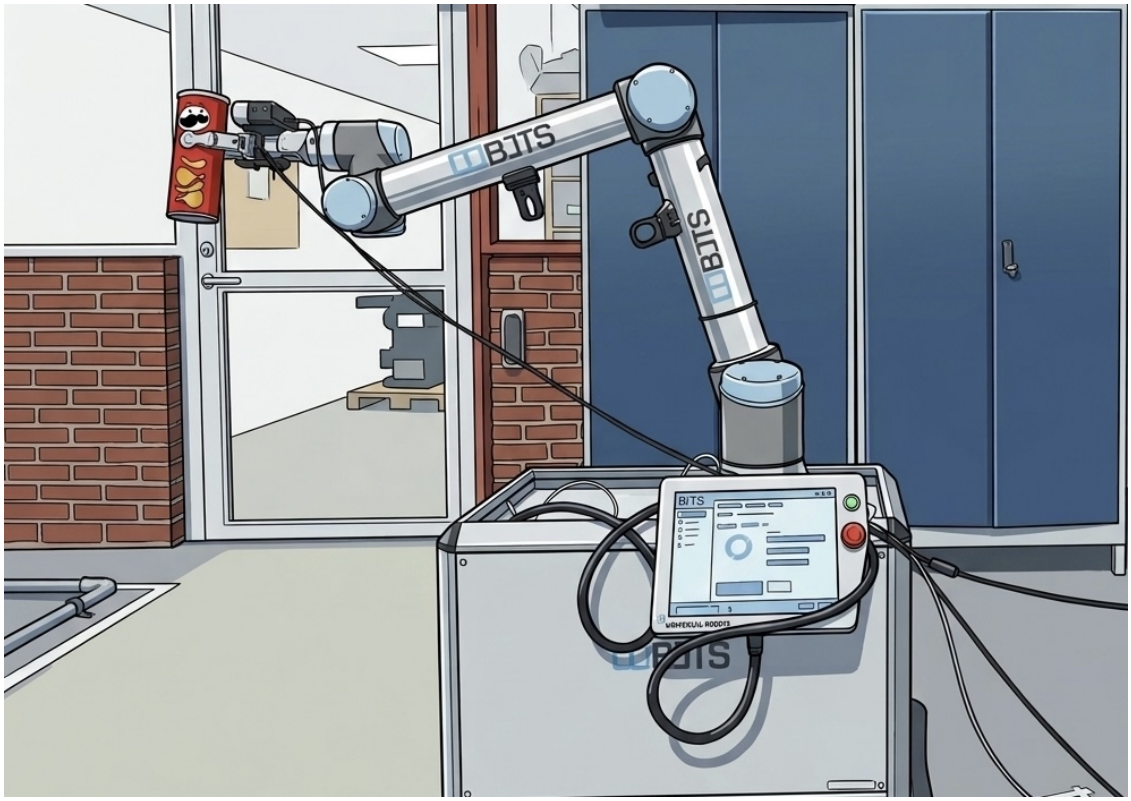
CHALMERS
UNIVERSITY OF TECHNOLOGY



Reactive Whole-Body Control for Grasping Moving Objects with Mobile Manipulators

Master's thesis in Systems, Control and Mechatronics

SHIJIE ZHOU
DONGXIANG LI

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Reactive Whole-Body Control for Grasping Moving Objects with Mobile Manipulators

SHIJIE ZHOU
DONGXIANG LI



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2026

Reactive Whole-Body Control for Grasping Moving Objects with Mobile Manipulators
SHIJIE ZHOU
DONGXIANG LI

© SHIJIE ZHOU, DONGXIANG LI 2026.

Supervisor: Yasemin Bekiroglu, Department of Electrical Engineering
Co-supervisor: Ahmet Ercan Tekden, Department of Electrical Engineering
Co-supervisor: Francesco Gigante, Department of Electrical Engineering
Examiner: Yasemin Bekiroglu, Department of Electrical Engineering

Master's Thesis 2026
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An illustration of the proposed mobile manipulation system showing grasping of a moving object in the laboratory environment.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Reactive Whole-Body Control for Grasping Moving Objects with Mobile Manipulators

SHIJIE ZHOU
DONGXIANG LI
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Grasping moving objects remains challenging for fixed-base manipulators because their limited workspace prevents them from continuously tracking objects that move beyond their reachable area. This limitation is particularly important in human–robot collaborative scenarios. In this work, we propose a framework for grasping moving objects with a mobile manipulator that dynamically repositions its base to expand its reachable workspace and improve grasp feasibility. To coordinate the system motion, we adopt a reactive whole-body control strategy that treats the mobile base and manipulator as a unified kinematic system. As the object moves, the target pre-grasp pose is updated based on the estimated object pose, allowing the robot to track and approach a grasp-relevant target. The proposed method is validated in both simulation and real-world experiments, demonstrating the effectiveness of the unified control strategy for coordinated tracking and grasp-oriented motion generation when interacting with moving objects.

Keywords: whole-body control, mobile manipulation, obstacle avoidance, reactive grasping.

Acknowledgements

We would like to express our sincere gratitude to our supervisor, Yasemin Bekiroglu, for her guidance, support, and valuable feedback throughout this thesis project. Her advice helped us clarify the research direction and improve the quality of this work.

We are also very grateful to Ahmet Ercan Tekden and Francesco Gigante for their continuous support during the project. Through our weekly meetings, they provided many helpful suggestions on the system design, experiment setup, implementation details, and result analysis. Their feedback was especially valuable in helping us identify problems and improve the experimental workflow.

Finally, we would like to thank our families and friends for their encouragement and support during our master's studies. Their understanding and support helped us stay motivated throughout the thesis process.

Shijie Zhou, Dongxiang Li, Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AMCL	Adaptive Monte Carlo Localization
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
FIFO	First-In-First-Out
FPFH	Fast Point Feature Histogram
HQP	Hierarchical Quadratic Programming
ICP	Iterative Closest Point
IoU	Intersection over Union
MAE	Mean Absolute Error
MiR	Mobile Industrial Robot
MLP	Multi-Layer Perceptron
PVS	Promptable Visual Segmentation
QP	Quadratic Programming
RANSAC	Random Sample Consensus
RG2-FT	RG2 Gripper with Force/Torque Sensor
RGB-D	Red, Green, Blue and Depth
ROS	Robot Operating System
RViz	ROS Visualization Tool
SAM 2	Segment Anything Model 2
SE(3)	Special Euclidean Group in Three Dimensions
SO(3)	Special Orthogonal Group in Three Dimensions
SPFH	Simplified Point Feature Histogram
SVD	Singular Value Decomposition
TF	Transform
UR10	Universal Robots UR10 Manipulator
ViT	Vision Transformer
VOS	Video Object Segmentation
WBC	Whole-Body Control

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i	Index of a robot joint
n	Number of manipulator joints
t	Time or time index

Sets

$\mathbb{R}^n$	n-dimensional real-valued vector space
$SO(3)$	Special orthogonal group representing three-dimensional rotations
$SE(3)$	Special Euclidean group representing three-dimensional rigid-body transformations

Parameters

b	Baseline distance between the left and right cameras
f	Camera focal length
f_x, f_y	Focal lengths in the camera intrinsic matrix
c_x, c_y	Principal point coordinates in the camera intrinsic matrix
$\mathbf{x}_{min}$	Lower bound of the QP decision variable
$\mathbf{x}_{max}$	Upper bound of the QP decision variable

Variables

$\mathbf{q}$	Generalized robot configuration
$\mathbf{q}_b$	Mobile base configuration
$\mathbf{q}_a$	Manipulator joint configuration

$\dot{\mathbf{q}}$	Generalized robot velocity
$\dot{\mathbf{q}}_b$	Mobile base velocity
$\dot{\mathbf{q}}_a$	Manipulator joint velocity
q_i	Position of the i-th manipulator joint
$\dot{q}_i$	Velocity of the i-th manipulator joint
v_b	Forward linear velocity of the mobile base
ω_b	Yaw angular velocity of the mobile base
θ_b	Heading angle of the mobile base
$\mathbf{p}_b$	Planar pose or position of the mobile base
$\mathbf{p}_e$	Position of the end-effector
$\mathbf{p}_{center}$	Position of the controlled `grasp_center` frame
$\mathbf{p}_{target}$	Position of the target-related `grasp_target` frame
$\mathbf{x}_e$	End-effector task-space pose
$\mathbf{v}_e$	End-effector spatial velocity
$\mathbf{v}^*$	Desired end-effector spatial velocity
$\boldsymbol{\nu}$	Linear velocity component of a spatial velocity
$\boldsymbol{\omega}$	Angular velocity component of a spatial velocity
$\mathbf{T}$	Homogeneous transformation matrix
${}^{A}\mathbf{T}_{B}$	Pose of frame $\{B\}$ expressed in frame $\{A\}$
$\mathbf{T}_{ep}$	Expected end-effector pose
$\mathbf{T}_{err}$	Relative pose error transformation
$\mathbf{R}$	Rotation matrix
${}^{A}\mathbf{R}_{B}$	Orientation of frame $\{B\}$ expressed in frame $\{A\}$
${}^{A}\mathbf{p}_{B}$	Position of frame $\{B\}$ expressed in frame $\{A\}$
$\mathbf{K}$	Camera intrinsic matrix
$\mathbf{J}(\mathbf{q})$	Whole-body Jacobian matrix
$\mathbf{J}_e(\mathbf{q})$	Jacobian of the controlled end-effector frame
$\mathbf{x}$	QP decision variable
$\mathbf{Q}$	Quadratic cost matrix of the QP problem
$\mathbf{c}$	Linear cost vector of the QP problem
$\mathbf{A}_{eq}$	Equality constraint matrix
$\mathbf{b}_{eq}$	Equality constraint vector
$\boldsymbol{\delta}$	Task-space slack variable
e_t	Translational pose error
$e(t)$	Tracking error at time t

Contents

List of Acronyms viii

Nomenclature x

List of Figures xiv

List of Tables xvi

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Mobile Manipulation for Moving Objects . . . 2
- 1.3 Challenges in Reactive Tracking . . . 2
- 1.4 Aim and Scope . . . 3
- 1.5 Related Work . . . 4
- 1.6 Contributions . . . 7

2 Theory 8
- 2.1 Mobile Manipulation . . . 8
- 2.2 Coordinate Frames and Rigid Body Transformations . . . 11
- 2.3 Robot Kinematics . . . 14
- 2.4 Visual Perception . . . 17
  - 2.4.1 Depth Sensing with the Femto Bolt Camera and Point Cloud Representation . . . 18
  - 2.4.2 Segment Anything Model 2 (SAM 2) . . . 19
  - 2.4.3 Calculating Point Cloud Normal Vectors . . . 20
  - 2.4.4 Point Cloud Filtering . . . 21
  - 2.4.5 Point Cloud Registration . . . 21
    - 2.4.5.1 Fast Point Feature Histogram (FPFH) . . . 22
    - 2.4.5.2 Random Sample Consensus(RANSAC) . . . 23
    - 2.4.5.3 Colored Point Cloud Registration . . . 24
- 2.5 Whole-Body Control of Mobile Manipulators . . . 25
- 2.6 Quadratic Programming for Reactive Control . . . 26
- 2.7 Obstacle Avoidance in Mobile Manipulation . . . 28

3 Methods 32
- 3.1 Robot Platform and Simulation Environment . . . 32
- 3.2 Software Framework and ROS Architecture . . . 34

3.2.1 Object grasping pose generation . . . 34
3.2.1.1 Image Acquisition and Segmentation . . . 34
3.2.1.2 Point Cloud Extraction and Preprocessing . . . 35
3.2.1.3 Dual-track Pose Estimation State Machine . . . 35
3.2.1.4 Grasp Strategy and Pose Generation . . . 35
3.2.2 Mapping and Localization . . . 36
3.2.3 Navigation and Costmap Representation . . . 37
3.2.4 ROS Communication and Coordinate Transformations . . . 38
3.3 Whole-body Controller . . . 40
3.3.1 Generalized Velocity Representation . . . 41
3.3.2 Pose Servoing . . . 41
3.3.3 QP-Based Velocity Optimization . . . 42
3.3.4 Controller State Machine . . . 43
3.3.5 Command Publication . . . 44
3.3.6 Summary . . . 44

4 Results **45**
4.1 Simulation Results . . . 45
4.1.1 Trajectory Tracking Results . . . 46
4.1.2 Obstacle-related Simulation Results . . . 48
4.2 Real-world Grasping Results . . . 49
4.2.1 Experimental Setup . . . 49
4.2.2 Grasping Results with Different Objects . . . 50
4.2.3 Quantitative Data Analysis . . . 55
4.3 Summary of Results . . . 56

5 Conclusion **58**

Bibliography **60**

List of Figures

1.1 Overview of the proposed reactive mobile manipulation system. . . . 4

2.1 Example of a coordinate frame tree in a mobile manipulation system. Transformations between connected frames can be composed to express poses in a common reference frame. 14
2.2 Overview of QP-based whole-body control. The expected pose and robot constraints are formulated as an optimization problem, and the solution provides whole-body velocity commands. 28
2.3 Distance-based obstacle avoidance concept. The robot should maintain a distance greater than the minimum safety distance $d_{\min}$ from obstacles. 29

3.1 Simulation environment used in this thesis. 33
3.2 Vision-guided tracking, grasping, and control 34
3.3 Map generated using `hector_mapping`. 36
3.4 Visualization of the costmap in RViz. 38
3.5 Main TF chain from map to manipulator. 40
3.6 Main coordinate transformation structure of the mobile manipulation system. 40

4.1 Tracking evaluation for the sideway motion trajectory in CoppeliaSim. 46
4.2 Tracking evaluation for the backward motion trajectory in CoppeliaSim. 47
4.3 Tracking evaluation for the nonlinear motion trajectory in CoppeliaSim. 47
4.4 Obstacle-related simulation setup in CoppeliaSim. The scene includes the mobile manipulator, target, and obstacle elements. 48
4.5 Example obstacle avoidance scenario in CoppeliaSim. 49
4.6 Real-world experimental setup using the MiR mobile base, UR10 manipulator, RG2-FT gripper, and wrist-mounted RGB-D camera. . . . 50
4.7 Initial RViz visualization of the Pringles can grasping experiment. The detected target pose and robot state are visualized before the grasping motion. 51
4.8 Final RViz visualization of the Pringles can grasping experiment. The end-effector reaches the target region after whole-body tracking and final grasp execution. 51
4.9 Trajectory comparison between `grasp_center` and `grasp_target` during the Pringles can experiment. The comparison shows how the controlled end-effector follows the target reference trajectory. 52

4.10 Real-world grasping sequence for the cleanser bottle. 53
4.11 Real-world grasping sequence for the cracker box. 53
4.12 Real-world grasping sequence for the Pringles can. 54
4.13 Real-world grasping sequence for the baseball. The three images show the initial state, intermediate tracking state, and final grasping state. 54
4.14 Tracking error over time during the selected tracking stage of the Pringles can experiment. The peak error occurs when the target is moved quickly toward the final grasping position. 55
4.15 Tracking error distribution during tracking stage of the Pringles can experiment. 56

List of Tables

2.1 Main components of the mobile manipulation platform. 10
2.2 Interpretation of common QP elements in robot control. 28

4.1 Average pre-grasp tracking error in simulation. 48

1
Introduction

1.1 Background

Robotic manipulation has been an important research topic in robotics for several decades [1]. Manipulation tasks such as grasping [2, 3, 4], throwing objects into bins [5], pick-and-place [6], object transportation, and object interaction are essential for enabling robots to operate in human-centered environments. In industrial applications, fixed-base manipulators have been widely used due to their high accuracy, repeatability, and reliability. When the target object is located within the reachable workspace of the manipulator and the environment is structured, such systems can perform manipulation tasks effectively.

However, fixed-base manipulators are inherently limited by their reachable workspace. If the target object is located outside the workspace of the arm, the task cannot be completed without repositioning the robot or modifying the environment. This limitation becomes more significant in unstructured or dynamic environments, where objects may appear at different locations or move during the task. In such cases, relying only on the motion of the robotic arm is often insufficient.

Mobile manipulators address this limitation by combining a mobile base with a robotic arm. The mobile base extends the operational workspace of the robot, while the manipulator provides the dexterity required for interaction with objects. This combination allows the robot to approach objects from different positions, maintain reachability, and perform manipulation tasks over a larger area. As a result, mobile manipulators are suitable for applications such as service robotics, warehouse automation, assistive robotics, and mobile object handling [7].

Despite these advantages, mobile manipulation introduces additional control challenges. The motion of the mobile base and the robotic arm must be coordinated in a way that supports the manipulation task [8]. If the base and the arm are controlled independently, the system may fail to fully exploit the redundancy of the mobile manipulator. Therefore, whole-body control strategies, where the base and arm are treated as a unified system, are important for achieving efficient and reactive mobile manipulation [9, 10].

1.2 Mobile Manipulation for Moving Objects

Many manipulation systems are designed under the assumption that the target object is static or changes slowly. In these scenarios, the robot can first estimate the target pose, plan a suitable motion, and then execute the manipulation task. For a mobile manipulator, this often means that the mobile base first moves to a desired position, after which the robotic arm performs the grasping or manipulation action.

This sequential strategy can be effective for static environments. However, it becomes less suitable when the target object is moving. When the object pose changes continuously, the robot must react to the motion of the target in real time [9, 11]. The desired grasp or pre-grasp pose must be updated continuously, and the robot motion must be adapted accordingly. If the object moves outside the reachable workspace of the arm, the mobile base must also move to maintain a suitable configuration for manipulation.

Grasping moving objects therefore requires the integration of perception, target tracking, motion generation, and control [12, 13, 14, 15, 16]. The robot must estimate the pose of the moving object, generate a grasp-related target pose, and control the whole robot to follow this target. Any delay or inaccuracy in perception or control may reduce the tracking performance and make the grasping task more difficult. Compared with grasping static objects, grasping moving objects requires the robot to respond continuously rather than relying on a single planned motion.

In this thesis, the focus is on reactive grasp-oriented tracking of moving objects using a mobile manipulator. Instead of treating the mobile base and robotic arm as separate subsystems, the robot is controlled as a whole-body kinematic system. This allows the base and arm to move simultaneously, enabling the robot to track target poses generated from moving objects while maintaining a suitable configuration for grasping.

1.3 Challenges in Reactive Tracking

Reactive grasp-oriented tracking with a mobile manipulator involves several challenges. First, the robot needs to continuously estimate the pose of the target object. In a vision-based system, this requires processing sensor data such as RGB images, depth images, or point clouds. The estimated object pose is then used to generate a grasp or pre-grasp pose, which serves as the target for the controller.

Second, the controller must generate coordinated motion for both the mobile base and the robotic arm. The mobile base provides large-scale motion and extends the reachable workspace, while the robotic arm provides local dexterity and precise end-effector motion. These two components have different kinematic properties and motion constraints. A suitable control framework must therefore coordinate them in a consistent and task-oriented manner.

Third, the system must operate in real time. Since the object is moving, the target pose may change before the robot reaches it. The controller must continuously update the robot motion based on the latest target estimate. This requires a reactive control strategy rather than a purely offline planning approach.

Fourth, obstacle avoidance must be considered when the robot moves in the environment. During tracking, the mobile base may need to reposition itself to maintain reachability. At the same time, the robot must avoid collisions with obstacles. This introduces an additional constraint on the robot motion and makes the tracking task more complex.

Together, these challenges require a framework that can transform continuously updated target estimates into coordinated base–arm commands in real time. In this thesis, whole-body control addresses the coordination and reachability challenges by treating the mobile base and the manipulator as a unified kinematic system. The QP-based formulation enables the controller to track the updated target pose while accounting for robot velocity and joint constraints. Visual perception provides the continuously updated target reference, while obstacle-related constraints are evaluated in simulation to investigate the feasibility of obstacle-aware tracking.

1.4 Aim and Scope

The aim of this thesis is to investigate a reactive whole-body control framework for grasp-oriented tracking of moving objects with a mobile manipulator. The proposed framework is designed to coordinate the motion of a mobile base and a robotic arm so that the robot can continuously follow grasp-related target poses generated from a moving object.

Figure 1.1 shows the overview of the proposed system. The mobile manipulator used in this work consists of a MiR mobile base and a UR10 robotic arm. The system is evaluated in simulation and, where applicable, on a real robotic platform. The simulation experiments are used to analyze the tracking behavior of the controller under different object motion patterns, including sideways motion, backward motion, and nonlinear motion. Obstacle avoidance is also considered to evaluate whether the robot can maintain target tracking while avoiding collisions.

The scope of this thesis includes the development, integration, and evaluation of the reactive whole-body control framework. The focus is mainly on coordinated base-arm motion, grasp-oriented target tracking, and experimental validation. The work does not aim to solve all aspects of autonomous grasping in unstructured environments. Instead, it focuses on enabling the robot to reactively track moving grasp-related target poses using whole-body motion.

