Master's Thesis

CAPerMoMa: Coupled Active Perception for Mobile Manipulation in Unknown Environments

Hamsa Datta Perur

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fulfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Sebastian Houben
Prof. Dr. Maren Bennewitz
Rohit Menon, M.Sc.
Dr. Alex Mitrevski

October 2024

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work. The report was, in part, written with the help of the AI assistants (Grammarly, ChatGPT, Github Copilot) as described in the appendix. I am aware that content generated by AI systems is no substitute for careful scientific work, which is why all AI-generated content has been critically reviewed by me, and I take full responsibility for it.

Date	Hamsa Datta Perur

Abstract

The rise of mobile manipulator (MoMa) systems in robotics has opened new avenues for robots to operate autonomously in indoor and unstructured environments such as homes and offices. MoMa is often tasked with ambiguous directives such as "get me the cup," requiring them to identify and locate objects without specific details on their location or orientation. The complexity of this task increases as the robot must explore unfamiliar environments while also searching for objects simultaneously. We propose CAPerMoMa (Coupled Active Perception for Mobile Manipulation), a system designed to improve MoMa's ability to explore indoor environments, detect objects, and execute precise manipulations. In our work, MoMa is equipped with actively controllable cameras that focus (gaze control) on the target object once it is detected - which we term "coupled active perception". The 'coupled' aspect refers to this dual objective of gazing at the object for mapping while planning for a base placement for grasping.

In our evaluation, we found that the continuous gaze control significantly reduced the mapping time and, in turn, improved total execution time. One of our primary goals is to get the best placement and maximize grasp success. Through our evaluation, we achieved a 70% grasp success rate, evaluated over four different scenarios with increasing complexity. Finally, we also show that our system generalizes well with new objects.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Prof. Dr. Sebastian Houben, Prof. Dr. Maren Bennewitz, Rohit Menon, and Dr Alex Mitrevski, for their invaluable guidance and support throughout this project. I am particularly thankful to Rohit for his help at various stages of this project and for helping me set up the volumetric mapping pipeline. Special thanks to Alex for helping me with the questions related to Toyota HSR's APIs and for his support whenever required.

I would like to thank Vamsi Krishna Kalagaturu and Allen Isaac Jose for helping me with my technical questions regarding simulations. Additionally, I would like to acknowledge the constructive discussions and support from my friends at Hochschule Bonn-Rhein-Sieg and the University of Bonn. Lastly, I thank my family for their unwavering moral support during the project.

Contents

1

Introduction

Service robots, particularly those designed for domestic settings, have witnessed remarkable advancements and acceptance in recent years [2] . Notable among these are cleaning robots, such as the Roomba [1] by iRobot, which have demonstrated significant success in household environments (see Figure 1.1). These robots excel at performing specific tasks like cleaning, navigating efficiently within the home to maintain cleanliness. However, their functionality is predominantly confined to navigation and surface-level cleaning [3].

Figure 1.1: Examples of service robots. Top image: Roomba robot cleaning the floor (taken from [1]). Bottom image: Toyota HSR searching for an object.

The evolution of service robots now focuses on the deployment of more complex systems capable of perception and mobile manipulation (MoMa) [4]. These advanced robots are being designed to operate in everyday environments, where their tasks extend beyond just navigation. In unstructured domestic settings, service robots are often confronted with ambiguous directives such as "get me the green cup," requiring them to identify and locate objects without specific details on their location or orientation. This task becomes more complex due to the need for the robot to explore unfamiliar environments and search for objects simultaneously [3].

The challenge for MoMa-equipped robots lies in their ability to successfully manipulate everyday objects like apples or cups, utilizing embodied cameras for active perception. This thesis aims to evaluate the manipulation success of such objects by service robots in domestic environments. Through this work, we seek to investigate the capabilities of MoMa-equipped robots, enabling them to perform more complex tasks in household settings.

