of the YOLO-based perception pipeline and the MPC controller required careful management of Python libraries such as NumPy, PyTorch, and Ultralytics. Version mismatches led to runtime errors (e.g., missing modules and incompatible APIs), which directly affected the perception-to-control pipeline.

To restore a stable and consistent environment, the following corrective actions were performed:

- Reinstallation and verification of system-wide Python dependencies to ensure compatibility (e.g., aligning `numpy`, `torch`, and `ultralytics` versions)
- Re-sourcing the ROS 2 underlay environment: `/opt/ros/humble/setup.bash`
- Cleaning the workspace by removing `build`, `install`, and `log` directories to eliminate corrupted build artifacts
- Resolving package conflicts and rebuilding message packages (such as `yolomsgs`) separately before full workspace compilation



Figure 6.3: Gazebo simulation of the C4 hall.

- Rebuilding the workspace using `colcon build --symlink-install` to maintain proper linkage between packages
- Verifying package availability and controller integration using `ros2 pkg list` and runtime testing

Additionally, adjustments were made to ensure correct interaction between the perception modules and the MPC controller, including validation of detected object data flow and consistency of runtime environments across nodes.

After completing these steps, the ROS 2 environment was successfully stabilized. All required packages were correctly detected, the build process became reliable, and the full system - including perception, navigation, and MPC could be launched and executed without critical errors.

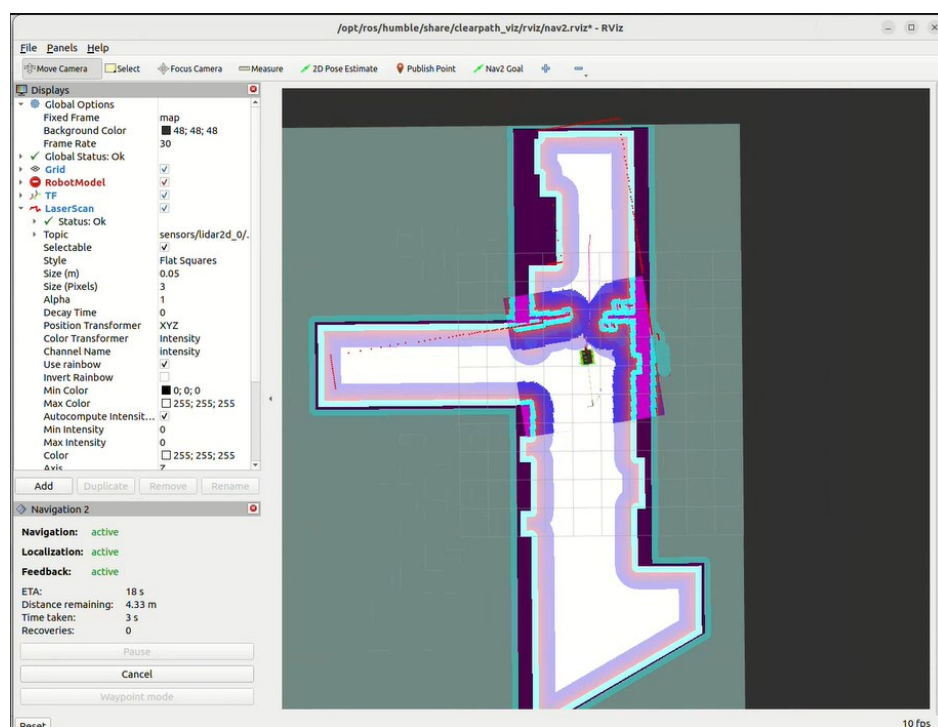


Figure 6.4: Rviz simulation of the C4 hall with moving robot and Nav2 path.

Chapter 7: System Design and Implementation

7.1 Software Architecture

The system architecture was significantly revised to ensure stability, modularity, and compatibility across all components. The final architecture includes:

- a ROS 2 Humble workspace with modular and decoupled packages,
- parameterized launch files supporting simulation,
- the Nav2 stack as the global navigation framework,
- a custom Model Predictive Control (MPC) controller for local trajectory optimization,
- integration with a perception pipeline (YOLO-based human detection),
- consistent Python environment aligned across all ROS nodes (NumPy, PyTorch, Ultralytics).

