Thesis

Motion planning for mobile manipulators in multi-agent settings using MPC

MSc Robotics

Danning Zhao

Delft University of Technology

TUDelft

Thesis

Motion planning for mobile manipulators in multi-agent settings using MPC

MSc Robotics

by

Danning Zhao

Student Name	Student Number
Danning Zhao	5459583

Supervisor:	Dr. J. (Javier) Alonso Mora
Daily Supervisor:	Ir. S. (Saray) Bakker
Faculty:	Faculty of Mechanical Engineering, Delft
Department:	Cognitive Robotics (CoR), Autonomous Multi-robots Lab
Duration:	May 13, 2024 – January 13, 2025

TU Delft

Abstract

Mobile manipulators, which integrate a robotic arm on a mobile base, are increasingly being explored and deployed in sectors such as healthcare, logistics, and aerospace. While motion planning for these systems has been studied in single-agent scenarios, the use of multiple robots to enhance efficiency and accelerate task completion in multi-agent settings remains largely unexplored, particularly in real-world environments. Extending motion planning to multi-mobile manipulators introduces challenges in real-time performance, collision avoidance, and coordination. To address these, this thesis proposes a decentralized Model Predictive Control (MPC) framework with a double integrator as dynamic model, denoted as MPC-d, tailored for multi-mobile manipulators operating in shared workspaces. It integrates optimization-based planning with robust state estimation, ensuring effective collision avoidance. Furthermore, a prioritized heuristic is introduced, leveraging the prediction horizon of MPC to resolve potential livelocks. The framework is validated through simulations and real-world experiments. Simulations compare MPC-d with MPC using a triple-integrator model (MPC-t) and a state-of-the-art geometric planner, called Geometric Fabrics (GF). Results demonstrate that MPC-d achieves comparable task success rates and collision avoidance compared to GF in pick-and-place scenarios while requiring less computation time than MPC-t. Real-world experiments confirm the framework's viability, showcasing effective collision avoidance, enhanced efficiency from the prioritized heuristic, and consistency with simulation outcomes. Although MPC-d incurs higher computational costs than reactive geometric methods, it provides reliable performance and motion prediction of other agents in multi-agent settings.

Additionally, this thesis provides an open-source codebase to support further advancements in multi-robot motion planning. Code: `https://github.com/DianeZhao/dinova_mpc`

Contents

List of Figures

List of Tables

1 Introduction

A mobile manipulator (MM) is a system that integrates a robot manipulator onto a mobile platform, offering the advantages of both high manipulation capabilities and the mobility of a moving base [1]. It finds applications across various sectors, including aerospace industries [2, 3], warehouse logistics [4] and healthcare [5, 6]. In general, these environments are inherently dynamic multi-agent environments. For instance, as shown in Fig. 1.1a, the YuMi mobile manipulator can operate in real-world settings such as hospitals. It has the potential to dispense medications, transport them to necessary locations and deliver medical supplies to staff.

Moreover, employing multiple robotic systems holds the potential for increasing efficiency and accelerating task completion [7–9]. In the future, multiple mobile manipulators could be employed in environments shared with humans and other robots that assist the customers or staff. For instance, as illustrated in Fig. 1.1b, in a restaurant scenario each customer can be assigned a corresponding server mobile manipulator, with each robot operating autonomously. Each robot needs to ensure that its movements do not interfere with those of other robots or individuals, thereby maintaining smooth and efficient service in multi-agent environments. This coordination is crucial for enhancing the overall customer experience and optimizing efficiency.

For autonomous service robots, tasks typically involve guiding the end-effector (EE) to a specific reference point while avoiding collisions with the environment. This process, known as motion planning, determines valid actions for the robot to navigate from one point to another while adhering to physical and operational constraints [10]. In dynamic environments, such as a restaurant where customers may pass by, real-time motion planning is essential for ensuring safe and efficient operation, accounting for collision avoidance, physical limits and precise end-effector positioning.

With the rise of robots operating simultaneously, motion planning faces new challenges. While task assignment can manage scheduling to reduce interference [11–13], the ability of controllers to quickly replan in dynamic environments is vital. In these cases, motion planners must balance multiple constraints and maintain low time complexity for real-time performance. The computational demands, especially with high-dimensional mobile manipulators, grow with system complexity, making efficient and scalable planning a significant challenge.

This thesis investigates the extent to which a single-agent mobile manipulator optimization-based local planner can generalize to multi-agent environments while maintaining real-time performance. We propose to extend a promising Model Predictive Control (MPC) framework [14] towards multiple mobile manipulators and compare several variations against geometric motion planning. Section 1.1 first examines whole-body local planner methods for single mobile manipulators, with a particular focus on those validated through real-world experiments. It then explores existing research on multi-manipulator systems to provide insights and inspiration for adapting these approaches to multi-mobile manipulator environments. The contributions of this thesis are outlined in Section 1.2.

(a) ABB's mobile and autonomous YuMi laboratory robot concept designed to work alongside medical staff and lab workers [15]

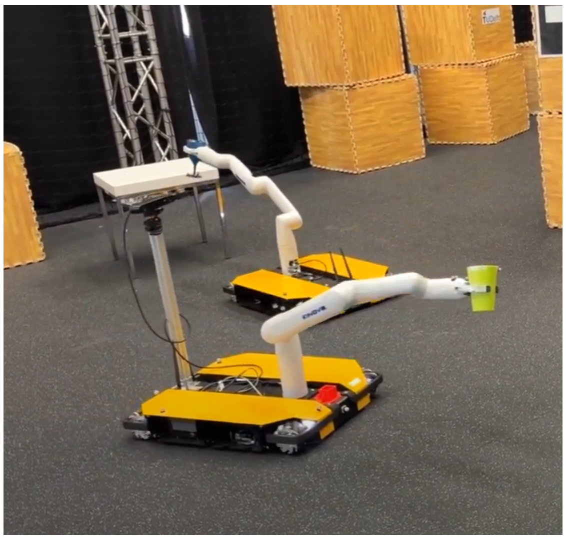

(b) Two mobile manipulators performing pick-and-place tasks, efficiently delivering cups to customers [16]

Figure 1.1: (a) A mobile manipulator works alongside humans (b) A demonstration of mobile manipulators operating in a multi-agent setting to complete pick-and-place tasks.

1.1. Related work

1.1.1. Local planners for a single mobile manipulator

Obstacle-free motion planning for single robotic manipulators has been extensively explored over the past few decades [1]. Whole-body motion planning for mobile manipulators can be viewed as an extension of fixed-base manipulator planning, integrating the Degrees of Freedom (DOF) of both the mobile base and the manipulator arm into a unified framework. These whole-body motion planning algorithms for mobile manipulators are typically classified into global and local planning methods. Global planning methods, mainly composed of sampling-based methods [17–20] and global trajectory optimization methods [21, 22], are integrated into open source motion planning frameworks, such as [23, 24]. However, when applied for systems with high DOF robots, they are unsuitable for dynamic environments due to long planning times. While local planners have a slightly lower success rate [25], they exhibit real-time trajectory updates, enabling them to adapt to dynamic environments. These planners can be roughly categorized into geometric methods [26, 27], sampling-based methods [28, 29] and optimization-based approaches solved by numerical solvers [14, 25, 30, 31].

Geometric methods [26, 32] are used for reactive and local motion planning which doesn't require solving an optimization problem. Desired system behavior is described using second-order differential equations. Different components or tasks, such as collision avoidance and joint limit avoidance, are described using a second-order differential equation for each task and combined within the configuration space. Due to the absence of an optimization problem and control horizon, the computation time is fast (approx. 1 ms) which is beneficial for dynamic real-world environments. Geometric fabrics are thereby extended to dynamic fabrics [27] accounting for the velocities of obstacles and to multi-agent scenarios [33]. However, the lack of a prediction horizon can lead to unintuitive responses to dynamic agents and no hard constraints are implemented for collision avoidance, which is present in optimization-based local motion planners.

In addition to geometric-based methods, advanced motion planning techniques also include sampling-based methods like Model Predictive Path Integral (MPPI). MPPI uses a sampling-based optimization technique to generate trajectories, evaluating them against a predefined cost function to select an optimal path. Its key advantage lies in its ability to handle non-smooth or non-differentiable cost terms and constraints, offering flexibility in dynamic and unstructured environments. However, MPPI relies on penalizing unsafe trajectories rather than enforcing strict constraints, leading to potential violations of critical safety or feasibility requirements. This limitation often necessitates post-processing steps, such as the incorporation of Control Barrier Functions (CBF), to ensure stability and collision avoidance [34].

In contrast, optimization-based methods like MPC and QP-based approaches directly incorporate system dynamics and constraints into their formulations, ensuring hard constraint satisfaction. QP-based methods are highly reactive and computationally efficient, with low execution times ranging from 4 ms to 10 ms for a single mobile manipulator [25, 35]. These methods excel in real-time scenarios but lack a prediction horizon, which limits their foresight in dynamic environments. Meanwhile, MPC incorporates a prediction horizon, solving a constrained optimization problem to produce locally optimal trajectories. This approach enables MPC to handle more complex scenarios involving dynamic obstacles, but at the cost of higher computation times, typically ranging from 20 ms to 100 ms [30, 36]. While optimization-based methods offer robust constraint handling and safety guarantees, they also face challenges. The inclusion of multiple constraints can lead to infeasibility, potentially causing the robot to stop moving. Strategies such as relaxing constraints [37] or carefully tuning solver parameters are often required to mitigate these issues. Furthermore, achieving real-time performance demands not only efficient solver design but also a deep understanding of numerical optimization and control theory. Practical implementations frequently reveal nuanced issues, underscoring the importance of real-world testing and iterative refinement.

Table 1.1 summarizes the information about the optimization-based local planners discussed in this chapter. It elaborates on their machine setup, cost functions, constraints, computation time, solver, and global planner. In the cost function and constraints columns, an 'x' denotes that the algorithm explicitly defines the corresponding term to meet this standard. Computation times are listed, and for methods with prediction horizons, the horizon time T is also included. A blank cell indicates that the paper does not explicitly mention the frequency of its controller, typically because these works primarily focus on the total task completion time.

1.1.2. Multi-Robot Motion Planning

In general, Multi-robot Motion Planning (MRMP) is an active research field which has gained attention over the years and is continually evolving within mobile robot systems like Unmanned Aerial Vehicles (UAVs) [38–40] and Unmanned Ground Vehicles (UGVs) [41]. In scenarios involving multiple (mobile) manipulators, the problem can be specified as leveraging multiple (mobile) manipulators that use end-effectors to accomplish individual tasks in close proximity while ensuring collision-free movement among themselves. Planning a feasible and local optimal trajectory for several (mobile) manipulators remains a challenging problem, as a single mobile manipulator is already computationally demanding due to its complex kinematic infrastructure as discussed in Section 1.1.1

One approach involves treating a multi-arm robot system as a single composite robot, with its resulting DOF being the sum of all individual robot DOF [42]. This centralized method offers a unified perspective on planning trajectories for multi-robot systems. Another approach treats each robot as an individual agent and aims to avoid collisions with other robots [12, 33, 43] by using well-developed single-agent local planners. This decentralized approach, emphasizes the autonomy of each robot in navigating its environment. Deadlocks may occur, where robots come to a halt and are unable to achieve their objective, caused by the robot's reactiveness in deciding the next steps based on current local observations of the environment. Similarly, livelocks, where robots continuously adjust their states without making progress toward the goal, can also hinder task completion. These are well-known challenges in decentralized systems that prevent efficient operation [39, 44]. To address these issues, prioritization frameworks can be employed, setting and communicating a hierarchy of goals within the group of affected robots to resolve conflicts and generate a new policy [33, 43].

Despite these advances, most research on multi-manipulator systems is limited to simulations. Real-world examples of multiple mobile manipulators operating in shared environments remain even rarer. The challenges in real-world applications, such as hardware limitations, noisy sensor data, and communication constraints, introduce greater complexity compared to simulations. The limited exploration of multi-mobile manipulator systems in practical settings highlights the need for further investigation and development in this area.

1.2. Contribution

This work extends real-time optimization-based local planners to enable the simultaneous operation of multiple high-DOF mobile manipulators within a shared workspace, as shown in Figure 1.1b. Inspired by the framework proposed by Adam et al. [14], which utilizes MPC for nonprehensile object transportation with obstacle avoidance in single mobile manipulators, this research applies MPC to multi-mobile manipulator systems. The investigation focuses on object delivery and pick-and-place tasks in dynamic, multi-agent environments. The effectiveness of these methods is validated through real-world experiments to ensure practical applicability in real operational settings.

This thesis contributes:

1. **Extension of whole-body MPC to multi-mobile manipulator systems with real-time performance**: To the best of the authors' knowledge, MPC has not been previously applied to multi-mobile manipulator systems in the real world. A decentralized MPC approach is proposed, demonstrating real-time performance, which is crucial for safe operation in dynamic multi-agent environments.
2. **A comparison study with prior work and state-of-the-art geometric-based local planner in simulation**: This work conducts a comparison between MPC and a purely reactive geometric-based method [27]. The study evaluates their performance in various simulated multi-agent scenarios. This comparative analysis illustrates the differences between reactive geometric methods and predictive optimization-based approaches, particularly in dynamic, multi-agent environments.
3. **A heuristic approach for resolving livelocks**: A heuristic is developed to establish prioritization between mobile manipulators, resolving potential conflicts such as livelocks that can arise from decentralized planning approaches in multi-robot systems. This ensures efficient coordination and smooth operation in shared workspaces.
4. **Open-source codebase**: The code developed for this thesis, encompassing both simulation and real-world experiments, will be made available as open-source. This aims to facilitate further research and development in multi-robot motion planning and real-time control.

1.3. Overview

Chapter 2 provides a detailed explanation of the implementation of MPC and the heuristic deadlock resolution approach. Chapter 3 focuses on validating the proposed approach in various simulated environments, with a discussion of the corresponding results. Following this, Chapter 4 presents the real-world experiments and their results. Finally, Chapter 5 concludes the thesis with a critical discussion of the findings and outlines potential future directions to further advance this work.

Table 1.1: Table summarizing information about the optimization-based local planners discussed in this chapter, detailing their machine setup, cost functions, constraints, computation time, solver, and global planner. In the cost function and constraints columns, an 'x' denotes that the algorithm explicitly defines the corresponding term to meet this standard. Computation times are listed, and for methods with prediction horizons, the horizon time T is also included. A blank cell indicates that the paper does not explicitly mention the frequency of its controller, typically because these works primarily focus on the total task completion time.

	Algorithm	Citation	Year	Machine setup				Cost function and constraints											Solver	Computation time	Global planners		Codes available
				manipulator	manipulator+omnidirectional	manipulator+non-holonomic	manipulator+quadrupedal	base trajectory	base desired velocity	EE desired pose	EE desired configuration	EE desired velocity	physical limits	self-collision	static obstacles	dynamic obstacles	obstacle motion model	manipulability			base	manipulation	
QP-based	MMC	[45]	2020	x								x	x					x	qpsolvers	2.53ms			x
	NEO	[25]	2021	x								x	x		x	x		x	qpsolvers	10ms			x
	Holistic	[35]	2022		x	x						x	x					x	qpsolvers	4.8ms			x
	VMC	[46]	2022			x						x	x	x				x	qpsolvers				x
	MotM	[47]	2023			x			x			x	x					x	qpsolvers				
	Reactive Base MotM	[48]	2024		x	x			x			x	x		x	x			qpsolvers		STAA*		
	IDK	[49]	2021		x							x	x		x				qpOASES [50]	8ms			
MPC	Fully-integrated	[51]	2019			x		x		x	x				x				OCS2(SLQ)	10ms(T=1s)	RRT*		
	Perceptive-MPC	[51]	2020			x				x			x		x	x			OCS2(SLQ)	50ms(T=2s)		RRT*	x
	Articulated Object Interaction	[30]	2021			x				x			x	x	x	x			OCS2(SLQ)	33ms(T=4s)			x
	Collision-Free MPC	[52]	2022				x			x			x	x	x	x			OCS2(SLQ)	14ms(T=1s)			
	Keep Upright	[14]	2023		x					x			x		x	x	x		OCS2(SQP)	23ms(T=1s)			x
	Free Space Decomposition	[36]	2021			x		x			x		x		x	x	x		ForcesPro (Interior point)		cubic Bezier Curve		x
MPPI	STORM	[28]	2021	x						x			x	x	x	x		x		20ms			x
	MPPI-IsaacGym	[29]	2023	x		x				x			x	x	x	x				40ms(T=6s)			x
	Robust-sampling-mpc	[34]	2023		x					x			x	x	x					10ms(T=1s)			x

2 Methodology

2.1. Method Overview

This chapter details the proposed framework for motion planning in multi-agent settings, combining multi-robot Model Predictive Control (MPC), robust state estimation, and a heuristic for livelock resolution to enable efficient and collision-free operation in shared workspaces. At its core, the framework leverages multi-robot MPC to plan motions across all degrees of freedom for the ego mobile manipulator, integrating input reference targets provided by either a user or the heuristic. State estimation plays a crucial role, ensuring accurate determination of the robot's full state while tracking the states of dynamic obstacles, such as other robots. These estimated states are then communicated to the controller to dynamically adapt the motion plans to the environment. To address the common challenge of livelock in decentralized multi-agent systems, the framework introduces a heuristic that effectively coordinates agents, enabling smooth operation in complex environments.

The chapter is structured as follows: key concepts foundational to the proposed approach are introduced in Section 2.2, including planning spaces and kinematic fundamentals, to establish the context for subsequent sections. Second, Section 2.3 presents a detailed description of multi-robot MPC, including its formulation and the application of Sequential Quadratic Programming (SQP) for solving the constrained nonlinear optimization problem. Following this, Section 2.4 discusses state estimation for both the ego robot and non-ego robots. Finally, the functionality of a heuristic to solve livelocks is introduced in Section 2.5. The chapter concludes with a summary of the complete methodology pipeline, presented in Section 2.6 as Fig. 2.1, which visually illustrates the interactions and data flow among the framework's key components.