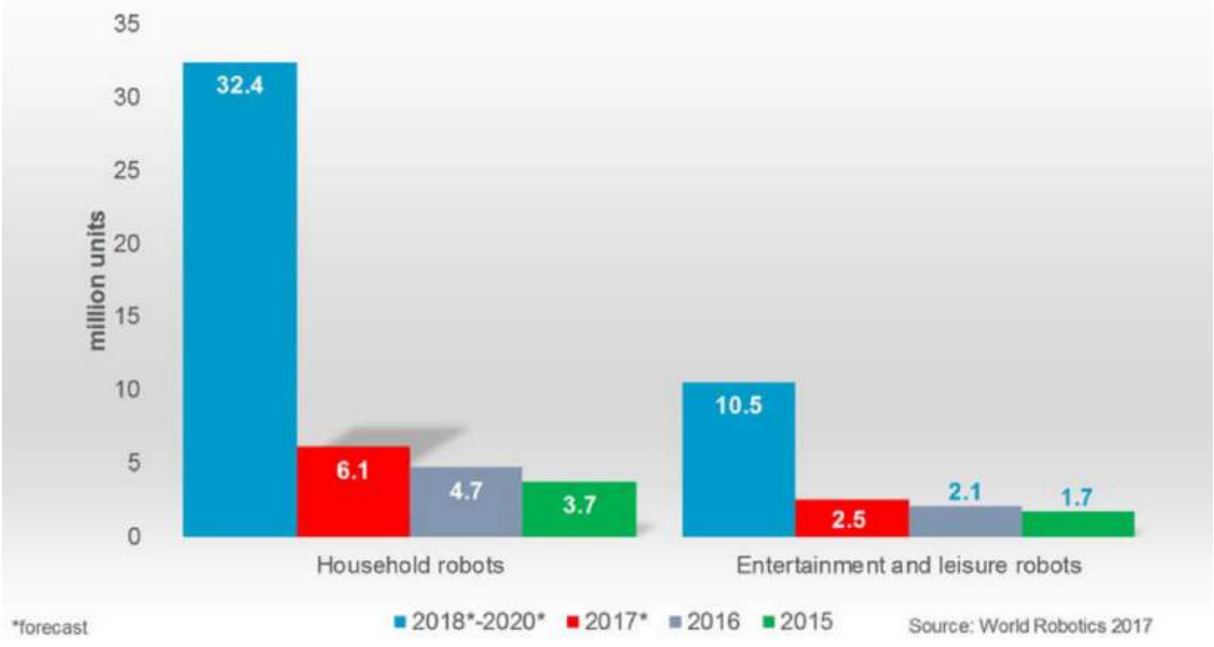


Figure 1.2: Domestic service robots unit sales 2015-2016, and forecast for 2017-2020 (taken from [5]).

1.1 Motivation

According to the International Federation of Robotics (IFR) service robots are defined as follows *"A service robot is a robot that works semi or completely autonomously to perform useful services for the well-being of humans and equipment, excluding manufacturing operations"* [2]. In an IFR report, the global sales of professional service robots experienced a significant growth of 48% in 2022 [2]. The IFR Statistical Department reported a total of nearly 158,000 units sold worldwide [2]. Additionally, the Robot-as-a-Service (RaaS) fleet expanded by 50%, surpassing 21,000 units [2]. Notably, mobile robot solutions have become well-established in transportation and logistics, with a remarkable 44% increase in sales during the same period [2]. These statistics show the growing importance and demand for service robots.

In 2021, the market for mobile manipulators (also see Figure 1.2) was valued at $470.2 million, with projections indicating a compound annual growth rate (CAGR) ranging from 11.1% to 47.1% from 2022 to 2028 [6]. This increasing demand is driven by a shortage of cost-efficient labor [6] and in turn increasing the demand for service robots. In this context, as service robots gain prevalence in unstructured settings like homes, they encounter directives with complex instructions without specific details on location or orientation. For instance, the target object to grasp could be anywhere on a table in a room that the robot has not yet seen or interacted with. MoMa are thus faced with the challenge of simultaneously exploring the environment and searching for the object (see Figure 1.3). The key objective is to locate and successfully grasp the object in the shortest amount of time, with a high rate of success, rather than constructing a comprehensive map of the surroundings and consuming more time. Therefore, the mapping

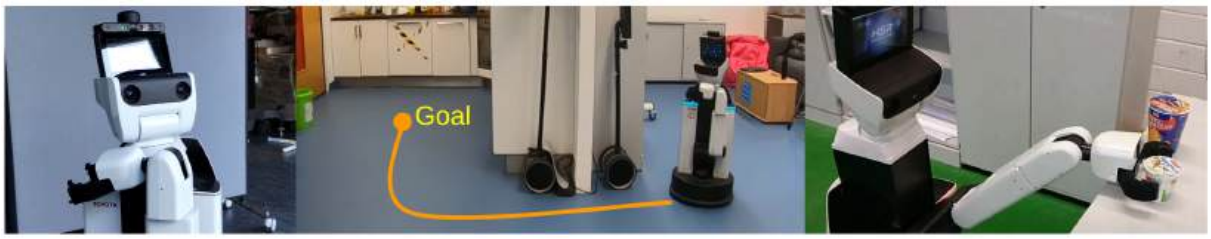


Figure 1.3: Different phases in mobile manipulation: searching, navigating, grasping (rightmost image taken from [7]).

performed by the robot should be just sufficient to facilitate successful grasping.

Unlike passive perception systems that simply take in environmental data without interaction (i.e. no camera movement), active perception actively adjusts its sensors e.g. camera to maximize information while minimizing movement cost and time. This approach is useful for mapping and pose estimation. According to Bajcsy *et al.* [8], The service industry, in particular, can greatly benefit from this advancement, as it relies on robots for tasks such as providing elderly assistance, and performing deliveries, all of which require sophisticated perception and manipulation capabilities [8]. However, in this work, we propose to couple active perception with manipulation i.e. while we plan to find desired poses for the mobile manipulator base, we balance between active exploration and object mapping on one hand and grasping success on the other. The 'coupled' aspect refers to this dual objective: any chosen pose should ideally enhance the robot's grasp potential while simultaneously contributing to the environmental map, just enough to support the primary goal of manipulation. This enables MoMa to operate more efficiently in complex and unknown domestic settings.

1.2 Challenges

The below are some of the common challenges MoMa face during gasping a target object in an unknown environment.

- Houses are visually congested spaces with a variety of items, and lighting. Accurately perceiving and recognizing objects is necessary for a robot, but this can be challenging because of things like similar-looking objects, occlusions (objects partially hidden), and changing lighting. Sometimes it is not possible to train a custom object detection model; we rely on generic models trained on large-scale data. Moreover, misidentifying objects is a problem.