The main research questions addressed in this thesis are:

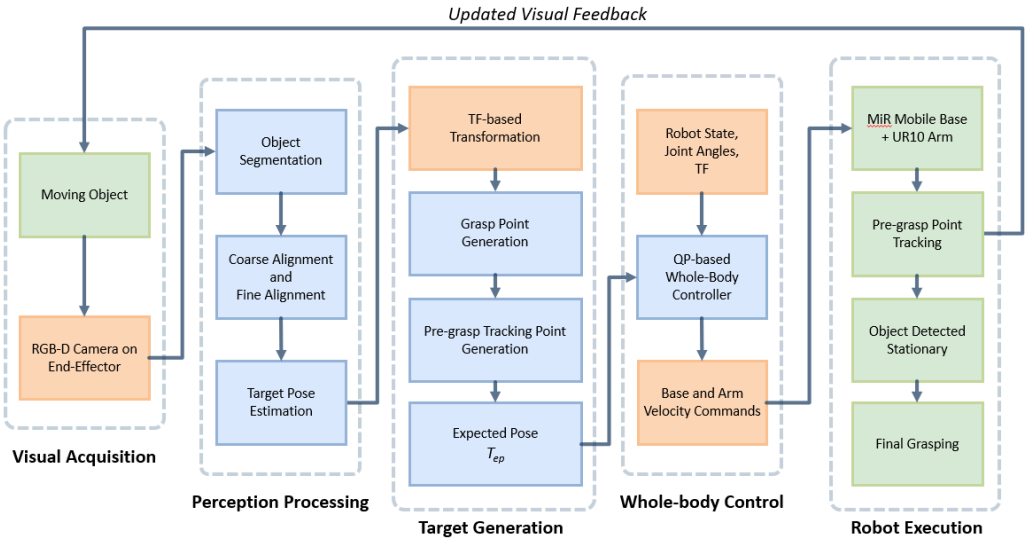


Figure 1.1: Overview of the proposed reactive mobile manipulation system.

- How can a mobile base and robotic arm be coordinated as a unified system for reactive tracking of moving objects?
- How can grasp-related target poses be continuously followed when the object pose changes over time?
- How does the proposed whole-body control framework perform under different target motion patterns?
- Can obstacle avoidance be integrated while maintaining grasp-oriented tracking performance?
- How does the proposed framework perform in real-world tracking and grasping experiments on a physical mobile manipulator?

1.5 Related Work

Mobile manipulators have been widely investigated because they combine the mobility of autonomous mobile robots with the dexterity of robotic manipulators [17, 18]. Compared with fixed-base manipulators, a mobile manipulator can operate over a larger workspace and perform tasks at different locations in the environment. This capability makes mobile manipulators suitable for applications such as logistics, service robotics, industrial assistance, inspection, and human-centered environments. Surveys on wheeled mobile manipulation have highlighted that the combination of base mobility and arm manipulation enables a wide range of interaction tasks, but also introduces challenges in decision making, motion coordination, perception, and task execution [19, 20]. In industrial contexts, mobile manipulators are also considered important for flexible automation, since they can support both transportation and manipulation functions within the same robotic platform.

Despite these advantages, mobile manipulation remains a complex system-level prob-

lem. A practical mobile manipulation system usually requires the integration of localization, mapping, navigation, perception, object detection, motion planning, control, and robot execution [20, 21]. Recent system-level studies have shown that building a multifunctional mobile manipulation robot often requires the combination of perception, mapping, navigation, object detection, and grasping within a unified architecture. This integration is difficult because the different modules operate at different time scales and with different sources of uncertainty. For example, localization and mapping provide global information about the robot and the environment, while perception and control must often respond to local and rapidly changing target states. Therefore, mobile manipulation is not only a problem of designing a robotic arm controller, but also a problem of coordinating multiple subsystems in a consistent and real-time manner.

A common approach in mobile manipulation is to decompose the task into sequential stages [4, 22, 23]. In such a pipeline, the mobile base first navigates to a suitable pose near the target, and the manipulator then executes the manipulation task while the base remains stationary. This "stop-and-manipulate" strategy is practical and has been widely used because it simplifies the control problem [24]. By separating base navigation from arm manipulation, the system can use existing navigation planners and manipulator controllers independently. However, this separation also limits the ability of the robot to react to moving targets or changing environments. When the target pose changes during execution, the robot may need to replan, stop, or reposition the base before the arm can continue the task. This can lead to inefficient motion and reduced robustness in dynamic scenarios.

To overcome these limitations, recent research has explored reactive mobile manipulation, where the robot performs manipulation while the base is moving or continuously adapting its configuration. Haviland et al. proposed a holistic approach to reactive mobile manipulation, in which the arm and base degrees of freedom are treated as a single integrated structure rather than independent subsystems [9]. Their work demonstrates that treating the mobile manipulator holistically can improve the fluidity and responsiveness of the resulting motion. Similarly, Burgess-Limerick et al. proposed an architecture for reactive mobile manipulation on-the-move, aiming to reduce task execution time by allowing the robot to manipulate objects while the base continues moving toward the next objective [24]. These studies are closely related to this thesis because they emphasize continuous coordination between base motion and arm motion instead of a strictly sequential navigation-then-manipulation pipeline.

Whole-body control provides an important framework for achieving such coordination. In whole-body control, the mobile base and the robotic arm are modeled as a coupled system [9, 10], and control commands are computed for all relevant degrees of freedom. This allows the controller to exploit the kinematic redundancy of the mobile manipulator. For example, the manipulator may compensate for small target motions using the arm, while the mobile base may reposition the system to keep the target within a favorable workspace. Optimization-based methods are commonly

used in this context because they can express task objectives and constraints in a systematic way. Quadratic programming (QP) formulations are especially useful for handling equality tasks, inequality constraints, joint limits, velocity limits, and secondary objectives [25].

Kim et al. presented a whole-body controller for a non-holonomic mobile manipulator based on hierarchical quadratic programming and continuous task transition [10]. Their work is relevant because many mobile manipulators, including differential-drive platforms, are subject to non-holonomic motion constraints. HQP-based control allows multiple tasks to be arranged according to priority, while still considering equality and inequality constraints. More generally, Escande et al. showed how hierarchical least-squares or hierarchical quadratic programming can be used for fast online robot motion generation under kinematic constraints [25]. These methods provide a theoretical basis for using optimization in real-time whole-body motion generation. In this thesis, a QP-based controller is used to coordinate the velocity commands of the MiR mobile base and the UR10 manipulator, with the goal of maintaining the desired end-effector motion while respecting practical constraints such as joint velocity limits.

Another important aspect of mobile manipulation is the use of visual feedback. Visual servoing refers to the use of visual information in the control loop to guide robot motion [26]. Classical visual servoing methods are often categorized into image-based visual servoing, where control is based directly on image features, and position-based visual servoing, where the target pose is estimated and used as feedback [26, 27]. For mobile manipulation, visual feedback is particularly useful because the target may not remain fixed relative to the robot. If the target is moving, the robot must continuously update the target pose and adjust its motion accordingly. Therefore, perception-based control is essential for tasks involving dynamic objects or uncertain object positions.

However, applying visual feedback to a mobile manipulator is more challenging than applying it to a fixed-base manipulator [26, 9]. First, the camera, base, arm, and end-effector may all belong to different coordinate frames, so the system must maintain accurate transformations between these frames. Second, perception data may contain noise, delay, and occasional detection failures. Third, the controller must generate commands that are feasible for both the manipulator and the mobile base. These challenges become more significant when the target is moving, because perception, transformation, optimization, and command execution must be repeated continuously. Therefore, a practical system must not only estimate the target pose, but also ensure that the estimated pose can be used reliably by the whole-body controller.

Navigation and obstacle avoidance are also closely related to mobile manipulation. In static manipulation tasks, the base can often be placed at a predefined pose before the arm starts moving. In contrast, dynamic or reactive tasks require the mobile base to continue moving while still avoiding obstacles and maintaining a

suitable configuration for manipulation. This creates a potential conflict between navigation objectives and manipulation objectives. For example, the base may need to move around an obstacle, while the end-effector must still maintain a desired pose relative to the target. As a result, mobile manipulation systems often need to combine global navigation, local obstacle avoidance, and manipulation control. In ROS-based systems, this is commonly handled through navigation modules, costmap representations, and local planners, while the manipulation controller handles the end-effector task [28, 29]. The difficulty is to make these components work together without producing inconsistent or delayed commands.

The related work shows that mobile manipulation has gradually moved from sequential pipelines toward more integrated and reactive frameworks. Existing research has demonstrated the value of treating the mobile base and manipulator as a coordinated system, especially when the robot must operate in dynamic environments [9, 24]. At the same time, implementing such a system on a real mobile manipulator remains challenging because the controller must be connected to perception, navigation, coordinate transformations, and hardware execution. This thesis builds on these ideas by developing a ROS-based whole-body mobile manipulation framework for dynamic target tracking. The proposed system coordinates a MiR mobile base and a UR10 manipulator through a QP-based controller, while incorporating perception feedback and navigation-related information. The system is evaluated in both CoppeliaSim simulation and real-world experiments, with the aim of demonstrating the feasibility of coordinated mobile manipulation for moving-target scenarios.

1.6 Contributions

The main contributions of this thesis are summarized as follows:

- A reactive whole-body control framework is implemented for coordinated control of a mobile base and robotic arm.
- A grasp-oriented tracking pipeline is integrated, allowing the robot to continuously update and follow target poses associated with moving objects.
- The mobile manipulator is modeled and evaluated as a unified kinematic system, enabling simultaneous motion of the base and the arm.
- The proposed framework is tested in simulation using different moving target trajectories, including sideways, backward, and nonlinear motion patterns.
- Obstacle avoidance is incorporated into the control framework to evaluate the robot's ability to track moving targets in more complex environments.
- Real-world experiments are conducted to validate the feasibility of the framework on a physical mobile manipulator platform.

2
Theory

This chapter presents the theoretical background required for understanding the proposed reactive whole-body mobile manipulation framework. The focus is on mobile manipulation, coordinate transformations, robot kinematics, visual perception, grasp-oriented target generation, whole-body control, quadratic programming, and obstacle avoidance. These concepts provide the foundation for the methodology and implementation described in the following chapters.

2.1 Mobile Manipulation

Mobile manipulation refers to robotic systems that combine a mobile base with one or more robotic manipulators. The mobile base provides locomotion in the environment, while the manipulator provides dexterous local interaction with objects. Compared with a fixed-base manipulator, a mobile manipulator can extend its operational workspace by changing the base pose. This makes mobile manipulators suitable for tasks in which the object is located outside the arm's immediate reachable workspace, or where the target pose changes during task execution.

A fixed-base manipulator can only operate within the workspace determined by its joint limits and link geometry. In contrast, a mobile manipulator has a workspace that depends not only on the manipulator configuration, but also on the pose of the mobile base. If the base pose is denoted by $\mathbf{q}_b$ and the arm joint configuration is denoted by $\mathbf{q}_a$, the generalized configuration of a mobile manipulator can be written as

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_b \\ \mathbf{q}_a \end{bmatrix}. \tag{2.1}$$

For a planar mobile base, the pose of the base in the world frame can be represented as

$$\mathbf{p}_b = \begin{bmatrix} x_b & y_b & \theta_b \end{bmatrix}^T, \tag{2.2}$$

where x_b and y_b describe the position of the base, and θ_b denotes the heading angle. Although the planar pose contains three variables, the MiR base is treated as a non-holonomic platform with two controllable velocity inputs:

$$\mathbf{u}_b = \begin{bmatrix} v_b & \omega_b \end{bmatrix}^T, \tag{2.3}$$

where v_b is the forward linear velocity and ω_b is the angular velocity around the vertical axis. The lateral velocity of the base is not directly controllable. The

manipulator configuration is given by

$$\mathbf{q}_a = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}^T, \tag{2.4}$$

where n is the number of manipulator joints. In this thesis, the robotic arm is a UR10 manipulator with six revolute joints.

The mobility of the base and the dexterity of the arm provide complementary capabilities. The base is suitable for large-scale displacement and repositioning in the environment, while the arm is suitable for accurate local motion near the target. However, these two subsystems also have different motion characteristics. The mobile base is usually slower and less precise than the manipulator, and its motion may be constrained by non-holonomic kinematics and obstacles in the environment. The manipulator, on the other hand, can generate accurate end-effector motion, but its workspace is limited and affected by joint limits and singular configurations. Therefore, effective mobile manipulation requires coordinated control of the base and the arm.

In this thesis, the mobile manipulation platform consists of a MiR mobile base and a UR10 robotic arm. The MiR base provides planar mobility, allowing the robot to reposition itself in the environment, while the UR10 arm provides the local dexterity required for reaching and grasp-oriented tracking. The combination of these two subsystems makes it possible to approach moving targets from different positions and maintain reachability when the target moves beyond the arm's immediate workspace.

The pose of the end-effector depends on both the base pose and the arm joint configuration. This relationship can be expressed by the forward kinematic mapping

$$\mathbf{x}_e = f(\mathbf{q}_b, \mathbf{q}_a), \tag{2.5}$$

where $\mathbf{x}_e$ denotes the task-space pose of the end-effector. In rigid-body form, the end-effector pose can also be written as a homogeneous transformation

$${}^{w}\mathbf{T}_e = {}^{w}\mathbf{T}_b(\mathbf{q}_b)\,{}^{b}\mathbf{T}_e(\mathbf{q}_a), \tag{2.6}$$

where ${}^{w}\mathbf{T}_b$ is the transformation from the world frame to the mobile base frame, and ${}^{b}\mathbf{T}_e$ is the transformation from the base frame to the end-effector frame. This equation shows that the end-effector motion is influenced by both base motion and arm motion. As a result, the same end-effector pose may be achieved by multiple combinations of base and arm configurations.

This redundancy is one of the main advantages of mobile manipulation. When the robot has more controllable degrees of freedom than the task strictly requires, the additional degrees of freedom can be used to satisfy secondary objectives. For example, the robot may keep the arm away from joint limits, maintain a favorable manipulation posture, avoid obstacles, or reposition the mobile base to improve

reachability. In velocity form, the relationship between the generalized velocity and the end-effector velocity can be written as

$$\mathbf{v}_e = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}, \tag{2.7}$$

where $\dot{\mathbf{q}}$ is the generalized velocity and $\mathbf{J}(\mathbf{q})$ is the whole-body Jacobian. The generalized velocity can be divided into base and arm components:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\mathbf{q}}_b \\ \dot{\mathbf{q}}_a \end{bmatrix}. \tag{2.8}$$

Accordingly, the whole-body Jacobian can be written as

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_b(\mathbf{q}) & \mathbf{J}_a(\mathbf{q}) \end{bmatrix}, \tag{2.9}$$

where $\mathbf{J}_b$ describes the effect of base motion on the end-effector velocity, and $\mathbf{J}_a$ describes the effect of arm joint motion. This formulation provides the foundation for whole-body control, where the base and the arm are controlled together to achieve a desired end-effector motion.

In many conventional mobile manipulation systems, the task is divided into separate navigation and manipulation stages. The base first moves to a planned pose near the object, and the arm then performs the manipulation task. This sequential strategy is effective for static objects because it simplifies the problem into two relatively independent parts. However, it is less suitable for moving targets. If the object moves while the robot is executing the task, a precomputed base pose may no longer be appropriate. The robot must instead continuously adapt its motion according to the updated target pose.

The mobile manipulation setting in this thesis is therefore different from that of a purely navigation-oriented mobile robot or a fixed-base manipulation system, as the mobile base and the robotic arm are treated as a unified system. Rather than planning the base motion and arm motion separately, the controller computes coordinated motion for both subsystems. In this setting, the desired task is expressed as a target end-effector pose, while the controller determines how much of the velocity should be produced by the base and how much should be produced by the arm.

Table 2.1: Main components of the mobile manipulation platform.

Component	**Role in the system**
MiR mobile base	Provides planar mobility and extends the reachable workspace
UR10 robotic arm	Provides local dexterity for reaching and grasp-oriented tracking
End-effector / gripper	Interacts with the target object during grasping
RGB-D camera	Provides visual and depth information for object pose estimation
Whole-body controller	Coordinates base and arm motion for reactive target tracking

2.2 Coordinate Frames and Rigid Body Transformations

Robotic systems often involve multiple coordinate frames. A mobile manipulation system may include frames attached to the world, the mobile base, the robotic arm, the end-effector, sensors, and objects in the environment. Since poses and measurements are usually expressed with respect to different reference frames, coordinate transformations are required to describe them in a common frame. This is especially important in mobile manipulation, where perception, control, navigation, and robot execution may all use different reference frames.

A coordinate frame defines a reference origin and orientation. In this thesis, several types of frames are used. A world or odometry frame provides a global reference for the mobile robot. A base frame is attached to the mobile platform and moves with the robot. The arm frames are attached to the links and joints of the UR10 manipulator. The end-effector frame represents the pose of the tool or gripper. Camera frames are used to express visual and depth measurements, while object or target frames are used to describe the pose of the detected target. To control the robot correctly, poses estimated in a camera frame must often be transformed into a robot-related frame such as the odometry frame, base frame, or end-effector frame.

The pose of one frame relative to another can be described by a rigid body transformation. In three-dimensional space, a rigid body transformation consists of a rotation and a translation. The rotation describes the relative orientation between two frames, while the translation describes the displacement between their origins. A common representation of a rigid body transformation is the homogeneous transformation matrix

$$ {}^{A}\mathbf{T}_{B} = \begin{bmatrix} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{p}_{B} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \in SE(3), \tag{2.10} $$

where ${}^{A}\mathbf{T}_{B}$ denotes the pose of frame $\{B\}$ expressed in frame $\{A\}$. The matrix ${}^{A}\mathbf{R}_{B} \in SO(3)$ represents the orientation of frame $\{B\}$ relative to frame $\{A\}$, and ${}^{A}\mathbf{p}_{B} \in \mathbb{R}^3$ represents the position of the origin of frame $\{B\}$ expressed in frame $\{A\}$.

The rotation matrix belongs to the special orthogonal group $SO(3)$ and satisfies

$$ \mathbf{R}^{T}\mathbf{R} = \mathbf{I}, \qquad \det(\mathbf{R}) = 1. \tag{2.11} $$

These properties indicate that a rotation matrix preserves lengths and angles. They are also important when computing inverse transformations, since the inverse of a rotation matrix is equal to its transpose.

A point expressed in frame $\{B\}$ can be transformed into frame $\{A\}$ using homogeneous coordinates. If a point is written as

$$ {}^{B}\bar{\mathbf{p}} = \begin{bmatrix} {}^{B}\mathbf{p} \\ 1 \end{bmatrix}, \tag{2.12} $$

then the same point expressed in frame $\{A\}$ is given by

$$ {}^{A}\bar{\mathbf{p}} = {}^{A}\mathbf{T}_{B}\, {}^{B}\bar{\mathbf{p}}. \tag{2.13} $$

Equivalently, this can be written in non-homogeneous form as

$$ {}^{A}\mathbf{p} = {}^{A}\mathbf{R}_{B}\, {}^{B}\mathbf{p} + {}^{A}\mathbf{p}_{B}. \tag{2.14} $$

This operation is commonly used to express sensor measurements, object positions, and robot poses in a desired reference frame.

Rigid body transformations can also be composed. If the pose of frame $\{B\}$ relative to frame $\{A\}$ and the pose of frame $\{C\}$ relative to frame $\{B\}$ are known, then the pose of frame $\{C\}$ relative to frame $\{A\}$ is given by

$$ {}^{A}\mathbf{T}_{C} = {}^{A}\mathbf{T}_{B}\, {}^{B}\mathbf{T}_{C}. \tag{2.15} $$

This property allows transformations to be chained through a kinematic or sensor frame tree. For example, a target pose detected in the camera frame can be transformed into the odometry frame through

$$ {}^{odom}\mathbf{T}_{target} = {}^{odom}\mathbf{T}_{base}\, {}^{base}\mathbf{T}_{camera}\, {}^{camera}\mathbf{T}_{target}. \tag{2.16} $$

This type of transformation is required when the perception module estimates a target pose in the camera coordinate frame, but the controller requires the target pose in a world, odometry, or robot base frame.

The inverse of a homogeneous transformation is used to transform quantities in the opposite direction. Given

$$ {}^{A}\mathbf{T}_{B} = \begin{bmatrix} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{p}_{B} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}, \tag{2.17} $$

the inverse transformation is

$$ {}^{B}\mathbf{T}_{A} = \left({}^{A}\mathbf{T}_{B}\right)^{-1} = \begin{bmatrix} ({}^{A}\mathbf{R}_{B})^{T} & -({}^{A}\mathbf{R}_{B})^{T}\, {}^{A}\mathbf{p}_{B} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}. \tag{2.18} $$

This follows from the orthonormality of rotation matrices. In practice, inverse transformations are frequently used when changing the reference frame of a pose, for example when expressing a target pose relative to the end-effector or base frame.