2.2. Fundamentals: Planning spaces and kinematics

Before investigating specific mobile manipulator planning methods, it's essential to clarify terminologies and common definitions used in motion planning algorithms.

2.2.1. Planning Space

Configuration Space (CS)

The configuration of a robot uniquely determines the position of every point in the system relative to a fixed reference frame. When the parameters used to define the configuration are independent, they are referred to as the generalized coordinates of the system, and the number of these independent parameters is termed the Degrees of Freedom (DOF) of the system.

The space in which a configuration is represented as a point is referred to as the Configuration Space C and the number of parameters used to specify a configuration is denoted as the dimension N [10]. For the case of a mobile manipulator, the configuration usually consists of both the base and arm configurations, denoted as $\boldsymbol{q} = [\boldsymbol{q}_{base}^T, \boldsymbol{q}_{arm}^T]^T$. To be more precise, the base configuration can be described as the 2D position coordinates $[x, y]$ in the horizontal plane and orientation θ, i.e., $\boldsymbol{q}_{base} =$

$[x, y, \theta]$. The arm configuration is composed of all joint angles, expressed as $\boldsymbol{q}_{arm} = [q_1, q_2, ..., q_{n_{arm}}]$, where n_{arm} is the number of degrees of freedom in the arm. In addition, for manipulators, the term Joint Space is interchangeably used to refer to the Configuration Space.

Task Space (TS)

In broad terms, Task Space refers to the space where a specific behavior is specified [27]. An example is defining it as the intended position and orientation of the end-effector (EE). An alternative instance is the avoidance of obstacles between a body connection and the surrounding environment. The analysis in this context focuses on the operational space of the end-effector. Typically, Cartesian coordinates are used to define the translational position of the end-effector in three-dimensional Euclidean space R^3, while the rotation group $SO(3)$ is employed to describe its orientation. Together, these components constitute the task space, represented mathematically as $R^3 \times SO(3) = SE(3)$ [10, 53].

2.2.2. Kinematic Fundamentals

Forward Kinematics (FK)

To determine the position and orientation of a point $\boldsymbol{p}$ attached to any link of a mobile manipulator, the function $\mathcal{K}$, known as the forward kinematics (FK) function, can be employed. This function maps the configuration $\boldsymbol{q}$ of the mobile manipulator to a pose $\boldsymbol{X_p} \in SE(3)$. The relationship is expressed as:

$$\boldsymbol{X_p} = \mathcal{K}_{\boldsymbol{p}}(\boldsymbol{q}), \tag{2.1}$$

where $\boldsymbol{X_p}$ corresponds to the pose of point $\boldsymbol{p}$ within the robot's kinematic chain, including its position and orientation.

Differential Kinematics (DK)

To obtain the velocity of the point $\boldsymbol{p}$ attached to a link of the mobile manipulator, differential kinematics (DK) can be utilized. This method establishes a relationship between the joint velocities $\boldsymbol{v} = [\dot{\boldsymbol{q}}_{base}^T, \dot{\boldsymbol{q}}_{arm}^T]^T$ and the velocity $\boldsymbol{\nu_p}$ of point $\boldsymbol{p}$ through the manipulator Jacobian $\boldsymbol{J}(\boldsymbol{q})$. The relationship is expressed as:

$$\boldsymbol{\nu_p} = \boldsymbol{J_p}(\boldsymbol{q})\boldsymbol{v}, \tag{2.2}$$

where $\boldsymbol{\nu_p} = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^T$ denotes the linear and angular velocity of point $\boldsymbol{p}$ in 3D space.

2.3. Multi-robot model predictive control

Model Predictive Control (MPC) is a family of advanced control techniques that use a model of the system to predict its future behavior over a defined horizon. At each time step, it solves a constrained optimization problem to determine the optimal control input $\boldsymbol{u}$, guiding the system toward its defined objectives. In this work, the MPC framework is tailored to enable the ego robot to reach target end-effector poses while avoiding collisions with other robots and obstacles in the environment. More specifically, the multi-robot MPC problem is formulated to optimize control inputs over a finite prediction horizon T, minimizing a cost function subject to system dynamics and constraints. This can be expressed formally as the following optimization problem:

$$\begin{aligned}
\min_{\boldsymbol{X},\boldsymbol{U}} \quad & \sum_{k=0}^{N-1} \frac{1}{2} L(\boldsymbol{x}_k, \boldsymbol{u}_k) + \frac{1}{2} L(\boldsymbol{x}_N), && \text{(2.3a)}\\
\text{subject to} \quad & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad \forall k = 0, 1, \ldots, N-1, && \text{(2.3b)}\\
& \boldsymbol{x}_0 = \boldsymbol{x}_{init}, && \text{(2.3c)}\\
& \boldsymbol{0} \leq \boldsymbol{d}(\boldsymbol{x}_k), \qquad \forall k = 0, 1, \ldots, N, && \text{(2.3d)}\\
& \underline{\boldsymbol{x}} \leq \boldsymbol{x}_k \leq \overline{\boldsymbol{x}}, \qquad \forall k = 0, 1, \ldots, N, && \text{(2.3e)}\\
& \underline{\boldsymbol{u}} \leq \boldsymbol{u}_k \leq \overline{\boldsymbol{u}}, \qquad \forall k = 0, 1, \ldots, N-1. && \text{(2.3f)}
\end{aligned}$$

where $\boldsymbol{X} = [\boldsymbol{x}_0^T, \cdots, \boldsymbol{x}_N^T]^T$ and $\boldsymbol{U} = [\boldsymbol{u}_0^T, \cdots, \boldsymbol{u}_{N-1}^T]^T$ are the sequences of state and input variables, respectively. $L(\cdot)$ is a time-varying stage cost. The optimization is subjected to the initial condition (Eq. 2.3c), system dynamics (Eq. 2.3b), collision avoidance constraints (Eq. 2.3d), state limits (Eq. 2.3e) and input limits (Eq. 2.3f).

The main components of this MPC formulation are outlined as follows. Section 2.3.1 discusses the system dynamics, followed by the end-effector tracking and regularization cost function in Section 2.3.2. Collision avoidance constraints are presented in Section 2.3.3, and the solver details are provided in Section 2.3.4.

2.3.1. System Model

Robot Model

The robot considered is a velocity-controlled mobile manipulator, with the state vector defined as $\boldsymbol{x}_r = [\boldsymbol{q}_r^T, \boldsymbol{v}_r^T]^T$. Here, $\boldsymbol{q}_r = [\boldsymbol{q}_{r_{base}}^T, \boldsymbol{q}_{r_{arm}}^T]^T$ represents the generalized configuration and the generalized velocity is given by $\boldsymbol{v}_r = [\dot{\boldsymbol{q}}_{r_{base}}^T, \dot{\boldsymbol{q}}_{r_{arm}}^T]^T$. The input $\boldsymbol{u}_r$ is taken to be acceleration $\boldsymbol{a}_r = [\ddot{\boldsymbol{q}}_{r_{base}}^T, \ddot{\boldsymbol{q}}_{r_{arm}}^T]^T$. This input is integrated to obtain the velocity commands sent to the actual robot. With $n_r = dim(\boldsymbol{q}_r)$ representing the dimension of the generalized configuration vector, the system dynamics are modelled as a second-order integrator within the prediction horizon of the Model Predictive Control, and are expressed as:

$$\dot{\boldsymbol{x}}_r = \boldsymbol{A}_r \boldsymbol{x}_r + \boldsymbol{B}_r \boldsymbol{u}_r, \tag{2.4}$$

where

$$\boldsymbol{A}_r = \begin{bmatrix} \mathbf{0}_{n_r \times n_r} & \boldsymbol{I}_{n_r \times n_r} \\ \mathbf{0}_{n_r \times n_r} & \mathbf{0}_{n_r \times n_r} \end{bmatrix} \in \mathbb{R}^{2n_r \times 2n_r}, \quad \boldsymbol{B}_r = \begin{bmatrix} \mathbf{0}_{n_r \times n_r} \\ \boldsymbol{I}_{n_r \times n_r} \end{bmatrix} \in \mathbb{R}^{2n_r \times n_r}. \tag{2.5}$$

Obstacles Model

Obstacles encountered during the planning process can be categorized into two types: static obstacles, whose positions remain constant throughout the entire process, and dynamic obstacles, which require updates to their states in real time. Dynamic obstacles may have known velocities, or they may simply require position updates. Static obstacles are straightforward to incorporate into the MPC framework by including their descriptions via a Unified Robot Description Format (URDF) file. In contrast, when dynamic obstacles are present, their states are integrated with the robot's states to facilitate collision avoidance and motion prediction. These dynamic obstacles are modelled as spheres with a known radius. Similar to the robot's state, the state of each obstacle i for $i = 1, 2, \ldots, n_o$, where n_o represents the total number of obstacles, comprises its position and velocity, represented as $\boldsymbol{x}_{o_i} = [\boldsymbol{q}_{o_i}^T, \boldsymbol{v}_{o_i}^T]^T$. Here, $\boldsymbol{q}_{o_i} = [x_{o_i}, y_{o_i}, z_{o_i}]$ denotes the obstacle's position in the global frame, while $\boldsymbol{v}_{o_i} = [\dot{x}_{o_i}, \dot{y}_{o_i}, \dot{z}_{o_i}]$ represents its translational velocity. It is important to note that the initial states of the obstacles are provided. To predict their future states along the prediction horizon, a constant velocity model is employed at each time step, which approximates their motion dynamics. In summary, the dynamic model of obstacle i can be described as $\dot{\boldsymbol{x}}_{o_i} = \boldsymbol{A}_o \boldsymbol{x}_{o_i}$, where $\boldsymbol{A}_{o_i} = \begin{bmatrix} \mathbf{0}_{3\times 3} & \boldsymbol{I}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \end{bmatrix}$.

System Model

In summary, for a mobile manipulator considering n_o dynamic obstacles, the augmented system state is

$$\boldsymbol{x} = [\boldsymbol{x}_r, \boldsymbol{x}_{o_1}, \ldots, \boldsymbol{x}_{o_{n_o}}] = \begin{bmatrix} \boldsymbol{q}_r^T & \boldsymbol{v}_r^T & \boldsymbol{q}_{o_1}^T & \boldsymbol{v}_{o_1}^T & \cdots & \boldsymbol{q}_{o_{n_o}}^T & \boldsymbol{v}_{o_{n_o}}^T \end{bmatrix}^T \in \mathbb{R}^{2n_r + 6n_o}. \tag{2.6}$$

The input $\boldsymbol{u}$ is the acceleration input $\boldsymbol{u}_r$ for the mobile manipulator. The dynamics of the entire system in Eq. 2.3a is described as:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \tag{2.7}$$

where

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_r & & & \\ & \boldsymbol{A}_{o_1} & & \\ & & \ddots & \\ & & & \boldsymbol{A}_{o_{n_o}} \end{bmatrix} \in \mathbb{R}^{(2n_r+6n_o)\times(2n_r+6n_o)}, \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_r \\ \mathbf{0}_{3\times 3} \\ \vdots \\ \mathbf{0}_{3\times 3} \end{bmatrix} \in \mathbb{R}^{(n_r+3n_o)\times n_r}. \tag{2.8}$$

To implement the MPC scheme, the continuous system dynamics in Eq. 2.7 needs to be discretized. The discrete-time model is derived from the continuous model using a discretization method, such as the backward Euler or Runge-Kutta method. As a result, the discretized state transition is expressed as:

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{2.9}$$

where k represents the time step index, and $f(\cdot)$ is the discrete state transition function.

2.3.2. Cost Function

The stage cost function $L(\boldsymbol{x}_k, \boldsymbol{u}_k)$ comprises the end-effector tracking error $\|\boldsymbol{\epsilon}(\boldsymbol{x}_k)\|^2_{\boldsymbol{W}_\epsilon}$, the state regularization cost $\|\boldsymbol{x}_k\|^2_{\boldsymbol{W}_x}$, and the control effort regularization cost $\|\boldsymbol{u}_k\|^2_{\boldsymbol{W}_u}$, with weighting matrices $\boldsymbol{W}_r$, $\boldsymbol{W}_x$, and $\boldsymbol{W}_u$, respectively. In summary, the stage cost function L can be expressed as:

$$L(\boldsymbol{x}_k, \boldsymbol{u}_k) = \|\boldsymbol{\epsilon}(\boldsymbol{x}_k)\|^2_{\boldsymbol{W}_\epsilon} + \|\boldsymbol{x}_k\|^2_{\boldsymbol{W}_x} + \|\boldsymbol{u}_k\|^2_{\boldsymbol{W}_u}. \tag{2.10}$$

The following sections describe the formulation of cost terms for end-effector tracking, as well as for state and input regularization.

Task Space Tracking Cost

The pose of the robot's end-effector, $\boldsymbol{r}_{ee}(\boldsymbol{q}_r) \in SE(3)$, is determined from the robot's full-body configuration $\boldsymbol{q}_r$ using the forward kinematics function $\mathcal{K}$. As discussed in Section 2.2.2, this function maps the robot's current configuration $\boldsymbol{q}_r$ to the position and orientation of the end-effector within the task space.

Deviations $\boldsymbol{\epsilon}(\boldsymbol{x}_k)$ in the end-effector pose $\boldsymbol{r}_{ee}$ from the reference pose $\hat{\boldsymbol{r}}_{ee}$ are penalized. Although the desired end-effector pose may vary over time, this work simplifies the scenario by assuming that the end-effector tracks a single constant reference point throughout the entire time horizon T. The end-effector pose deviation is defined as $\boldsymbol{\epsilon} = \begin{bmatrix}\boldsymbol{\epsilon}_{pos}^T & \boldsymbol{\epsilon}_O^T\end{bmatrix}^T \in \mathbb{R}^6$. For the translational error $\boldsymbol{\epsilon}_{pos} \in \mathbb{R}^3$, the difference between the current and desired position vectors is used:

$$\boldsymbol{\epsilon}_{pos} = \boldsymbol{r}_{pos} - \hat{\boldsymbol{r}}_{pos}. \tag{2.11}$$

To compute the rotational error $\boldsymbol{\epsilon}_O \in \mathbb{R}^3$, an orientation error formulation based on quaternions representing the current pose $\boldsymbol{\xi}$ and the desired pose $\hat{\boldsymbol{\xi}}$ is utilized [54]. The current pose is represented by the quaternion $\boldsymbol{\xi} = [\xi_w, \xi_x, \xi_y, \xi_z]$, where ξ_w is the scalar part, and the vector part is given by the components $[\xi_x, \xi_y, \xi_z]$. Similarly, the desired pose is represented by the quaternion $\hat{\boldsymbol{\xi}} = [\hat{\xi}_w, \hat{\xi}_x, \hat{\xi}_y, \hat{\xi}_z]$, where $\hat{\xi}_w$ indicates the scalar part, and $[\hat{\xi}_x, \hat{\xi}_y, \hat{\xi}_z]$ denotes the vector part of the desired quaternion.

The rotational error $\boldsymbol{\epsilon}_O$ is computed using the following formulation:

$$\boldsymbol{\epsilon}_O(\xi, \hat{\xi}) = \xi_w \cdot [\hat{\xi}_x, \hat{\xi}_y, \hat{\xi}_z]^T - \hat{\xi}_w \cdot [\xi_x, \xi_y, \xi_z]^T + [\hat{\xi}_x, \hat{\xi}_y, \hat{\xi}_z]^T \times [\xi_x, \xi_y, \xi_z]^T. \tag{2.12}$$

In summary, the cost function $\|\boldsymbol{\epsilon}(\boldsymbol{x}_k)\|^2_{\boldsymbol{W}_\epsilon}$ can be detailed as:

$$\|\boldsymbol{\epsilon}(\boldsymbol{x}_k)\|^2_{\boldsymbol{W}_\epsilon} = \boldsymbol{\epsilon}(\boldsymbol{x}_k)^T \boldsymbol{W}_\epsilon \boldsymbol{\epsilon}(\boldsymbol{x}_k) = \boldsymbol{\epsilon}_{pos}(\boldsymbol{x}_k)^T \boldsymbol{W}_{\epsilon_{pos}} \boldsymbol{\epsilon}_{pos}(\boldsymbol{x}_k) + \boldsymbol{\epsilon}_O(\boldsymbol{x}_k)^T \boldsymbol{W}_{\epsilon_O} \boldsymbol{\epsilon}_O(\boldsymbol{x}_k). \tag{2.13}$$

Moreover, in practice, there exists a trade-off between tracking cost and regularization cost. When the end-effector is far from the goal, a high regularization cost leads to smoother motion and reduced controller overshoot. Conversely, when the end-effector approaches the goal, a high regularization cost can dominate the cost function, resulting in slower progress and prolonging the time needed to reach the goal. To mitigate this issue, a dynamic scaling of the tracking cost weight is employed, denoted as α_r. Specifically, when the end-effector is within a certain threshold distance, represented as α_t, the tracking weight is increased to counterbalance the regularization cost, thereby accelerating convergence toward the goal. It is important to note that both α_r and α_t are constants that can be fine-tuned to optimize performance.