- Even if an object is detected, determining a suitable base placement for manipulation is important. This base placement must also be navigable by MoMa, ensuring that it avoids hitting obstacles like furniture or tables while positioning itself for grasping. The navigable location is generally determined when MoMa is very close to the object, if a grasp is not possible, it plans for the next best base location. This process increases both the navigation time and the overall execution time.

- Because household things come in a variety of shapes, sizes, and textures, it might be difficult to consistently grasp objects. Robots need grasp strategy to modify their grasp pose and technique

based on the target. The approach direction and the presence of clutter make it even more difficult for a successful grasp.

- In order to have a successful grasp pose we might need a good 3D representation. In case of MoMa we can not always have high quality 3D representations as it might consume more time visiting different view points. So we might just rely on the go mapping by actively looking at the object during motion.

1.3 Problem Overview

The main problem addressed in the thesis proposal revolves around the challenge of enabling MoMa to efficiently operate in unknown and unstructured environments, such as homes, where they receive vague directives like "get me the green cup." The key difficulty lies in the robot's need to simultaneously explore the environment and locate, map, and grasp objects without prior knowledge of their exact location. The proposed solution is a coupled active perception approach that integrates active exploration and object mapping with manipulation tasks. This method aims to enhance task efficiency by balancing the robot's focus between gathering environmental information and successfully executing object grasping, thereby improving both the success rate and the time required to complete tasks.

1.4 Problem Statement

We propose our approach aimed at improving the capabilities of MoMa through the integration of actively controllable cameras for coupled active perception.

Hypothesis: The underlying hypothesis of our approach posits that this coupled active perception with MoMa yields several key advantages. Firstly, by actively controlling the perception process, MoMa can significantly reduce the time required for mapping the environment. Rather than passively observing the surroundings, MoMa directs its sensors to gather task-relevant information efficiently, thereby expediting the mapping process. As MoMa's knowledge of the environment grows, the balance between mapping and manipulation begins to shift. whenever possible, the robot seeks to achieve both mapping and grasping in tandem, prioritizing actions that contribute to both goals.

MoMa's initial goal is to detect/map the object sufficiently to enable further action. In the early stages of a task, when the environment (or the object) is largely unknown, the robot may concentrate more on object detection and mapping. However, once the object is sufficiently reconstructed, it prioritizes grasping when selecting new poses. We hypothesize that this coupled approach will not only improve the success rate but also the task completion time for mobile manipulation.

Approach: Our approach is divided into four main phases as described in the Figure 1.4. Each phase is explained as below.

- **Exploration with object search:** The process begins with MoMa initiating the exploration and navigation phase. As MoMa moves, it continually checks for the presence of objects. If no object is detected, the process loops back to continue exploration and navigation. When an object is detected, MoMa switches to active gaze control to focus its actively controllable cameras on the object.

- **Object approach with gaze control:** Gaze control ensures that MoMa is looking towards the target object. During the active gaze control two steps are performed. (i) Object detection - MoMa identifies and detects the specific instance of the object from the rest of the scene. (ii) Depth segmentation - MoMa evaluates the depth information to understand the spatial layout and how far the object is. Once the object is detected MoMa proceeds towards object mapping.

- **Targeted object mapping:** In this phase, MoMa still utilizes the gaze control. As it approaches the object, it initiates the targeted volumetric mapping for object reconstruction. Typically, multiple viewpoints are needed to create a sufficiently detailed object reconstruction that is usable for grasp pose detection. In our approach, MoMa samples base locations and observation viewpoints based on the currently detected environment. We use volumetric mapping pipeline whose point cloud output is fed to a grasping detection network that scores grasps on the target object's point cloud sample. As the object is sufficiently reconstructed, grasp candidates with good enough scores are detected. MoMa then selects a pose that favours good reachability and grasping success, thus leading to faster grasp execution.

- **Object Manipulation:** Once a grasp candidate with a high enough score is detected, MoMa attempts to find a collision-free path to execute the grasp. After reaching the base location it gets ready for manipulation by going to pregrasp configuration and then plans for the grasp. If the grasps fail due to the inability to find collision-free paths for the arm, MoMa reattempts the grasp with few more execution. Once a collision-free path is found, the grasp is executed and the task is assumed to be completed.

1.5 Thesis Outline

- Chapter 2 - Literature Review: Conducts a literature review on related works in mobile manipulation, identifying gaps and establishing the context for the current study. In the end we identify the limitations of the current state-of-the-art techniques.

- Chapter 3 - Preliminaries: Introduces our MoMa platform in our work. We also provide necessary background on topics like 3D mapping and grasp synthesis.

- Chapter 4 - Methodology: Explains the CAPerMoMa architecture, detailing each of its components.

- Chapter 5 - Experiments and Results: Describes the design of the experiments conducted, the metrics used for evaluation, and presents the results. The chapter also includes a discussion of the results.

- Chapter 6 - Conclusion: Summarizes the findings, discusses their implications, and suggests potential areas for future work.