In addition to transforming points and poses, it is often necessary to transform velocities. A rigid-body velocity in three-dimensional space can be represented as a twist,

$$ \mathbf{v} = \begin{bmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{bmatrix} \in \mathbb{R}^{6}, \tag{2.19} $$

where $\boldsymbol{\nu} \in \mathbb{R}^3$ is the linear velocity and $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular velocity. If a twist is transformed from frame $\{B\}$ to frame $\{A\}$, the adjoint transformation associated with ${}^{A}\mathbf{T}_{B}$ can be used:

$$ {}^{A}\mathbf{v} = \mathrm{Ad}_{{}^{A}\mathbf{T}_{B}} {}^{B}\mathbf{v}, \tag{2.20} $$

where

$$ \mathrm{Ad}_{{}^{A}\mathbf{T}_{B}} = \begin{bmatrix} {}^{A}\mathbf{R}_{B} & [{}^{A}\mathbf{p}_{B}]_{\times} {}^{A}\mathbf{R}_{B} \\ \mathbf{0}_{3\times 3} & {}^{A}\mathbf{R}_{B} \end{bmatrix}. \tag{2.21} $$

Here $[{}^{A}\mathbf{p}_{B}]_{\times}$ is the skew-symmetric matrix of the translation vector. For a vector $\mathbf{p} = [p_x, p_y, p_z]^T$, the skew-symmetric matrix is

$$ [\mathbf{p}]_{\times} = \begin{bmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{bmatrix}. \tag{2.22} $$

The adjoint transformation is useful in whole-body control because task-space velocities, Jacobians, and pose errors may need to be expressed in a consistent frame before being used in the controller.

Pose errors are also computed using coordinate transformations. Let ${}^{w}\mathbf{T}_{e}$ be the current end-effector pose and ${}^{w}\mathbf{T}_{d}$ be the desired end-effector pose, both expressed in the world or odometry frame. The relative transformation from the current pose to the desired pose can be written as

$$ {}^{e}\mathbf{T}_{d} = ({}^{w}\mathbf{T}_{e})^{-1} {}^{w}\mathbf{T}_{d}. \tag{2.23} $$

This relative transformation contains both translational and rotational errors. The translational error can be obtained from the position difference,

$$ \mathbf{e}_{p} = {}^{w}\mathbf{p}_{d} - {}^{w}\mathbf{p}_{e}, \tag{2.24} $$

while the rotational error can be described by the relative rotation matrix

$$ \mathbf{R}_{err} = ({}^{w}\mathbf{R}_{e})^{T} {}^{w}\mathbf{R}_{d}. \tag{2.25} $$

In a velocity-based controller, these errors can be converted into a desired end-effector spatial velocity, which is then mapped to joint and base velocities through the robot Jacobian.

In robotics, transformations are often organized as a frame tree. Each edge in the tree represents the pose of one frame relative to another. This structure makes it possible to query the transformation between any two connected frames by composing the transformations along the path between them. In a mobile manipulation system, such transformations are important because perception, planning, and control may each use different reference frames.

In the Robot Operating System (ROS), the TF framework is used to maintain and query transformations between coordinate frames. Static transformations can be used for fixed relationships, such as the transform between the robot base and a camera mounted on the robot. Dynamic transformations are used for time-varying relationships, such as the pose of the mobile base in the odometry frame or the pose of the end-effector as the manipulator moves. By using TF, a target pose detected by the perception module can be transformed into the frame required by the controller.

For the system in this thesis, coordinate transformations are required in several places. The camera provides visual and depth measurements in the camera optical frame. The target or grasp pose estimated from perception must be transformed into the odometry or base frame before being used as a control target. The whole-body controller then compares the desired target pose with the current end-effector pose and computes velocity commands for the mobile base and arm. Therefore, a consistent frame representation is essential for connecting perception, control, and robot execution.

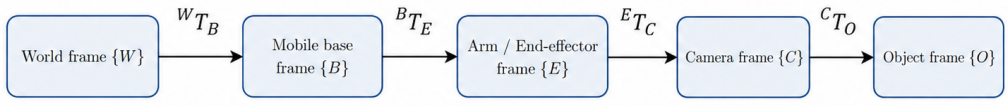


Figure 2.1: Example of a coordinate frame tree in a mobile manipulation system. Transformations between connected frames can be composed to express poses in a common reference frame.

2.3 Robot Kinematics

Robot kinematics describes the relationship between the configuration of a robot and the pose or motion of its end-effector, without considering the forces and torques that cause the motion. In manipulation tasks, kinematics is fundamental because the robot must determine how changes in joint positions or base motion affect the position and orientation of the end-effector. For a mobile manipulator, this relationship includes both the robotic arm and the mobile base.

For a robotic manipulator, the joint configuration can be represented by a vector

$$\mathbf{q}_a = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}^T, \tag{2.26}$$

where q_i denotes the position of the i-th joint and n is the number of joints of the manipulator. In this thesis, the manipulator is a UR10 arm with six revolute joints, and therefore $n = 6$. The corresponding joint velocity vector is

$$\dot{\mathbf{q}}_a = \begin{bmatrix} \dot{q}_1 & \dot{q}_2 & \cdots & \dot{q}_n \end{bmatrix}^T. \tag{2.27}$$

The pose of the end-effector is determined by the forward kinematics of the robot. In general form, this can be written as

$$\mathbf{x}_e = f(\mathbf{q}_a), \tag{2.28}$$

where $\mathbf{x}_e$ represents the end-effector pose and $f(\cdot)$ is the nonlinear mapping from joint space to task space. The end-effector pose can be represented by a position and orientation,

$$\mathbf{x}_e = \begin{bmatrix} \mathbf{p}_e \\ \boldsymbol{\phi}_e \end{bmatrix}, \tag{2.29}$$

where $\mathbf{p}_e \in \mathbb{R}^3$ is the end-effector position and $\boldsymbol{\phi}_e$ represents the orientation using a suitable parameterization, such as Euler angles, angle-axis representation, or a quaternion.

The same pose can also be represented as a homogeneous transformation matrix. For a serial manipulator, the transformation from the arm base frame $\{B\}$ to the end-effector frame $\{E\}$ is obtained by composing the transformations between consecutive links:

$${}^{B}\mathbf{T}_{E}(\mathbf{q}_a) = {}^{B}\mathbf{T}_{1}(q_1){}^{1}\mathbf{T}_{2}(q_2)\cdots{}^{n-1}\mathbf{T}_{E}(q_n), \tag{2.30}$$

where each transformation describes the relative pose between two frames in the kinematic chain. This equation expresses the forward kinematics of the manipulator in rigid-body form.

For a mobile manipulator, the end-effector pose also depends on the pose of the mobile base. The planar pose of the base in the world or odometry frame can be written as

$$\mathbf{p}_b = \begin{bmatrix} x_b & y_b & \theta_b \end{bmatrix}^T, \tag{2.31}$$

where x_b and y_b describe the position of the base, and θ_b is the heading angle. Although the planar pose contains three variables, the mobile base is treated with two controllable velocity inputs:

$$\mathbf{u}_b = \begin{bmatrix} v_b & \omega_b \end{bmatrix}^T, \tag{2.32}$$

where v_b is the forward linear velocity and ω_b is the angular velocity around the vertical axis. The lateral velocity of the base is not directly controllable.

The kinematic model of such a planar base can be written as

$$\begin{bmatrix} \dot{x}_b \\ \dot{y}_b \\ \dot{\theta}_b \end{bmatrix} = \begin{bmatrix} \cos\theta_b & 0 \\ \sin\theta_b & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_b \\ \omega_b \end{bmatrix}. \tag{2.33}$$

This model shows that the base can move forward along its heading direction and rotate around its vertical axis, but it cannot directly move sideways. Therefore, the base contributes two controllable velocity degrees of freedom to the whole-body

controller.

The complete end-effector pose of the mobile manipulator can be expressed by combining the base pose and the arm forward kinematics:

$$^{W}\mathbf{T}_{E}(\mathbf{p}_b, \mathbf{q}_a) = {}^{W}\mathbf{T}_{B}(\mathbf{p}_b){}^{B}\mathbf{T}_{E}(\mathbf{q}_a), \tag{2.34}$$

where $\{W\}$ denotes the world or odometry frame, $\{B\}$ denotes the mobile base or arm base frame, and $\{E\}$ denotes the end-effector frame. This equation shows that the end-effector pose is affected by both the motion of the mobile base and the motion of the robotic arm.

For velocity-level control, differential kinematics relates the generalized velocity of the robot to the end-effector spatial velocity. For the manipulator alone, the differential kinematic relationship is

$$\mathbf{v}_e = \mathbf{J}_a(\mathbf{q}_a)\dot{\mathbf{q}}_a, \tag{2.35}$$

where $\mathbf{v}_e \in \mathbb{R}^6$ is the end-effector spatial velocity, $\dot{\mathbf{q}}_a$ is the joint velocity vector, and $\mathbf{J}_a(\mathbf{q}_a)$ is the manipulator Jacobian. The spatial velocity contains both linear and angular velocity components:

$$\mathbf{v}_e = \begin{bmatrix} \boldsymbol{\nu}_e \\ \boldsymbol{\omega}_e \end{bmatrix}, \tag{2.36}$$

where $\boldsymbol{\nu}_e \in \mathbb{R}^3$ is the linear velocity and $\boldsymbol{\omega}_e \in \mathbb{R}^3$ is the angular velocity.

In this thesis, the base is represented in the controller by two virtual velocity variables corresponding to forward translation and yaw rotation. These virtual base velocities are combined with the UR10 joint velocities in the whole-body velocity vector. This allows the controller to compute coordinated commands for the mobile base and arm in a single velocity-level formulation.

The Jacobian matrix is central to many robot control methods because it allows motion commands to be formulated in task space and then mapped to joint or base velocities. If a desired end-effector spatial velocity $\mathbf{v}_d$ is given, the velocity-level control problem can be written as

$$\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \approx \mathbf{v}_d. \tag{2.37}$$

When the robot has more controllable degrees of freedom than the dimension of the task, the system is redundant. In such cases, there may be infinitely many velocity vectors $\dot{\mathbf{q}}$ that produce the same end-effector velocity. This redundancy can be used to optimize secondary objectives, such as avoiding joint limits, improving manipulability, or maintaining a favorable base position.

One common way to solve the velocity level kinematic problem is to use the pseudo-inverse of the Jacobian:

$$\dot{\mathbf{q}} = \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{v}_d, \tag{2.38}$$

where $\mathbf{J}^{\dagger}$ denotes the pseudo-inverse. For redundant robots, a null space term can be added:

$$\dot{\mathbf{q}} = \mathbf{J}^{\dagger}\mathbf{v}_d + \left(\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J}\right)\dot{\mathbf{q}}_0, \tag{2.39}$$

where $\dot{\mathbf{q}}_0$ is a secondary velocity command projected into the null space of the task. This means that the secondary motion does not affect the primary end-effector task. Although this formulation is useful theoretically, practical mobile manipulation often requires inequality constraints, velocity limits, and task weighting. Therefore, this thesis uses a quadratic programming formulation for whole-body control, which is introduced in later sections.

When the Jacobian loses rank, the robot may enter a singular configuration. Near singularities, certain end-effector motions become difficult or impossible to generate, and very large joint velocities may be required to produce small end-effector motions. Manipulability is a commonly used measure for evaluating how easily a robot can move its end-effector in different directions. One common manipulability measure is

$$w(\mathbf{q}_a) = \sqrt{\det\left(\mathbf{J}_a(\mathbf{q}_a)\mathbf{J}_a^T(\mathbf{q}_a)\right)}, \tag{2.40}$$

where a larger value generally indicates that the manipulator is in a configuration with better ability to generate end-effector motion in multiple directions. A low value may indicate that the robot is close to a singular configuration.

For a redundant system, manipulability can also be treated as a secondary objective. For example, the controller may prefer motions that increase the manipulability measure or avoid configurations where the arm becomes close to singular. This is relevant for mobile manipulation because the base can help reposition the robot so that the arm remains in a more favorable configuration during tracking.

Robot kinematics provides the mathematical relationship between base motion, joint motion, and end-effector motion. For a mobile manipulator, the end-effector pose is determined by both the mobile base and the robotic arm. The velocity-level relationship is described by the whole-body Jacobian, which provides the foundation for coordinated base-arm control. These concepts are used later in the whole-body control and quadratic programming sections.

2.4 Visual Perception

This section presents the visual perception pipeline that underpins the manipulation system. The pipeline consists of three stages: (1) depth estimation via the Femto Bolt camera, which recovers the 3D geometry of the scene using indirect Time-of-Flight sensing. (2) object segmentation using SAM 2, which isolates the

task-relevant object from the raw point cloud. (3) surface normal estimation on the segmented point cloud, which provides the orientation information required for downstream grasp pose planning. The remainder of this section introduces each stage in turn.

2.4.1 Depth Sensing with the Femto Bolt Camera and Point Cloud Representation

To obtain dense depth information of the workspace, this work uses an Orbbec Femto Bolt camera, an indirect Time-of-Flight (iToF) depth camera, whose underlying sensing principle is briefly reviewed below.

An iToF camera estimates depth by actively illuminating the scene rather than by triangulating between two viewpoints. The camera emits infrared light that is modulated at a known frequency f, and each pixel on the sensor measures the phase shift $\Delta\phi$ between the emitted and the reflected signal. Since this phase shift accumulates with the time the light takes to travel to the scene and back, the time-of-flight t for a given pixel can be recovered as Equation (2.41):

$$t = \frac{\Delta\phi}{2\pi f} \tag{2.41}$$

The corresponding depth z, i.e., the distance from the camera to the scene point, follows directly from the time-of-flight and the speed of light c, as shown in Equation (2.42):

$$z = \frac{c \cdot t}{2} \tag{2.42}$$

where the factor of 2 accounts for the round trip of the light. Repeating this measurement in parallel across all pixels yields a per-pixel depth map of the scene for each frame.

Unlike the binocular stereo matching used in some depth sensors, this active sensing approach does not depend on scene texture, which makes it more robust in the textureless or low-contrast regions often present in a manipulation workspace, at the cost of being more sensitive to strong ambient infrared light and to multi-path interference from highly reflective surfaces.

The resulting depth map is then combined with the camera's intrinsic parameters to recover the 3D structure of the scene. The camera intrinsic matrix K is given by Equation (2.43):

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.43}$$

Using K, every pixel $[u, v]^T$ with depth z can be back-projected into a 3D point $[X, Y, Z]^T$ in the camera coordinate system, as shown in Equation (2.44):

$$z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.44}$$

Applying this back-projection to every pixel of a depth frame yields the raw point cloud of the scene. This raw point cloud, however, contains the entire workspace rather than only the object to be manipulated. The next subsection introduces how the task-relevant object is isolated from this point cloud through visual segmentation.

2.4.2 Segment Anything Model 2 (SAM 2)

Before a reliable grasp can be planned, the target object must first be separated from the background and other clutter in the raw point cloud produced by the binocular camera. This work adopts SAM 2 for this purpose, as it supports promptable segmentation directly on video streams, allowing the target object to be consistently tracked and re-segmented across consecutive frames rather than being re-identified from scratch at every frame.

SAM 2 is a universal promptable visual segmentation framework that successfully generalizes the spatial segmentation capabilities of fundamental vision models on static images to the spatio-temporal domain of continuous videos. To overcome the bottleneck of traditional Video Object Segmentation (VOS) algorithms in balancing flexible interaction and real-time processing of long sequences, SAM 2 proposes a Transformer [30] architecture with a Streaming Memory Mechanism to tackle the Promptable Visual Segmentation (PVS) task. Its core paradigm is that, when processing the current video frame, the model relies not only on the current frame's visual features and real-time interactive prompts but is also deeply conditioned on the memory containing the target's historical states. The overall network architecture consists of four key components: an image encoder, a memory bank, a memory attention module, and a prompt and mask decoder [31], summarized briefly below.

Streaming Image Encoder. Rather than processing an entire video clip as a 3D tensor, SAM 2 consumes video frame by frame. It uses the Hiera architecture [32], pre-trained with Masked Autoencoders [33], as its backbone. Unlike a standard Vision Transformer (ViT) [34], Hiera produces multi-scale feature maps in a single forward pass, with low-resolution features feeding the memory attention module and high-resolution features passed directly to the decoder via skip connections to preserve segmentation edge quality.

Spatio-Temporal Memory Bank. To track an object across frames, SAM 2 maintains a memory bank of two FIFO queues: one holding the N most recent frames, and one holding up to M frames with explicit user prompts. Each memory entry combines a 2D spatial feature map (preserving shape and location) with a compact 1D object pointer (encoding high-level semantic identity). Temporal position encoding is applied only to the recent-frame queue, since prompted frames may be temporally sparse or even come from the future relative to the current frame.

Memory Attention. A stack of L Transformer blocks first computes self-attention within the current frame, then cross-attention between the current frame and the memory bank (using 2D Rotary Positional Embedding [35] for spatial alignment). This step lets the model match the same physical object across frames despite changes in appearance or motion.

Mask Decoder. Building on the two-way cross-attention decoder of the original

SAM [36], SAM 2 outputs multiple candidate masks to resolve prompt ambiguity, automatically propagating the highest-IoU candidate when no further prompt is given. It also includes a dedicated occlusion-prediction head, allowing the model to explicitly signal when the tracked object leaves the frame or becomes fully occluded — a scenario not handled by static-image segmentation models. After decoding, the predicted mask is fused back into the frame's features to generate a new memory entry for subsequent frames.
Applying SAM 2 to each stereo frame yields a binary mask that isolates the target object. Projecting this mask back through the depth map obtained in the previous subsection produces a segmented point cloud containing only the object of interest. Accurately planning a grasp on this object, however, requires knowing not just its 3D position but also the local surface orientation at each point, which motivates the normal vector estimation described next.

2.4.3 Calculating Point Cloud Normal Vectors

Surface normals describe the local orientation of a point cloud and are a prerequisite for downstream grasp planning, since a feasible grasp pose must be aligned with the object's surface geometry rather than its raw point positions. Normal vector estimation is also widely used in other point cloud algorithms, including feature extraction, segmentation, and denoising.
Normal vector estimation generally proceeds in three steps: neighborhood selection, local surface fitting, and a global consistency check, described below.
Neighborhood Selection. Starting from the point p to be estimated, a method (such as k-nearest neighbors search or radius search) is used to select this point and its neighboring points to form a point set, denoted as P.
Surface Approximation and Normal Calculation. The points in P are used to fit a local plane via least squares, as shown in Eq. (2.45) and Eq. (2.46):

$$Ax + By + Cz + D = 0 \tag{2.45}$$

$$A^2 + B^2 + C^2 = 1 \tag{2.46}$$

where A, B, C, D are the plane parameters to be estimated, and Eq. (2.46) normalizes the normal vector (A, B, C).
The parameters are solved by minimizing the sum of squared point-to-plane distances over all n points in P, subject to the normalization constraint. This standard least-squares fitting problem reduces, after eliminating D via the centroid of the neighborhood and applying the method of Lagrange multipliers, to the eigenvalue problem shown in Eq. (2.47):

$$M \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \lambda \begin{bmatrix} A \\ B \\ C \end{bmatrix}, \quad M = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix} \tag{2.47}$$

where M is the covariance matrix of the neighboring points (e.g., $cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$), and λ denotes an eigenvalue of M. The eigenvector corresponding to the minimum eigenvalue of M is the estimated normal vector at point p.

Consistency Check. Since the plane-fitting step above only determines the normal direction up to a sign, the normals computed independently at each point are not guaranteed to point consistently toward the same side of the surface (e.g., all outward or all inward). A consistency check is therefore applied afterward to re-orient normals so that they agree across the point cloud.
With the segmented object point cloud and its associated surface normals available, the perception pipeline provides the geometric information — object position, extent, and local surface orientation — required by the grasp planning module described in the following section.

2.4.4 Point Cloud Filtering

The raw point cloud produced by the sensing pipeline is dense and therefore costly to process directly; to speed up the subsequent segmentation and normal estimation steps, this work applies voxel filtering to downsample the point cloud before further processing.
Voxel filtering divides the continuous three-dimensional space into a regular grid of cubes, called voxels, and replaces all points falling inside the same voxel with a single representative point, typically their centroid. Given a voxel containing the points $\{p_1, p_2, \ldots, p_n\}$ with coordinates (x_i, y_i, z_i), the representative point $p = (x, y, z)$ is computed as the mean position of the points within the voxel, as shown in Eq. (2.48)–(2.50):

$$x = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.48}$$

$$y = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{2.49}$$

$$z = \frac{1}{n} \sum_{i=1}^{n} z_i \tag{2.50}$$

By reducing the number of points while preserving the overall geometric structure of the point cloud, voxel filtering lowers the computational cost of the segmentation and normal estimation steps described in the following subsections.
Visual perception plays an important role in robotic manipulation because the robot needs information about the target object and the surrounding environment. For grasping or tracking tasks, the robot must estimate where the target object is located and how it is oriented in space. In mobile manipulation, visual perception is especially important because both the robot and the object may change their positions over time.