In summary, the dynamic tracking cost weight can be formulated as:

$$\boldsymbol{W}_\epsilon = \begin{cases} \boldsymbol{W}_\epsilon & \text{if} \quad \|\boldsymbol{\epsilon}_{pos}\|_2 > \alpha_t \\ \alpha_r \cdot \boldsymbol{W}_\epsilon & \text{if} \quad \|\boldsymbol{\epsilon}_{pos}\|_2 \le \alpha_t. \end{cases} \tag{2.14}$$

Regularization cost

In this work, the regularization term is formulated to penalize significant deviations in the system's states $\boldsymbol{x}_k$ and control inputs $\boldsymbol{u}_k$. Specifically, both the reference state $\boldsymbol{x}_{ref}$ and reference input $\boldsymbol{u}_{ref}$ are set to zero. It is crucial to note that the penalization primarily focuses on the velocity components of the state vector, while the deviations of the configuration component are not directly penalized because their associated cost is set to zero. This design choice effectively encourages the system to minimize excessive control efforts and large state deviations in velocity, thereby promoting smoother and more stable motion during operation. The regularization cost is defined as $\|\boldsymbol{x}_k\|^2_{\boldsymbol{W}_x} + \|\boldsymbol{u}_k\|^2_{\boldsymbol{W}_u}$, where $\boldsymbol{W}_x$ and $\boldsymbol{W}_u$ are weight matrices that determine the level of penalization for deviations in the states and control inputs, respectively.

As described in Section 2.3.1, the system state $\boldsymbol{x}$ includes both robot states and obstacle states. However, only the robot states are penalized in the regularization cost, as the obstacle states are used solely for collision avoidance. Therefore, the cost weights associated with the obstacle states are set to zero. With $\boldsymbol{W}_{x_r} \in \mathbb{R}^{2n_r \times 2n_r}$ representing the cost weight matrix for penalizing the robot's states, and $\boldsymbol{W}_{u_r} \in \mathbb{R}^{n_r \times n_r}$ for penalizing the inputs, the regularization cost matrices are given by:

$$\boldsymbol{W}_x = \begin{bmatrix} \boldsymbol{W}_{x_r} & & & \\ & \mathbf{0}_{6\times6} & & \\ & & \ddots & \\ & & & \mathbf{0}_{6\times6} \end{bmatrix} \in \mathbb{R}^{(2n_r+6n_o)\times(2n_r+6n_o)}, \quad \boldsymbol{W}_u = \boldsymbol{W}_{u_r} \in \mathbb{R}^{n_r \times n_r}. \tag{2.15}$$

Quadratic approximation of the cost function

To deploy Model Predictive Control on real-world robotic platforms, it is essential to ensure the optimization problem remains well-conditioned to avoid numerical issues. More specifically, for the Sequential Quadratic Programming (SQP) method, which is discussed in detail in Section 2.3.4, a positive semidefinite (p.s.d.) Hessian is used to ensure that the resulting quadratic program (QP) subproblems are convex and can be solved efficiently [55].

For regularization cost functions in Eq. 2.15, the second derivatives are represented by the weighting matrices $\boldsymbol{W}_x$ and $\boldsymbol{W}_u$, which can be made positive semidefinite through careful selection of their weight parameters. However, for the end-effector cost in Eq. 2.13, where the robot's forward kinematics are typically non-p.s.d., the second derivatives are approximated using the Gauss-Newton method. This method has proven effective in practice [31, 56], resulting in the following approximation for the Hessian:

$$\nabla^2_{\boldsymbol{x}\boldsymbol{x}} \left(\|\boldsymbol{\epsilon}(\boldsymbol{x})\|^2_{\mathbf{W}_\epsilon}\right) \approx 2\nabla_{\boldsymbol{x}}\boldsymbol{\epsilon}^T_{pos}\boldsymbol{W}_{\epsilon_{pos}}\nabla_{\boldsymbol{x}}\boldsymbol{\epsilon}_{pos} + 2\nabla_{\boldsymbol{x}}\boldsymbol{\epsilon}^T_{O}\boldsymbol{W}_{\epsilon_O}\nabla_{\boldsymbol{x}}\boldsymbol{\epsilon}_{O}. \tag{2.16}$$

2.3.3. Constraints

This subsection examines the constraints integral to the Model Predictive Control framework, including state and input limitations that maintain the system's physical boundaries and collision avoidance constraints that prevent interactions with obstacles.

State and Input Constraints

In addition to regularization costs, hard constraints on both input and state variables ensure the system operates within physical limits. These constraints are crucial for safe and reliable robot operation, as they enforce the hardware and environmental boundaries directly.

As defined in the system state equation Eq. 2.6, the overall state comprises both the robot's and obstacles' states. However, only the robot's states are subject to these limits. Therefore, the state constraints in Eq. 2.3e can be rewritten as:

$$\underline{\boldsymbol{x}}_r \leq \boldsymbol{x}_r \leq \overline{\boldsymbol{x}}_r, \tag{2.17}$$

where $\underline{\boldsymbol{x}}_r$ and $\overline{\boldsymbol{x}}_r$ represent the lower and upper bounds of the robot's state, respectively. For input constraints in Eq. 2.3f, no reformulation is necessary, since the system input is the same as the robot's input.

Collision Constraints

Self-collision and obstacle collision avoidance are achieved by ensuring that the robot maintains a distance greater than a minimal safety threshold from links on the ego-robot and obstacles. The robot is represented using primitive collision bodies, and GJK-based distance queries are utilized within FCL [57]. Let n_c denote the total number of collision pairs. The index i_c ranges over the set $\{1, 2, \ldots, n_c\}$.

For each i_c-th collision body pair, the shortest distance d_{i_c} is defined as the signed distance between the nearest points $\boldsymbol{p}_{i_c}^A$ and $\boldsymbol{p}_{i_c}^B$ in the inertial frame. This distance can be expressed as follows:

$$r_{i_c}(\boldsymbol{x}) = \pm\|\boldsymbol{p}_{i_c}^A(\boldsymbol{x}) - \boldsymbol{p}_{i_c}^B(\boldsymbol{x})\|_2. \tag{2.18}$$

A negative sign indicates colliding bodies, while a positive sign indicates non-overlapping cases. The vector $\boldsymbol{d}(\boldsymbol{x})$ captures the differences between the minimum distances of all pairs of collision spheres and safety thresholds ϵ_{i_c}. Each element of $\boldsymbol{d}(\boldsymbol{x})$ can be expressed as:

$$d_{i_c}(\boldsymbol{x}) = r_{i_c}(\boldsymbol{x}) - \epsilon_{i_c}. \tag{2.19}$$

Consequently, the collision avoidance constraint in Eq. 2.3d can be expressed as:

$$\boldsymbol{0} \leq \boldsymbol{d}(\boldsymbol{x}_k) = [d_1(\boldsymbol{x}_k), \dots, d_{i_c}(\boldsymbol{x}_k), \dots, d_{n_c}(\boldsymbol{x}_k)]. \tag{2.20}$$

Linear approximations of the constraints

The gradient of the distance function is necessary for formulating the quadratic program problem in the SQP framework. According to the approach outlined in [52, 58, 59], the gradient of the distance function with respect to the robot's state can be derived as:

$$\nabla_{\boldsymbol{x}} d_{i_c} = \pm \hat{\boldsymbol{n}}_{i_c}^T \left(\boldsymbol{J}_{i_c}^A - \boldsymbol{J}_{i_c}^B\right), \tag{2.21}$$

where $\boldsymbol{J}_{i_c}^A$ and $\boldsymbol{J}_{i_c}^B$ are the Jacobians of $\boldsymbol{p}_{i_c}^A$ and $\boldsymbol{p}_{i_c}^B$, respectively. The normal vector $\hat{\boldsymbol{n}}_{i_c}$ is defined as:

$$\hat{\boldsymbol{n}}_{i_c} = \frac{\boldsymbol{p}_{i_c}^A - \boldsymbol{p}_{i_c}^B}{\|\boldsymbol{p}_{i_c}^A - \boldsymbol{p}_{i_c}^B\|_2}. \tag{2.22}$$

2.3.4. Numerical Optimization

In nonlinear Model Predictive Control, all decision variables can be collected into a vector $\boldsymbol{w} = [\boldsymbol{X}^T, \boldsymbol{U}^T]^T$, where $\boldsymbol{X} = [\boldsymbol{x}_0, \dots, \boldsymbol{x}_N]$ and $\boldsymbol{U} = [\boldsymbol{u}_0, \dots, \boldsymbol{u}_{N-1}]$ represent the state and input sequences, respectively. The optimization problem can be formulated as a Nonlinear Programming Problem (NLP) problem. This formulation captures the cost to be minimized over the prediction horizon and the constraints to be satisfied. The general NLP formulation is:

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \phi(\boldsymbol{w}), \\ \text{subject to:} \quad & \boldsymbol{F}(\boldsymbol{w}) = \boldsymbol{0}, \\ & \boldsymbol{G}(\boldsymbol{w}) \leq \boldsymbol{0}, \end{aligned} \tag{2.23}$$

where $\phi(\boldsymbol{w})$ is the cost function, $\boldsymbol{F}(\boldsymbol{w}) \in \mathbb{R}^{n_F}$ represents the collection of initial state and system dynamics constraints, and $\boldsymbol{G}(\boldsymbol{w}) \in \mathbb{R}^{n_G}$ represents the collection of general inequality constraints.

Sequential Quadratic Programming (SQP)

To address the NLP problem, the Sequential Quadratic Programming (SQP) method is employed. It approximates the original nonlinear problem by iteratively solving a sequence of quadratic subproblems. Each subproblem optimizes a quadratic approximation of the Lagrangian while adhering to linearized constraints. In every iteration, the method leverages gradient information from the original problem to refine the solution [60]. The Lagrangian function for the general NLP problem is defined as:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = \phi(\boldsymbol{w}) + \boldsymbol{\lambda}^T \boldsymbol{F}(\boldsymbol{w}) + \boldsymbol{\nu}^T \boldsymbol{G}(\boldsymbol{w}), \tag{2.24}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{n_F}$ are the Lagrange multipliers associated with the equality constraints $\boldsymbol{F}(\boldsymbol{w}) = \boldsymbol{0}$, and $\boldsymbol{\nu} \in \mathbb{R}^{n_G}$ are the multipliers associated with the inequality constraints $\boldsymbol{G}(\boldsymbol{w}) \leq \boldsymbol{0}$.

At each optimization iteration n, the SQP method solves a quadratic subproblem that approximates the original nonlinear problem by minimizing a quadratic model of the Lagrangian, subject to linearized constraints. The quadratic subproblem is given by:

$$\min_{\delta \boldsymbol{w}_n} \quad \frac{1}{2}\delta \boldsymbol{w}_n^T \nabla^2_{\boldsymbol{w}\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_n, \boldsymbol{\lambda}_n, \boldsymbol{\nu}_n)\delta \boldsymbol{w}_n + \nabla_{\boldsymbol{w}} \phi(\boldsymbol{w}_n)^T \delta \boldsymbol{w}_n, \tag{2.25a}$$

$$\text{subject to:} \quad \nabla_{\boldsymbol{w}} \boldsymbol{F}(\boldsymbol{w}_n)^T \delta \boldsymbol{w}_n + \boldsymbol{F}(\boldsymbol{w}_n) = \boldsymbol{0}, \tag{2.25b}$$

$$\nabla_{\boldsymbol{w}} \boldsymbol{G}(\boldsymbol{w}_n)^T \delta \boldsymbol{w}_n + \boldsymbol{G}(\boldsymbol{w}_n) \leq \boldsymbol{0}, \tag{2.25c}$$

where $\delta \boldsymbol{w}_n$ represents the search direction, indicating the change in the decision variables. The term $\nabla^2_{\boldsymbol{w}\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_n, \boldsymbol{\lambda}_n, \boldsymbol{\nu}_n)$ denotes the Hessian of the Lagrangian with respect to $\boldsymbol{w}$, evaluated at the current iteration n. Additionally, $\nabla_{\boldsymbol{w}} \phi(\boldsymbol{w}_n)$ signifies the gradient of the objective function. Furthermore, the gradients $\nabla_{\boldsymbol{w}} \boldsymbol{F}(\boldsymbol{w}_n)$ and $\nabla_{\boldsymbol{w}} \boldsymbol{G}(\boldsymbol{w}_n)$ correspond to the gradients of the equality and inequality constraints, respectively.

After solving the quadratic subproblem, the new iterate $\boldsymbol{w}_{n+1}$ is updated using the equation $\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \alpha_n \delta \boldsymbol{w}_n$. In this equation, $\delta \boldsymbol{w}_n$ represents the search direction obtained from the subproblem solution, while α_n is the step size determined through a line search. The line search is formulated as

$$\alpha_n = \arg\min_{\alpha > 0} \phi(\boldsymbol{w}_n + \alpha \delta \boldsymbol{w}_n), \tag{2.26}$$

which aims to find the optimal step size that minimizes the objective function along the direction of $\delta \boldsymbol{w}_n$. Furthermore, the Lagrange multipliers $\boldsymbol{\lambda}_n$ and $\boldsymbol{\nu}_n$ are updated based on the dual variables obtained from the solution of the subproblem.

Sub-QP problem

In practice, when setting up the quadratic program problem in Eq. 2.25 for a solver, the Lagrangian multipliers $\boldsymbol{\lambda}_n$ and $\boldsymbol{\nu}_n$ are typically not explicitly included in the Hessian of the cost function provided to the QP solver [56]. To ensure efficient numerical solvability, it is crucial that the optimization problem is well-conditioned. Specifically, when the Hessian in Eq.2.25a is positive semidefinite (p.s.d.), the resulting quadratic program becomes convex, enabling efficient solution [55]. As discussed in Section 2.10, this condition is satisfied by using an approximate positive semidefinite Hessian for the tracking cost in Eq. 2.16, instead of the full Hessian of the Lagrangian. Moreover, soft constraints are employed to relax all constraints except for the system dynamics. For a general inequality constraint $g(\boldsymbol{x}, \boldsymbol{u}) \leq 0$, a slack variable $s \geq 0$ is introduced as an additional decision variable. This modifies the inequality to $g(\boldsymbol{x}, \boldsymbol{u}) \leq s$, allowing the system to handle constraint violations more flexibly. To minimize violations, a penalty term for s is added to the objective function in Eq. 2.25a. Specifically, an L_2-norm penalty of the form $w_s s^2$ is introduced for each slack variable, where $w_s > 0$ is a tunable weight that controls the trade-off between constraint violation and optimality.

2.4. State Estimation

Accurate state estimation is essential for effective motion planning, particularly for both the ego robot and other non-ego robots. In a simulation environment, states can be obtained directly from the physics engine, simplifying the process. However, in real-world applications, accurately estimating the robot's state is more challenging due to factors such as sensor noise. Precise knowledge of the robots' states is crucial for safe operation and collision detection, as this information is provided to the controller and processed through forward kinematics to determine the positions of potential collision pairs. Additionally, state information is necessary to ensure that the robot operates within its physical limits, preventing damage and maintaining operational safety.

In this section, the methods for real-world state estimation for both the ego robot and the non-ego agents are detailed. A total of m robots are considered, represented by the set $R = \{1, 2, \dots, m\}$. In this set, the ego robot is indexed by i, while the other robots are represented by the index j in the set $R \setminus \{i\}$. For the analysis, one robot, denoted as j, is selected from this set of non-ego robots. Specifically, Section 2.4.1 introduces the use of a Kalman Filter (KF) for state estimation of the ego robot i, while Section 2.4.2 covers the numerical differential method for estimating the states of the selected non-ego robot j.

2.4.1. State Estimation for the Ego-Robot Using a Kalman Filter (KF)

A Kalman Filter [61] is employed for state estimation in the hardware experiments to estimate the ego robot's state vector $\boldsymbol{x}_r = [\boldsymbol{q}_r^T, \boldsymbol{v}_r^T]^T$. The Kalman filter is a recursive algorithm that combines predictions from a system model with noisy sensor measurements, providing an updated estimate of the state and reducing uncertainty over time. The Kalman filter operates in two main phases:

- **Prediction step**: In this step, the current state estimate is propagated forward using the robot's dynamic model, represented by the discrete-time equation:

  $$\boldsymbol{x}_r^+ = \bar{\boldsymbol{A}}_r \boldsymbol{x}_r + \bar{\boldsymbol{B}}_r \boldsymbol{u}, \tag{2.27}$$

  where $\bar{\boldsymbol{A}}_r$ and $\bar{\boldsymbol{B}}_r$ are the system matrices derived from the Taylor series expansion of the continuous-time dynamics. These matrices are defined as:

  $$\bar{\boldsymbol{A}}_r = \begin{bmatrix} \boldsymbol{I}_{n_r} & \delta t \boldsymbol{I}_{n_r \times n_r} \\ \boldsymbol{0}_{n_r \times n_r} & \boldsymbol{I}_{n_r \times n_r} \end{bmatrix} \in \mathbb{R}^{2n_r \times 2n_r}, \quad \bar{\boldsymbol{B}}_r = \begin{bmatrix} \frac{1}{2}\delta t^2 \boldsymbol{I}_{n_r \times n_r} \\ \delta t \boldsymbol{I}_{n_r \times n_r} \end{bmatrix} \in \mathbb{R}^{2n_r \times n_r}. \tag{2.28}$$

  Here, δt is the sampling time for each iteration of the robot control loop. The prediction step also updates the associated uncertainty through the process covariance matrix $\bar{\boldsymbol{Q}}$, which accounts for uncertainties in the system model due to process noise:

  $$\bar{\boldsymbol{Q}} = \bar{\boldsymbol{B}}_r \boldsymbol{Q} \bar{\boldsymbol{B}}_r^T, \tag{2.29}$$

  where $\boldsymbol{Q}$ represents the process noise covariance. The predicted state covariance is updated as:

  $$\boldsymbol{P}_r^+ = \bar{\boldsymbol{A}}_r \boldsymbol{P}_r \bar{\boldsymbol{A}}_r^T + \bar{\boldsymbol{Q}}. \tag{2.30}$$

- **Update step**: After obtaining new sensor data z, the predicted state is updated using the observation model:

  $$\boldsymbol{z} = \boldsymbol{C}\boldsymbol{x}_r, \tag{2.31}$$

  where $\boldsymbol{C}$ is the matrix that maps the state vector to the sensor measurements:

  $$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I}_{n_r \times n_r} & \boldsymbol{0}_{n_r \times n_r} \end{bmatrix} \in \mathbb{R}^{n_r \times 2n_r}. \tag{2.32}$$

  The Kalman gain $\boldsymbol{K}$ is then calculated to weigh the predicted state and the measurements, based on their respective uncertainties:

  $$\boldsymbol{K} = \boldsymbol{P}_r^+ \boldsymbol{C}^T (\boldsymbol{C}\boldsymbol{P}_r^+ \boldsymbol{C}^T + \boldsymbol{R})^{-1}, \tag{2.33}$$

  where $\boldsymbol{R}$ is the measurement covariance matrix that represents the uncertainty in the sensor readings. The Kalman gain $\boldsymbol{K}$ determines how much the filter relies on the sensor measurements $\boldsymbol{z}$ relative to the predicted state $\boldsymbol{x}_r^+$, based on their respective uncertainties. The updated state estimate is computed as:

  $$\boldsymbol{x}_r = \boldsymbol{x}_r^+ + \boldsymbol{K}(\boldsymbol{z} - \boldsymbol{C}\boldsymbol{x}_r^+), \tag{2.34}$$

  where $\boldsymbol{z} - \boldsymbol{C}\boldsymbol{x}_r^+$ is the innovation term, representing the difference between the actual measurements and the predicted measurements. This equation corrects the predicted state by blending it with the new sensor measurements, weighted by the Kalman gain. Finally, the state covariance is updated to reflect the reduced uncertainty after incorporating the measurements:

  $$\boldsymbol{P}_r = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{C})\boldsymbol{P}_r^+. \tag{2.35}$$

In this work, the ego-robot's state is measured through two sources: the mobile base pose is captured using a Vicon motion capture system, and the robot arm's joint angles are measured using joint encoders. These measurements are fused with the system model through the Kalman Filter to continuously refine the robot's state estimate over time.