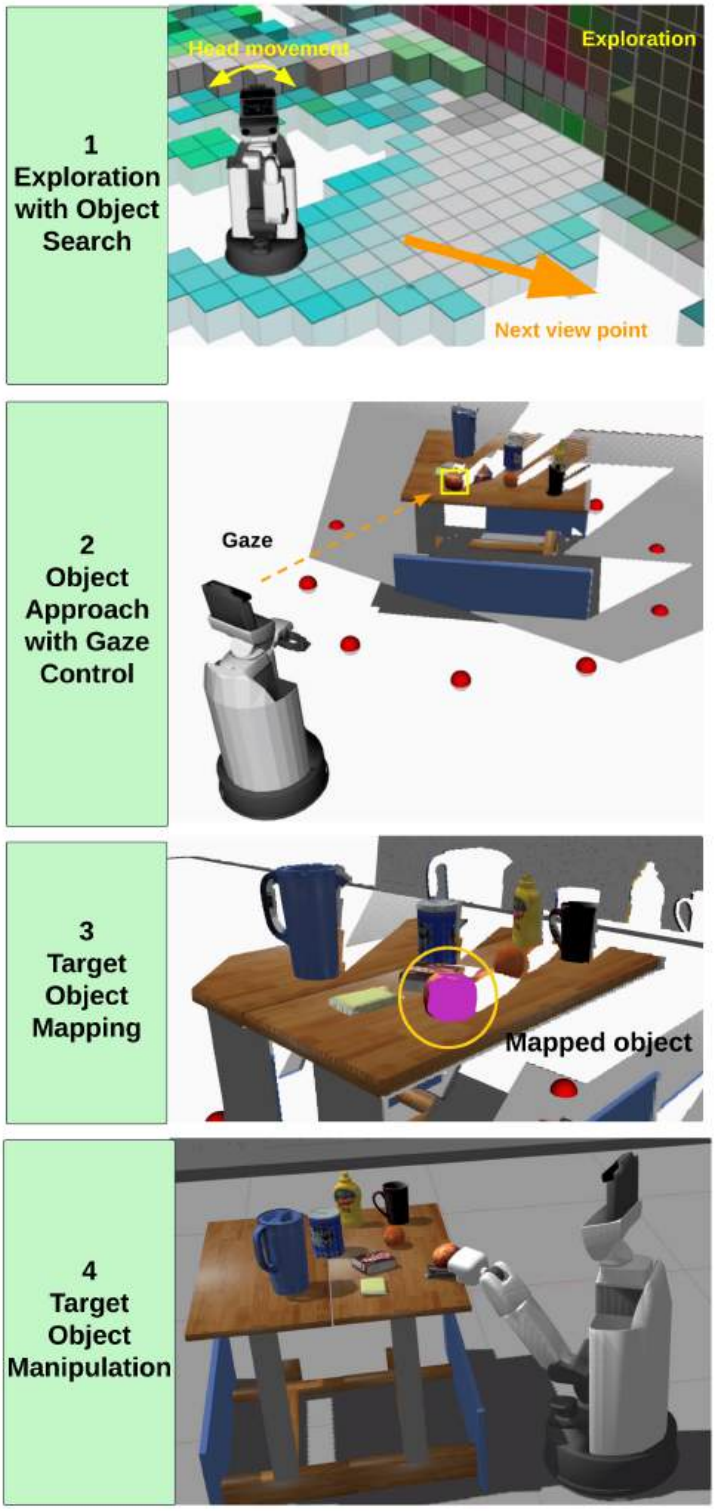


Figure 1.4: Stages of our proposed approach: exploration, approach with gaze control, object mapping, and manipulation.

2

Literature Review

The chapter presents a review of the previous work that forms the basis of some of the key areas of mobile manipulation and perception. In this chapter we will discuss about works related to active perception with manipulators, exploration with mobile manipulators, reachability of robotic systems, tabletop mobile manipulation, and hybrid methods in mobile manipulation. Most topics in each of the individual sections present insights of important contributions that are relevant to us.

2.1 Mobile Manipulation in Unstructured, Dynamic Environments

Early methods from S. Chitta *et al.* [9] present an integrated approach to mobile pick-and-place tasks in unstructured human environments, addressing the challenges posed by novel unknown objects, cluttered workspaces, and noisy sensor data [10]. While the approach significantly improves the robot's ability to interact with unstructured environments, challenges remain in perception under high levels of uncertainty, such as severe occlusions or extremely cluttered environments [9]. Similarly, work by B. Hamner *et al.* [11] showcases the integration of coordinated control, visual and force servoing, and reactive task control strategies to navigate and manipulate in uncertain environments successfully. While the system shows high reliability in controlled conditions, its adaptability to varied and less structured environments remains a concern.

More recent work by B. Burgess-Limerick *et al.* [12] demonstrates reactive manipulation tasks, such as picking and placing objects, without halting the mobile base, thereby improving efficiency and operational gracefulness. F. Reister *et al.* [13] address the challenge of efficiently executing mobile manipulation tasks with integrated cost considerations. Unlike previous methods that primarily focus on either navigation or manipulation independently, this work presents a holistic approach that simultaneously optimizes navigation and manipulation costs. This dual focus significantly reduces the total task completion time. All the previous approaches assume the knowledge of the target's location, requiring distinct stages for navigating to the target, and mapping for object localization at each stage.