2.4.5 Point Cloud Registration

Point cloud registration refers to the process of aligning two point clouds that are expressed in different coordinate frames, for example due to different viewpoints or different acquisition times, into a common reference frame. This is a necessary step whenever information from multiple observations needs to be combined, such as when the segmented object point cloud must be aligned with a reference

model to estimate its 6D pose. Registration is typically carried out in two stages: a coarse (global) registration step that finds an initial alignment without requiring prior knowledge of the relative pose, followed by a fine registration step that refines this estimate. In this work, coarse registration is performed by matching Fast Point Feature Histogram (FPFH) descriptors between the two point clouds and estimating the initial alignment with Random Sample Consensus(RANSAC), due to its computational efficiency and robustness to noise and variations in point density. This initial alignment is then refined using Colored Iterative Closest Point(ICP), which jointly exploits geometric and color information to achieve a more accurate fine registration than geometry-only ICP variants.

2.4.5.1 Fast Point Feature Histogram (FPFH)

The extraction of the FPFH is a deterministic process that densely traverses the point cloud, aiming to generate a 33-dimensional geometric descriptor with rotation and translation invariance for each point. Combining the classical theoretical framework with the underlying engineering optimization of Open3D, its complete pipeline can be summarized into the following four steps [37]:

Constructing the Local Reference Frame

For every query point p_q (with normal vector n_q) in the point cloud, the algorithm utilizes a KD-tree to search for all its neighboring points p_k (with normal vectors n_k) within a given neighborhood radius. For each pair (p_q, p_k), the displacement vector $\Delta p = p_k - p_q$ is defined, and a local 3D orthogonal coordinate system (u, v, w) is constructed:

$$u = n_q \tag{2.51}$$

$$v = \frac{\Delta p \times u}{\|\Delta p \times u\|} \tag{2.52}$$

$$w = u \times v \tag{2.53}$$

Extracting Relative Pose Features

Based on the aforementioned local coordinate system, three independent angular features (α, ϕ, θ) are calculated, which can completely describe the relative spatial pose of the two points:

$$\alpha = \text{atan2}(w \cdot n_k, u \cdot n_k) \tag{2.54}$$

$$\phi = v \cdot n_k \tag{2.55}$$

$$\theta = u \cdot \frac{\Delta p}{\|\Delta p\|} \tag{2.56}$$

Generating the Simplified Point Feature Histogram (SPFH)

The (α, ϕ, θ) values calculated between the query point p_q and all its neighbors are mapped into a discrete histogram. Each angular dimension is divided into 11 bins. These are concatenated to form a 33-dimensional (11×3) feature vector. To eliminate the impact caused by differences in the number of neighbors (point density), the frequencies falling into the bins are normalized so that the sum of this 33-dimensional vector defaults to 100.

Distance Weighting and Block-wise Independent Normalization
The SPFH only contains the relationship between the query point and its direct neighbors. To expand the receptive field of the feature, the algorithm needs to weight and integrate the SPFHs of the neighborhood onto the query point. The final formula adopted by Open3D is:

$$FPFH(p_q) = SPFH(p_q) + \mathcal{N}\left(\sum_{k=1}^{K} \frac{1}{d_k^2} \cdot SPFH(p_k)\right) \tag{2.57}$$

where d_k^2 is the squared Euclidean distance between the two points. $\mathcal{N}(\cdot)$ represents the block-wise independent normalization operation, and its specific execution logic is as follows:

Subspace Partitioning: The weighted sum of the 33-dimensional neighborhood features is strictly partitioned into three 11-dimensional subspaces (corresponding to α, ϕ, and θ, respectively).

Independent Normalization: The sum of elements in each of these three subspaces is calculated separately, and the sum of each subspace is forcibly and independently scaled to 100.

Final Superposition: The normalized neighborhood features are added to the query point's own $SPFH(p_q)$.

2.4.5.2 Random Sample Consensus(RANSAC)

RANSAC is a robust model estimation method that can be used when the data contains outliers. The main idea is to repeatedly sample a small subset of data, estimate a model, and then evaluate how many data points agree with the model. This makes RANSAC useful for estimating geometric primitives or initial transformations in noisy point clouds [38].
After extracting the FPFH features, the algorithm proceeds through the following steps to estimate the initial global transformation matrix between the source and target point clouds:

1. **Candidate Matching Initialization:** Based on the extracted FPFH descriptors, the algorithm searches for the nearest neighbor in feature space for each point in the source point cloud to establish a candidate matching set. A Mutual Filter is applied to retain only bidirectional matches, effectively eliminating a significant number of unidirectional false matches.

2. **Random Sampling:** Randomly select n pairs of matched points from the candidate matching set (i.e., n points in the source point cloud and their corresponding n points in the target point cloud).

3. **Early Rejection (Geometric Consistency Check):** Before executing computationally expensive matrix operations, a geometric consistency check is performed on the n sampled point pairs. If there is a significant discrepancy (exceeding a preset tolerance threshold) in the side lengths between the polygon formed by the n points in the source point cloud and the polygon

formed by the corresponding n points in the target point cloud, the sample is discarded, and the algorithm returns to the previous step to resample.

4. **Model Estimation:** For the n pairs of sampled points that pass the consistency check, Singular Value Decomposition (SVD) is utilized to compute the 3D rigid body transformation matrix (including the rotation matrix R and the translation vector t).

5. **Model Evaluation:** Apply the computed transformation matrix to the entire source point cloud. Calculate the spatial Euclidean distance between the points in the transformed source point cloud and their corresponding points in the target point cloud. Matching pairs with distances less than a preset distance threshold are defined as inliers.

6. **Iteration and Termination:** Compare the number of inliers of the current transformation model with the historical maximum. If the current model has a higher number of inliers, update the optimal transformation matrix. Repeat the aforementioned steps until the preset maximum number of iterations is reached or the convergence condition is satisfied, ultimately outputting the optimal global transformation model.

2.4.5.3 Colored Point Cloud Registration

In the fine registration stage of point clouds, the traditional Point-to-Plane Iterative Closest Point (ICP) algorithm [39] relies solely on spatial geometric distances for pose constraints. When facing scenarios such as smooth planes that lack significant geometric variations, pure geometric constraints are highly susceptible to Sliding Degeneracy, which causes the optimization algorithm to converge to local optima. In contrast, the Colored ICP algorithm fuses the color texture features of the point cloud with geometric features to construct a joint objective function.
Let $\mathbf{p}$ be a point in the source point cloud, $\mathbf{q}$ be its corresponding point in the target point cloud, and $\mathbf{n_q}$ be the surface normal vector at the target point. The rigid transformation matrix to be solved is $\mathbf{T}$. The joint error function [40] $E(\mathbf{T})$ for Colored ICP is defined as the weighted sum of the geometric error $E_G(\mathbf{T})$ and the photometric error $E_C(\mathbf{T})$:

$$E(\mathbf{T}) = (1-\lambda)E_C(\mathbf{T}) + \lambda E_G(\mathbf{T}) \tag{2.58}$$

where $\lambda \in [0,1]$ is a weight parameter used to balance the contribution ratio between geometric and color constraints.
The geometric error term $E_G(\mathbf{T})$ follows the point-to-plane distance penalty model. By minimizing the orthogonal projection distance from the transformed source point to the tangent plane containing the target point, an initial alignment of the spatial structure is achieved:

$$E_G(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \left((\mathbf{Tp}-\mathbf{q})\cdot\mathbf{n_q}\right)^2 \tag{2.59}$$

where $\mathcal{K}$ is the set of effective corresponding point pairs established based on Euclidean distance in the current iteration step.

The photometric error term $E_C(\mathbf{T})$ is used to measure the surface texture discrepancy between point pairs after spatial alignment. Because point cloud data is discretely distributed in space, directly calculating the brightness difference between the source point and the target point cannot provide a continuous optimization gradient. Therefore, the algorithm introduces the continuous color space gradient $\nabla C_{\mathbf{q}}$ of the target point cloud at $\mathbf{q}$ and employs a first-order Taylor expansion to linearize and approximate the color error:

$$E_C(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \left(C_p(\mathbf{p}) - C_q(\mathbf{q}) - \nabla C_{\mathbf{q}} \cdot (\mathbf{T}\mathbf{p} - \mathbf{q})\right)^2 \tag{2.60}$$

where $C_p(\mathbf{p})$ and $C_q(\mathbf{q})$ represent the brightness values of the source and target points after conversion to a single channel, respectively. The gradient inner product term $\nabla C_{\mathbf{q}} \cdot (\mathbf{T}\mathbf{p} - \mathbf{q})$ carries significant physical constraint meaning: when the point cloud slides along a smooth surface (i.e., the geometric error approaches zero), this term produces a sharp numerical change whenever there is a color gradient variation in the direction of sliding. This provides a globally unique minimum anchor point for the optimization function, effectively overcoming the phenomenon of sliding degeneracy.
For the joint objective function, the algorithm parameterizes the continuous rigid transformation matrix $\mathbf{T}$ into a 6-degree-of-freedom vector (containing 3 rotation parameters and 3 translation parameters). By computing the Jacobian matrix of the error function with respect to the pose parameters, a non-linear least squares problem is constructed and solved iteratively using the Gauss-Newton Method until the error criteria are met.

2.5 Whole-Body Control of Mobile Manipulators

Whole-body control coordinates the mobile base and the robotic arm within a unified control problem [9, 10]. Based on the whole-body differential kinematics introduced previously, both subsystems contribute to the motion of the end-effector. The control objective is therefore to determine a generalized robot velocity that produces the desired end-effector motion while satisfying the kinematic and physical constraints of the system.

For reactive pose tracking, the desired end-effector spatial velocity is generated from the error between the current end-effector pose and the target pose. In simplified form, this relationship can be expressed as

$$\mathbf{v}_e^* = \lambda \mathbf{e}, \tag{2.61}$$

where $\mathbf{e}$ contains the translational and rotational pose errors, and $\lambda > 0$ is a control gain. The corresponding whole-body control objective is

$$\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \approx \mathbf{v}_e^*, \tag{2.62}$$

where $\mathbf{J}(\mathbf{q})$ is the whole-body Jacobian and $\dot{\mathbf{q}}$ contains the controllable base and arm velocities.

Because the mobile manipulator is kinematically redundant, different combinations of base and arm velocities may achieve the same end-effector motion. This redundancy allows the controller to distribute motion between the two subsystems while considering additional objectives and constraints, such as velocity limits, joint limits, and favorable arm configurations. In this thesis, this velocity allocation is formulated as a quadratic programming problem, as described in the following section.

2.6 Quadratic Programming for Reactive Control

Quadratic programming is an optimization method that minimizes a quadratic objective function subject to linear equality and inequality constraints. It is widely used in robot control because many control objectives and physical limitations can be expressed in a structured optimization form. For whole-body control, quadratic programming provides a convenient way to compute robot velocities while considering task-space tracking, joint limits, velocity limits, and other constraints.

A general quadratic programming problem can be written as

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq}, \\ & \mathbf{A}_{ineq}\mathbf{x} \leq \mathbf{b}_{ineq}, \\ & \mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max}. \end{aligned} \tag{2.63}$$

In this formulation, $\mathbf{x}$ is the decision variable, $\mathbf{Q}$ is the quadratic cost matrix, and $\mathbf{c}$ is the linear cost vector. The equality constraints describe conditions that should be satisfied exactly, while the inequality constraints and bounds describe limitations on the solution.

In the context of velocity-level robot control, the decision variable often represents the robot velocity. For a whole-body controller, this can include both mobile base velocity and manipulator joint velocity. In a general form, the decision variable can be written as

$$\mathbf{x} = \dot{\mathbf{q}}, \tag{2.64}$$

where $\dot{\mathbf{q}}$ is the generalized velocity of the robot.

The main task in end-effector control is usually expressed through the differential kinematics relationship

$$\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{v}_e^*, \tag{2.65}$$

where $\mathbf{J}(\mathbf{q})$ is the Jacobian matrix and $\mathbf{v}_e^*$ is the desired end-effector spatial velocity. In a QP formulation, this relationship can be included as an equality constraint.

This allows the optimizer to compute robot velocities that satisfy the desired task-space motion.

However, in practice, the desired end-effector velocity may not always be exactly achievable. This can happen because of joint limits, velocity limits, singular configurations, or conflicts with other constraints. To improve feasibility, slack variables can be introduced. With slack variables, the decision variable becomes

$$\mathbf{x} = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{s} \end{bmatrix}, \tag{2.66}$$

where $\mathbf{s}$ denotes the slack variable. The task constraint can then be written as

$$\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{s} = \mathbf{v}_e^*. \tag{2.67}$$

The slack variable relaxes the strict task constraint, allowing the optimization problem to remain feasible even when the exact task velocity cannot be achieved. To prevent excessive task error, the slack variable is usually penalized in the objective function.

The objective function in a QP-based controller can include different terms. A common objective is to minimize the magnitude of the robot velocity, which avoids unnecessary motion:

$$\min_{\dot{\mathbf{q}}} \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{W}\dot{\mathbf{q}}, \tag{2.68}$$

where $\mathbf{W}$ is a positive semi-definite weighting matrix. The weighting matrix can be used to assign different costs to different degrees of freedom. For example, some motions may be penalized more strongly than others depending on the desired behavior of the robot.

When slack variables are included, the objective can be extended as

$$\min_{\dot{\mathbf{q}},\mathbf{s}} \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{W}_q\dot{\mathbf{q}} + \frac{1}{2}\mathbf{s}^T\mathbf{W}_s\mathbf{s}, \tag{2.69}$$

where $\mathbf{W}_q$ weights the robot velocity and $\mathbf{W}_s$ weights the slack variables. A larger weight on $\mathbf{s}$ encourages the controller to satisfy the task-space tracking objective more accurately.

Inequality constraints are also important in QP-based whole-body control. They can be used to represent physical and safety-related limitations. For example, joint velocity limits can be written as

$$\dot{\mathbf{q}}_{min} \leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}_{max}, \tag{2.70}$$

where $\dot{\mathbf{q}}_{min}$ and $\dot{\mathbf{q}}_{max}$ are the lower and upper velocity limits. Similar constraints can be used to avoid joint limits or to restrict motions that may lead to unsafe configurations.

The advantage of QP-based control is that task objectives and constraints can be handled within the same optimization problem. This is especially useful for mobile manipulators, where the controller must coordinate multiple degrees of freedom while respecting kinematic and safety constraints. Instead of solving the end-effector tracking problem separately from other requirements, a QP formulation allows these requirements to be considered simultaneously.

Table 2.2: Interpretation of common QP elements in robot control.

QP element	**Interpretation in robot control**
Decision variable $\mathbf{x}$	Robot velocity, slack variables, or other control variables
Quadratic cost $\mathbf{Q}$	Penalizes motion effort or task relaxation
Linear cost $\mathbf{c}$	Encourages or discourages specific directions of motion
Equality constraints	Enforce task-space tracking relationships
Inequality constraints	Represent safety, joint-limit, or obstacle-related constraints
Bounds	Limit feasible values of control variables
Slack variables	Relax strict constraints to improve feasibility

In summary, quadratic programming provides a mathematical framework for computing robot motion under multiple objectives and constraints. For whole-body control, QP is particularly useful because it allows task-space tracking, motion regularization, velocity limits, and additional constraints to be considered in a unified way.

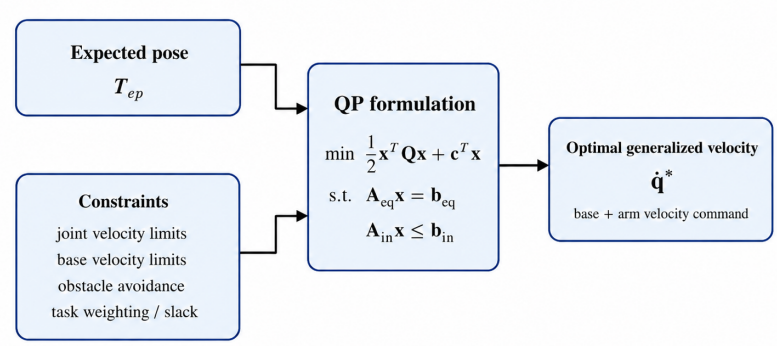


Figure 2.2: Overview of QP-based whole-body control. The expected pose and robot constraints are formulated as an optimization problem, and the solution provides whole-body velocity commands.

2.7 Obstacle Avoidance in Mobile Manipulation

Obstacle avoidance is an essential requirement for robots operating in real environments [41]. A robot must not only achieve its task objective, such as reaching or tracking a target pose, but also avoid collisions with surrounding objects, humans, and the environment. For mobile manipulators, obstacle avoidance is especially important because both the mobile base and the robotic arm may move during the task.

In mobile manipulation, obstacle avoidance can be more challenging than in pure mobile robot navigation or fixed-base manipulation. A mobile robot mainly needs to avoid obstacles while moving its base through the environment, whereas a fixed-base manipulator mainly needs to avoid collisions between its arm and nearby objects. A mobile manipulator combines both challenges. The base may need to reposition itself, while the arm may simultaneously move to achieve a manipulation objective. Therefore, obstacle avoidance must consider the motion of the whole robot.

Obstacle avoidance methods can generally be divided into planning-based methods and reactive methods [42]. Planning-based methods compute a collision-free path or trajectory before execution. These methods are useful when the environment is known or changes slowly. However, when the target or environment changes during execution, a precomputed path may become invalid. Reactive methods, on the other hand, update the robot motion online based on the current obstacle information. This makes them suitable for dynamic tasks where the robot needs to respond continuously.

A common concept in obstacle avoidance is the distance between the robot and the nearest obstacle. Let d denote the distance between a point or part of the robot and an obstacle. To avoid collision, the distance should remain larger than a minimum safety distance $d_{\min}$:

$$d \geq d_{\min}. \tag{2.71}$$

Here, $d_{\min}$ defines the minimum allowed distance between the robot and the obstacle. If d becomes smaller than this threshold, the robot is considered too close to the obstacle.

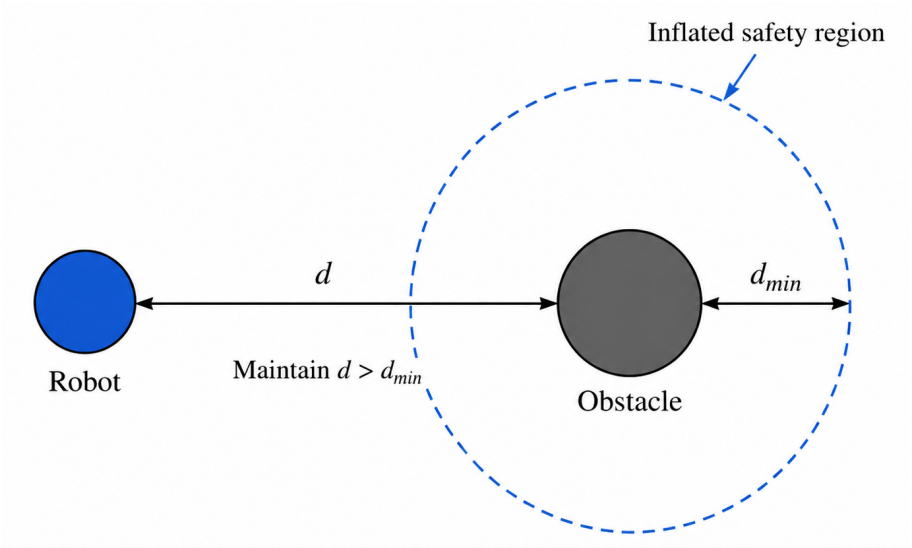


Figure 2.3: Distance-based obstacle avoidance concept. The robot should maintain a distance greater than the minimum safety distance $d_{\min}$ from obstacles.

For velocity-level control, it is often useful to express obstacle avoidance in terms of

the rate of change of the distance. The distance rate $\dot{d}$ describes whether the robot is moving closer to or farther away from the obstacle. A simple safety condition can be written as

$$\dot{d} \geq -\eta(d - d_{\min}), \tag{2.72}$$

where $\eta > 0$ is a gain that determines how strongly the robot reacts when it approaches the safety boundary. This constraint allows the robot to approach an obstacle when it is far away, but restricts motion toward the obstacle when the distance becomes close to $d_{\min}$. In other words, the closer the robot is to the obstacle, the more limited its motion toward the obstacle becomes.

Another common way to represent obstacle information is through an occupancy map or a costmap. An occupancy map describes whether each region of the environment is free, occupied, or unknown. A costmap extends this idea by assigning higher costs to regions near obstacles. The closer a location is to an obstacle, the higher its cost becomes. This allows the robot to prefer safer regions while still moving toward the task objective.

The cost around an obstacle is often inflated to create a safety margin. This inflated region does not necessarily mean that the space is physically occupied, but it indicates that the robot should avoid moving too close to the obstacle. Such representations are commonly used in mobile robot navigation and can also support obstacle-aware mobile manipulation.

In optimization-based control, obstacle avoidance can be incorporated as an inequality constraint. This is useful because the robot can consider the main task objective and safety requirements in the same control problem. For example, the robot may track a desired end-effector velocity while also satisfying a distance-based safety constraint. In this way, obstacle avoidance does not need to be treated as a completely separate module, but can be integrated into the motion generation process.