2.4.2. State Estimation for the Non-Ego Robots

Accurate knowledge of the full state vector of the non-ego robot $\boldsymbol{x}_{or} = [\boldsymbol{q}_{or}^T, \boldsymbol{v}_{or}^T]^T$ is necessary for collision avoidance. This state vector includes the whole-body configuration $\boldsymbol{q}_{or}$ and velocity $\boldsymbol{v}_{or}$, which are used to model the positions and velocities of obstacles representing the non-ego robot. The base configuration and arm joint positions $\boldsymbol{q}_{or}$ are obtained from the Vicon motion capture system and arm encoders, respectively. The arm joint velocities $\boldsymbol{v}_{or}^{arm}$ are measured by the encoders, while the base velocity $\boldsymbol{v}_{or}^{base}$ is estimated via numerical differentiation of the base's position data. For simplicity, the index j for the non-ego robot and the subscripts for the base configuration $\boldsymbol{q}_{or}^{base}$ and velocity $\boldsymbol{v}_{or}^{base}$ are omitted in the following equations. The estimated base velocity is expressed as:

$$\boldsymbol{v}_{measured} = \frac{\boldsymbol{q} - \boldsymbol{q}_{prev}}{\Delta t}, \tag{2.36}$$

where $\boldsymbol{q}$ represents the current configuration of the robot's base, $\boldsymbol{q}_{prev}$ is the previous configuration, and Δt denotes the time interval between consecutive measurements. To mitigate the noise introduced by numerical differentiation, an exponential smoothing filter is applied, described by the following equation:

$$\boldsymbol{v}_{filtered} = (1 - \beta)\boldsymbol{v}_{prev} + \beta \boldsymbol{v}_{measured}. \tag{2.37}$$

In this equation, β represents the smoothing factor, $\boldsymbol{v}_{filtered}$ denotes the filtered base velocity, and $\boldsymbol{v}_{prev}$ is the previously filtered velocity. The smoothing factor β is determined using the relationship for both linear and angular velocities:

$$\beta = 1 - e^{-\frac{\Delta t}{\tau}}. \tag{2.38}$$

Here, τ is the time constant for velocity smoothing. Different time constants τ_{linear} and $\tau_{angular}$ are employed for linear and angular velocities.

2.4.3. Collision Avoidance in Multi-agent Settings

In the formulation of multi-robot Model Predictive Control, as presented in Section 2.3, the obstacle states $\boldsymbol{x}_{o_i} = \begin{bmatrix}\boldsymbol{q}_{o_i}^T & \boldsymbol{v}_{o_i}^T\end{bmatrix}^T$ are assumed to be known. This section details how these states are derived based on the ego-robot and non-ego robot states obtained in Section 2.4.1 and Section 2.4.2.

In robotic systems, efficient collision detection is crucial for ensuring safe motion. One common approach to simplify this process is to approximate the geometry of the robot's links using bounding spheres, rather than relying on the robot's actual geometric shapes. This method significantly reduces computational complexity, as calculating distances or detecting intersections between spheres is far simpler and faster than performing these tasks on arbitrary shapes [62]. By using spheres to represent the robot's links, the collision detection process becomes more efficient, especially in dynamic environments with numerous moving parts or obstacles.

Assuming a collision sphere i is attached to a robot link l by its centre, the state of the sphere is represented as $\boldsymbol{x}_{o_i} = \begin{bmatrix}\boldsymbol{q}_{o_i}^T & \boldsymbol{v}_{o_i}^T\end{bmatrix}^T$. Once the full-body state of the robot, $\boldsymbol{x} = \begin{bmatrix}\boldsymbol{q}^T, \boldsymbol{v}^T\end{bmatrix}^T$, is obtained, the position $\boldsymbol{q}_{o_i}$ and velocity $\boldsymbol{v}_{o_i}$ of the sphere can be determined using forward kinematics (FK) and differential kinematics (DK), as outlined in Section 2.2.2. For example, for spheres representing the non-ego robot, the configuration and velocity are given by $\boldsymbol{q}_{o_i} = \mathcal{K}_{o_i}(\boldsymbol{q}_{or})$ and $\boldsymbol{v}_{o_i} = J_{o_i}(\boldsymbol{q}_{or})\boldsymbol{v}_{or}$.

2.5. A Prioritized Heuristic to Solve Livelocks

In multi-agent robotic systems, deadlocks and livelocks can lead to inefficiencies and delays, especially when robots fail to resolve conflicts in their trajectories or task assignments. For multiple mobile manipulators, such challenges often arise during navigation to distant targets for object delivery or when end-effectors operate in close proximity for grasping or placing objects. This thesis addresses livelocks in object delivery scenarios, which frequently occur when robots approach each other. For instance, in a two-robot case, the slightly faster robot may inadvertently hinder the slower robot's progress by pushing it away from its target during collision avoidance. To address this, a prioritized heuristic is proposed, leveraging the inherent prediction horizon of the decentralized MPC framework. The heuristic is structured around livelock detection and resolution, as summarized in Algorithm 1.

The livelock detection criterion, outlined in line 5 of Algorithm 1, comprises two conditions. First, robots' end-effectors should be within a distance threshold $d_{ee,l}$ of each other, as livelocks are more likely to

Algorithm 1 Task Execution Loop with the Heuristic

```
1: Input: $(\hat{\boldsymbol{r}}_{ee_1}, \hat{\boldsymbol{r}}_{ee_2})$                    ▷ End-effector reference targets
2: goal_reached ← False
3: while NOT goal_reached do
4:     $\boldsymbol{r}_{ee_1}(t), \boldsymbol{r}_{ee_2}(t)$                       ▷ Get current end-effector poses
5:     livelock = Eq. 2.40                                  ▷ Detect livelock
6:     if livelock then
7:         if $\|\hat{\boldsymbol{r}}_{pos_1} - \boldsymbol{r}_{pos_1}(t)\|_2 < \|\hat{\boldsymbol{r}}_{pos_2} - \boldsymbol{r}_{pos_2}(t)\|_2$ then
8:             publish$(\hat{\boldsymbol{r}}_{ee_1}, \boldsymbol{r}_{ee_2}(t))$        ▷ Reset robot-2 target and let it standstill
9:         else
10:            publish$(\boldsymbol{r}_{ee_1}(t), \hat{\boldsymbol{r}}_{ee_2})$        ▷ Reset robot-1 target and let it standstill
11:        end if
12:    end if
13:    if $\|\boldsymbol{r}_{pos_1}(t) - \boldsymbol{r}_{pos_2}(t)\|_2 > d_{ee,r})$ then
14:        livelock ← False                                 ▷ Livelock solved
15:        publish$(\hat{\boldsymbol{r}}_{ee_1}, \hat{\boldsymbol{r}}_{ee_2})$                 ▷ Publish original targets
16:    end if
17:    if $\|\hat{\boldsymbol{r}}_{pos_1} - \boldsymbol{r}_{pos_1}(t)\|_2 < d_{ee,\hat{r}} \wedge \|\hat{\boldsymbol{r}}_{pos_2} - \boldsymbol{r}_{pos_2}(t)\|_2 < d_{ee,\hat{r}}$ then
18:        goal_reached ← True
19:    end if
20: end while
```

occur when robots operate in close proximity. Furthermore, the MPC framework, which continuously predicts the trajectories of dynamic obstacles representing non-ego robots with constant velocity, generates preemptive policies that slow down the ego-robot to avoid collisions. This decrease in velocity occurs before the actual clearance between any collision pair reaches the minimum pairwise distance threshold ϵ_{i_c} defined in the collision avoidance constraint in Section 2.3.3. Given this behavior, the heuristic evaluates the velocity $\boldsymbol{v}_i$ for each robot i, defined as the rate of change in the Euclidean distance between the robot's end-effector position $\boldsymbol{r}_{pos,i}(t)$ and its target position $\hat{\boldsymbol{r}}_{pos,i}$. This measure quantifies the robot's progress toward its target without the knowledge of the optimized trajectory within the whole prediction horizon. The average velocity $\bar{\boldsymbol{v}}_i$ over a specified past duration T_l is computed as:

$$\bar{\boldsymbol{v}}_i = \frac{1}{n_v} \sum_{k=1}^{n_v} \frac{\|\boldsymbol{r}_{pos_i}(t_k) - \hat{\boldsymbol{r}}_{pos_i}\|_2 - \|\boldsymbol{r}_{pos_i}(t_{k-1}) - \hat{\boldsymbol{r}}_{pos_i}\|_2}{t_k - t_{k-1}}, \tag{2.39}$$

where n_v is the number of velocity values recorded at discrete time steps within T_l, and $t_k - t_{k-1}$ is the time difference between consecutive steps. In summary, in a two-robot scenario, a livelock is detected if the following condition holds:

$$(\|\boldsymbol{r}_{pos_1}(t) - \boldsymbol{r}_{pos_2}(t)\|_2 < d_{ee,l}) \wedge (\bar{\boldsymbol{v}}_1 > \bar{\boldsymbol{v}}_l \vee \bar{\boldsymbol{v}}_2 > \bar{\boldsymbol{v}}_l), \tag{2.40}$$

where $\bar{\boldsymbol{v}}_l$ is a predefined negative velocity threshold. Since $\bar{\boldsymbol{v}}_l$ represents the decrease in distance to the target, a value exceeding this threshold indicates insufficient progress toward the target, or even positive, which would mean the robot is moving away from its target. This condition effectively identifies livelocks where robots remain close together but fail to make meaningful progress toward their goals. The duration T_l represents a sufficient time window over which the robot's behavior is monitored to ensure that any decresed velocity to the target persists long enough to be indicative of a livelock rather than a temporary adjustment.

After a livelock is detected, as detailed in lines 6–12 of Algorithm 1, a hierarchy is established based on the proximity of the robots to their respective targets. The robot whose end-effector is closer to its target is assigned a higher priority and allowed to proceed first. Conversely, the robot whose end-effector is farther from its target is assigned a lower priority. The lower-priority robot temporarily resets its target to its current end-effector pose, $\boldsymbol{r}_{ee}(t)$, and remains stationary. When the distance between the end-effectors exceeds the specified threshold $d_{ee,r}$, the robots are considered sufficiently separated, and the livelock is resolved, as shown in lines 13-14 of Algorithm 1. The lower-priority robot then restores its original target $\hat{r}_{ee}$ and continues moving toward it. The task is considered finished when the distances to targets of both robots' end-effector are within threshold $d_{ee,\hat{r}}$.

2.6. Summary of the Methodology Pipeline

At the conclusion of this chapter, the pipeline of the proposed framework is summarized, emphasizing the interaction between the key components for motion planning in multi-agent settings. This overview demonstrates how multi-robot Model Predictive Control, state estimation, and a heuristic for livelock resolution work together to ensure efficient, collision-free operation in shared environments. The following diagram Fig. 2.1, as an example for two robots, provides a clear visual representation of the sequence of processes involved, with key input and output variables indicated for each block.

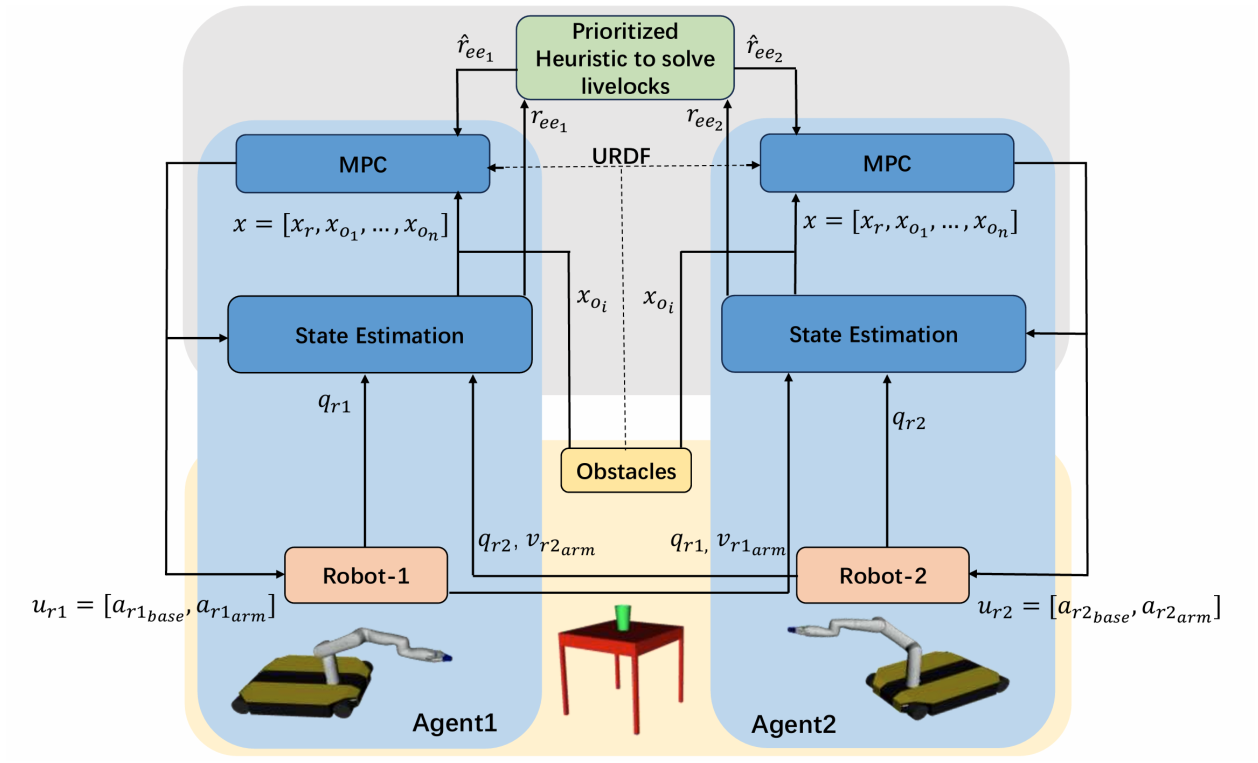


Figure 2.1: Overview of the multi-robot MPC and a pioritized heuristic in a multi-agent settings

3 Simulation experiments

This section describes all simulation experiments used to validate and benchmark the proposed MPC with a double integrator dynamic model (denoted as MPC-d). The quantitative experiments compare the proposed method with Geometric Fabrics (GF) [27] and the original approach, MPC with a triple integrator dynamic model [14] (denoted as MPC-t). Differences between optimization-based and geometric-based methods are highlighted, along with the benefits of the proposed approach. Additionally, the variation in computation time for GF, MPC-d, and MPC-t as the number of obstacles increases is analyzed.

The section begins with an overview of the experimental framework in Section 3.1, including the MPC hyperparameters. This is followed by a comparative study of the performance of MPC-d, MPC-t and GF in various realistic pick-and-place scenarios in Section 3.2. Finally, Section 3.3 presents a detailed comparison of the computation times.

3.1. Experimental Framework

Simulations are conducted to evaluate the controller's performance in an idealized environment using the simulated platform Pybullet, comprising a 9-DOF mobile manipulator with a Clearpath Dingo omnidirectional base and a Kinova GEN3 Lite arm. As detailed in Section 2.3.4, the MPC formulation described in Eq.2.3 is solved using Sequential Quadratic Programming (SQP) implemented in the Optimal Control of Switched Systems (OCS2) toolbox, with QP subproblems solved by the HPIPM [63]. The automatic differentiation library CppAD [64] is used to obtain Jacobians, and GJK-based distance queries in Flexible Collision Library (FCL) [57] enables collision detection. The simulation experiments are conducted on a desktop system with a Dell Precision 5820 Tower, featuring an Intel Xeon W-2123 processor running at 3.6 GHz with eight cores, and 8 GB of RAM.