2.2 Mobile Manipulation with Exploration

Exploration is an important task in robotics when the environment is fully unknown. This allows MoMa to gather task relevant information like obstacles, objects of interest. Exploration is mainly useful for search and rescue task where the environment is very complex. One similar work by Z. Wang *et al.* [14]

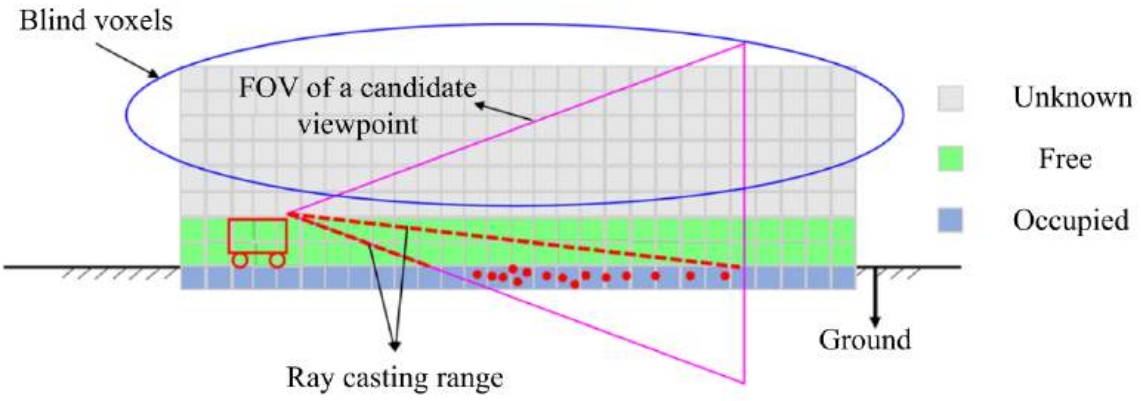


Figure 2.1: Blind voxels problem: Image taken from Z. Wang *et al.* [14].

introduces a 3D exploration methods using their multi layer information maps and reactive planning as compared to traditional methods [15], [16], [17] that use the size of unknown voxels as an indicator for exploration [14]. As their work is mainly focused on outdoor environment they face with the problem of blind voxels as shown in Figure 2.1. The blind voxels presented in this work may also serve as a exploration indicator but the number of such blind voxels are very high thus increasing the compute time [14]. To avoid such problem Z. Wang *et al.* propose hierarchical reactive planning, where they break down the task of 3D exploration into subtasks such as 2D exploration, 3D reconstruction, localization, and collision avoidance into a hierarchy, improving task efficiency by dynamically adjusting them based on real-time environmental perception [14]. Some more details on the previous work is also discussed in section 4.2.2. In the previous work the authors focused on outdoor 3D mapping, but in the work by D. Deng *et al.* [18] they focus on 2D indoor mapping. The authors introduce a new way of measuring and optimizing information gain for 2D exploration. As shown in Figure 2.2 the exploration process begins with the robot in an initial sensor pose. The robot takes depth measurements and creates a probability occupancy map, mapping the environment into known and unknown spaces. The authors come up with "boundariness map" which helps locate frontier cells that separate known and unknown areas. Based on these boundary cells' information, the next best view [19] estimator selects the optimal path for further exploration. This process is repeated until no more boundary cells are left to explore, achieving sufficient coverage. Similarly, inspired by the work by [18], J. Huang *et al.* [20] propose their work on fast exploration for large scale environments. They address computational challenges in autonomous exploration by incorporating a fast preprocessing stage and a path optimization formulation that balances motion distance, information gain, and coverage efficiency [20]. The framework includes a preprocessing stage where basic information such as frontiers, viewpoints, and road-maps is computed at high speed [20]. The system detects frontiers (regions between known and unknown areas) using a 3D grid map, and candidate viewpoints are generated based on the robot's position and the terrain [20]. Instead of traditional methods, visible frontiers are counted to evaluate information gain, greatly reducing computation time [20]. The path planning algorithm selects

the best path by considering A utility function which ensures less revisitations.

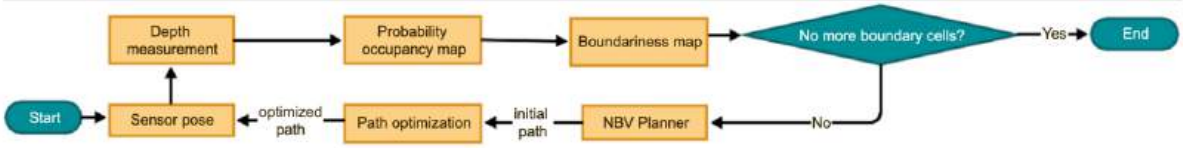


Figure 2.2: Exploration planning strategy: Image taken from D. Deng *et al.* [18].