However, obstacle avoidance can conflict with task execution. For example, the shortest or most direct motion toward a target may bring the robot close to an obstacle. In such cases, the controller must balance task performance and safety. This trade-off is particularly important for mobile manipulators, since the base and the arm may have different possibilities for avoiding obstacles. The base may move around an obstacle, while the arm may adjust its configuration to keep the end-effector close to the target.

For mobile manipulation tasks, obstacle avoidance should ideally preserve the ability of the robot to perform the manipulation objective. Avoiding obstacles by stopping the robot completely may be safe, but it may prevent the robot from completing the task. A more useful strategy is to modify the robot motion so that it remains safe while still moving toward the target. This motivates the use of reactive and constraint-based obstacle avoidance methods in whole-body control.

In summary, obstacle avoidance provides the theoretical basis for generating safe robot motion in environments with obstacles. Distance constraints, costmaps, and inequality constraints are common ways to represent obstacle-related information. For mobile manipulators, obstacle avoidance must be considered together with task execution, since both the mobile base and the robotic arm contribute to the overall motion of the robot.

3
Methods

3.1 Robot Platform and Simulation Environment

The robotic platform used in this thesis is a mobile manipulator consisting of a MiR mobile base and a UR10 robotic arm. The main purpose of this platform is to combine the mobility of an autonomous mobile robot with the manipulation capability of a six-degree-of-freedom industrial manipulator. The mobile base allows the robot to reposition itself in the environment, while the robotic arm enables local reaching and end-effector motion around the target object.

The MiR base provides planar motion on the ground. In the control framework, the base is treated as a non-holonomic mobile platform with two controllable velocity inputs: the forward linear velocity and the yaw angular velocity. This enables the base to move toward or around a target while maintaining a feasible configuration for the arm. Compared with a fixed-base manipulator, the mobile base extends the operational workspace of the whole system and allows the robot to continue tracking even when the target moves beyond the immediate reach of the arm.

The UR10 arm is mounted on the mobile base and provides six revolute joints for manipulation. Its role in the system is to generate accurate local motion of the end-effector. During tracking, the arm is responsible for reducing the local pose error between the current end-effector pose and the expected pose. Since the arm has a limited workspace, coordinated control of the mobile base and the arm is required in order to maintain reachability and tracking performance during dynamic tasks.

An end-effector-mounted camera is used for visual perception. Since the camera is attached near the end-effector, the measured target pose is naturally related to the motion of the manipulator. Through coordinate transformations, the detected target pose can be expressed in the odometry or world frame and then converted into the expected end-effector pose used by the controller.

Figure 3.1 shows a simulation environment which is developed in CoppeliaSim to support the design and validation of the proposed method. The simulation model follows the same overall structure as the real system, including the MiR mobile base, the UR10 arm and the end-effector frame. Task objects and obstacle elements are also included in the scene so that both tracking and obstacle avoidance behaviors could be evaluated.

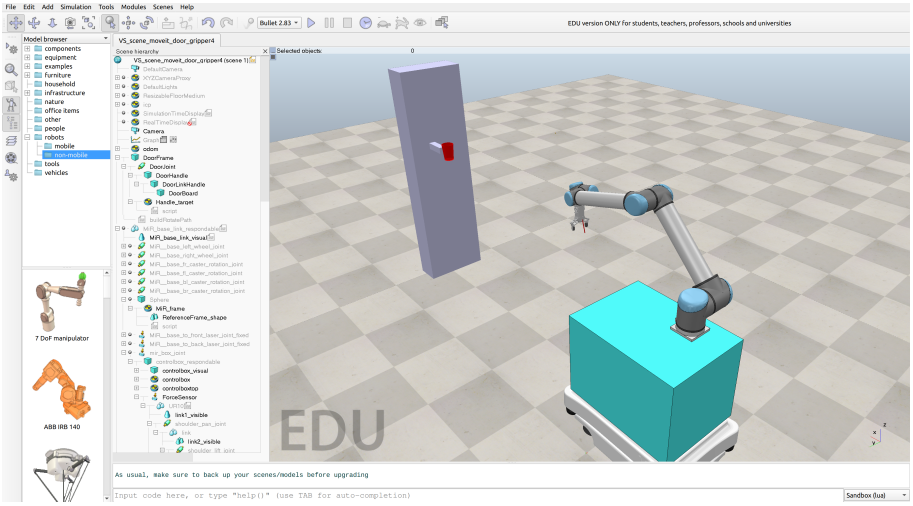


Figure 3.1: Simulation environment used in this thesis.

The first role of the CoppeliaSim environment is to validate the tracking performance of the proposed whole-body controller. Before deployment on a real-world platform, the controller is tested in a simulation environment to verify whether the mobile base and robotic arm could generate coordinated motion while tracking the desired target motion. Specifically, we design three reference trajectories: sideway motion, backward motion, and nonlinear motion, to evaluate the controller's performance under different tracking conditions. These trajectories are designed to provide representative motion patterns for testing the tracking ability of the proposed controller. By evaluating the controller on multiple trajectories rather than on a single motion pattern, the tracking performance can be assessed more comprehensively.

The second role of the CoppeliaSim environment is to evaluate obstacle avoidance. In addition to tracking-only scenarios, obstacle-populated simulation scenes are constructed to test whether the controller could maintain safe motion while preserving task execution. By introducing environmental obstacles into the workspace, it becomes possible to examine how the robot adjusts its motion when obstacle-related constraints are activated.

Therefore, this thesis uses CoppeliaSim for two purposes. First, it provides a controlled environment for validating the tracking performance of the whole-body controller using multiple reference trajectories. Second, it enables safe and repeatable testing of the obstacle avoidance strategy before transferring the method to a physical platform.

The system is developed through a staged validation process. The whole-body controller is first implemented and evaluated in CoppeliaSim under controlled and repeatable conditions. After its tracking behavior is verified in simulation, the control framework is implemented on the physical robot platform, with adjustments for ROS communication, sensor feedback, and hardware interfaces.

3.2 Software Framework and ROS Architecture

The system is implemented using ROS Noetic as its main software framework. ROS is used to connect the simulation environment, map building and localization module, navigation component, perception module, robot state feedback, visualization tools, and whole-body control commands. ROS's modular structure allows different functional components to run as independent nodes and exchange information through topics, while coordinate transformation between the robot coordinate system and the sensor coordinate system is handled through the TF framework.

3.2.1 Object grasping pose generation

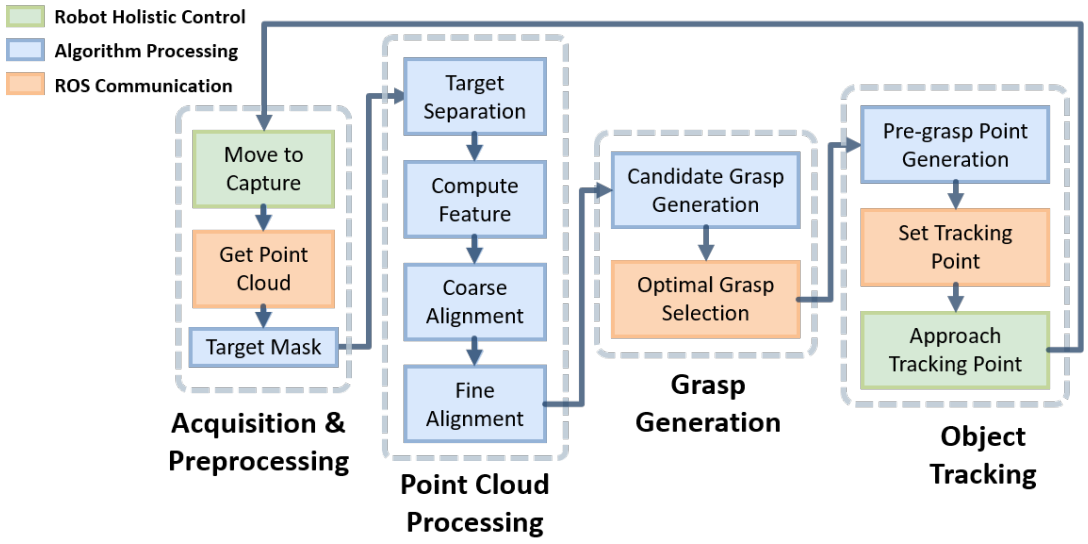


Figure 3.2: Vision-guided tracking, grasping, and control

The overall pipeline for target grasp pose generation is illustrated in Figure 3.2. It primarily consists of four core modules: Image Acquisition and Segmentation, Point Cloud Extraction and Preprocessing, Dual-track Pose Estimation State Machine, and Grasp Strategy and Pose Generation.

3.2.1.1 Image Acquisition and Segmentation

This module is primarily responsible for acquiring and parsing visual information from the scene. The system synchronously captures RGB and depth images containing the target object using an RGB-D camera (Femto Bolt) mounted on a mobile manipulator. Subsequently, the RGB image is transmitted to the underlying SAM 2 (Segment Anything Model 2) server module. Leveraging its robust zero-shot segmentation capability and streaming video memory mechanism, SAM 2 performs high-precision real-time segmentation of the target object. The generated 2D mask results are returned to the ROS node, providing an accurate pixel-level Region of Interest (ROI) for the subsequent 3D reconstruction.

3.2.1.2 Point Cloud Extraction and Preprocessing

This module aims to elevate 2D semantic information into 3D geometric space. First, the mask output by SAM 2 is mapped back to the corresponding depth image to extract the local 3D point cloud of the target object. To eliminate edge noise (such as background flying pixels) and reduce computational complexity, spatial threshold cropping (Z-axis Crop) and morphological filtering are applied. The result is ultimately converted into a standard PointCloud2 format containing RGB color information, which is published via ROS for real-time visual verification.

For the predefined target source point cloud (template), the algorithm executes a dual-scale downsampling preprocessing based on different voxel sizes. It generates a sparse template with FPFH high-dimensional features for RANSAC global coarse matching, and a dense template with consistent normal vectors for Colored ICP local fine matching. This establishes a robust geometric foundation for the subsequent multi-stage pose estimation.

3.2.1.3 Dual-track Pose Estimation State Machine

The core objective of this module is to calculate the 6-DoF pose of the target object in the current camera coordinate system in a real-time and robust manner. To achieve this, the algorithm designs an adaptive "tracking-relocation" dual-track state machine strategy:

- **Relocation Mode:** This mode is triggered at the first frame when the system starts capturing the target, or when tracking is lost (i.e., the registration fitness falls below a preset threshold). The system utilizes FPFH features combined with the RANSAC algorithm featuring a geometric consistency pruning strategy to perform global coarse matching. Subsequently, an ICP with an enlarged search radius is introduced for residual absorption to obtain a reliable initial pose.
- **Tracking Mode:** Starting from the second frame, if the historical pose is highly reliable, the algorithm directly inherits the transformation matrix from the previous frame as the initial estimation and calls the Colored ICP algorithm based on joint geometric-photometric optimization. Colored ICP effectively overcomes the sliding degeneracy problem of pure geometric ICP on flat surfaces, enabling the continuous updating of the target pose at extremely high frame rates (e.g., real-time speed).

Furthermore, considering that RANSAC relocation may cause discontinuous jumps in the pose, this module synchronously broadcasts a `used_ransac` flag containing the pose source status while publishing the current pose. This allows downstream motion planning nodes to adopt corresponding trajectory smoothing strategies.

3.2.1.4 Grasp Strategy and Pose Generation

This module converts the registered 6D object pose into executable grasp commands for the robotic arm's end-effector. First, based on the registration matrix T_{inv}, the object's centroid coordinates and spatial principal axis (typically set as the Z-axis)

direction are extracted. The X, Y, and Z axes of the gripper are rigorously constructed using the orthogonal cross-product algorithm to ensure that the generated grasp rotation matrix possesses absolute mathematical orthogonality.

In the design of the actual grasp strategy, the system generates not only the final Grasp Pose but also a Pre-grasp Pose based on geometric constraints. The core purpose of this design is to guide the end-effector to a preparatory position while ensuring the camera maintains continuous observation of the target object, thereby realizing visual servoing and tracking of dynamic objects. Only after the visual algorithm determines that the target object has entered a stationary state does the system execute the final approach and grasping actions.

3.2.2 Mapping and Localization

Map building and localization are essential for representing a robot pose and its surrounding environment within a consistent reference frame. We first perform map building to obtain a two-dimensional occupancy grid map of the environment. Once the map is generated, localization techniques are used to estimate the pose of the mobile platform on the known map while the robot is in operation.

For map building, the `hector_mapping` package is used to construct a 2D occupancy grid map based on laser scan data. Hector mapping enables the robot to incrementally build a map of its surroundings by matching newly acquired laser scan data with the current map estimate. The resulting map indicates free, occupied, and unknown areas. This map provides the structure required for localization, target navigation, and obstacle avoidance movement of the mobile platform.

Figure 3.3 shows the map generated using `hector_mapping`.

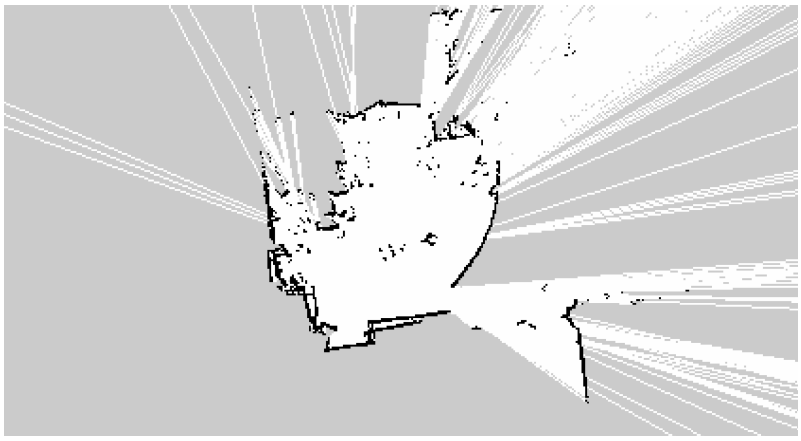


Figure 3.3: Map generated using `hector_mapping`.

After the map is generated, localization is performed using the `amcl` package. AMCL (Adaptive Monte Carlo Localization) estimates the pose of a mobile robot on a known map by combining laser scan observations with odometry information. In this system, AMCL is used to estimate the pose of the MiR mobile platform relative

to the map coordinate system. The estimated base pose provides the transformation relationship between the global map coordinate system and the robot's odometry frame (or base frame), which is essential for map-based navigation and consistent spatial representation.

The maps generated by `hector_mapping` and the localization results obtained from `amcl` form the foundation of mobile platform navigation and obstacle representation in this system. The maps are used to capture the static structure of the environment, while localization provides the robot's current pose during operation.

3.2.3 Navigation and Costmap Representation

The navigation and obstacle mapping components of this system are built on the ROS navigation stack. After generating maps using `hector_mapping` and estimating the robot's pose using `amcl`, the navigation-related ROS packages utilize these maps and localization data to support navigation and obstacle avoidance for the mobile platform. We use the `map_server` package to load and publish the generated map. This map provides a static representation of the environment, including walls and fixed obstacles. Combined with the localization results, this map enables a consistent representation of the robot's pose in the map coordinate system. This provides a spatial foundation for global path planning and environmental representation.

Obstacle information is represented using the `costmap_2d` package. The costmap represents the environment as a grid of traversal costs, where occupied cells correspond to obstacles and inflated regions around obstacles represent safety margins. These inflated regions are important because they prevent the mobile base from planning paths too close to obstacles. In this way, the costmap provides a practical representation of both obstacle locations and safe navigation space.

Two types of costmaps are used in the navigation stack: the global costmap and the local costmap. The global costmap is mainly based on the static occupancy grid map and represents obstacle information in the global map frame. It is used by the global planner to generate a path from the current robot pose to the assigned goal. The local costmap is centered around the robot and is updated during execution using local sensor data. It is used by the local planner to generate short-term motion commands while reacting to nearby obstacles.

Figure 3.4 shows the visualization of the navigation and costmap representation in RViz. The occupancy grid map, robot pose, laser scan data, planned path, and costmap layers can be inspected together. This visualization is useful for verifying that the map, localization result, and obstacle representation are aligned correctly before running mobile manipulation experiments.

The navigation stack is used to verify whether the mobile base can navigate to the target pose specified in RViz while avoiding obstacles in the map environment. This

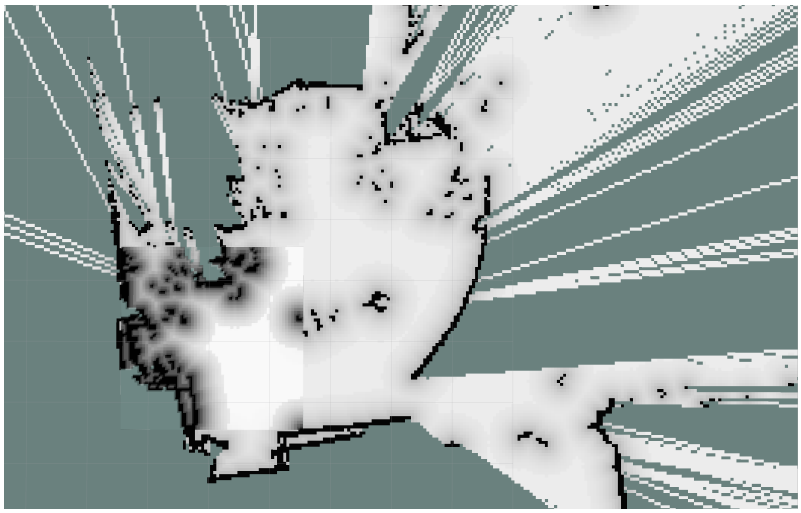


Figure 3.4: Visualization of the costmap in RViz.

functionality provides an important foundation for the mobile operating system, as reliable localization, navigation, and obstacle representation are crucial for the safe operation of the mobile base in structured environments.

While the whole-body controller focuses on coordinating tracking between the base and the robot arm, the navigation and cost map modules provide the environmental representation required by the mobile base. The cost map-based representation also provides useful information for extending the system to obstacle-avoiding whole-body movements. Therefore, this navigation framework can function both as a independent mobile base navigation function, or as a supporting component of the mobile operation architecture.

3.2.4 ROS Communication and Coordinate Transformations

The system's software architecture is based on ROS topics, messages, and coordinate transformations. Different modules, including simulation, mapping, localization, perception, navigation, robot state feedback, and control, communicate via ROS topics. This modular structure allows for the separation of perception, navigation, and control components while maintaining connectivity through a common communication framework.

In the implemented system, the navigation-related modules and the whole-body controller utilize different parts of the ROS framework. The navigation process relies on the map, localization, and cost map modules to support target-based mobile base navigation. In contrast, the whole-body controller primarily relies on robot state feedback, target pose information, and the transformation chain between the coordinate systems of the mobile base, manipulator, end effector, and camera.

The whole-body controller receives robot state and target pose information via ROS topics. The pose of the mobile base is obtained through odometry feedback, while

the joint states of the robotic arm are provided through joint state messages. The target or expected pose is published in the form of pose messages and used by the controller as a tracking reference. After resolving the control problem, the controller issues velocity commands to the mobile base and the robotic arm.

In the implemented system, the base movement commands are represented using the `geometry_msgs/Twist` message. The linear velocity component controls the forward motion of the base, while the angular velocity component controls the yaw rotation. The robotic arm commands are represented as joint velocity commands. Thus, the output of the whole-body controller is decomposed into base movement velocity commands and robotic arm joint velocity commands, both of which are calculated within the same coordinated control framework.

The system uses a variety of standard ROS message types. The `geometry_msgs` package is used to represent pose, transformation and velocity commands. For example, the `PoseStamped` message is used to represent the target pose or the expected end-effector pose, whilst the `Twist` message is used to represent velocity commands for the mobile chassis. The `sensor_msgs` package is used for sensor-related data, including joint states, laser scans, images and point clouds. The `nav_msgs` package is used for map, odometry, path and occupancy grid information. The `std_msgs` package is used for simple commands and trigger signals.

Coordinate transformations are handled by the `tf` and `tf2_ros` packages. These packages maintain the relationships between the map frame, odometry frame, base frame, robot arm frame, end-effector frame, camera frame, and target-relative frame.

For the navigation pipeline, the main transformation chain can be described as

$$\texttt{map} \rightarrow \texttt{odom} \rightarrow \texttt{base_footprint} \rightarrow \texttt{base_link}. \tag{3.1}$$

For the whole-body controller, the transformation chain is mainly related to the mobile base, manipulator, end-effector, camera, and target poses. It can be described as

$$\texttt{odom} \rightarrow \texttt{base_footprint} \rightarrow \texttt{base_link} \rightarrow \texttt{arm links} \rightarrow \texttt{end-effector} \rightarrow \texttt{camera}. \tag{3.2}$$

Figure 3.5 and Figure 3.6 illustrate the main coordinate transformation structure used in the system. To improve readability, the TF tree is presented in two parts. Figure 3.5 shows the global and robot-body transformation chain, including the map, odometry, mobile base, and manipulator-related frames. Figure 3.6 focuses on the end-effector region, including the gripper and camera-related frames. These frames are mainly used for localization, whole-body tracking control, visual perception, and target pose representation.