Table 3.1 summarizes the parameters used for the MPC-d and MPC-t. The general parameters, such as sampling time, prediction horizon, tracking error weights, slack weights, safety thresholds and physical limits are shared across both MPC methods. The table also lists parameters unique to each controller, such as reference state weights and control input weights which are tailored to their respective dynamic models. Notably, the regularization costs for the arm's velocity are set high to mitigate overshooting during goal tracking, especially along the z-axis. Specifically, for MPC-t, the state upper boundaries are defined as $\overline{\boldsymbol{x}}_r = [\overline{\boldsymbol{q}}_r, \overline{\boldsymbol{v}}_r, \overline{\boldsymbol{a}}_r] = [\overline{\boldsymbol{q}}_{base}, \overline{\boldsymbol{q}}_{arm}, \overline{\boldsymbol{v}}_{base}, \overline{\boldsymbol{v}}_{arm}, \overline{\boldsymbol{a}}_{base}, \overline{\boldsymbol{a}}_{arm}]$, while for MPC-d, they are given by $\overline{\boldsymbol{x}}_r = [\overline{\boldsymbol{q}}_r, \overline{\boldsymbol{v}}_r] = [\overline{\boldsymbol{q}}_{base}, \overline{\boldsymbol{q}}_{arm}, \overline{\boldsymbol{v}}_{base}, \overline{\boldsymbol{v}}_{arm}]$. In both MPC-based controllers, the lower boundaries are defined as $\underline{\boldsymbol{x}}_r = -\overline{\boldsymbol{x}}_r$, $\underline{\boldsymbol{u}} = -\overline{\boldsymbol{u}}$. For GF, the physical limits, including configuration, velocity, and acceleration boundaries, are the same as those used in the MPC. This consistency ensures that all controllers operate within the same dynamic and physical constraints.

The robot's states x_r are directly obtained from the simulation environment. These states are either provided as inputs to the ego robot's controller or used for forward kinematics (FK) and differential kinematics (DK) to derive the states of obstacles x_{o_i} representing non-ego robots. To assess collision avoidance, the ego-robot is approximated by 5 collision spheres and the non-ego robot is simplified using 3 collision spheres. Their parent links and radii are detailed in Table 3.2. In practice, the Pinocchio library [65] is used to efficiently compute the positions and velocities of the collision spheres attached to specific robot links by leveraging a kinematic model derived from the robot's URDF file and the full-body states.

Table 3.1: Model Predictive Control Parameter Settings

(a) Common MPC Parameter Settings

Parameter	Symbol	Value
Sampling Time	Δt	$0.1\,\mathrm{s}$
Prediction Horizon	T	$2\,\mathrm{s}$
Tracking Error Weights	$\boldsymbol{W}_\epsilon$	$diag(1.5, 1.5, 5, 2, 2, 2) \in \mathbb{R}^{6\times6}$
Slack Weights	w_s	100
Base Configuration Limits	$\overline{\boldsymbol{q}}_{base}$	$[10\mathrm{m}, 10\mathrm{m}, 10\mathrm{rad}]$
Arm Configuration Limits	$\overline{\boldsymbol{q}}_{arm}$	$[2.69, 1.76, 2.62, 2.62, 2.53, 2.60]\,\mathrm{rad}$
Base Velocity Limits	$\overline{\boldsymbol{v}}_{base}$	$[0.3\mathrm{m/s}, 0.3\mathrm{m/s}, 0.5\mathrm{rad/s}]$
Arm Velocity Limits	$\overline{\boldsymbol{v}}_{arm}$	$[0.4, 1.1, 1.1, 1, 1, 1]\,\mathrm{rad/s}$
Safety Threshold	ϵ_{i_c}	$0.1\,\mathrm{m}$

(b) MPC with Triple Integrator as Dynamic Model

Parameter	Symbol	Value
Reference State Weights	$\boldsymbol{W}_{x_r}$	$diag(0, 0, 0, 2, 0, 0, 0, 0, 0, 4, 4, 2, 3, 7, 4, 3, 3, 3,$ $0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01)$
Base Acceleration Limits	$\overline{\boldsymbol{a}}_{base}$	$\left[2.5\mathrm{m/s^2}, 2.5\mathrm{m/s^2}, 1\mathrm{rad/s^2}\right]$
Arm Acceleration Limits	$\overline{\boldsymbol{a}}_{arm}$	$[5, 5, 5, 9, 9, 9]\,\mathrm{rad/s^2}$
Control Input Weights	$\boldsymbol{W}_u$	$0.001\,\boldsymbol{I}_9$
Base Input Limits	$\overline{\boldsymbol{u}}_{base}$	$[20\mathrm{m/s^3}, 20\mathrm{m/s^3}, 20\mathrm{rad/s^3}]$
Arm Input Limits	$\overline{\boldsymbol{u}}_{arm}$	$[80, 80, 80, 80, 80, 80]\mathrm{rad/s^3}$

(c) MPC with Double Integrator as Dynamic Model

Parameter	Symbol	Value
Reference State Weights	$\boldsymbol{W}_{x_r}$	$diag(0, 0, 0, 2, 0, 0, 0, 0, 0, 4, 4, 2, 3, 7, 4, 3, 3, 3)$
Control Input Weights	$\boldsymbol{W}_u$	$0.1\,\boldsymbol{I}_9$
Base Input Limits	$\overline{\boldsymbol{u}}_{base}$	$\left[2.5\mathrm{m/s^2}, 2.5\mathrm{m/s^2}, 1\mathrm{rad/s^2}\right]$
Arm Input Limits	$\overline{\boldsymbol{u}}_{arm}$	$[5, 5, 5, 9, 9, 9]\,\mathrm{rad/s^2}$
Dynamic Scale	α_t	5
Threshold Distance	α_d	$0.5\,\mathrm{m}$

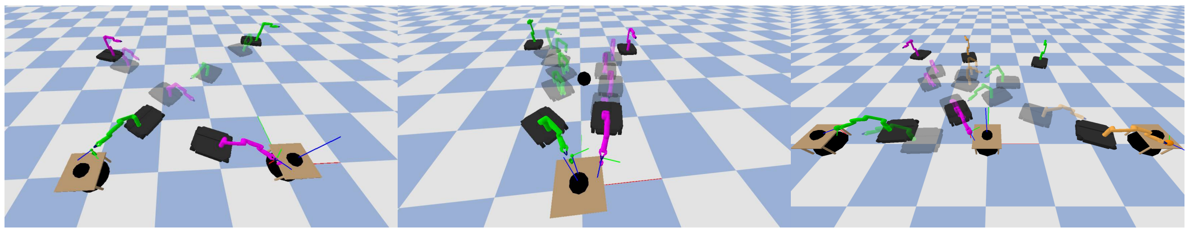
Table 3.2: Collision link names, corresponding parent link names, offsets (in meters) to their parent links, and sphere radii (in meters) for both the agent and obstacle robots.

Ego Robot			
Collision Link Name	**Parent Link Name**	**Offset** $[\mathrm{m}]$	**Sphere Radius** $[\mathrm{m}]$
`tool_frame_collision_link`	`tool_frame`	[0,0,0]	0.10
`upper_wrist_collision_link`	`upper_wrist_link`	[0,0,0]	0.10
`lower_wrist_collision_link`	`lower_wrist_link`	[0,0,0]	0.10
`forearm_collision_link`	`forearm_link`	[0,0,0]	0.10
`base_collision_link`	`base_link`	[0,0,0]	0.45
Non-Ego Robot			
Collision Link Name	**Parent Link Name**	**Offset** $[\mathrm{m}]$	**Sphere Radius** $[\mathrm{m}]$
`tool_frame_collision_link`	`tool_frame`	[0,0,0]	0.20
`lower_wrist_collision_link`	`lower_wrist_link`	[0,0,0]	0.17
`base_collision_link`	`base_link`	[0,0,0]	0.50

3.2. Experiment 1: Simulated pick-and-place scenarios

3.2.1. Setup

The proposed method is evaluated across three distinct realistic pick-and-place scenarios and compared against two baseline methods: GF [32] and the original MPC-t approach [14]. In each scenario, the robots' end-effectors are required to reach a static pre-grasp target in $SE(3)$, encompassing both position and orientation. Each scenario is simulated in 30 randomized environments, with variations in the robots' initial configurations and the positions of obstacles and objects, as illustrated in Fig. 3.1. The starting positions of the holonomic bases are uniformly sampled within the range $[x, y, \theta] \in [-3.0, 3.0]\,\mathrm{m} \times [2.0, 5.0]\,\mathrm{m} \times [-2.0, 2.0]\mathrm{rad}$, with the arm initialized in a fixed home configuration $\boldsymbol{q}_{arm}^{home} = [0.0, 0.0, 1.54, 0.0, 0.0, 0.0]\mathrm{rad}$. The tables are positioned near the origin of the world frame at $[x, y, z] = [-3.0, 0.0, 0.0]\,\mathrm{m}$, $[0.0, 0.0, 0.0]\,\mathrm{m}$, and $[3.0, 0.0, 0.0]\,\mathrm{m}$, and each is modelled as two vertically stacked collision spheres to capture the key spatial constraints for collision avoidance. Object poses are uniformly sampled within an annular region around the centre of the table, defined by an inner radius of 0.2 m and an outer radius of 0.3 m. Each robot is assigned to reach a static target pose, determined based on the object's position. Task success is defined as all agents reaching their respective target positions within a task-specific Euclidean distance tolerance without any collisions occurring. For comparison with the baseline methods, success rate, time to reach the goal, computation time, and collision rate are evaluated. Collision violations are identified by checking for contact between all rigid bodies at each timestep by using the collision check functionality of the Pybullet physics engine.



(a) Scenario 1 (S1): Two tables **(b)** Scenario 2 (S2): One table **(c)** Scenario 3 (S3): Three robots

Figure 3.1: Illustration of three simulated scenarios. In each scenario, the robots' end-effector are required to reach a static pre-grasp target in $SE(3)$. The grasp target is represented by a marker placed next to the generated cup, whose color matches that of the corresponding robot's arm. The robots are tasked with achieving their targets while avoiding collisions with each other and static obstacles.

3.2.2. Results

Table 3.3 presents the performance of GF, MPC-d, and MPC-t across three scenarios, evaluated using metrics such as success rate, time-to-success, computation time, and collision rate. Specifically, MPC-d exhibits performance comparable to the state-of-the-art GF in terms of success rate and collision rate across all scenarios. Regarding computation time, although MPC-d incurs higher computational costs compared to GF, it demonstrates a significant reduction relative to MPC-t. This efficiency allows MPC-d to maintain real-time performance across all scenarios, highlighting its potential for application in real-world settings.

Table 3.3: Statistics for three different scenarios of the proposed method are presented, compared against the whole-body Geometric Fabrics (GF) and MPC with a triple integrator dynamic model (denoted as MPC-t). Each scenario is randomized across 30 different environments.

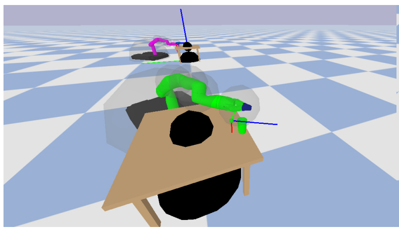
Scenario	Method	Success rate [%]	Time-to-Success [s]	Computation time [ms]	Collision-rate [%]
Two tables	GF	63.3	23.09 $\pm$ 7.96	1.30 $\pm$ 0.29	6.7
	MPC-t	70.0	15.92 $\pm$ 4.85	35.39 $\pm$ 2.68	6.7
	MPC-d	80.0	12.85 $\pm$ 4.35	16.23 $\pm$ 1.46	6.7
One table	GF	50.0	23.53 $\pm$ 3.92	1.27 $\pm$ 0.26	0
	MPC-t	53.3	17.10 $\pm$ 5.33	34.91 $\pm$ 2.3	0
	MPC-d	80.0	13.66 $\pm$ 4.68	16.02 $\pm$ 1.35	0
Three agents	GF	46.7	45.74 $\pm$ 12.50	1.456 $\pm$ 0.33	10.0
	MPC-t	70.0	22.73 $\pm$ 8.84	99.33 $\pm$ 5.07	10.0
	MPC-d	70.0	15.89 $\pm$ 4.07	38.13 $\pm$ 2.08	3.3

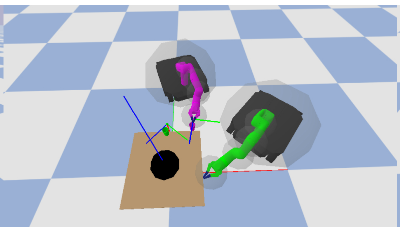
In Scenario 1, mobile manipulators are tasked with picking up a cup placed on two separate tables. Some scenarios require the robots to cross paths while avoiding collisions, as depicted in Fig. 3.1a. The grasp target poses are generated randomly, often near the center of the tables, necessitating the robotic arms to extend to reach the goal. The MPC-t and MPC-d methods achieve success rates of 70.0% and 80.0%, respectively, slightly outperforming GF, which achieves a success rate of 63.3%. Failures occur when the robot's end-effector cannot reach its target due to collision avoidance constraints, often involving the robot's upper wrist link and the table, as shown in Fig. 3.2a. Collisions are observed across all methods, particularly in scenarios where the robots' trajectories intersect. For GF, inter-robot or self-collisions are the primary causes, as illustrated in Fig. 3.2e and Fig. 3.2f. Conversely, the MPC-based methods primarily experience collisions between robots and tables, typically during livelock situations where one robot's arm blocks another's path to its goal, as seen in Fig. 3.2d. Although MPC-based methods maintain a minimum distance between the robot's collision spheres and the table's bounding spheres, the corners of the tables are not adequately captured by the current collision models. To mitigate such issues, a heuristic approach, as discussed in Section 2.5, could resolve livelocks by managing the robots' navigation priorities before reaching their goals. Additionally, future work should focus on improving obstacle modelling to enhance collision avoidance performance.

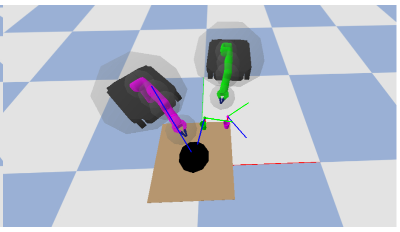
In Scenario 2, each of the two mobile manipulators is assigned to grasp an allocated cup on the same table as illustrated in Fig. 3.1b. Robots need to navigate in close proximity to the other agent while avoiding a static obstacle near their trajectories. The higher 80% success rate achieved by the MPC-d method, compared to the 50% and 53.3% success rates of GF and MPC-t respectively is due to its ability to manage crossed trajectories at grasp poses. Recall that in MPC-d, the target tracking weight $\boldsymbol{W}_\epsilon$ is dynamically adjusted based on the robot's proximity to its goal. As the robot approaches its goal, the tracking weight increases, resulting in a higher speed of its end-effector. In scenarios where one robot reaches its goal, this increase in target tracking weight helps the other robot make faster progress toward its own goal. Although the first robot may initially block the second robot's trajectory, its end-effector is moved away to avoid a collision, allowing the second robot to proceed. This dynamic adjustment of the tracking weight helps to resolve local minima and improve overall performance. In contrast, GF tends to result in localized solutions and struggles to handle trajectory conflicts effectively. However, the dynamic weight adjustment and collision avoidance alone cannot resolve all scenarios involving grasp poses with crossed trajectories. Deadlock still occurs when one robot blocks another's path, or when both robots are hindered by collision avoidance between bases near their goals, as shown in Fig.3.2b and Fig.3.2c. Therefore, a higher-level task planner will be necessary in the future to address these complex interactions more effectively.

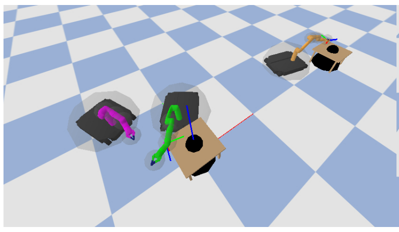
To demonstrate the scalability of the proposed method to an arbitrary number of agents, Scenario 1 is extended to include three decentralized mobile manipulators in Scenario 3. As illustrated in Fig. 3.1c, the trajectories of the three robots may intersect. It is observed that the MPC-based methods, MPC-d and MPC-t, achieve higher success rates of 70% and 70%, respectively, compared to the 46.7% success rate of GF. Notably, while the computation time of GF remains relatively unchanged across scenarios, MPC-t experiences a significant increase in computation time due to the higher number of collision-avoidance constraints, resulting in a loss of real-time performance. In contrast, MPC-d maintains real-time performance even in this more complex scenario. Due to the crowded nature of the scenario, collisions occur across the compared decentralized methods. As in Scenario 1, GF primarily suffers from inter-robot and self-collisions. On the other hand, while the MPC-based methods successfully eliminate inter-robot collisions in deadlock situations, the robotic arms may collide with the tables during attempts to avoid other robots.

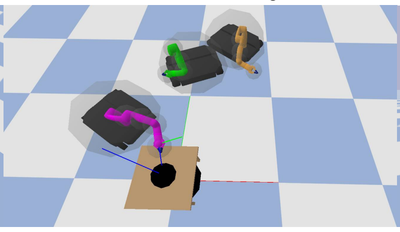
In summary, benchmarking different local motion planning methods for mobile manipulators presents significant challenges, as there are no existing off-the-shelf solutions or published statistics for direct comparison. Additionally, achieving consistent behavior across multiple planners is complex and requires extensive adjustments. While configuration, velocity, and acceleration bounds are set identically across all controllers, differences in the design of cost and constraint functions among methods can still result in varying outcomes and behaviors. Moreover, achieving consistent behavior requires expert knowledge and a deep understanding of the specific characteristics and underlying principles of each approach. When evaluating time-to-success across all scenarios, GF consistently shows higher values than MPC-based methods. This difference is likely due to the different tracking cost weights, with larger weights promoting more aggressive behavior that helps the robot reach its goal faster.


(a) S1: Robot-1 (green) is unable to reach its goal due to collision avoidance constraints between its upper-wrist link and the table.