2.3 Mobile Manipulation with Reachability

It is important to know the limits and capabilities of a robotic platform, reachability is one such property [21]. Reachability analysis helps in planning for manipulation related tasks [21]. We call the representation of the reachable positions of a robotic arm a reachability map [21], introduced for the first time by R. Diankov [22]. In our work, we likewise use forward and inverse kinematics to construct the reachability maps [21]. The main advantage of using the reachability maps is that they can be computed offline as computing the inverse kinematics online is relatively demanding [21]. An early work by M.A. Roa *et al.* [23] introduce the concept of "graspability map" similar to reachability. But the work is only focused on robotic hands. N. Vahrenkamp *et al.* [24] introduced "Reachability Inversion" toward robot base placement based on reachability analysis. They show that a precomputed reachability data can be used as a means of generating potential robot base poses suited for grasping tasks by re-projecting them back to the ground (see left image in Figure 2.3). Following the work from [24] S. Jauhri *et al.* [10] extend it for MoMa based systems. The authors use reinforcement learning (RL) based method to to improve MoMa tasks like reaching and fetching in real-world environments. The authors use reachability map to guide the robot in selecting optimal base poses for manipulation, reducing learning time in RL [10]. The authors validate their approach on the TIAGo++ robot as shown in Figure 2.3.

2.4 Active Perception with Mobile Manipulation

In this section we will discuss works that are related to active perception with mobile manipulators. The works related to this topic are most relevant to our work as they combine different approaches like active perception, object search, planning, and mobile manipulation.

Active perception refers to the process by which a system, such as a robot or an autonomous vehicle, actively selects and controls its sensors to gather information from the environment in a way that maximizes its ability to accomplish a specific task or goal [8]. This involves not just passively receiving sensory data but also actively directing attention and sensing resources to areas of interest [25]. Mobile manipulation refers to the integration of mobility and manipulation capabilities in robotic systems, enabling them to move autonomously in their environment while also manipulating objects or interacting with their surroundings [26]. One of the key advantages of active perception lies in its ability to enhance the robot's situational awareness and decision-making capabilities [27]. Conventional passive perception systems, which observe the environment without actively shaping their sensing processes, may struggle to effectively gather task-relevant information amidst this uncertainty [8].

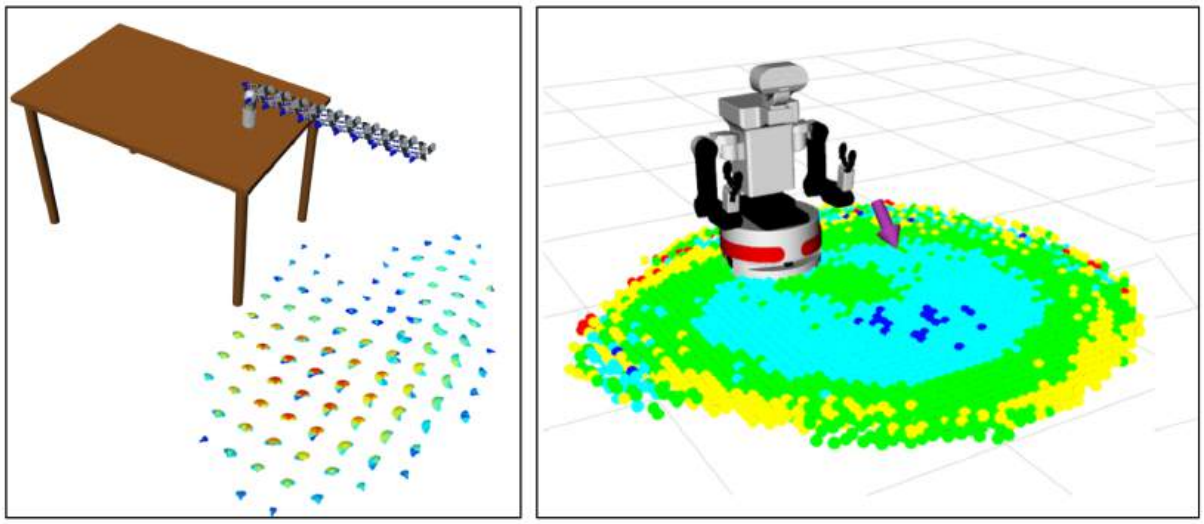


Figure 2.3: Left - base placement map: Image taken from N. Vahrenkamp *et al.* [24], Right - inverse reachability map in a MoMa scenario: Image taken from S. Jauhri *et al.* [10].

Recent work by S. Jauhri *et al.* [4] introduces a new approach that combines exploration for visual information gain and exploitation of task-specific information like grasp reachability and executability, without prior scene knowledge. The authors come up with path-wise utility calculations over robot poses, a method for generating motion objectives that balance between maximizing visual information gain and exploiting task-oriented outcomes such as grasp success, using a unique blend of path-wise utilities [4]. They also employ a receding horizon control approach that enables the robot to adaptively and reactively generate motions based on newly acquired scene data, demonstrating a significant advancement in mobile manipulator control [4]. However, this approach presumes prior knowledge of the target object's approximate position.