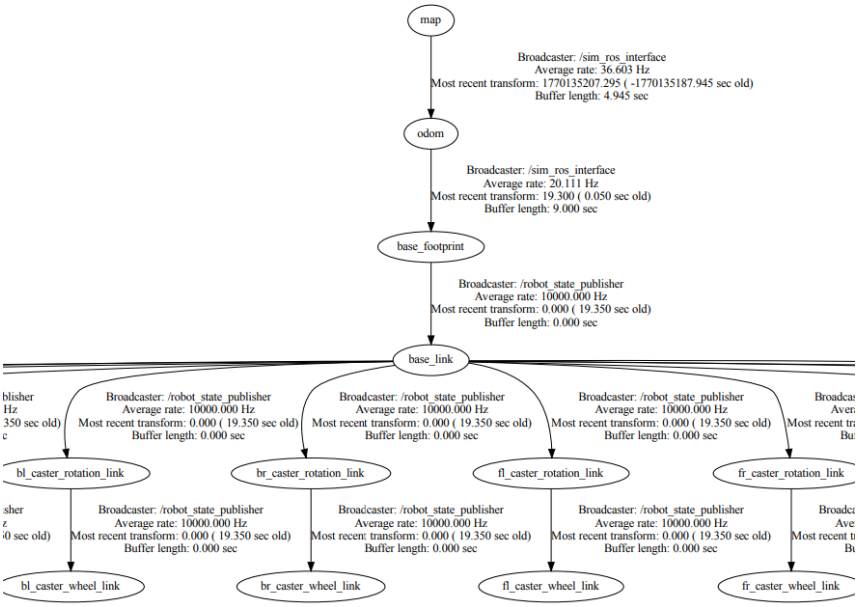


Figure 3.5: Main TF chain from map to manipulator.

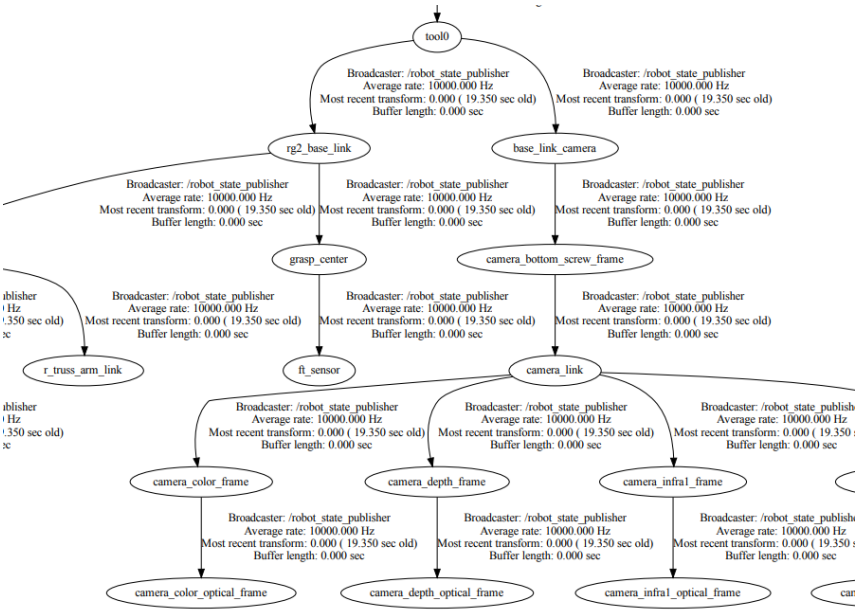


Figure 3.6: Main coordinate transformation structure of the mobile manipulation system.

3.3 Whole-body Controller

The whole-body controller is the core component of the proposed mobile manipulation system. Its purpose is to generate coordinated velocity commands for the MiR mobile base and the UR10 manipulator so that the end-effector can reactively track a target-related grasp pose. Instead of controlling the mobile base and the arm

as two independent subsystems, the controller treats them as a unified kinematic system and computes their velocities simultaneously.

The controller receives the current robot state and the target-related pose as input. The robot state includes the joint positions of the UR10 arm and the coordinate transformations between the odometry frame, mobile base frame, arm frames, end-effector frame, and target-related frames. Based on this information, the current end-effector pose and the desired target-related pose are expressed in a consistent reference frame. A velocity-level control problem is then formulated to reduce the pose error between the current end-effector pose and the desired pose.

3.3.1 Generalized Velocity Representation

The mobile manipulator is represented using a generalized velocity vector that includes both the mobile base and the robotic arm. In this implementation, the optimized velocity vector is written as

$$\dot{\mathbf{q}} = \begin{bmatrix} \omega_b & v_b & \dot{q}_1 & \dot{q}_2 & \dot{q}_3 & \dot{q}_4 & \dot{q}_5 & \dot{q}_6 \end{bmatrix}^T, \tag{3.3}$$

where ω_b is the yaw angular velocity of the mobile base, v_b is the forward linear velocity of the mobile base, and $\dot{q}_1, \ldots, \dot{q}_6$ are the joint velocities of the UR10 manipulator. This representation allows the controller to decide how much of the required end-effector motion should be generated by the base and how much should be generated by the arm.

The controlled end-effector frame is denoted as `grasp_center`. Its current pose is computed from the robot kinematic model as

$$^b\mathbf{T}_e = f(\mathbf{q}), \tag{3.4}$$

where $^b\mathbf{T}_e$ is the current end-effector pose expressed in the mobile base frame, and $\mathbf{q}$ is the current robot configuration.

3.3.2 Pose Servoing

At each control cycle, the controller computes the relative pose error between the current end-effector pose and the desired target-related pose. This error can be written as

$$\mathbf{T}_{err} = \left(^b\mathbf{T}_e\right)^{-1} {}^b\mathbf{T}_{ep}, \tag{3.5}$$

where $^b\mathbf{T}_{ep}$ denotes the expected end-effector pose used as the tracking reference. The translational error is computed from the position component of $\mathbf{T}_{err}$:

$$e_t = \left\|\mathbf{p}_{err}\right\|_2. \tag{3.6}$$

A pose servoing method is then used to convert the pose error into a desired end-effector spatial velocity,

$$\mathbf{v}^* = \text{servo}\left({}^b\mathbf{T}_e, {}^b\mathbf{T}_{ep}\right), \tag{3.7}$$

where $\mathbf{v}^* \in \mathbb{R}^6$ contains both linear and angular velocity components. This desired spatial velocity is used as the main tracking objective in the QP-based whole-body controller.

3.3.3 QP-Based Velocity Optimization

Following the general QP formulation introduced in Section 2.6, the controller solves a velocity-level optimization problem at each control cycle. The implementation uses the decision variable

$$\mathbf{x} = \begin{bmatrix} \dot{\mathbf{q}} \\ \boldsymbol{\delta} \end{bmatrix}, \tag{3.8}$$

where $\dot{\mathbf{q}} \in \mathbb{R}^8$ contains the yaw and forward velocities of the MiR base together with the six UR10 joint velocities, and $\boldsymbol{\delta} \in \mathbb{R}^6$ is the task-space slack variable.
The end-effector tracking task is imposed through

$$\mathbf{J}_e(\mathbf{q})\dot{\mathbf{q}} + \boldsymbol{\delta} = \mathbf{v}^*, \tag{3.9}$$

or equivalently,

$$\mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq}, \qquad \mathbf{A}_{eq} = \begin{bmatrix} \mathbf{J}_e(\mathbf{q}) & \mathbf{I}_6 \end{bmatrix}, \qquad \mathbf{b}_{eq} = \mathbf{v}^*. \tag{3.10}$$

The implemented objective is

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x}, \tag{3.11}$$

where $\mathbf{Q}$ penalizes the generalized robot velocity and task-space slack, while $\mathbf{c}$ represents the secondary objective for maintaining a favorable arm configuration. The optimization is subject to bounds on the base velocities, arm joint velocities, and slack variables:

$$\mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max}. \tag{3.12}$$

During the visual tracking stage, a field-of-view (FoV) constraint is incorporated into the QP in order to maintain continuous observation of the target using the end-effector-mounted camera. Let the target position expressed in the camera frame be

$${}^c\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^T. \tag{3.13}$$

The corresponding normalized image-plane feature is defined as

$$\mathbf{e}_{fov} = \frac{1}{z}\begin{bmatrix} x \\ y \end{bmatrix}. \tag{3.14}$$

The projected feature is required to remain within predefined image bounds,

$$\mathbf{e}_{min} \leq \mathbf{e}_{fov} \leq \mathbf{e}_{max}. \tag{3.15}$$

The dynamics of the image feature are given by

$$\dot{\mathbf{e}}_{fov} = \mathbf{L}_e \mathbf{v}_c, \tag{3.16}$$

where $\mathbf{L}_e$ is the image interaction matrix and $\mathbf{v}_c$ denotes the spatial velocity of the camera. The camera velocity is related to the generalized robot velocity by

$$\mathbf{v}_c = \mathbf{J}_c(\mathbf{q})\dot{\mathbf{q}}, \tag{3.17}$$

where $\mathbf{J}_c(\mathbf{q})$ is the camera Jacobian.

Using this relationship, the FoV requirement is incorporated into the QP as an additional inequality constraint,

$$\underbrace{\begin{bmatrix}\mathbf{L}_e\mathbf{J}_c(\mathbf{q}) & \mathbf{0}\end{bmatrix}}_{\mathbf{A}_{\text{FoV}}} \mathbf{x} \leq \underbrace{\mathbf{e}_{max} - \mathbf{e}_{fov}}_{\mathbf{b}_{\text{FoV}}}. \tag{3.18}$$

The zero block indicates that the FoV constraint does not act on the task-space slack variables. As the target approaches the image boundary, the admissible camera motion in the corresponding direction is progressively restricted, helping to keep the target inside the camera field of view during reactive tracking. In the final grasping phase, the FoV constraint is disabled because the controller primarily focuses on reaching the final grasp pose.

By solving the QP problem at each control cycle, the controller obtains a feasible whole-body velocity command that balances end-effector tracking, base motion, arm motion, velocity limits, joint constraints, and visual tracking requirements.

3.3.4 Controller State Machine

The controller is organized as a simple state machine with three main phases: `tracking`, `stopping`, and `final_pose`. In the `tracking` phase, the robot continuously follows the updated target-related pose. The detected target pose is monitored over a short time window to determine whether it has become stable.

When the target pose is considered stable, the robot enters the `stopping` phase. During this phase, zero velocity commands are sent to both the mobile base and the arm to ensure a smooth transition. The target pose is then fixed in the odometry frame and used as the reference for the final grasping motion.

In the `final_pose` phase, the controller moves the mobile base and the arm toward the final grasping pose using the same QP-based whole-body control formulation. The grasping motion is considered complete when the translational error becomes smaller than a predefined threshold.

3.3.5 Command Publication

After the QP problem is solved, the optimized generalized velocity vector is divided into base and arm commands. The first two elements are used as the mobile base velocity command:

$$\omega_b = \dot{q}_0, \quad v_b = \dot{q}_1. \tag{3.19}$$

The yaw angular velocity ω_b is published to the `angular.z` field, while the forward linear velocity v_b is published to the `linear.x` field. The remaining six elements are sent as UR10 joint velocity commands:

$$\dot{\mathbf{q}}_a = \begin{bmatrix} \dot{q}_1 & \dot{q}_2 & \dot{q}_3 & \dot{q}_4 & \dot{q}_5 & \dot{q}_6 \end{bmatrix}^T. \tag{3.20}$$

These joint velocities are published to the arm velocity controller. Therefore, at each control cycle, the whole-body controller outputs one velocity command for the mobile base and one velocity command for the UR10 arm. If no valid solution is available, zero velocity commands are sent to keep the system safe.

3.3.6 Summary

The proposed whole-body controller combines pose servoing, differential kinematics, and QP-based velocity optimization. By optimizing the mobile base and arm velocities together, the controller enables the end-effector to reactively track target-related grasp poses while maintaining feasible robot motion. This provides the control foundation for the simulation and real-world grasping experiments presented in the following chapter.

4
Results

This chapter presents the experimental results of the proposed whole-body mobile manipulation system. The evaluation is divided into two parts: simulation experiments and real-world grasping experiments. The simulation experiments are used to evaluate the behavior of the proposed controller under controlled and repeatable conditions, while the real-world experiments are used to validate the complete perception-control pipeline on the physical mobile manipulator.

In simulation, the controller is evaluated using representative target motion patterns and obstacle-related scenarios. Three target trajectories are considered: sideway motion, object motion toward the robot, and nonlinear motion. These trajectories are designed to test whether the controller can continuously follow updated target poses while coordinating the mobile base and the robotic arm. In addition, obstacle-related simulation scenes are used to examine the motion behavior of the robot in constrained environments.

In the real-world experiments, the proposed system is evaluated using several objects with different shapes and sizes. The perception module estimates the grasp target pose from sensor data, and the whole-body controller generates coordinated velocity commands for the MiR mobile base and the UR10 manipulator. The results are analyzed qualitatively through experiment snapshots and quantitatively using recorded tracking data.

4.1 Simulation Results

Simulation experiments are conducted in CoppeliaSim before testing the system on the physical robot. The simulation environment includes the MiR mobile base, the UR10 manipulator, the end-effector frame, target objects, predefined reference trajectories, and obstacle elements. This environment provides a safe and repeatable platform for evaluating the behavior of the controller under different target motions.

The main purpose of the simulation experiments is to verify whether the whole-body controller can follow continuously updated target poses. During each experiment, the expected target trajectory and the actual end-effector trajectory are recorded and compared. The expected trajectory represents the target-related pre-grasp pose, while the actual trajectory represents the motion of the controlled end-effector frame.

4.1.1 Trajectory Tracking Results

Three representative target motion patterns are designed for the trajectory tracking evaluation: sideway motion, object motion toward the robot, and nonlinear motion. These trajectories cover different motion characteristics. The sideway motion mainly evaluates lateral tracking behavior. The object motion toward the robot evaluates whether the controller can maintain tracking when the target approaches the robot. The nonlinear trajectory introduces changes in target direction and therefore provides a more challenging tracking scenario.

To quantify the tracking performance, the position tracking error is computed as the Euclidean distance between the expected end-effector position and the actual end-effector position:

$$e_p(t) = \|\mathbf{p}_{ep}(t) - \mathbf{p}_e(t)\|_2 \,, \tag{4.1}$$

where $\mathbf{p}_{ep}(t)$ is the position component of the expected pose and $\mathbf{p}_e(t)$ is the position component of the actual end-effector pose

Figure 4.1 shows the tracking result for the sideway motion trajectory. In this experiment, the target moves laterally with respect to the robot. The result shows that the end-effector is able to follow the updated target pose throughout the motion.

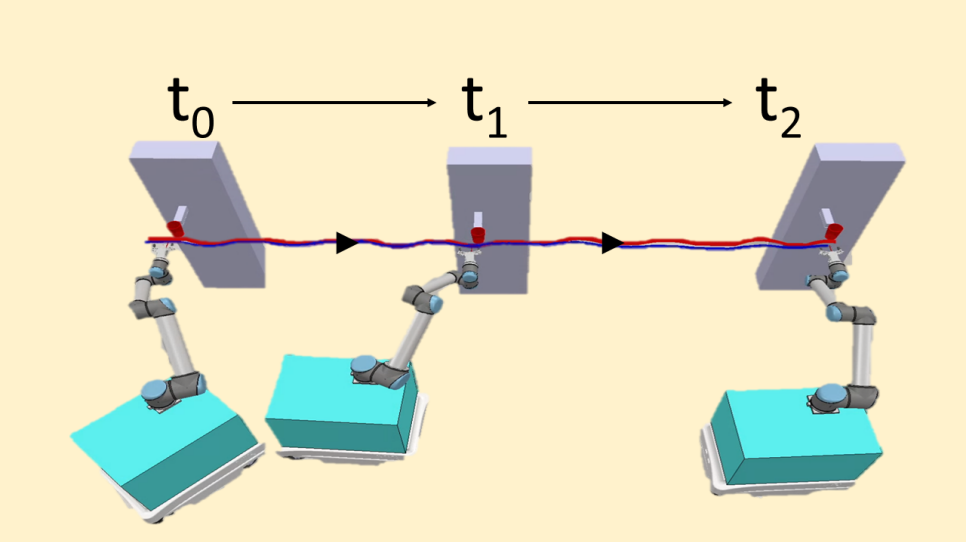


Figure 4.1: Tracking evaluation for the sideway motion trajectory in CoppeliaSim.

Figure 4.2 shows the tracking result for the trajectory where the object moves toward the robot. In this case, the target gradually approaches the robot, which requires the controller to adjust both the mobile base and the arm configuration. This motion is important for evaluating whether the controller can maintain a suitable pre-grasp configuration when the target moves closer to the platform.

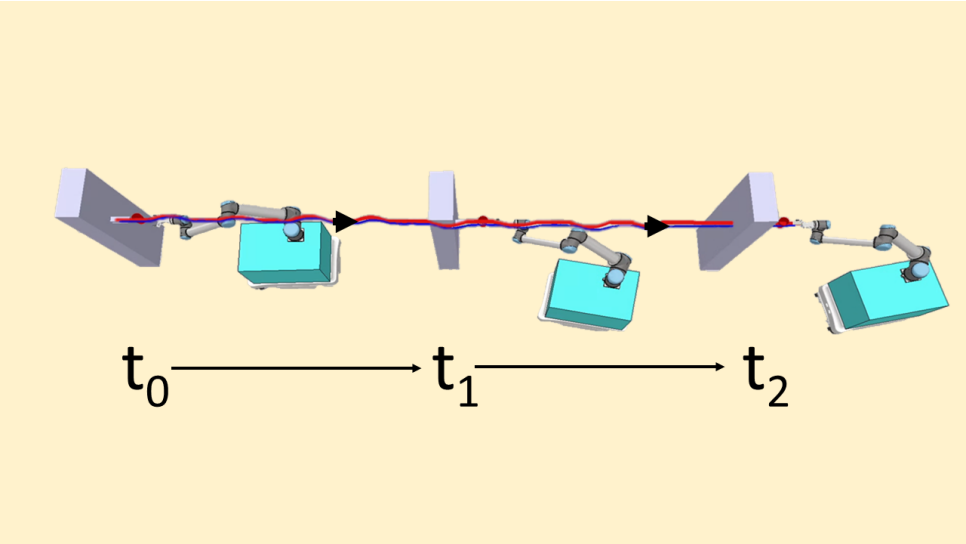


Figure 4.2: Tracking evaluation for the backward motion trajectory in CoppeliaSim.

Figure 4.3 shows the tracking result for the nonlinear motion trajectory. Compared with the previous two cases, the nonlinear trajectory contains more variation in target direction. This makes the tracking task more challenging, since the controller must react to changing target poses while maintaining coordinated base-arm motion.

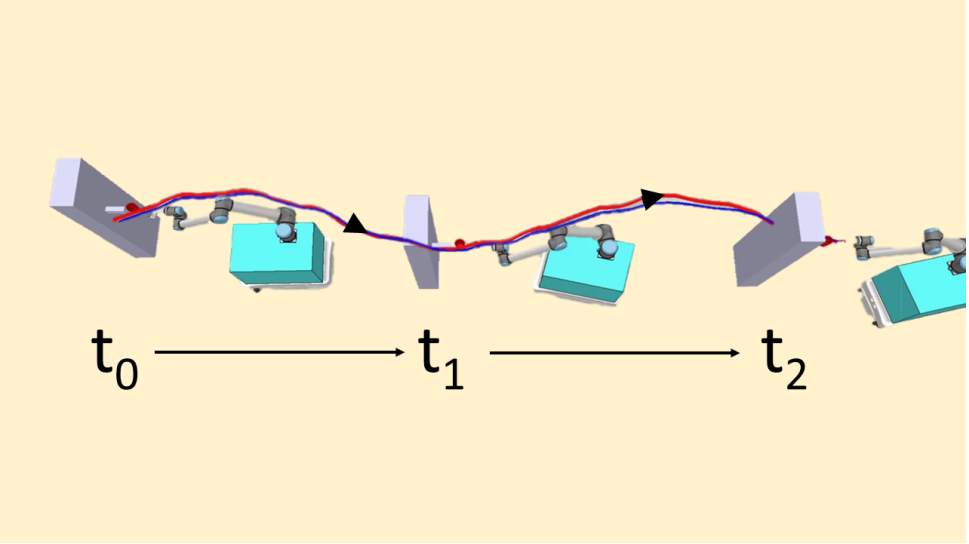


Figure 4.3: Tracking evaluation for the nonlinear motion trajectory in CoppeliaSim.