(b) S2: After robot-2 (magenta) reaches its goal, it blocks robot-1's path while trying to keep its end effector at the target. As a result, robot-1 cannot reach its own goal.


(c) S2: Both robots navigate to their respective goals, but as they get closer and avoid collisions with each other's base, both stop and neither of them reaches their goal.


(d) S1 and S3 (MPC): Deadlock occurs when robot-2 (magenta) attempts to reach its goal which is blocked by robot-1 (green). To avoid colliding with robot-2, robot-1 collides with the edge of the table.


(e) S1 and S3 (GF): An inter-robot collision occurs between robot-1 (green) and robot-3 (orange) as they attempt to reach their goals by using GF

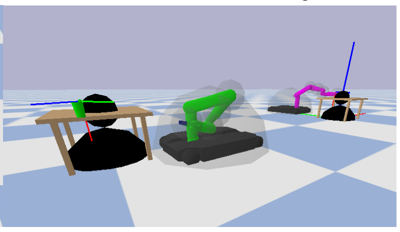

(f) S1 and S3 (GF): The arm of robot-1 (green) folds, resulting in a self-collision as the robot attempts to reach its goal by using GF.

Figure 3.2: Screenshots capturing task failures across the three scenarios are provided. Different colors distinguish the robots: robot-1 (green), robot-2 (magenta), and robot-3 (orange). Each robot's goal is indicated by a marker next to the generated cup, whose color matches the corresponding robot's arm. The collision spheres used for each ego robot's collision avoidance are visualized as semi-transparent gray spheres.

3.3. Experiment 2: Computation Time

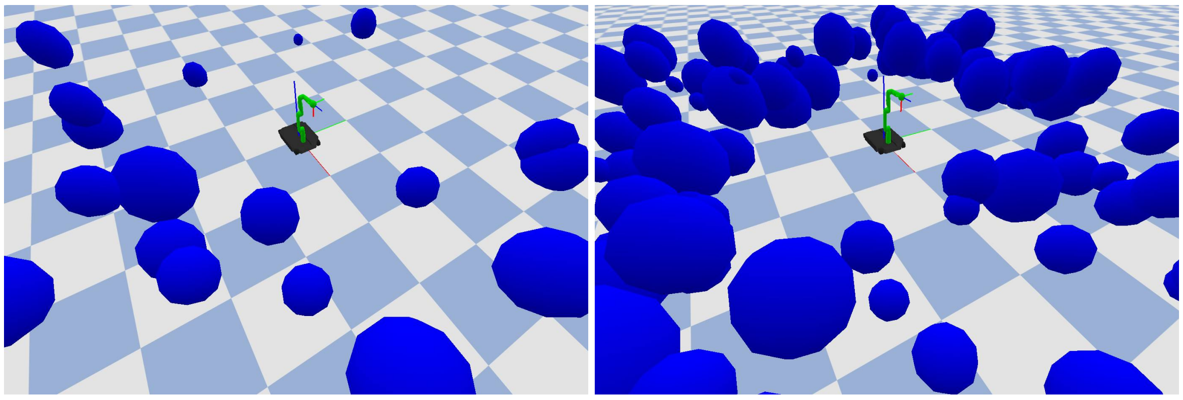
This section presents a comparison study on the increase in computation time across three different methods as the number of obstacles increases. Since each method uses sphere-sphere collision avoidance inequality constraints, it is crucial to evaluate their scalability with respect to the growing number of obstacles.

The GF method does not differentiate between static and dynamic obstacles. In contrast, for MPC-t and MPC-d, if the obstacle states need to be updated during runtime, the states x_{o_i} of these obstacles have to be stacked with system states. This increases the optimization problem's dimensionality, thereby increasing computation time. To mitigate this issue, obstacles can be categorized into two types:

1. **Static obstacles**: These obstacles have fixed states that remain unchanged during the planning process. For instance, stationary objects like tables can be treated as static obstacles. They can be represented using a URDF file and provided to the controller during initialization. GJK-based distance queries from Flexible Collision Library (FCL) [57] are used to calculate the minimum distance between each collision pair, ensuring collision avoidance during runtime. Inequality collision avoidance constraints are applied to each sphere representing the ego robot. Consequently, for every static obstacle added, 5 additional inequality constraints are introduced into the optimization problem, as all the collision spheres representing the ego-robot as detailed in Section 3.1 are considered.
2. **Dynamic obstacles**: The obstacles states are updated at runtime. Their states x_{o_i} are stacked with the system's states to propagate their behavior over the prediction horizon. To accommodate these updates, the controller increases the dimensionality of the optimization problem, which raises computation time. For MPC-d, the state includes the 3D position and velocity, totalling 6 dimensions. For MPC-t, the state includes acceleration as well, adding 9 additional dimensions for each dynamic obstacle. Similarly to static obstacles, inequality collision avoidance constraints are applied to each sphere representing the ego robot. As a result, every dynamic obstacle added introduces 5 additional inequality constraints to the optimization problem. The controller ensures that the dynamic obstacles' states are propagated during the prediction horizon and that constraints are satisfied throughout this period.

3.3.1. Setup

The robot is positioned at the origin of the world with a fixed arm configuration. In all cases, the robot's end-effector is required to remain stationary at its initial pose. Obstacle positions are uniformly sampled within the range $[x, y, z] \in [-4.0, 4.0]\ \mathrm{m} \times [-4.0, 4.0]\ \mathrm{m} \times [0.0, 1.0]\ \mathrm{m}$, while their radii are uniformly sampled within the range $[0.1, 0.5]\ \mathrm{m}$, as illustrated in Fig. 3.3. The velocity and acceleration of the obstacles are set to zero. Computation times are recorded throughout the simulation sessions, each with a fixed duration. For each method and each specific number of obstacles, 10 tests are conducted, with obstacle positions randomized for each run.



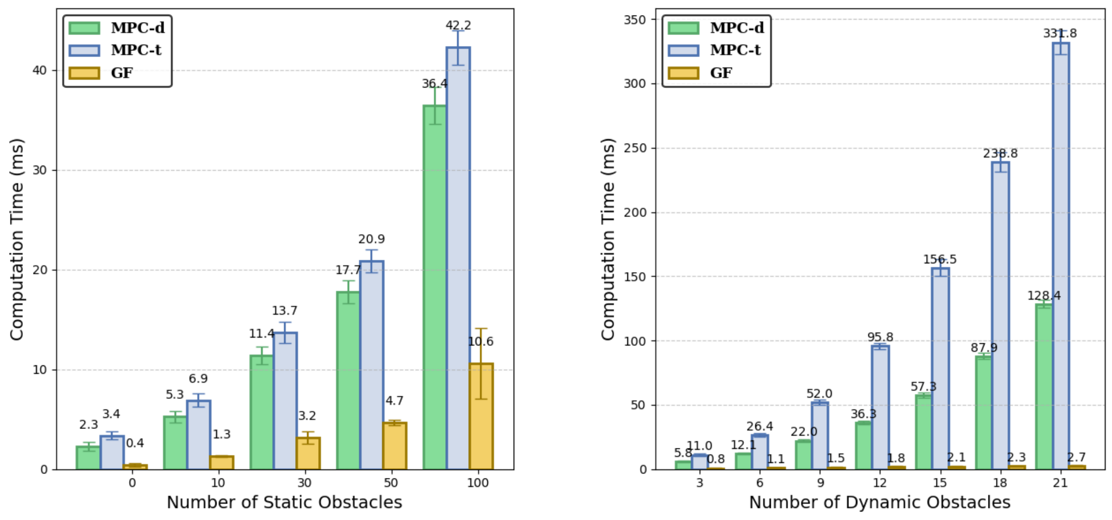
(a) 20 obstacles **(b)** 100 obstacles

Figure 3.3: Illustrations depicting scenarios with varying numbers of obstacles. The robot ensures its end-effector remains at its initial pose as indicated by the marker. It checks the minimum distance among all collision pairs and ensures the collision inequality constraints.

3.3.2. Results

Figure 3.4a illustrates the computation times of GF, MPC-d, and MPC-t with increasing numbers of static obstacles. The number of static obstacles ranges from 0 to 100, allowing for a comprehensive analysis of performance across varying environmental complexities. The MPC-based methods exhibit a noticeable increase in computation time as the number of obstacles grows. Specifically, MPC-d rises from 2.3 ms for 0 obstacles to 36.4 ms for 100 obstacles, while MPC-t follows a similar trend, increasing from 3.4 ms to 42.2 ms. Although these computation times are higher compared to GF, both MPC methods maintain real-time performance even at the upper limit of 100 obstacles, demonstrating their capability in handling complex environments. In contrast, the GF method exhibits remarkable scalability, with a much more gradual increase in computation time as the number of static obstacles increases. Beginning at just 0.4 ms for 0 obstacles, GF maintain a linear progression, reaching only 10.6 ms for 100 obstacles. This underscores its efficiency and suitability for real-time applications, particularly in environments with high complexity.

Figure 3.4b presents the results for dynamic obstacles, where the number of obstacles varies between 3 and 21. Different from the static obstacles scenarios, in this scenario, a significant difference emerges between the MPC-based methods and GF. Similar to the static obstacles scenario, GF exhibits minimal increases in computation time and maintains real-time performance throughout the range. In contrast, the MPC-based methods show a substantial increase in computation time as the number of dynamic obstacles rises, with MPC-d growing from 5.8 ms for 3 obstacles to 128.4 ms for 21 obstacles, and MPC-t increasing from 11.0 ms to 331.8 ms over the same range. Notably, MPC-t exceeds the real-time performance threshold of 50.0 ms when the number of obstacles surpasses 9, reaching 52.0 ms, while MPC-d loses real-time performance at 57.3 ms when the number of obstacles exceeds 15. Additionally, it is worth noting that MPC-d consistently achieves approximately half the computation time of MPC-t for the same number of dynamic obstacles. The higher computation times in MPC-based methods stem from their need to account for real-time updates of dynamic obstacle states and propagate these behaviors over the prediction horizon, increasing the optimization problem's dimensionality. This demonstrates the direct correlation between increased system dimensionality and computation cost in handling dynamic obstacles.



(a) (b)

Figure 3.4: Comparison of controller computation time performance with increasing numbers of static and dynamic obstacles.

The results highlight a clear trade-off between the methods. GF offer excellent scalability and minimal computational overhead, maintaining real-time performance even in complex environments with both static and dynamic obstacles. On the other hand, by making a split between static and dynamic obstacles, the MPC-based methods achieve real-time performance in static environments with a large number of obstacles and in dynamic environments with fewer than 9 obstacles for MPC-t and fewer than 15 obstacles for MPC-d. The advantage of MPC lies in its ability to propagate obstacle states across the entire prediction horizon, providing enhanced collision avoidance, particularly for high-speed obstacles representing non-ego robots. In contrast, GF can only update obstacles positions at runtime, making it purely reactive. To optimize MPC performance, separating static and dynamic obstacles could be a beneficial approach. A limitation of this setup is that after the first policy is generated, the MPC system operates with reduced computational complexity for subsequent policies, especially when the robot's trajectory stabilizes or it nears its goal. This might lead to an underestimation of the true computational cost, as the system faces fewer environmental changes. Furthermore, the simple task configurations used in the experiments do not fully capture real-world scenarios, such as continuously moving obstacles or complex manipulation tasks. For future improvements, a more dynamic environment with high-speed obstacles should be introduced, alongside tasks that require continuous re-planning, like complex pick-and-place operations. This would better reflect real-world challenges and more thoroughly evaluate the capacity of MPC-based methods to handle complex, changing conditions.

4 Real-world experiments

Experiments are conducted to evaluate the real-world applicability of the proposed framework. The robot's state estimation integrates data from a Vicon motion capture system and onboard sensors. For the ego robot, joint encoders capture the arm's positions $\boldsymbol{q}_{arm}$, while the Vicon system precisely tracks the base's position and orientation $\boldsymbol{q}_{base}$. These measurements are fused using a Kalman Filter (KF) to estimate the full robot state $\boldsymbol{x}_r = [\boldsymbol{q}_r^T, \boldsymbol{v}_r^T]^T$, as described in Section 2.4.1. For non-ego robots, joint encoders provide arm joint positions and velocities, while base velocities are computed via numerical differentiation, as detailed in Section 2.4.2. Collision sphere states $\boldsymbol{x}_{o_i} = [\boldsymbol{q}_{o_i}^T, \boldsymbol{v}_{o_i}^T]^T$ for non-ego robots are determined using forward kinematics (FK) and differential kinematics (DK), as outlined in Section 2.4.3. Additional collision pairs, such as those between the robot's end-effector and the ground, are incorporated to ensure safe operations. Static tables in the environment are modeled as cubes with known dimensions and positions supplied by the Vicon system. These parameters are used to generate URDF files, which are then integrated into the MPC for planning, as discussed in Section 3.3. The MPC-d framework, illustrated in Fig.2.1, runs on a laptop with an Intel Core i5-10300H CPU at 2.50 GHz and 16 GiB of RAM. Robots receive velocity commands published by the laptop. The MPC parameters used in real-world experiments remain consistent with the simulation, as listed in Table 3.1 and Table 3.2. State estimation parameters specific to the real-world setup are provided in Table 4.1, and the heuristic parameters are summarized in Table 4.2.

A video demonstration of the experiments is available at `https://autonomousrobots.nl/msc_projects_finished/25-danningzhao-multiagentmpc`.

Table 4.1: State Estimation Parameter Settings

(a) Kalman Filter Parameter Settings

Parameter	Symbol	Value
Sampling Time	δt	8ms
Process Noise Covariance	$\boldsymbol{Q}$	$10\boldsymbol{I}_{9\times9}$
Measurement Noise Covariance	$\boldsymbol{R}$	$0.001\boldsymbol{I}_{9\times9}$
State Transition Matrix	$\bar{\boldsymbol{A}}_r$	$\begin{bmatrix} \boldsymbol{I}_9 & \delta t\boldsymbol{I}_{9\times9} \\ \boldsymbol{0}_{9\times9} & \boldsymbol{I}_{9\times9} \end{bmatrix} \in \mathbb{R}^{18\times18}$
Control Input Matrix	$\bar{\boldsymbol{B}}_r$	$\begin{bmatrix} \frac{1}{2}\delta t^2\boldsymbol{I}_{9\times9} \\ \delta t\boldsymbol{I}_{9\times9} \end{bmatrix} \in \mathbb{R}^{18\times9}$
Measurement Model	$\boldsymbol{C}$	$\begin{bmatrix} \boldsymbol{I}_{9\times9} & \boldsymbol{0}_{9\times9} \end{bmatrix} \in \mathbb{R}^{9\times18}$
Initial Covariance Estimate	$\boldsymbol{P}_0$	$0.1\boldsymbol{I}_{18\times18}$

(b) Numerical Differentiation Parameters

Parameter	Symbol	Value
Time Interval	Δt	8ms
Time Constant-linear	τ_{linear}	0.045
Time Constant-angular	$\tau_{angular}$	0.025

Table 4.2: Parameter Settings for the Heuristic to Solve Livelocks

Parameter	Symbol	Value
Livelock Detection Distance Threshold	$d_{ee,l}$	$1\,\mathrm{m}$
Average Velocity Threshold for Detection	$\bar{\boldsymbol{v}}_l$	$-0.3\,\mathrm{m/s}$
Duration for Calculating $\bar{v}$	T_l	$0.5\,\mathrm{s}$
Livelock Resolution Distance Threshold	$d_{ee,r}$	$1\,\mathrm{m}$
Task Completion Tolerance	$d_{ee,\hat{r}}$	$0.07\,\mathrm{m}$

4.1. A single robot experiment

Before progressing to multi-robot experiments, it is essential to validate the functionality of MPC on a single robot. This section tests the static obstacle avoidance capabilities of MPC-d.