Special attention was given to resolving dependency conflicts in Ubuntu and Python environments. Inconsistent versions of `numpy`, `torch`, and `ultralytics` previously caused runtime failures in perception nodes. These were resolved by enforcing compatible versions and cleaning redundant installations from both system and user-level paths.

7.2 MPC Integration and Enhancements

The MPC module was fully integrated as a replacement for the default Nav2 local planner. The controller operates by:

- subscribing to odometry and robot state estimates,
- predicting future states over a finite time horizon,
- optimizing control inputs (linear and angular velocities),
- enforcing dynamic and safety constraints during optimization.

Compared to the initial design, the MPC implementation was extended with:

- an increased number of dynamic constraints handled simultaneously,
- integration of updated Safety Separation Margin (SSM) equations,
- improved parameter tuning to match the updated system dynamics and environment,
- partial coupling with perception outputs (human detection data from YOLO).

7.2.1 MPC Integration: Current Status

The MPC controller package (`mpc_low`) has been successfully compiled, integrated, and deployed within the navigation stack. The system can be launched reliably, and the interaction between Nav2, Gazebo simulation, and the MPC controller is stable during execution.

During the development process, several runtime issues were encountered, primarily related to timing constraints and system performance. These included missed control loop deadlines and occasional Behavior Tree timing warnings. To address these issues, the control frequency was reduced to 5 Hz, which ensured consistent and stable controller execution without overloading the system.

Additional tuning of MPC parameters was performed to improve overall behavior, including adjustments to velocity limits, constraint handling, and cost function weights. These changes resulted in smoother navigation and more reliable controller performance in dynamic scenarios.

The perception system is fully operational and correctly detects humans in the environment using the YOLO-based pipeline. The detected positions are successfully transformed and propagated to the MPC module, where they are incorporated into the control framework. Debugging confirmed that the human position data is consistently received and utilized within the MPC solver, ensuring proper interaction between perception and control components.

Overall, the MPC controller operates as intended, and the previously observed errors have been resolved. The system has evolved from an unstable prototype into a fully functional and integrated navigation framework, providing a solid foundation for further improvements in robustness, safety, and real-time performance.

Chapter 8: Testing and Evaluation

8.1 Simulation Testing

The system was validated through a series of simulation experiments in Gazebo and RViz. The following aspects were successfully verified:

- correct spawning of the robot in the C4 corridor environment,
- proper functioning of LiDAR and IMU sensors,
- successful initialization and operation of the Nav2 navigation stack,
- correct visualization and data flow in RViz.

8.1.1 Simulation Testing: Current Observations

Extensive simulation testing was conducted to evaluate the stability and performance of the fully integrated system. The robot model consistently spawns in the simulation environment, and all required nodes launch without critical errors.

During earlier stages of development, several runtime warnings were observed, including control loop timing issues and occasional transform inconsistencies. These were systematically addressed through environment cleanup, dependency alignment, and controller tuning. In particular:

- Control loop instability was resolved by adjusting the controller frequency to 5 Hz
- Timing and synchronization issues within Nav2 were mitigated through parameter tuning
- Package and dependency conflicts affecting runtime behavior were eliminated

Following these improvements, the system demonstrates stable and reliable operation. The following components are fully functional:

- Gazebo simulation launches consistently
- Robot description, sensors, and TF tree operate correctly
- Nav2 stack performs global planning without failures
- MPC controller executes reliably during navigation

The simulation environment is now considered both structurally and dynamically stable under MPC control.