The average tracking errors for the three simulation trajectories are summarized in Table 4.1. The sideway motion shows a larger average tracking error than the other two trajectories. This is mainly related to the non-holonomic constraint of the MiR mobile base. Since the base cannot generate lateral velocity directly, sideways target motion cannot be followed by pure sideward translation of the platform. Instead, the controller has to coordinate base rotation, forward motion, and arm

motion to indirectly compensate for the lateral displacement. As a result, the end-effector tracking error becomes larger during sideway motion compared with the object moving toward the robot and nonlinear motion cases. The object motion toward the robot and the nonlinear motion show smaller average errors, indicating that the controller can maintain the end-effector close to the expected pre-grasp pose under these motion patterns.

Table 4.1: Average pre-grasp tracking error in simulation.

Trajectory	Average tracking error (cm)
Sideway motion	6.21
Backward motion	3.20
Nonlinear motion	3.34

Overall, the simulation tracking results show that the controller can generate coordinated motion for the mobile base and the UR10 arm under different target motion patterns. The mobile base provides large-scale repositioning, while the manipulator performs local end-effector adjustment. This coordination allows the system to continuously follow pre-grasp poses during object motion.

4.1.2 Obstacle-related Simulation Results

In addition to trajectory tracking, obstacle-related simulation scenes are created in CoppeliaSim to evaluate the robot motion behavior in constrained environments. The purpose of these experiments is to visualize how the robot moves when obstacles are present and to assess the feasibility of extending the framework toward obstacle-aware whole-body mobile manipulation.

Figure 4.4 shows the obstacle-related simulation setup. The scene contains the mobile manipulator, target motion, and obstacle elements.

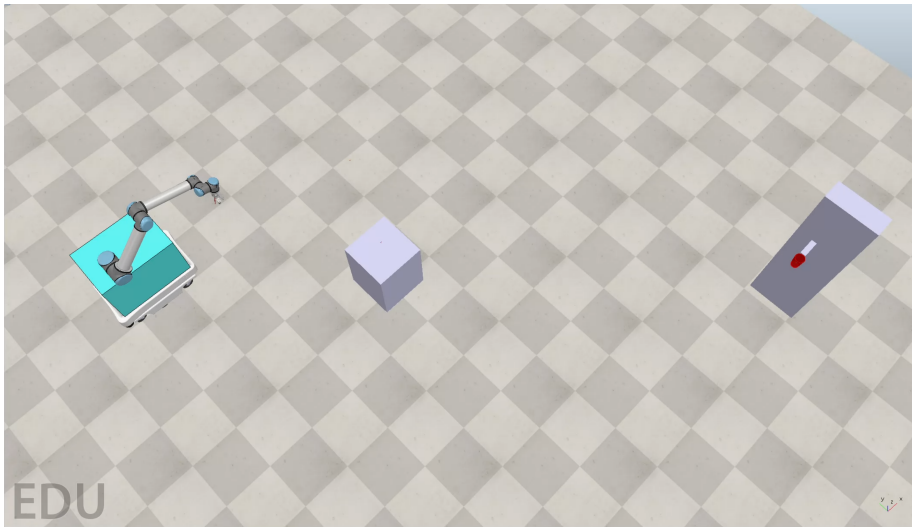


Figure 4.4: Obstacle-related simulation setup in CoppeliaSim. The scene includes the mobile manipulator, target, and obstacle elements.

Figure 4.5 shows the resulting robot motion in the obstacle-related simulation. The result illustrates that the robot can still reach the pre-grasp region while operating in a constrained environment.

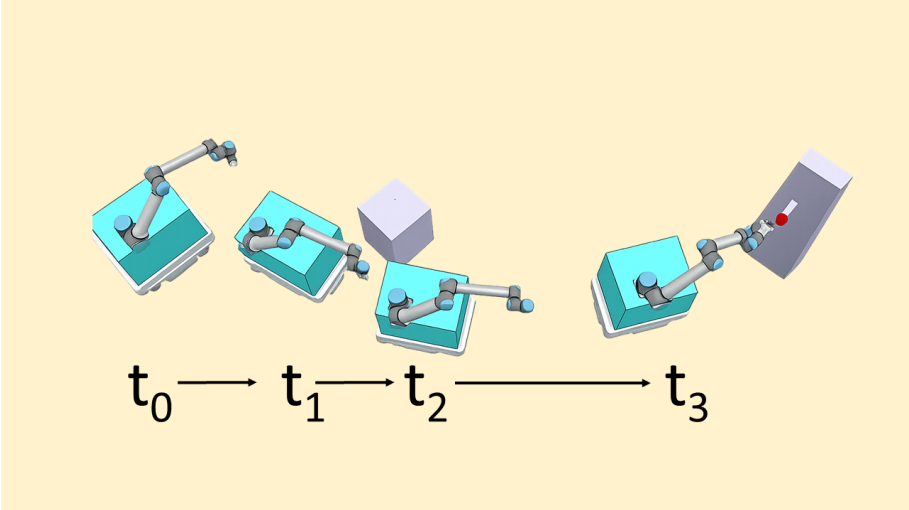


Figure 4.5: Example obstacle avoidance scenario in CoppeliaSim.

The obstacle avoiding simulation provides a qualitative evaluation of the motion behavior of the mobile manipulator in constrained scenes. While the real-world experiments in this thesis mainly focus on visual tracking and grasp execution, these simulation results show the potential of extending the proposed whole-body control framework toward obstacle-aware mobile manipulation.

4.2 Real-world Grasping Results

Real-world experiments are conducted on the physical mobile manipulator to evaluate the complete perception-control pipeline. The experimental platform consists of a MiR mobile base, a UR10 manipulator, an RG2-FT gripper, and an end-effector-mounted RGB-D camera. The perception module estimates the grasp target pose from the observed object, and the whole-body controller generates coordinated commands for the mobile base and the robotic arm.

Different objects are used in the real-world experiments to evaluate the robustness of the system under varying object shapes and sizes. The tested objects include box-shaped objects, cylindrical objects, and spherical or irregular objects. These objects provide different perception and grasping conditions, making them suitable for evaluating the complete mobile manipulation pipeline.

4.2.1 Experimental Setup

Figure 4.6 shows the real-world experimental setup. The object is placed in front of the robot and observed by the wrist-mounted RGB-D camera. The estimated grasp pose is published as a target pose and transformed into the reference frame used by

the controller. The expected end-effector pose is generated by applying a predefined grasping offset to the detected target pose.

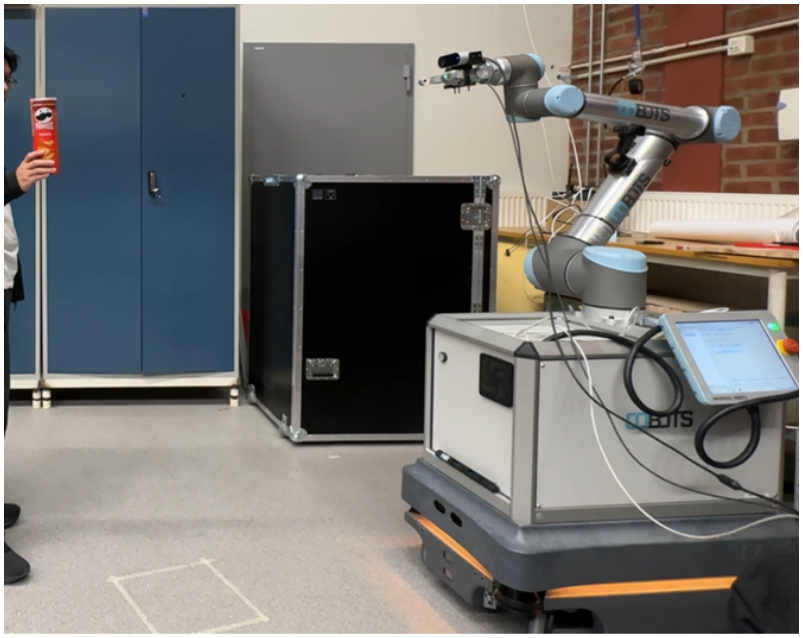


Figure 4.6: Real-world experimental setup using the MiR mobile base, UR10 manipulator, RG2-FT gripper, and wrist-mounted RGB-D camera.

The robot first tracks the detected grasp target and continuously updates the expected pre-grasp pose. Once the target pose becomes stable, it is fixed as the final grasping reference, and the robot moves the end-effector toward the object.

4.2.2 Grasping Results with Different Objects

The real-world grasping experiments are performed using multiple objects with different geometries. This section presents representative grasping results for these objects. Each result illustrates the complete process from whole-body tracking to final grasp execution.

The tested objects include a Pringles can, a cleanser bottle, a cracker box, and a baseball. These objects are selected to provide different shapes and grasping conditions, including cylindrical, bottle-shaped, box-shaped, and compact curved geometries. The purpose of this evaluation is to examine whether the proposed perception-control pipeline can be applied to different object types rather than only to a single target object.

Among these experiments, the Pringles can is presented as a representative case for detailed visualization and analysis. Figure 4.7 shows the initial RViz visualization of the Pringles can grasping experiment. At this stage, the target pose has been detected and visualized, and the robot starts tracking the expected pre-grasp pose. The visualization shows the spatial relationship between the robot, the controlled end-effector frame, and the target-related frame before the grasping motion.

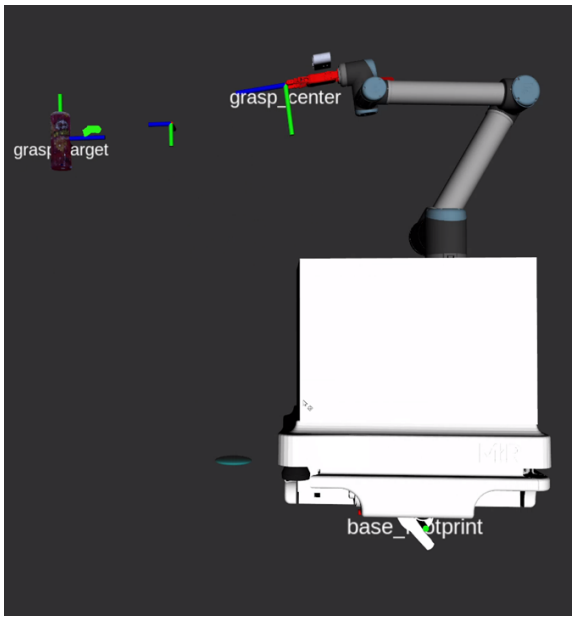


Figure 4.7: Initial RViz visualization of the Pringles can grasping experiment. The detected target pose and robot state are visualized before the grasping motion.

Figure 4.8 shows the RViz visualization after the grasping motion is completed. Compared with the initial state, the end-effector has moved toward the target object, and the final grasping pose is reached through the coordinated motion of the mobile base and the UR10 arm. This result demonstrates that the controller can use the estimated target pose to guide the robot toward the object in the real-world setup.

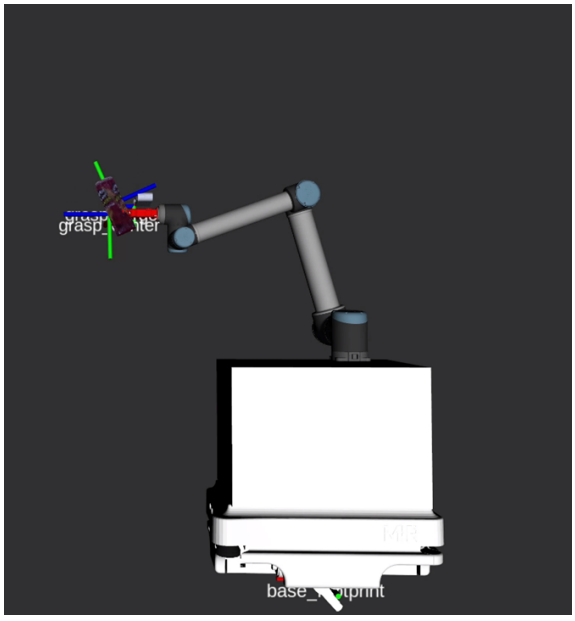


Figure 4.8: Final RViz visualization of the Pringles can grasping experiment. The end-effector reaches the target region after whole-body tracking and final grasp execution.

To further illustrate the tracking behavior during the Pringles can experiment, the

trajectories of `grasp_center` and `grasp_target` are compared in Figure 4.9. The green trajectory represents the target reference published as `grasp_target`, while the red trajectory represents the actual motion of the controlled end-effector frame, `grasp_center`. Their spatial relationship provides a qualitative illustration of how the end-effector follows the moving target reference during the tracking stage.

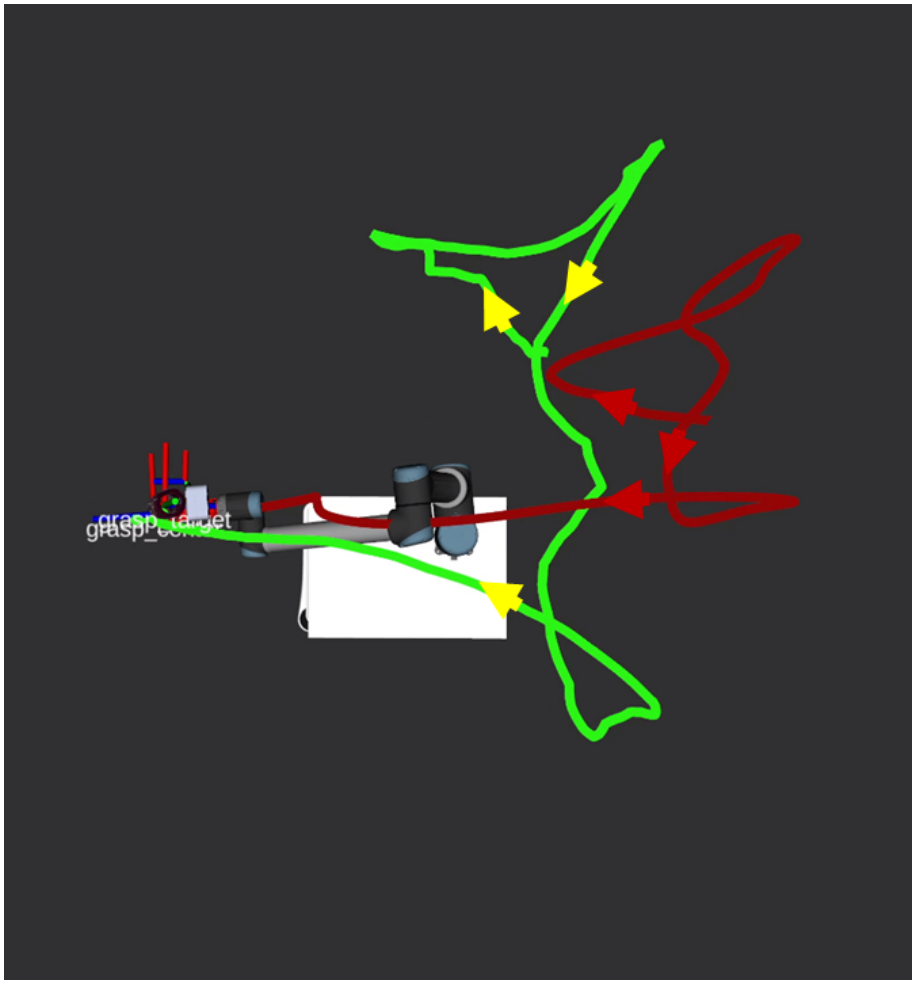


Figure 4.9: Trajectory comparison between `grasp_center` and `grasp_target` during the Pringles can experiment. The comparison shows how the controlled end-effector follows the target reference trajectory.

For each object, the grasping process is presented using three representative stages: the initial state, an intermediate tracking state, and the final grasping state. The initial state shows the robot before the grasping motion starts. The intermediate state shows the robot moving toward the object while updating the target grasp pose. The final state shows the robot after reaching the object and completing the grasping motion.

Figure 4.10 shows the grasping sequence for the cleanser bottle. The object has a complex shape and provides a vertical grasping target. The result shows that the robot is able to approach the object and complete the grasping motion using the same whole-body control pipeline.

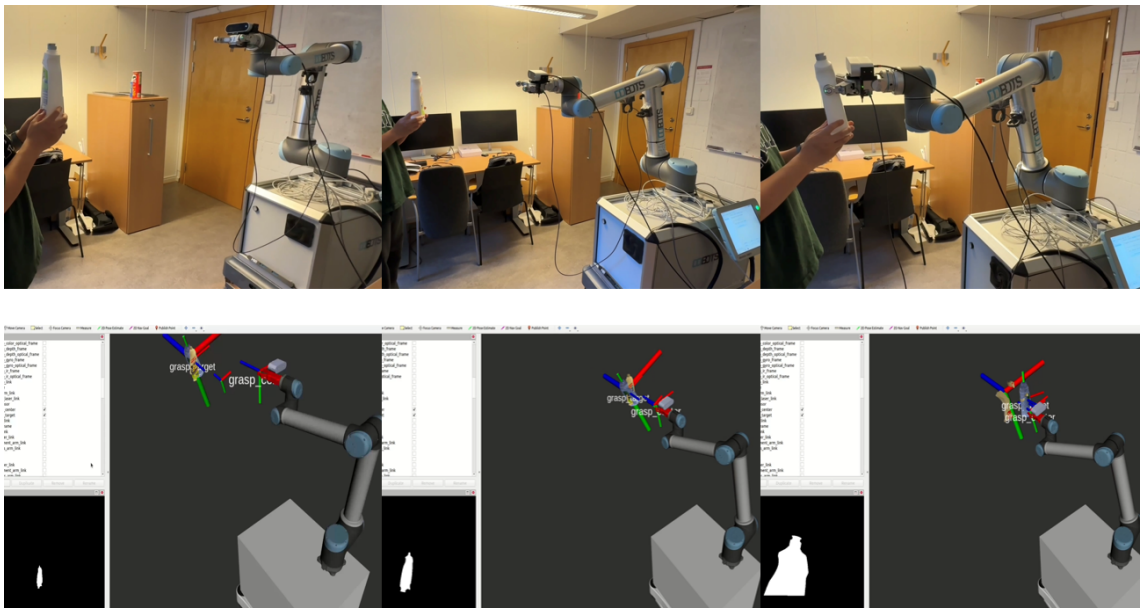


Figure 4.10: Real-world grasping sequence for the cleanser bottle.

Figure 4.11 shows the grasping sequence for the cracker box. Compared with the cleanser bottle, the cracker box has a rectangular shape, which provides a different geometric condition for pose estimation and grasp execution. The result demonstrates that the proposed pipeline can also be applied to box-shaped objects.

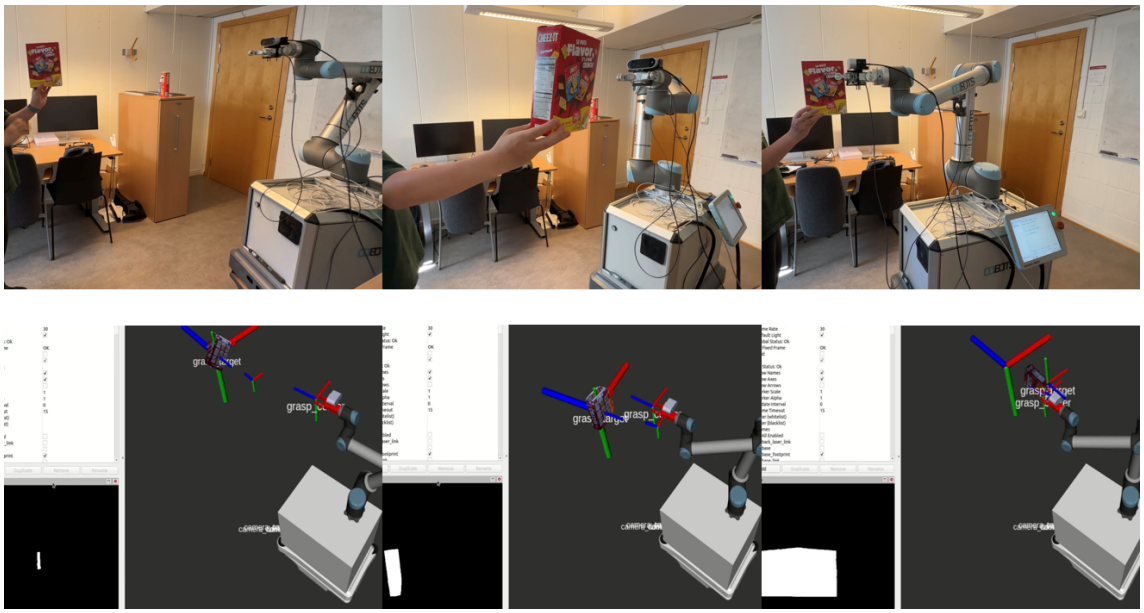


Figure 4.11: Real-world grasping sequence for the cracker box.

Figure 4.12 shows the grasping sequence for the Pringles can. The cylindrical shape provides a clear but different grasping geometry compared with the previous objects. During the experiment, the robot detects the target, moves toward the expected grasp-related pose, and completes the grasping motion.