4.1.1. Assessing single robot static obstacle avoidance

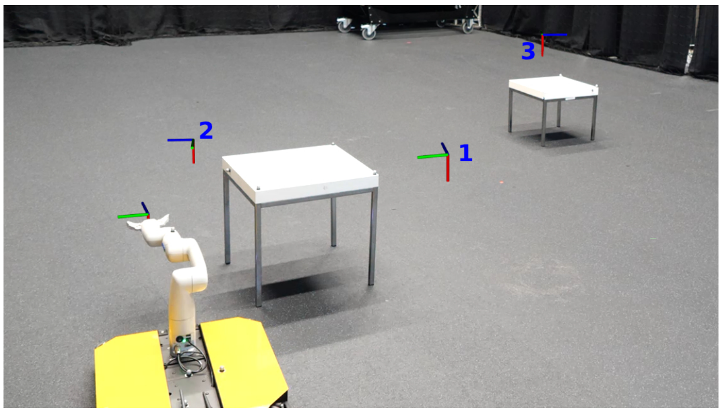


Figure 4.1: Real-world environment for static obstacle avoidance experiments. The robot sequentially reaches three end-effector targets, indicated by the markers in the figure

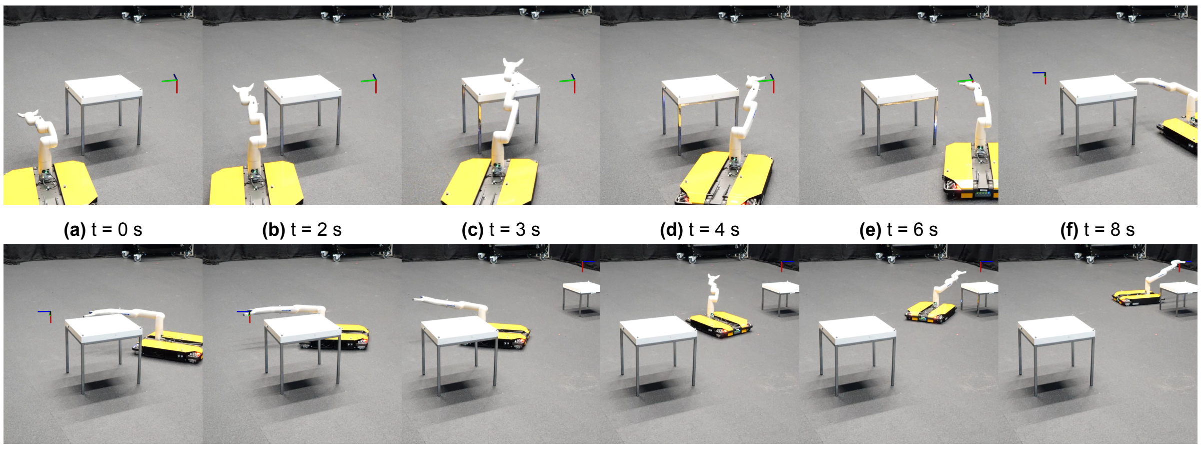
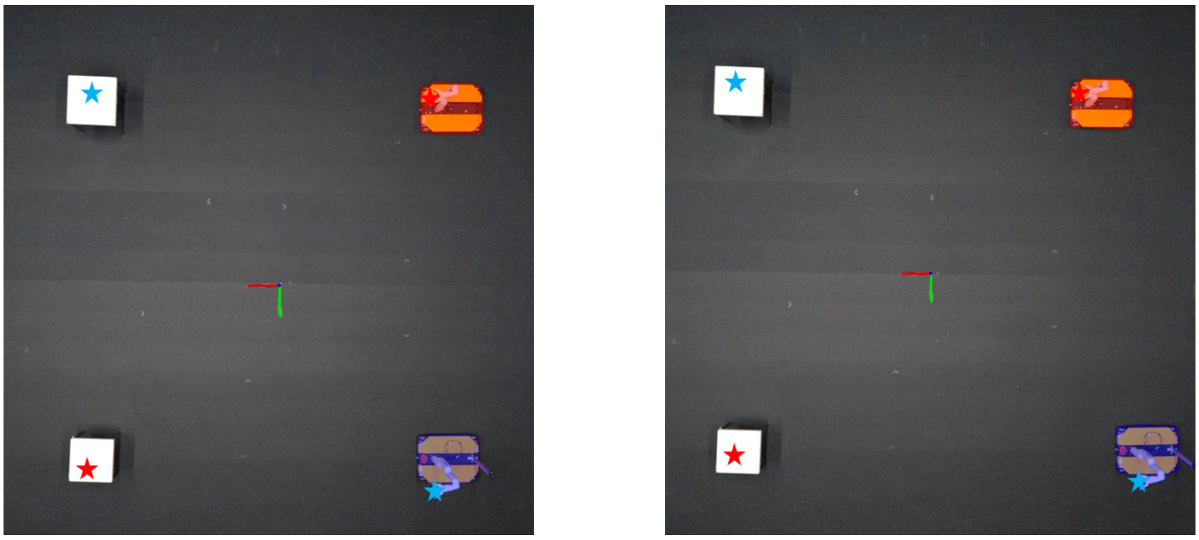


Figure 4.2: The mobile manipulator avoids the two tables modelled as static obstacles in the real world. **(a-e)** The end-effector moves to the other side of table and reaches goal-1. **(f-h)** The end-effector moves behind the table and reaches goal-2. **(i-l)** Then it turns back, relocating behind another table positioned far away and reaches goal-3.

Figure 4.1 illustrates the setup for the static obstacle avoidance experiment. Starting from the initial configuration shown, the robot's end-effector is tasked to reach three sequential goals, indicated by markers, without colliding with the tables. As illustrated in Fig. 4.2, the mobile manipulator can reach three different goals without any collision. The computation time of the MPC-d has a value of 6 ms on average, aligning with the results observed in the simulation experiments.

4.2. Multi-Robot Experiments

4.2.1. A prioritized heuristic to solve livelocks



(a) Setup 1. Robots are symmetrically positioned about the x-axis. Robot-1 (blue) starts with the base configuration $\boldsymbol{q}_{base_1} = [-2\mathrm{m}, 2\mathrm{m}, 0\mathrm{rad}]$, while robot-2 (red) starts with $\boldsymbol{q}_{base_2} = [-2\mathrm{m}, -2\mathrm{m}, 0\mathrm{rad}]$.

(b) Setup 2. Robot-2 (red) retains the same initial base configuration $\boldsymbol{q}_{base_2} = [-2\mathrm{m}, -2\mathrm{m}, 0\mathrm{rad}]$, while robot-1 (blue) starts at $\boldsymbol{q}_{base_1} = [-2.5\mathrm{m}, 2\mathrm{m}, 0\mathrm{rad}]$.

Figure 4.3: Two robots are tasked with reaching end-effector target poses, represented by colored stars, with their trajectories crossing. The initial configurations of the robots are shown in the figure, and the marker at the center denotes the origin of the world frame.

To evaluate the dynamic collision avoidance functionality and the efficiency of the heuristic in Section 2.5 in resolving livelock situations, two robots are tasked with reaching end-effector target poses that are symmetric about the x-axis, causing their trajectories to cross, as illustrated in Fig. 4.3. Robot-1, positioned at the bottom right and marked in blue, and robot-2, positioned at the top right and marked in red, start with their arm configurations fixed in a home position, $\boldsymbol{q}_{arm}^{home} = [0.0, 0.0, 0.0, 1.7, 1.57, -1.57]\mathrm{rad}$. The target positions for robot-1 and robot-2 are specified as $[x, y, z] = [2, -2, 0.45]\,\mathrm{m}$ and $[x, y, z] = [2, 2, 0.45]\,\mathrm{m}$, respectively, with the desired quaternion for the end-effector orientation defined as $\hat{\boldsymbol{\xi}} = [\hat{\xi}_w, \hat{\xi}_x, \hat{\xi}_y, \hat{\xi}_z] = [0.0, 0.707, 0.0, 0.707]$. The tables are positioned below the targets, with their coordinates in the world frame specified as $[x, y, z] = [2.0, 2.0, 0.0]\,\mathrm{m}$ and $[0.0, 0.0, 0.0]\,\mathrm{m}$. Two setups are designed to analyze the robots' different behavior during livelock. In setup 1, both robots are symmetrically positioned about the x-axis, with the initial base configuration for robot-1 defined as $\boldsymbol{q}_{base_1} = [x, y, \theta] = [-2\,\mathrm{m}, 2\,\mathrm{m}, 0\,\mathrm{rad}]$ and for robot-2 as $\boldsymbol{q}_{base_2} = [x, y, \theta] = [-2\,\mathrm{m}, -2\,\mathrm{m}, 0\,\mathrm{rad}]$, both within the world frame. In setup 2, robot-2 starts from the same initial base configuration as in setup 1, while robot-1 begins from $\boldsymbol{q}_{base_1} = [x, y, \theta] = [-2.5\mathrm{m}, 2\mathrm{m}, 0\mathrm{rad}]$. In practice, base positions deviate a maximum of 5 cm in x and y from this initial base configuration, while θ deviates a maximum of 10°. Task success is defined as both agents reaching their target positions within 7 cm of the goal without collisions. Performance of MPC-d and its extension MPC-d* with a heuristic livelock resolution is compared using metrics including time-to-goal, total path length, and collision rate. The total path length is calculated as the sum of the trajectory lengths based on the x, y-positions of both robots' chassis links. Collisions are identified visually by detecting contact between any rigid body, with ground collisions checked only for the arm.

4.2.2. Results

Given these start pose disturbances and other uncertainties, the results remain consistent and demonstrate the efficiencies of the proposed method under real-world conditions. Table 4.3 compares the performance of MPC-d and its extension, MPC-d*, which incorporates a heuristic for livelock resolution, across two setups. MPC-d* outperforms MPC-d in both time-to-success and total path length, demonstrating the effectiveness of the prioritized heuristic.

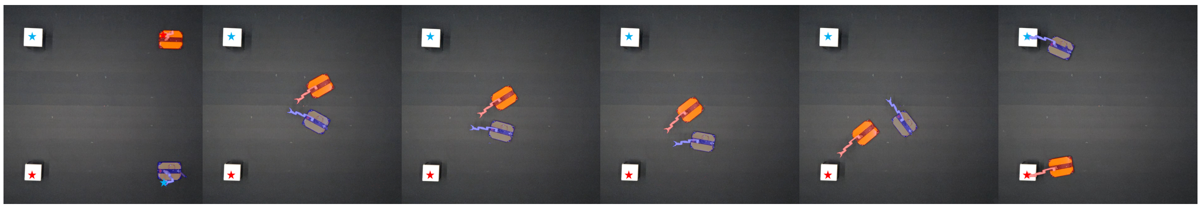
Table 4.3: Statistics for two setups comparing MPC-d with livelock resolution heuristic (denoted as MPC-d*) and MPC-d without it, each tested 10 times.

Setup	Method	Time-to-Success [s]	Total Path Length [m]	Collision-rate [%]
1	MPC-d	18.21 ± 1.12	13.83 ± 0.88	0
	MPC-d*	17.05 ± 0.84	12.94 ± 0.71	0
2	MPC-d	25.73 ± 1.02	17.50 ± 1.39	0
	MPC-d*	15.80 ± 0.55	12.99 ± 1.50	0

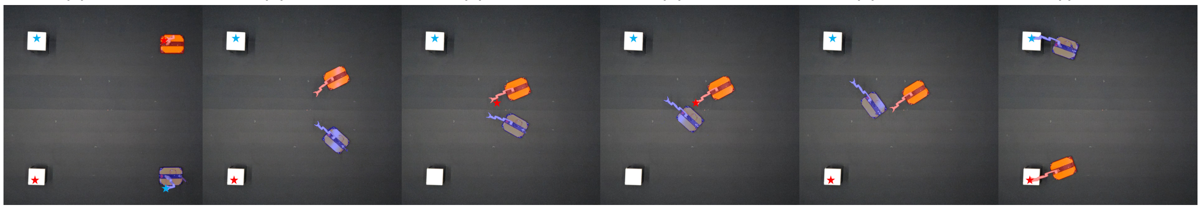
In setup 1, MPC-d* achieves an improved time-to-success of 17.05 $\pm$ 0.84 s and a total path length of 12.94 $\pm$ 0.71 m, compared to MPC-d with a time-to-success of 18.21 $\pm$ 1.12 s and a total path length of 13.83 $\pm$ 0.88 m. This slight improvement is mainly due to the resolution of the livelock. As shown in Fig. 4.4(a-f), the livelock occurs when the end-effectors approach each other, preventing one robot from reaching its goal. During the livelock, one robot pushes the other robot away from its own goal, requiring the hindered robot to spend an additional 1−2 s to escape and proceed toward its goal. With the prioritized heuristic, the livelock is detected early, allowing one robot to reset its target pose and let the other pass first, leading to reduced time-to-success and path length, as demonstrated in Fig. 4.4(g-l).

In setup 2, MPC-d* results in a much smaller time-to-success of 15.80 $\pm$ 0.55 s and a total path length of 12.99 $\pm$ 1.50 m, compared to MPC-d, which has a time-to-success of 25.73 $\pm$ 1.02 s and a total path length of 17.50 $\pm$ 1.39 m. As shown in Fig. 4.5(a-f), the livelock occurs when the slightly faster robot keeps pushing the slower one aside until it reaches its goal. This interaction leads to longer paths and higher times-to-success compared to setup 1. It is worth highlighting that no collisions occur between the hindered robot and tables, demonstrating the collision avoidance capability of MPC-d. By incorporating the heuristic, the livelock is resolved earlier, enabling both robots to efficiently progress toward their respective goals, as illustrated in Fig. 4.5(g-l).

In summary, the heuristic improves performance by reducing both time-to-success and path length while minimizing collision probability. Notably, the total path length of MPC-d* closely approximates the sum of straight-line paths connecting each robot to its goal, as the heuristic enables near-straight trajectories. The experiments primarily focus on object delivery scenarios, addressing challenges where robots are tasked to navigate to targets far away. Detecting deadlocks or livelocks in multi-agent scenarios remains a significant challenge, often requiring tailored actions depending on the specific type of problem encountered.

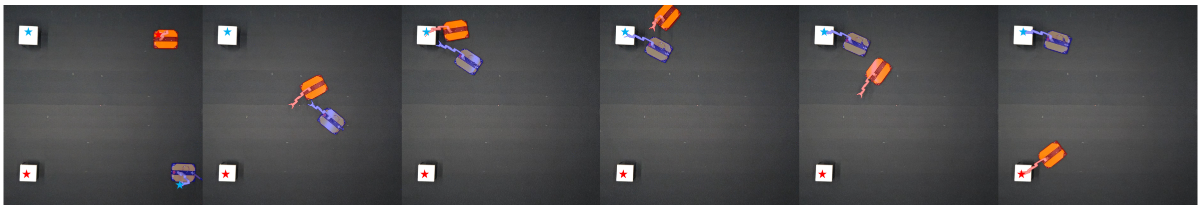


(a) t = 0 s **(b)** t = 6 s **(c)** t = 7 s **(d)** t = 8 s **(e)** t = 10 s **(f)** t = 18 s

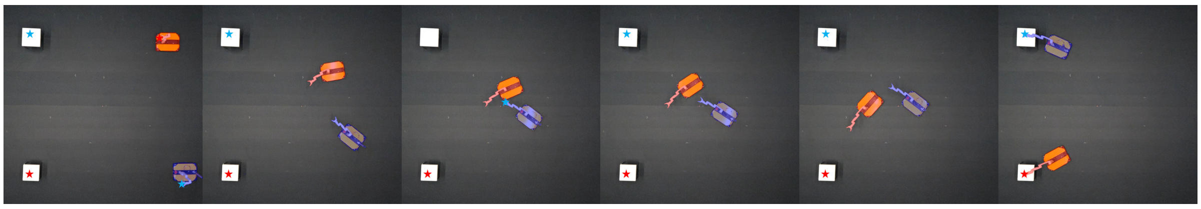


(g) t = 0 s **(h)** t = 4 s **(i)** t = 6 s **(j)** t = 8 s **(k)** t = 10 s **(l)** t = 18 s

Figure 4.4: Setup1. **(a-f) MPC-d**: A livelock occurs at 6 s when the end-effectors approach each other. The livelock persists as robot-2 (red) end-effector moves to its own goal and pushes robot-1 away till 8 s. Eventually, at 8 s, robot-1 gets rid of the livelock and begins moving toward its respective goal. **(g-l) MPC-d***: Both robots slow down at 6 s as they approach each other. A potential livelock is detected when the Euclidean distance between the EEs becomes smaller than the 1 m threshold. At 6 s, a livelock is confirmed when at least one robot's average velocity $\bar{v}$, as defined in Eq. 2.39, is larger than the -0.3 m/s threshold. The heuristic then resets robot-2's target pose, allowing it to stand still and let robot-1 pass. By 10 s, the Euclidean distance between the EEs exceeds 1.0 m, indicating that the livelock has been resolved. robot-2's original target pose is reset, and both robots reach their targets by 18 s.



(a) t = 0 s **(b)** t = 5 s **(c)** t = 12 s **(d)** t = 14 s **(e)** t = 20 s **(f)** t = 28 s



(g) t = 0 s **(h)** t = 4 s **(i)** t = 6 s **(j)** t = 8 s **(k)** t = 10 s **(l)** t = 16 s

Figure 4.5: Setup 2. **(a-f) MPC-d**: At 5 s, robot-1, aiming for its goal, pushes robot-2 to avoid collisions. The livelock persists until 14 s, when robot-1 reaches its goal and robot-2 starts moving toward its own target. **(g-l) MPC-d***: At 6 s, a livelock is detected as the average velocity $\bar{v}$ of robot-2 is larger than the threshold -0.3 m/s threshold. Robot-1's target pose is rest to its current EE pose, letting robot-2 pass. Finally, both robots reach their targets by 16 s.

5 Conclusions and Future Research

This chapter summarizes this thesis's key findings and contributions, emphasizing the advancements in motion planning for mobile manipulators in multi-agent settings using MPC. Additionally, it addresses the limitations encountered during the research and outlines potential directions for future work to refine and extend the proposed methods.

5.1. Conclusions

Research on motion planning for mobile manipulators has mainly focused on single-agent systems employing local planners such as geometric-based, sampling-based, and optimization-based methods. While these methods have achieved real-time performance in dynamic environments, extending them to multiple (mobile) manipulator systems presents additional challenges, especially for optimization-based methods. Existing studies on multi-manipulator motion planning have explored centralized and decentralized approaches, with the latter gaining attention for its scalability and adaptability. However, research addressing multiple mobile manipulators remains even rarer, particularly those tested and validated in real-world environments. Moreover, deadlocks and livelocks pose persistent challenges in decentralized frameworks, especially in dynamic, multi-agent settings.

To address these limitations, this thesis builds on the optimization-based local planner MPC [14], adapting it to multi-mobile manipulator systems operating in shared workspaces. A decentralized MPC framework is proposed in Chapter 2, with its core being the MPC-d, which uses a double integrator as the dynamic model introduced in Section 2.3. This modification ensures computational efficiency for real-time motion planning while maintaining predictive capabilities for obstacles. Furthermore, accurate state estimation detailed in Section 2.4 is essential for both ego and non-ego robots, enabling safe operation and collision avoidance by providing precise system states to the controller. To address the common livelock issue in decentralized frameworks, which arises from obstacle avoidance based on local observations, a heuristic is introduced in Section 2.5. It employs a prioritization framework that establishes a hierarchy of goals within livelock groups, effectively resolving conflicts and ensuring progress in multi-robot setups.