Below we explain the ActPerMoMa pipeline in Figure 2.4:

- First, the system gets an initial estimate of the position of the target object or area. It gathers the visual information from the camera of the robot RGBD data, and records the current pose of the robot position and orientation.
- The resultant visual data is used to create a 3D map via the "Truncated Signed Distance Field" (TSDF) [4] of the scene that helps the robot create an understanding of how the environment is structured. From this map data, a base occupancy map is generated, which later is used for collision avoidance.
- From this 3D map, the robot detects a set of possible 6D grasps for the object. On each camera posture, the system pre-estimates how much new information it would get, i.e., "Information Gain", which helps the robot plan an optimal next move [4].
- First, the robot creates a list of possible paths. The system assesses each path based on two

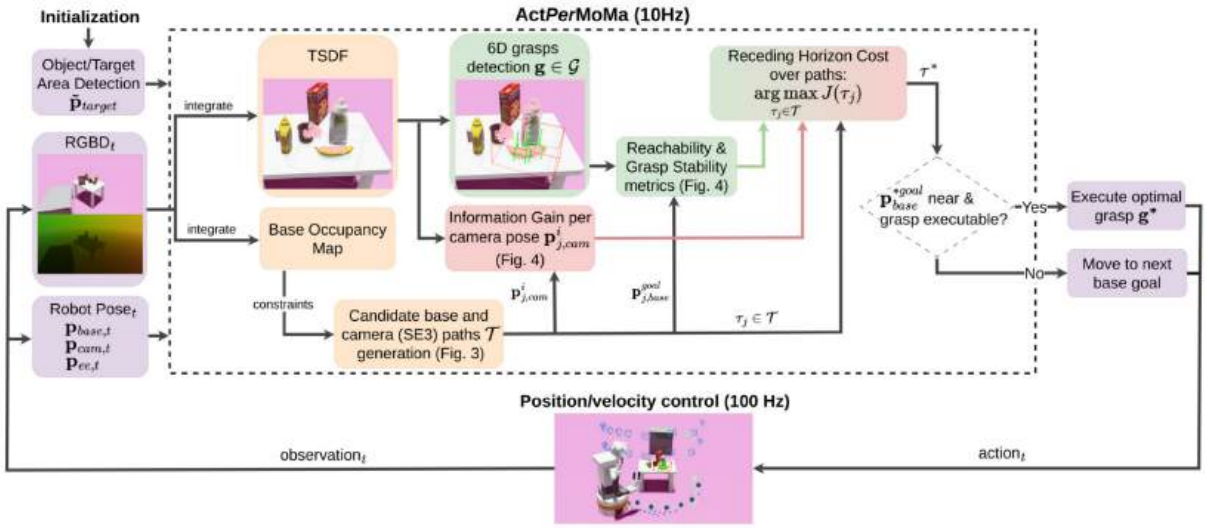


Figure 2.4: ActPerMoMa pipeline: Image taken from S. Jauhri *et al.* [4].

components: one considering how much information is gained and the other, how easily one can reach and execute the detected grasps from those base positions [4].

- The robot chooses the optimal path by using a method known as receding horizon control in such a way that it optimally balances information gathering with the reachability of the grasp. It uses cost function considering the reachability of the grasp such that this balance is achieved [4].
- If the selected grasp is nearby and executable, the robot performs the grasp; otherwise, it proceeds to the next base goal and re-plans.

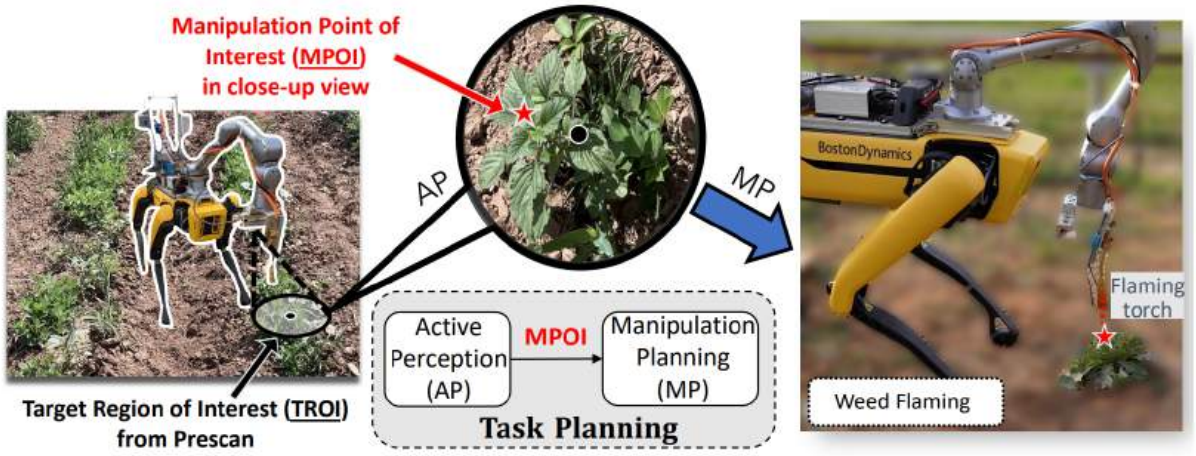


Figure 2.5: CAPM pipeline: Image taken from S. Xie *et al.* [28].

Another similar work by S. Xie *et al.* [28] introduce a "Coupled Active Perception and Manipulation"

(CAPM) [28] planning framework as shown in Figure 2.5. The authors show how to perceive and manipulate in precise tasks for mobile manipulators in complex environments, such as agricultural fields. The work also focues on how to get near view points for a successful manipulation [28]. As shown in Figure 2.5 the system first does an initial prescan at a distance to roughly locate the target, here a weed [28]. This prescan gives the robot an initial rough idea of where to navigate, but it is too random to provide details for task execution. The robot moves towards the Target Region of Interest (TROI) and applies active perception (AP) to capture a close-up view of the target [28]. Once a proper close-up view is captured, a good point of manipulation is determined, called MPOI (see Figure 2.5). This is the exact point where the robot should act, like using a flaming torch to burn the weed [28]. The information obtained from the AP step is used to plan the next phase: Manipulation Planning (MP) [28]. If the MPOI is reachable, the robot executes the task, such as flaming the weed. A robot-mounted arm with a flaming torch precisely targets the center of the weed for removal [28].