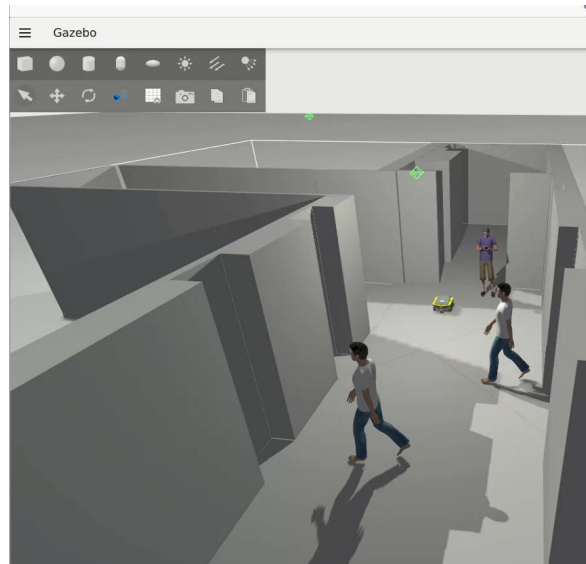


Figure 8.1: Simulation testing: obstacle avoidance with MPC in Gazebo

8.1.2 Simulation Testing: Obstacle Avoidance

Dynamic obstacle avoidance was evaluated by introducing one static and two moving human actors into the simulation environment. The robot, controlled by the MPC-based local planner, successfully adapts its trajectory in real time to avoid collisions.

The perception system detects human positions and propagates this information to the MPC controller, allowing it to incorporate dynamic constraints during optimization. As a result, the robot demonstrates safe navigation behavior in the presence of moving obstacles.

8.2 Hardware Testing

Previously, the robot that the project focused on was malfunctioning and could not be operated. After resolving the hardware problems, the Jackal robot can finally be operated with the joystick and a laptop used as its hardware.

In addition to this, robot can operate while navigating with Nav2. The SLAM (Simultaneous Localization and Mapping) issues were resolved, and currently JACKAL robot can draw the map of the surrounding environment in Rviz correctly.

However, there is a build failure during an attempt to launch MPC on the robot. Nav2 framework does not launch when the namespace we work on, j100_0001, is configured in the YAML file. As a result, without Nav2 testing MPC was not possible.

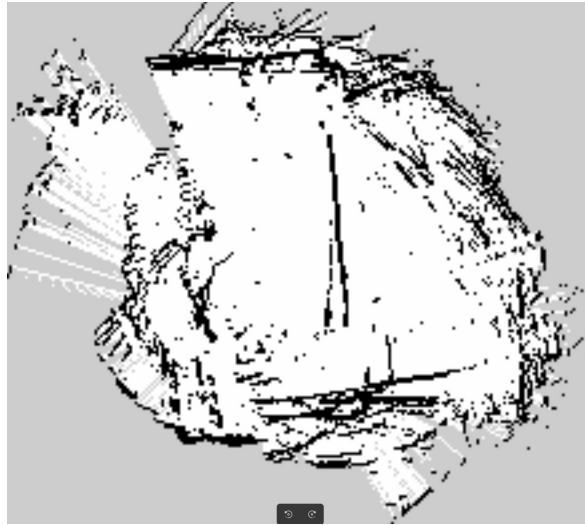


Figure 8.2: Map of the lab room drawn with SLAM before

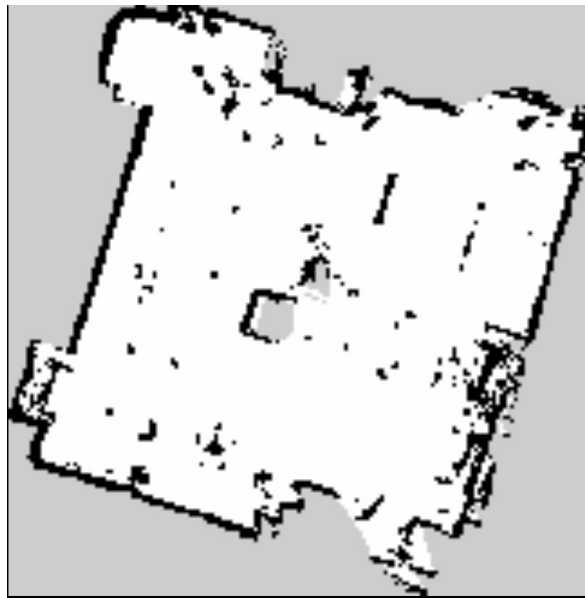


Figure 8.3: Map of the lab room drawn with SLAM after

Chapter 9: Experimental Validation of Speed and Separation Monitoring (SSM)

9.1 Introduction to SSM

Speed and Separation Monitoring (SSM) is a critical safety function for autonomous mobile robots that ensures a protective separation distance is maintained between the robot and any obstacles or personnel. Since this project involves a Jackal robot, weighing 17 kg and capable of carrying up to 20 kg load, it is important to ensure that possible harmful collision with human is avoided, while maximizing the speed and efficiency of the robot. This chapter details the empirical braking tests performed on the Jackal to define the safety parameters required for autonomous navigation.