To validate the performance of MPC-d, a comparison study with original work MPC-t [14], which uses a triple integrator as the dynamic model, and GF [27] was conducted in simulation, including two key experiments: pick-and-place scenarios and a computation time analysis, as detailed in Chapter 3. In Section 3.2, the pick-and-place experiments demonstrated that MPC-d performed comparably to GF in terms of success rate and collision avoidance. While MPC-d incurred higher computational costs than GF, it was more computationally efficient than MPC-t. In Section 3.3, GF showed excellent scalability, maintaining real-time performance even with up to 100 dynamic obstacles. In contrast, MPC-d maintained real-time performance in crowded static environments and in dynamic environments with up to 15 obstacles. The key advantage of MPC-d is its ability to propagate obstacle states over the prediction horizon, enabling better collision avoidance for high-speed obstacles, unlike the purely reactive GF.

In Chapter 4, real-world experiments validate the proposed multi-mobile manipulator MPC framework.

Section 4.1 presents a qualitative experiment demonstrating the goal-reaching and static obstacle avoidance capabilities of MPC-d on a single robot. Subsequently, quantitative experiments in Section 4.2 assess dynamic obstacle avoidance and the efficiency of the heuristic in resolving livelock situations in an object delivery scenario. Both experiments confirm effective collision avoidance between robots and environmental obstacles. Additionally, the heuristic enhances performance by reducing time-to-success and path length while maintaining minimal collision risk. The computation time of MPC-d aligns with simulation results, highlighting its adaptability for real-world applications.

Overall, the results demonstrate that MPC-d achieves performance comparable to the state-of-the-art geometric method GF in multi-agent settings. While MPC-d's inherent optimization-based formulation results in higher computation time compared to geometric methods like GF, it still maintains real-time performance, even in scenarios involving up to three robots. Additionally, the obstacle prediction horizon facilitates the design of a prioritized heuristic to proactively resolve livelock situations. Real-world experiments further validate the proposed pipeline's alignment with simulation results, confirming its efficient state estimation for both ego and non-ego robots, as well as its robust real-time performance. These findings underscore the adaptability and effectiveness of MPC-d in real-world applications.

5.2. Future Research

The proposed framework demonstrates reliable real-time performance and adaptability, validated through both simulation and real-world experiments. However, there remain several research directions for further improvement and exploration to enhance its scalability, adaptability, and performance in complex real-world environments.

First, future work could investigate replacing explicit pairwise collision checks with an Euclidean Signed Distance Field (ESDF) [66] for static obstacle avoidance. As discussed in Section 3.3, the computation time grows significantly with the number of obstacles due to the explicit pairwise constraints. An ESDF, which encodes the distance to the nearest obstacle, could serve as a cost function within the MPC framework, eliminating the need for explicitly defining the shapes of obstacles. Moreover, integrating the ESDF with a perception pipeline could enable the system to dynamically update the distance field using sensor data, allowing for operation in previously unknown environments. Prior research [30, 31, 52] has demonstrated the utility of ESDF in motion planning for mobile manipulators, showcasing its potential to improve scalability and maintain real-time performance in a complex environment.

Secondly, the proposed MPC-d pipeline serves as a local planner, focusing on short-term trajectory generation but lacking global guidance. As highlighted in the scenario experiments in Section 3.2, certain target poses become unreachable due to the conservative collision avoidance strategy. To address this, future research could explore integrating a grasp planner, as discussed in [67, 68]. Such a planner would adapt target poses by considering object geometry, environmental constraints, and robot kinematics, ensuring better reachability. Furthermore, while the heuristic introduced in Section 2.5 resolves livelocks in specific object delivery scenarios, deadlocks in multi-robot pick-and-place tasks, caused by inter-robot collision avoidance, remain a significant challenge. Optimizing task allocation and scheduling as explored in works such as [11, 12] can reduce deadlocks by improving coordination among robots.

Finally, another promising direction for future research involves leveraging learning-based methods to automate target selection and optimize parameter tuning within MPC frameworks for motion planning in multi-agent settings. For instance, deep Reinforcement Learning (RL) can be used to provide long-term guidance to the local MPC planner by recommending interaction-aware subgoals that account for the robot's progress and its interactions with surrounding agents [69]. Additionally, neural networks (NNs) could be employed to approximate the optimization process within the MPC framework, offering significant computational speedups while maintaining system stability and constraint satisfaction through safety augmentation techniques [70]. These combined methods could enable faster and more adaptive motion planning in dynamic, multi-agent environments, further enhancing the practicality of MPC-based systems in real-world applications.

References

[1] T. Sandakalum and M. H. Ang, “Motion planning for mobile manipulators—a systematic review,” *Machines*, vol. 10, no. 2, 2022. [Online]. Available: `https://www.mdpi.com/2075-1702/10/2/97`.

[2] M. Yang, E. Yang, R. C. Zante, M. Post, and X. Liu, “Collaborative mobile industrial manipulator: A review of system architecture and applications,” in *2019 25th International Conference on Automation and Computing (ICAC)*, 2019, pp. 1–6.

[3] K. Zhou *et al.*, “Mobile manipulator is coming to aerospace manufacturing industry,” in *2014 IEEE International Symposium on Robotic and Sensors Environments (ROSE) Proceedings*, 2014, pp. 94–99.

[4] J. Aleotti *et al.*, “Toward future automatic warehouses: An autonomous depalletizing system based on mobile manipulation and 3d perception,” *Applied Sciences*, vol. 11, no. 13, 2021. [Online]. Available: `https://www.mdpi.com/2076-3417/11/13/5959`.

[5] J. Gros, D. Zatyagov, M. Papa, C. Colloseus, S. Ludwig, and D. Aschenbrenner, “Unlocking the benefits of mobile manipulators for small and medium-sized enterprises: A comprehensive study,” *Procedia CIRP*, vol. 120, pp. 1339–1344, 2023.

[6] L. Tagliavini, L. Baglieri, G. Colucci, A. Botta, C. Visconte, and G. Quaglia, “Dot paquitop, an autonomous mobile manipulator for hospital assistance,” *Electronics*, vol. 12, no. 2, p. 268, 2023.

[7] Z. Feng, G. Hu, Y. Sun, and J. Soon, “An overview of collaborative robotic manipulation in multi-robot systems,” *Annual Reviews in Control*, vol. 49, pp. 113–127, 2020.

[8] H. Touzani, H. Hadj-Abdelkader, N. Séguy, and S. Bouchafa, “Multi-robot task sequencing & automatic path planning for cycle time optimization: Application for car production line,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1335–1342, 2021.

[9] H. Touzani *et al.*, “Efficient industrial solution for robotic task sequencing problem with mutual collision avoidance & cycle time optimization,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2597–2604, 2022.

[10] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, Available at http://planning.cs.uiuc.edu/.

[11] N. Gafur, V. Yfantis, and M. Ruskowski, “Optimal scheduling and non-cooperative distributed model predictive control for multiple robotic manipulators,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 390–397.

[12] A. Tika, N. Gafur, V. Yfantis, and N. Bajcinca, “Optimal scheduling and model predictive control for trajectory planning of cooperative robot manipulators,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9080–9086, 2020.

[13] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, “Long-horizon multi-robot rearrangement planning for construction assembly,” *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 239–252, 2022.

[14] A. Heins and A. P. Schoellig, “Keep it upright: Model predictive control for nonprehensile object transportation with obstacle avoidance on a mobile manipulator,” *IEEE Robotics and Automation Letters*, 2023.

[15] ABB. “Abb demonstrates concept of mobile laboratory robot for hospital of the future.” (2024), [Online]. Available: `https://new.abb.com/news/detail/37301/abb-demonstrates-concept-of-mobile-laboratory-robot-for-hospital-of-the-future`.

[16] A. M.-R. L. Delft. “Rss 2024 lab-tour demo - interact.” (2024), [Online]. Available: `https://www.youtube.com/watch?v=AldMFKnlW3M&list=PLOksz-MTFhN2T4gYQlLHR2lGCdCOBDkJw`.

[17] J. Kuffner and S. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, 995–1001 vol.2.

[18] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, “Manipulation planning on constraint manifolds,” in *2009 IEEE international conference on robotics and automation*, IEEE, 2009, pp. 625–632.

[19] D. Berenson, S. Srinivasa, and J. Kuffner, “Task space regions: A framework for pose-constrained manipulation planning,” *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[20] Z. Kingston, M. Moll, and L. E. Kavraki, “Exploring implicit spaces for constrained sampling-based planning,” *The International Journal of Robotics Research*, vol. 38, no. 10-11, pp. 1151–1178, 2019.

[21] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, “Chomp: Gradient optimization techniques for efficient motion planning,” in *2009 IEEE international conference on robotics and automation*, IEEE, 2009, pp. 489–494.

[22] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, “Stomp: Stochastic trajectory optimization for motion planning,” in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 4569–4574.

[23] *Moveit!* [Online]. Available: `https://moveit.ros.org/`.

[24] I. A. Sucan, M. Moll, and L. E. Kavraki, “The open motion planning library,” *IEEE Robotics Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[25] J. Haviland and P. Corke, “Neo: A novel expeditious optimisation algorithm for reactive motion control of manipulators,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1043–1050, 2021.

[26] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, “Riemannian motion policies,” *arXiv preprint arXiv:1801.02854*, 2018.

[27] M. Spahn, M. Wisse, and J. Alonso-Mora, “Dynamic optimization fabrics for motion generation,” *IEEE Transactions on Robotics*, 2023.

[28] M. Bhardwaj *et al.*, “Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation,” in *Conference on Robot Learning*, PMLR, 2022, pp. 750–759.

[29] C. Pezzato, C. Salmi, M. Spahn, E. Trevisan, J. Alonso-Mora, and C. H. Corbato, “Sampling-based model predictive control leveraging parallelizable physics simulations,” *arXiv preprint arXiv:2307.09105*, 2023.

[30] M. Mittal, D. Hoeller, F. Farshidian, M. Hutter, and A. Garg, “Articulated object interaction in unknown scenes with whole-body mobile manipulation,” in *2022 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2022, pp. 1647–1654.

[31] J. Pankert and M. Hutter, “Perceptive model predictive control for continuous mobile manipulation,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6177–6184, 2020.

[32] N. Ratliff and K. Van Wyk, “Fabrics: A foundationally stable medium for encoding prior experience,” *arXiv preprint arXiv:2309.07368*, 2023.

[33] S. Bakker, L. Knoedler, M. Spahn, W. Böhmer, and J. Alonso-Mora, “Multi-robot local motion planning using dynamic optimization fabrics,” in *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, IEEE, 2023, pp. 149–155.

[34] G. Rizzi, J. J. Chung, A. Gawel, L. Ott, M. Tognon, and R. Siegwart, “Robust sampling-based control of mobile manipulators for interaction with articulated objects,” *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1929–1946, 2023.

[35] J. Haviland, N. Sünderhauf, and P. Corke, “A holistic approach to reactive mobile manipulation,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3122–3129, 2022.

[36] M. Spahn, B. Brito, and J. Alonso-Mora, “Coupled mobile manipulation via trajectory optimization with free space decomposition,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 12 759–12 765.

[37] R. J. M. Afonso and R. K. H. Galvão, “Infeasibility handling in constrained mpc,” *Frontiers of Model Predictive Control*, pp. 47–64, 2012.

[38] Á. Madridano, A. Al-Kaff, D. Martín, and A. De La Escalera, “Trajectory planning for multi-robot systems: Methods and applications,” *Expert Systems with Applications*, vol. 173, p. 114 660, 2021.

[39] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, “Online trajectory generation with distributed model predictive control for multi-robot motion planning,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 604–611, 2020.

[40] C. E. Luis and A. P. Schoellig, “Trajectory generation for multiagent point-to-point transitions via distributed model predictive control,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 375–382, 2019.

[41] M. Dorigo, G. Theraulaz, and V. Trianni, “Swarm robotics: Past, present, and future [point of view],” *Proceedings of the IEEE*, vol. 109, no. 7, pp. 1152–1165, 2021.

[42] A. Tika and N. Bajcinca, “Predictive control of cooperative robots sharing common workspace,” *IEEE Transactions on Control Systems Technology*, 2023.

[43] N. Gafur, G. Kanagalingam, and M. Ruskowski, “Dynamic collision avoidance for multiple robotic manipulators based on a non-cooperative multi-agent game,” *arXiv preprint arXiv:2103.00583*, 2021.

[44] Y. Chen, M. Guo, and Z. Li, “Deadlock resolution and recursive feasibility in mpc-based multi-robot trajectory generation,” *IEEE Transactions on Automatic Control*, pp. 1–16, 2024.

[45] J. Haviland and P. Corke, “A purely-reactive manipulability-maximising motion controller,” *arXiv preprint arXiv:2002.11901*, 2020.

[46] K. He *et al.*, “Visibility maximization controller for robotic manipulation,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 8479–8486, 2022.

[47] B. Burgess-Limerick, C. Lehnert, J. Leitner, and P. Corke, “An architecture for reactive mobile manipulation on-the-move,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 1623–1629.

[48] B. Burgess-Limerick, J. Haviland, C. Lehnert, and P. Corke, “Reactive base control for on-the-move mobile manipulation in dynamic environments,” *IEEE Robotics and Automation Letters*, 2024.

[49] A. Heins, M. Jakob, and A. P. Schoellig, “Mobile manipulation in unknown environments with differential inverse kinematics control,” in *2021 18th Conference on Robots and Vision (CRV)*, 2021, pp. 64–71.

[50] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, “Qpoases: A parametric active-set algorithm for quadratic programming,” *Mathematical Programming Computation*, vol. 6, pp. 327–363, 2014.

[51] A. Gawel *et al.*, “A fully-integrated sensing and control system for high-accuracy mobile robotic building construction,” in *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2019, pp. 2300–2307.

[52] J.-R. Chiu, J.-P. Sleiman, M. Mittal, F. Farshidian, and M. Hutter, “A collision-free mpc for whole-body dynamic locomotion and manipulation,” in *2022 international conference on robotics and automation (ICRA)*, IEEE, 2022, pp. 4686–4693.

[53] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. John Wiley & Sons, 2020.

[54] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, 1st. Springer Publishing Company, Incorporated, 2008.

[55] D. Kouzoupis, G. Frison, A. Zanelli, and M. Diehl, “Recent advances in quadratic programming algorithms for nonlinear model predictive control,” *Vietnam Journal of Mathematics*, vol. 46, no. 4, pp. 863–882, 2018.

[56] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, “Perceptive locomotion through nonlinear model-predictive control,” *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3402–3421, 2023.

[57] J. Pan, S. Chitta, and D. Manocha, “Fcl: A general purpose library for collision and proximity queries,” in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 3859–3866.

[58] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, “Finding locally optimal, collision-free trajectories with sequential convex optimization.,” in *Robotics: science and systems*, Berlin, Germany, vol. 9, 2013, pp. 1–10.

[59] J. Schulman *et al.*, “Motion planning with sequential convex optimization and convex collision checking,” *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[60] S. J. Wright, *Numerical optimization*, 2006.

[61] T. D. Barfoot, *State Estimation for Robotics*. Cambridge University Press, 2017.

[62] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.

[63] G. Frison and M. Diehl, “Hpipm: A high-performance quadratic programming framework for model predictive control,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020.

[64] “Cppad: A package for differentiation of c++ algorithms.” (2024), [Online]. Available: `https://github.com/coin-or/CppAD` (visited on 10/07/2024).

[65] J. Carpentier *et al.*, “The pinocchio c++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives,” in *2019 IEEE/SICE International Symposium on System Integration (SII)*, 2019, pp. 614–619.

[66] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, “Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 1366–1373.

[67] R. Newbury *et al.*, “Deep learning approaches to grasp synthesis: A review,” *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3994–4015, 2023.

[68] Z. He, N. Chavan-Dafle, J. Huh, S. Song, and V. Isler, “Pick2place: Task-aware 6dof grasp estimation via object-centric perspective affordance,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 7996–8002.

[69] B. Brito, M. Everett, J. P. How, and J. Alonso-Mora, “Where to go next: Learning a subgoal recommendation policy for navigation in dynamic environments,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4616–4623, 2021.

[70] H. Hose, J. Köhler, M. N. Zeilinger, and S. Trimpe, “Approximate non-linear model predictive control with safety-augmented neural networks,” *arXiv preprint arXiv:2304.09575*, 2023.

Glossary

CBF Control Barrier Functions. 2
CS Configuration Space. 6, 7

DK differential kinematics. 7, 14, 17, 24
DOF Degrees of Freedom. 2, 3, 6, 17

EE end-effector. iii, 1, 3, 5, 7–10, 14–16, 19, 20, 22, 24–28
ESDF Euclidean Signed Distance Field. 30

FCL Flexible Collision Library. 17, 21
FK forward kinematics. 7, 10, 12, 14, 17, 24

GF Geometric Fabrics. i, iv, 17, 19–23, 29, 30
GJK Gilbert–Johnson–Keerthi. 10, 21

IDK inverse differential kinematics. 5

KF Kalman Filter. ii, 12, 13, 24

MM mobile manipulator. 1, 3
MotM manipulation on-the-move. 5
MPC Model Predictive Control. i, iii, iv, 1, 3–8, 10, 11, 14–30
MPPI Model Predictive Path Integral. 2, 5
MRMP Multi-robot Motion Planning. 3

NLP Nonlinear Programming Problem. 11
NNs neural networks. 30

OCS2 Optimal Control of Switched Systems. 5, 17

p.s.d. positive semidefinite. 10, 12

QP quadratic program. 3, 5, 10–12, 17

RL Reinforcement Learning. 30

SLQ Sequential Linear Quadratic. 5
SQP Sequential Quadratic Programming. 5, 6, 10, 11, 17

TS Task Space. 7, 9

UAVs Unmanned Aerial Vehicles. 3
UGVs Unmanned Ground Vehicles. 3
URDF Unified Robot Description Format. 8, 18, 21, 24

VMC Visibility Maximization Controller. 5