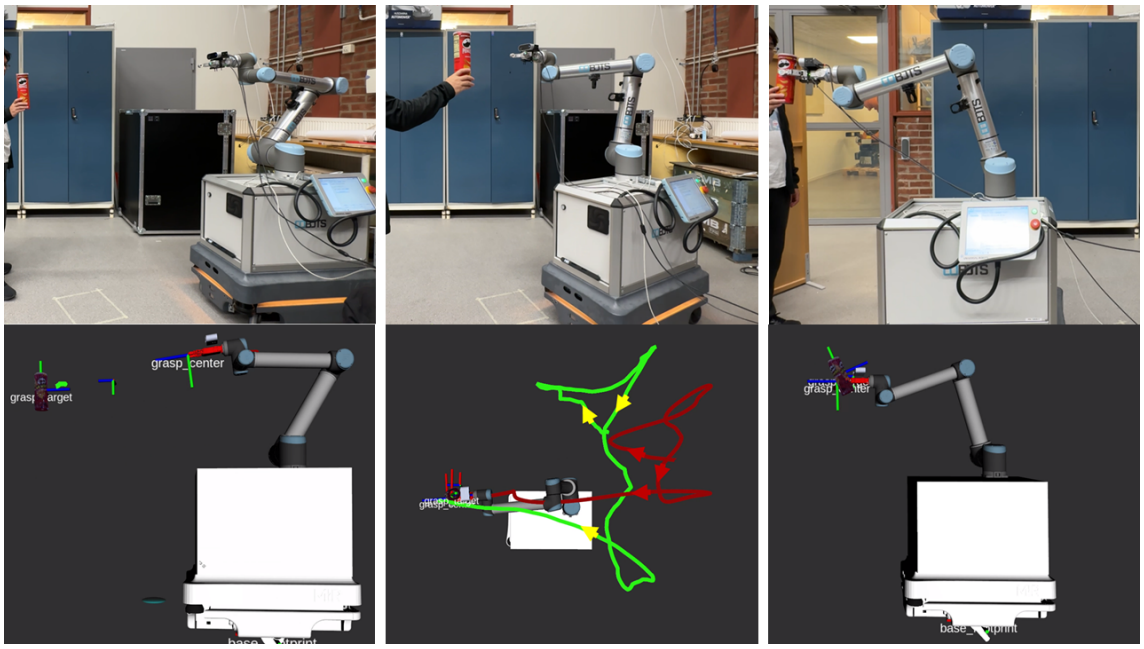


Figure 4.12: Real-world grasping sequence for the Pringles can.

Figure 4.13 shows the grasping sequence for the baseball. The baseball has a compact curved shape, which differs from the bottle-shaped, box-shaped, and cylindrical objects. The result provides an additional validation case for grasping objects with different geometry.

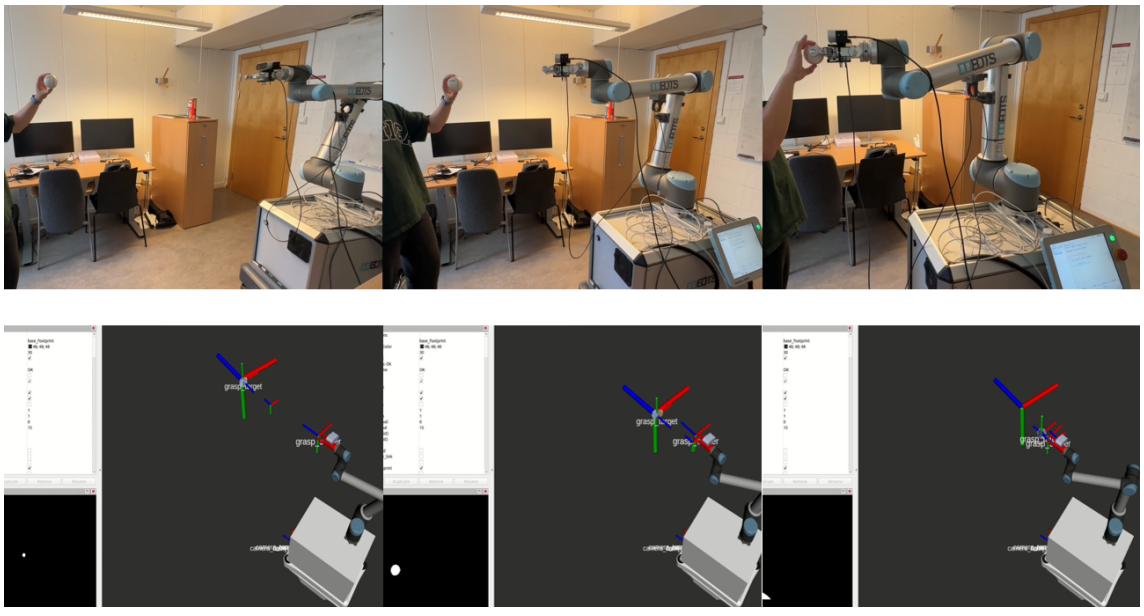


Figure 4.13: Real-world grasping sequence for the baseball. The three images show the initial state, intermediate tracking state, and final grasping state.

The results with different objects show that the proposed system can be applied to a range of object geometries. Although the object shapes are different, the same

overall pipeline is used in each case: the object pose is estimated, the expected grasp-related pose is generated, and the mobile base and manipulator move cooperatively toward the target. These experiments provide qualitative evidence that the proposed whole-body mobile manipulation framework is not limited to a single object type.

4.2.3 Quantitative Data Analysis

Quantitative data analysis is performed on the representative Pringles can grasping experiment. The analysis focuses on the middle tracking stage of the experiment, rather than the full execution process. The initial convergence period is excluded because the robot and the target may have a relatively large initial offset before the controller reaches a stable tracking behavior. The final grasping stage is also excluded because the system is no longer only performing continuous target tracking, but is moving toward the final grasping pose.

The tracking error is evaluated using the relative position difference between the controlled end-effector frame and the target-related reference frame. Specifically, the recorded trajectories of `grasp_center` and `grasp_target` are used to compute the tracking error during the selected tracking interval. The error is defined as

$$e(t) = \|\mathbf{p}_{center}(t) - \mathbf{p}_{target}(t)\|_2 \,, \tag{4.2}$$

where $\mathbf{p}_{center}(t)$ is the position of the `grasp_center` frame and $\mathbf{p}_{target}(t)$ is the position of the `grasp_target` frame.
Figure 4.14 shows the tracking error over time during the selected tracking interval. Most of the tracking process remains within a moderate error range, indicating that the controller can maintain the end-effector close to the target-related reference during continuous tracking.

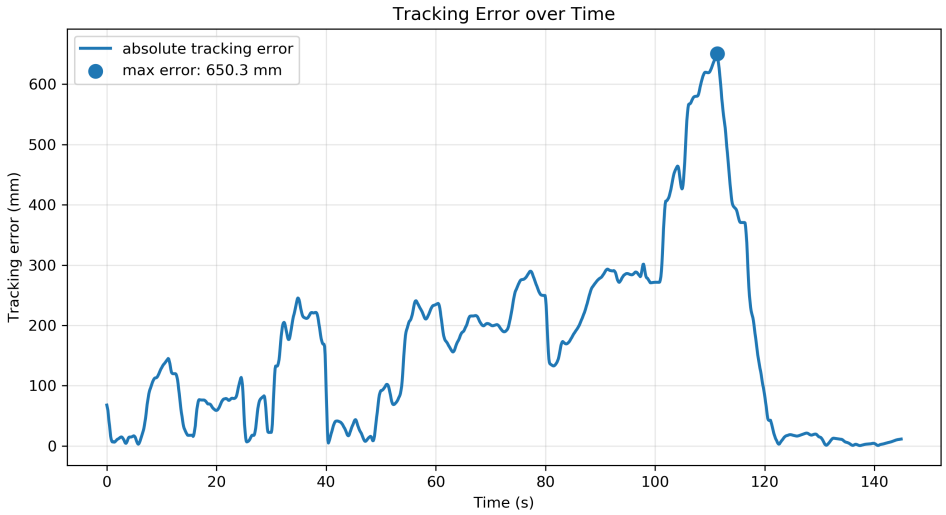


Figure 4.14: Tracking error over time during the selected tracking stage of the Pringles can experiment. The peak error occurs when the target is moved quickly toward the final grasping position.

The statistical results are summarized as follows. The mean error is 163.95 mm, the root mean square error is 224.54 mm, and the standard deviation is 169.46 mm. The final error is 11.84 mm.

Figure 4.15 shows the distribution of the tracking error. The histogram indicates that a large portion of the tracking samples is concentrated in the lower-error region. The long tail in the distribution is mainly caused by the transient high-error period observed in Figure 4.14. This suggests that the large error values are not present throughout the experiment, but are concentrated around a short interval where the target motion changes rapidly.

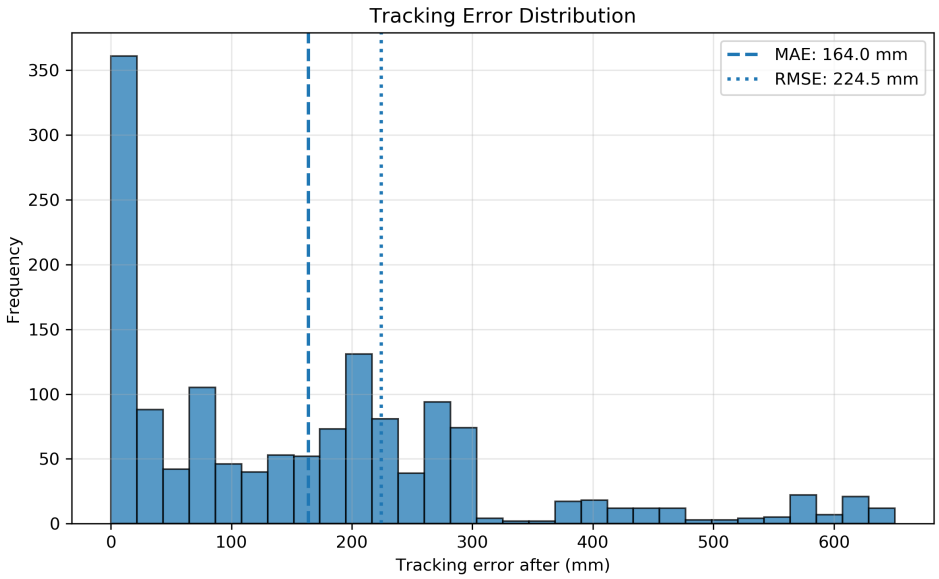


Figure 4.15: Tracking error distribution during tracking stage of the Pringles can experiment.

Overall, the quantitative analysis shows that the whole-body controller is able to maintain tracking during the selected real-world experiment. Although a large transient error occurs when the object is moved rapidly, the error decreases again after the disturbance, and the final error remains small. This indicates that the controller can recover from sudden target motion and continue tracking the target-related reference pose.

4.3 Summary of Results

The simulation and real-world results demonstrate the effectiveness of the proposed whole-body mobile manipulation framework. In simulation, the controller is evaluated using three representative target motions. The results show that the end-effector can continuously follow updated target-related pre-grasp poses while the mobile base and the arm move cooperatively. The obstacle-related simulation also provides a qualitative demonstration of robot motion behavior in constrained envi-

ronments.

In the real-world experiments, the Pringles can experiment provides the main validation case with RViz visualization, trajectory comparison, and quantitative data analysis. The results show that the system can detect the target object, generate the expected grasp-related pose, and move the end-effector toward the target using whole-body control. Additional object experiments further indicate that the pipeline can be applied to objects with different shapes.

Overall, the results show that the proposed framework can support reactive whole-body tracking and grasping of object target poses in both simulation and real-world settings.

5
Conclusion

This thesis investigates reactive whole-body tracking and grasping with a mobile manipulator consisting of a MiR mobile base and a UR10 robotic arm. The main contribution is a QP-based controller that treats the base and arm as a unified kinematic system and computes their velocity commands simultaneously. Combined with visual object-pose estimation and grasp-target generation, the controller enables the end-effector to follow continuously updated pre-grasp poses. When the estimated target pose becomes stable, it is fixed in the odometry frame and used as the reference for the final grasping motion.

The simulation results show that the controller tracks three representative target-motion patterns while coordinating the mobile base and manipulator. The average positional tracking errors are 6.21 cm for sideways motion, 3.20 cm for motion toward the robot, and 3.34 cm for nonlinear motion. The larger error during sideways motion is consistent with the non-holonomic constraint of the MiR base, which cannot generate lateral velocity directly. The obstacle-related simulations provide a qualitative demonstration of operation in constrained environments, although obstacle avoidance is not fully integrated into the real-world whole-body grasping pipeline.

The real-world experiments provide a feasibility validation of the complete perception–control pipeline using a cleanser bottle, a cracker box, a Pringles can, and a baseball. In the representative Pringles can experiment, the controller follows the moving target-related reference and recovers after a transient error caused by rapid object motion. The selected tracking interval has a mean error of 163.95 mm, an RMSE of 224.54 mm, and a final error of 11.84 mm. These results indicate that coordinated base–arm tracking and subsequent grasp execution are feasible on the physical platform, while also showing that tracking performance remains sensitive to rapid target motion and perception uncertainty.

Several limitations remain. First, the real-world experiments mainly focus on perception tracking and grasp execution, while obstacle-aware whole-body tracking is not fully integrated into the real-world grasping pipeline. Second, the performance of the system depends on the accuracy and stability of the perception module. Noise or delay in target pose estimation can affect the tracking behavior and final grasping accuracy.

Future work can address these limitations in several directions. Obstacle information from the navigation stack or costmap representation could be integrated into

the QP controller to enable real-time obstacle-aware whole-body tracking. In addition, the perception module could be improved to provide more stable and accurate object pose estimates under dynamic motion. Finally, more extensive real-world experiments with a larger set of objects and motion patterns would provide a more complete evaluation of the proposed mobile manipulation framework.

Bibliography

[1] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*, 2nd ed. Springer, 2016.

[2] A. Tekden, M. P. Deisenroth, and Y. Bekiroglu, "Grasp transfer based on self-aligning implicit representations of local surfaces," *IEEE Robotics and Automation Letters*, vol. 8, no. 10, pp. 6315–6322, 2023.

[3] A. Ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp pose detection in point clouds," *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1455–1473, 2017.

[4] F. Burget and M. Bennewitz, "Stance selection for humanoid grasping tasks by inverse reachability maps," in *2015 IEEE International conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 5669–5674.

[5] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[6] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2023, pp. 785–799.

[7] N. Ghodsian, K. Benfriha, A. Olabi, V. Gopinath, and A. Arnou, "Mobile manipulators in industry 4.0: A review of developments for industrial applications," *Sensors*, vol. 23, no. 19, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/19/8026

[8] Y. Wu, P. Balatti, M. Lorenzini, F. Zhao, W. Kim, and A. Ajoudani, "A teleoperation interface for loco-manipulation control of mobile collaborative robotic assistant," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3593–3600, 2019.

[9] 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, Apr. 2022, arXiv:2109.04749 [cs]. [Online]. Available: http://arxiv.org/abs/2109.04749

[10] S. Kim, K. Jang, S. Park, Y. Lee, S. Y. Lee, and J. Park, "Whole-body control of non-holonomic mobile manipulator based on hierarchical quadratic programming and continuous task transition," in *4th IEEE International*

Conference on Advanced Robotics and Mechatronics, ICARM 2019, Toyonaka, Japan, July 3-5, 2019. IEEE, 2019, pp. 414–419. [Online]. Available: https://doi.org/10.1109/ICARM.2019.8834269

[11] J. González Huarte and A. Ibarguren, "Visual servoing architecture of mobile manipulators for precise industrial operations on moving objects," *Robotics*, vol. 13, no. 5, 2024. [Online]. Available: https://www.mdpi.com/2218-6581/13/5/71

[12] F. Husain, A. Colomé, B. Dellen, G. Alenya, and C. Torras, "Realtime tracking and grasping of a moving object from range video," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 2617–2622.

[13] N. Marturi, M. Kopicki, A. Rastegarpanah, V. Rajasekaran, M. Adjigble, R. Stolkin, A. Leonardis, and Y. Bekiroglu, "Dynamic grasp and trajectory planning for moving objects," *Autonomous Robots*, vol. 43, no. 5, pp. 1241–1256, 2019.

[14] M. Tuscher, J. Hörz, D. Driess, and M. Toussaint, "Deep 6-dof tracking of unknown objects for reactive grasping," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 14 185–14 191.

[15] C.-C. Wong, M.-Y. Chien, R.-J. Chen, H. Aoyama, and K.-Y. Wong, "Moving object prediction and grasping system of robot manipulator," *IEEE Access*, vol. 10, pp. 20 159–20 172, 2022.

[16] E. Maranci, C. A. Avizzano, and S. D'Avella, "Reactive grasp controller for dynamic grasping of moving objects," *Robotics and Autonomous Systems*, p. 105401, 2026.

[17] S. Yenamandra, A. Ramachandran, K. Yadav, A. Wang, M. Khanna, T. Gervet, T.-Y. Yang, V. Jain, A. W. Clegg, J. Turner, Z. Kira, M. Savva, A. Chang, D. S. Chaplot, D. Batra, R. Mottaghi, Y. Bisk, and C. Paxton, "Homerobot: Open-vocabulary mobile manipulation," 2024. [Online]. Available: https://arxiv.org/abs/2306.11565

[18] P. Liu, Y. Orru, J. Vakil, C. Paxton, N. Shafiullah, and L. Pinto, "Demonstrating ok-robot: What really matters in integrating open-knowledge models for robotics," in *Robotics: Science and Systems XX*, ser. RSS2024. Robotics: Science and Systems Foundation, 2024. [Online]. Available: http://dx.doi.org/10.15607/RSS.2024.XX.091

[19] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger, and O. Madsen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing*, vol. 37, pp. 282–291, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584515000575

[20] S. Thakar, S. Srinivasan, S. Al-Hussaini, P. M. Bhatt, P. Rajendran, Y. Jung Yoon, N. Dhanaraj, R. K. Malhan, M. Schmid, V. N. Krovi, and S. K. Gupta, "A survey of wheeled mobile manipulation: A decision-making

perspective," *Journal of Mechanisms and Robotics*, vol. 15, no. 2, p. 020801, 07 2022. [Online]. Available: https://doi.org/10.1115/1.4054611

[21] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe, "HERB: a home exploring robotic butler," *Autonomous Robots*, vol. 28, no. 1, pp. 5–20, Jan. 2010. [Online]. Available: https://doi.org/10.1007/s10514-009-9160-9

[22] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Robot placement based on reachability inversion," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 1970–1975.

[23] A. Makhal and A. K. Goins, "Reuleaux: Robot base placement by reachability analysis," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2018, pp. 137–142.

[24] B. Burgess-Limerick, C. Lehnert, J. Leitner, and P. Corke, "An architecture for reactive mobile manipulation on-the-move," 2022. [Online]. Available: https://arxiv.org/abs/2212.06991

[25] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *Int. J. Rob. Res.*, vol. 33, no. 7, p. 1006–1028, Jun. 2014. [Online]. Available: https://doi.org/10.1177/0278364914521306

[26] F. Chaumette and S. Hutchinson, "Visual servo control. I. Basic approaches," *IEEE Robotics & Automation Magazine*, vol. 13, no. 4, pp. 82–90, Dec. 2006, conference Name: IEEE Robotics & Automation Magazine.

[27] ——, "Visual servo control. II. Advanced approaches [Tutorial]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 109–118, Mar. 2007, conference Name: IEEE Robotics & Automation Magazine.

[28] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 709–715.

[29] M. Quigley, "Ros: an open-source robot operating system," in *IEEE International Conference on Robotics and Automation*, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:6324125

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[31] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer, "Sam 2: Segment anything in images and videos," 2024. [Online]. Available: https://arxiv.org/abs/2408.00714

[32] C. Ryali, Y.-T. Hu, D. Bolya, C. Wei, H. Fan, P.-Y. Huang, V. Aggarwal, A. Chowdhury, O. Poursaeed, J. Hoffman, J. Malik, Y. Li, and C. Feichtenhofer, "Hiera: A hierarchical vision transformer without the bells-and-whistles," 2023. [Online]. Available: https://arxiv.org/abs/2306.00989

[33] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," 2021. [Online]. Available: https://arxiv.org/abs/2111.06377

[34] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[35] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," 2023. [Online]. Available: https://arxiv.org/abs/2104.09864

[36] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," 2023. [Online]. Available: https://arxiv.org/abs/2304.02643

[37] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 3212–3217.

[38] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[39] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Sensor Fusion IV: Control Paradigms and Data Structures*, P. S. Schenker, Ed., vol. 1611, International Society for Optics and Photonics. SPIE, 1992, pp. 586 – 606. [Online]. Available: https://doi.org/10.1117/12.57955

[40] J. Park, Q.-Y. Zhou, and V. Koltun, "Colored point cloud registration revisited," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 143–152.

[41] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 500–505.

[42] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se



CHALMERS
UNIVERSITY OF TECHNOLOGY