9.2 Experimental Methodology

To carry out the experiment, specific script was written that, once run, waited for short period of time to allow the user to move away from the robot to a safe distance, run the robot at a given speed, give a command to stop, measure the distance covered AFTER the command was given until the complete stop and record it for later analysis. The robot was commanded to various linear velocities (v) ranging from 0.5 m/s to 1.5 m/s. Upon reaching a steady-state velocity, a zero-velocity command (0.0 m/s) was given. The stopping distance was calculated by measuring the displacement from the EKF-filtered odometry (/j100_0001/platform/odom/filtered topic in the ROS2 system of the robot) between the timestamp of the stop command and the moment the robot reached a complete halt.

9.3 Results and Statistical Analysis

The experimental results demonstrate a clear correlation between velocity and braking performance. Table 9.1 summarizes the data across the trial sets.

9.4 Discussion of Safety Parameters

The maximum stopping distance observed at 1.0 m/s was approximately **0.1692 m**. While the robot can operate at 1.5 m/s, the experimental data reveals a significant increase in variance (Standard Deviation of 0.113 m) and a maximum stop distance of **0.3427 m**.

Table 9.1: Braking Performance Summary Across 20 Trials per Speed

Velocity (m/s)	Average Dist. (m)	Max Dist. (m)	Std. Dev (m)
0.5	0.0121	0.0229	0.0066
0.6	0.0414	0.0674	0.0199
0.7	0.1040	0.1232	0.0150
0.8	0.0844	0.1501	0.0559
0.9	0.1019	0.1643	0.0487
1.0	0.1514	0.1692	0.0125
1.5	0.2000	0.3427	0.1015

The Hokuyo LiDAR utilized in this project has a measurement uncertainty (Z_d) of ± 6 mm. When operating at 1.0 m/s, the sensor uncertainty represents only 3.5% of the total stopping distance, indicating that the safety system is limited by physical inertia rather than sensor precision. Including the maximum distance is essential since the safe distance should be based on the worst-case scenario that can occur. Considering the standard deviation is important to decide at which the robot should operate; low variance ensures that robot's stopping distance is unlikely to deviate from the average braking distance. Results suggest that stopping distance at linear velocity of 1.0 m/s has second lowest variance, after linear velocity of 0.5 m/s.

9.5 Implementation Recommendations

Based on the results, **1.0 m/s** is established as the maximum operational velocity. At this velocity, robot is fast enough to operate efficiently and less likely to deviate from the calculated stopping distance. To implement the SSM logic, the **Protective Separation Distance** (S_p) is defined as follows:

$$S_p = S_s + Z_d + S_{buffer} \quad (9.1)$$

For a velocity of 1.0 m/s:

- S_s (Max Stopping Distance): 0.1692 m
- Z_d (LiDAR Uncertainty): 0.006 m
- S_{buffer} (Safety Margin): 0.075 m (approx. 30% of total)

Final SSM Configuration: The robot is programmed to trigger a *Hard Stop* whenever an object is detected within a **0.25 meter (250 mm)** radius. This distance provides a robust safety envelope that accounts for the worst-case physical braking observed while maintaining the agility required for indoor navigation.

While this distance is necessary for a Hard stop, the implementation of Model Predictive Control requires to optimize robot motion by varying robot's speed and trajectory when encountering an obstacle rather than stopping completely as in traditional motion planning methods. This distance will remain as a safety measure, in case robot will not be able to find an optimal trajectory and the collision is inevitable if movement is continued.

However, for a MPC based motion planning, it is necessary to optimize the motion trajectory and robot's speed based on the distance to the obstacle in real time. To implement it, it is necessary to take into consideration SSM constraints formulated in ISO/TS 15066. "Their imposition ensures that, assuming that the human can move at any time in any direction with a fixed maximum speed v_h , a collision with the manipulator cannot happen while the latter is in motion." (Nurbayeva et al., 2023).

The SSM constraints take the following form:

$$\bar{v}_h(T_{dr} + \frac{v_i}{\bar{a}_i}) + v_i T_{dr} + \frac{v_i^2}{2\bar{a}_i} + \epsilon_s \leq d_{ih}, i \in N_r$$

where $\bar{v}_h = 1.6m/s$ is the assumed maximum human moving speed, which was set to be the most optimal value when testing the simulation, $T_{dr} = 0.2s$ is the time of detection and reaction, set to be twice as great as the sampling time, which is set to be 0.1 s, v_i is the robot speed, a variable that MPC will be able to set in accordance with these constraints, $\bar{a}_i = 2.94m/s^2$ maximum robot deceleration, derived from previous experiment, $\epsilon = 0.006m$ is the uncertainty of LiDAR sensor and d_{ih} is the distance between the edges of the robot's and human's spheres.

Using distance between the centers of the human and the robot is more convenient for computational solving rather than using the distance between the spheres. The following formula:

$$d_{ih} \min_{j \in N_h} \{d_{ij} - (R_{r,i} + R_{h,j})\}$$

where $R_r = 0.25m$ and $R_h = 0.25$ are radiuses of robot and human spheres respectively. Rearranging the equation gives us:

$$\bar{v}_h(T_{dr} + \frac{v_i}{\bar{a}_i}) + v_i T_{dr} + \frac{v_i^2}{2\bar{a}_i} + \epsilon_s + R_r + R_h \leq d$$

where d is the distance between centers of human and robot spheres and equals to:

$$d = \sqrt{(x_i - x_h)^2 + (y_i - y_h)^2}$$

The problem with this formulation is that the ACADOS package that is used for computation in MPC uses derivative of the function. The square root function (9.1) approaches 0 when x approaches zero, in this region of the curve derivative will approach infinity, which can destabilize the ACADOS solver. The plausible solution is to square both sides of the equation. Rearranging the equation into polynomial form and squaring both sides give:

$$((\frac{1}{2\bar{a}_i})v_i^2 + (T_{dr} + \frac{\bar{v}_h}{\bar{a}_i})v_i + (\bar{v}_h T_{dr} + \epsilon + R_r + R_h))^2 \leq (x_i - x_h)^2 + (y_i - y_h)^2$$

By implementing these constraints, MPC will be able to vary the speed of the robot to ensure obstacles are avoided and safe motion trajectory is chosen.

Another issue with the SSM configuration was the computation involving odd powers of v_i . For robot manipulators, velocity is computed using differential kinematics, in which, square root must be avoided as well. As a result, in a case study discussed in [19], approximation was used instead of directly implementing constraints. In 9.2, The red curve shows the allowed maximum velocity, while the black curve shows the calculated velocity due to approximation. As it can be observed, it is lower than the red curve, meaning that robot had to operate at a velocity lower than what SSM constraints could allow.

However, in our case, we are able to implement these SSM constraints directly, since the velocity of the robot is the control variable and is imposed based on those constraints.

As a result, less conservative SSM configuration is implemented in this MPC.

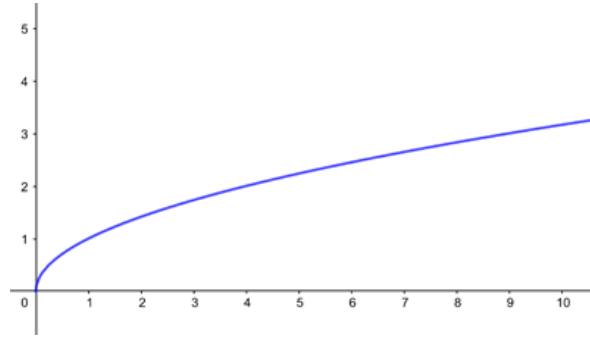


Figure 9.1: Square root function graph

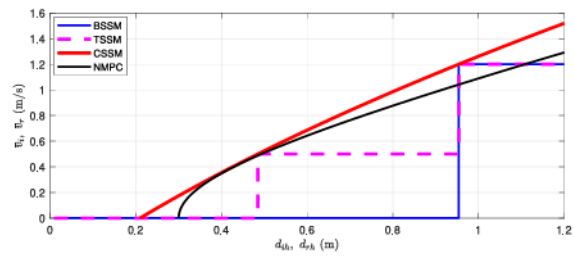


Figure 9.2: Velocity modulation profiles for the four controllers considered in the case study

Chapter 10: **Limitations**

Unfortunately, we could not carry out experiments on JACKAL robot due to technical issues we are still facing with the robot. Despite complete restoration of the ROS2 environment, installation of all necessary packages and setup, we faced namespace troubles. We encountered build failure when configuring namespace in a YAML file (a human-readable language, which allows humans to control robot behaviour). This failure does not allow Nav2 to operate, and since Nav2 is the main framework used for navigation, MPC can not be launched as well.

Chapter 11: Conclusions and Future Work

This project successfully achieved the integration of MPC-based navigation framework for the Clearpath Jackal robot within a ROS 2 Humble simulation environment. Throughout the development process, significant effort was dedicated to resolving system-level issues, stabilizing the software architecture, and ensuring compatibility across all components.

Key accomplishments of this work include:

- Refactoring and stabilizing the inherited codebase
- Creating a realistic C4 corridor simulation environment in Gazebo
- Restoring and configuring the ROS 2 environment on both the robot and development machine
- Resolving dependency conflicts in Ubuntu and Python libraries (NumPy, PyTorch, Ultralytics)
- Successful compilation and full integration of the MPC controller package, including an increased capacity to handle multiple dynamic obstacles and the integration of an updated Safety Separation Monitoring formulation
- Establishing a stable interaction between Nav2, MPC, and simulation
- Integration of a YOLO-based perception pipeline for human detection

All major issues encountered during earlier stages such as simulation launch failures, package inconsistencies, controller instability, and missing dependencies have been resolved. The system can now be launched reliably, and the robot operates correctly within the simulated environment.

The MPC controller is fully functional and actively used during navigation. It generates smooth control commands and successfully handles dynamic constraints. Additionally, the perception system detects human positions and propagates them to the MPC controller, enabling real-time trajectory adaptation.

Simulation experiments demonstrated that the robot is capable of navigating toward a goal while avoiding collisions with moving obstacles. This confirms the effectiveness of the integrated perception-control pipeline and the applicability of MPC for dynamic environments.

While the system is now stable and operational, several aspects can be further improved. Future work will focus on:

- Enhancing real-time performance and increasing control loop efficiency
- Further tuning MPC cost functions and safety constraints (e.g., SSM parameters)
- Improving robustness in more complex and crowded dynamic scenarios
- Extending validation to real-world experiments on the physical Jackal robot

- Performing quantitative evaluation of navigation performance and safety metrics

Overall, the project has progressed from an unstable and partially functional setup to a fully integrated and operational navigation system. The developed framework provides a solid foundation for further research and development in safe and intelligent robot navigation in dynamic environments.

Bibliography

- [1] Azizi, M. R., Rastegarpanah, A., Stolkin, R. (2021). Motion Planning and Control of an Omnidirectional Mobile Robot in Dynamic Environments. *Robotics*, 10(1), 48. <https://doi.org/10.3390/robotics10010048>
- [2] Bjelonic, M., Grandia, R., Geilinger, M., et al. (2022). Offline motion libraries and online MPC for advanced mobility skills. *International Journal of Robotics Research*, 41(9–10), 903–924. <https://doi.org/10.1177/02783649221102473>
- [3] Bos, M., Vandewal, B., Decre, W., Swevers, J. (2023). MPC-based Motion Planning for Autonomous Truck-Trailer Maneuvering. *IFAC-PapersOnLine*, 56(2), 4877–4882. <https://doi.org/10.1016/j.ifacol.2023.10.1258>
- [4] Chai, R., Niu, H., Carrasco, J., et al. (2024). Deep Reinforcement Learning-Based Fast Trajectory Planning and Control for Mobile Robot in Unknown Environment. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4), 5778–5792. <https://doi.org/10.1109/TNNLS.2022.3209154>
- [5] Fan, Z. (2023). Multi-Point path planning for robots based on deep reinforcement learning. *Journal of Physics: Conference Series*, 2580(1). <https://doi.org/10.1088/1742-6596/2580/1/012048>
- [6] Firouzmand, E., Talebi, H., Abdollahi, F. (2021). Robust MPC-based Abstraction for Motion Planning of a Mobile Robot. *ICRoM 2021*, 485–490. <https://doi.org/10.1109/ICRoM54204.2021.9663460>
- [7] Gao, Y., Messerer, F., Frey, J., et al. (2023). Collision-free Motion Planning for Mobile Robots by Zero-order Robust Optimization-based MPC. *ECC 2023*. <https://doi.org/10.23919/ECC57647.2023.10178311>
- [8] Jaramillo-Martinez, R., Chavero-Navarette, E., Ibarra-Perez, T. (2024). Reinforcement-Learning Based Path Planning: A Reward Function Strategy. *Applied Sciences*, 14(17). <https://doi.org/10.3390/app14177654>
- [9] Le, H., Saeedvand, S., Hsu, Ch. (2024). Review of Mobile Robot Navigation Using DRL in Crowded Environments. *Journal of Intelligent and Robotic Systems*, 110(4). <https://doi.org/10.1007/s10846-024-02198-w>
- [10] Li, J., Sun, J., Liu, L., Xu, J. (2021). MPC for tracking of autonomous mobile robot with local path planning. *Measurement and Control*, 54(9–10), 1319–1325. <https://doi.org/10.1177/00202940211043070>
- [11] Nippun, K., Kochuvila, S. (2023). Mobile Service Robot Path Planning Using Deep Reinforcement Learning. *IEEE Access*, 11, 100083–100096. <https://doi.org/10.1109/ACCESS.2023.3311519>
- [12] Ou, Y., Cai, Y., Sun, Y., Qin, T. (2024). Autonomous Navigation with Sensor Fusion and DRL. *Sensors*, 24(12). <https://doi.org/10.3390/s24123895>
- [13] Pai, N., Tsai, X., Chen, P., Lin, H. (2024). Indoor Mobile Robot Path Planning Using DRL. *Sensors and Materials*, 36(5), 1959–1982. <https://doi.org/10.18494/SAM4827>
- [14] Piccineli, N., Vesentini, F., Muradore, R. (2023). MPC-Based Motion Planning using Velocity Obstacles. *ECC 2023*. <https://doi.org/10.23919/ECC57647.2023.10178219>

- [15] Yang, L., Bi, J., Yuan, H. (2022). Dynamic Path Planning for Mobile Robots with DRL. *IFAC-PapersOnLine*, 55(11), 19–24. <https://doi.org/10.1016/j.ifacol.2022.08.042>
- [16] Yin, Y., Chen, Z., Liu, G., Guo, J. (2023). Mapless Local Path Planning Using DRL. *Sensors*, 23(4). <https://doi.org/10.3390/s23042036>
- [17] Nurbayeva, A., Shintemirov, A., Rubagotti, M. (2023). Deep Imitation Learning of Nonlinear Model Predictive Control Laws for a Safe Physical Human–Robot Interaction. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, 19(7), 8384–8395. <https://doi.org/10.1109/TII.2022.3217833>
- [18] Zhang, Q., Ma, W., Zheng, Q., et al. (2024). Path Planning in Dynamic Environments Based on DRL. *IEEE Access*, 12, 189136–189152. <https://doi.org/10.1109/ACCESS.2024.3507016>
- [19] Oleinikov, A., Kusdavletov, S., Shintemirov, A., Rubagotti, M.(2021). Safety-Aware Nonlinear Model Predictive Control for Physical Human-Robot Interaction. *IEEE Robotics and Automation Letters*, 6(3), 5665–5672. <https://doi.org/10.1109/LRA.2021.30835813>