One of the early works from A.D. Mezei *et al.* [29] using classical methods show how active perception techniques can be applied to object manipulation tasks, particularly in industrial environments. In this work A 3D depth sensor mounted on a robotic arm performs the data acquisition. The ICP algorithm [30] combines point clouds from multiple views to create a consistent model of the environment. After scene registration, the system divides the environment into object primitives, focusing on the target object [29]. These are then converted into a graph-based model used for object detection and manipulation [29].

2.5 Summary of Previous Work

Finally, Table 2.1 shows the different characteristics considered in our work as compared to other most relevant approaches discussed so far.

Approach	**Active Exploration**	**Object Mapping**	**Reactive Planning**	**Manipulability**
Z. Wang *et al.* [14]	✓	✗	✓	✗
D. Deng *et al.* [18]	✓	✗	✓	✗
J. Huang *et al.* [20]	✓	✗	✓	✗
N. Vahrenkamp *et al.* [24]	✗	✗	✓	✓
S. Jauhri *et al.* [10]	✗	✗	✓	✓
S. Xie *et al.* [28]	✗	✗	✓	✓
A.D. Mezei *et al.* [29]	✗	✗	✓	✓
S. Chitta *et al.* [9]	✗	✓	✓	✓
B. Hamner *et al.* [11]	✗	✗	✓	✗
B. B. Limerick *et al.* [12]	✗	✗	✓	✗
F. Reister *et al.* [13]	✗	✗	✓	✓
S. Jauhri *et al.* [4]	✗	✓	✓	✓
Ours	✓	✓	✓	✓

Table 2.1: Comparison of different approaches in active exploration, object mapping, reactive planning, and manipulability.

2.6 Limitations of Previous Work

The below are the most relevant limitations of the works discussed so far.

- Many works focus on only a subset of characteristics shown in Table 2.1, leading to gaps in their applicability to more complex scenarios. Thus, these previous works fall short in integrating all the key components—exploration, mapping, and manipulation—necessary for a robust mobile manipulation system capable of functioning autonomously.
  - Our Solution: We propose an integrated CAPerMoMa system combining all the characteristics shown in Table 2.1. Our CAPerMoMa system addresses this gap by combining these elements and also keeping execution time in mind. Refer Section 4.1 for more details.
- Work by S. Jauhri *et al.* [4] try to address the problem of mobile manipulation using active perception. The authors active perception is based on sampling head poses directed toward the target object and moreover they assume the target's location is already known prior, which in general is not always available.
  - Our Solution: We propose a continuous gaze control strategy that is coupled with the mobile exploration and manipulation. Refer Section 4.2.4 to get more details. We also include exploration in our system.
- Work by S. Chitta *et al.* [9] show mobile manipulation in uninstructed environments. They use motion planing for arm to check if the target pose is reachable or not in a online manner (i.e. forward and inverse kinematics), which is computationally very demanding thus increasing the total execution time.
  - Our Solution: To solve the problem of computing forward and inverse kinematics every time, we adapt reachability and inverse reachability maps. Inverse reachability maps give many candidate poses for base placement, we filter them out using robot footprint. Refer to Section 4.2.8 for more details.
- Most of the methods discussed so far only use simple or specific scenarios to evaluate their methodologies. Mobile manipulation in indoor scenarios requires navigating from one room to other hence evaluating the algorithms in complex search spaces is required.
  - Our Solution: In order to evaluate our system well, we propose four indoor scenarios with increasing search space and complexity. See Section 5.4 for more deatils.

3

Preliminaries

In this chapter, we introduce our MoMa platform which is the Toyota Human Support Robot (HSR) that plays a central role in our work. This chapter also provides an overview of the technical topics like mapping and grasping that we utilize, which is important for understanding its integration into our experimental framework. All of our work presented in this report is performed in the simulation environment.

3.1 MoMa: Toyota Human Support Robot (HSR)

Before delving into the CAPerMoMa system, it is important to discuss the MoMa utilized in this work. The concept of MoMa was briefly introduced in Chapter 1, and here, we provide a more detailed description of our chosen platform. For this project, we have selected the Toyota Human Support Robot (HSR), an assistive robot developed by Toyota Motor Corporation (TMC) [31] [32].

As illustrated in Figure 3.1, the HSR features a cylindrical form factor, equipped with a mobile differential drive base and a 2D LiDAR sensor for navigation. The robot also includes a 5-degree-of-freedom (DOF) arm with a two-finger gripper designed for object manipulation. Additionally, a depth camera (RGB-D sensor) is mounted on a movable head that supports both pan and tilt movements.

Figure 3.1: Toyota HSR: MoMa platform used in this work.

Some of the key technical specifications relevant to our work are as follows:

- Model: HSR type-B
- Base Footprint: 430 mm
- Maximum Gripper Opening: 135 mm
- End-Effector (EEF) link: hand_palm_link

3.2 3D Mapping

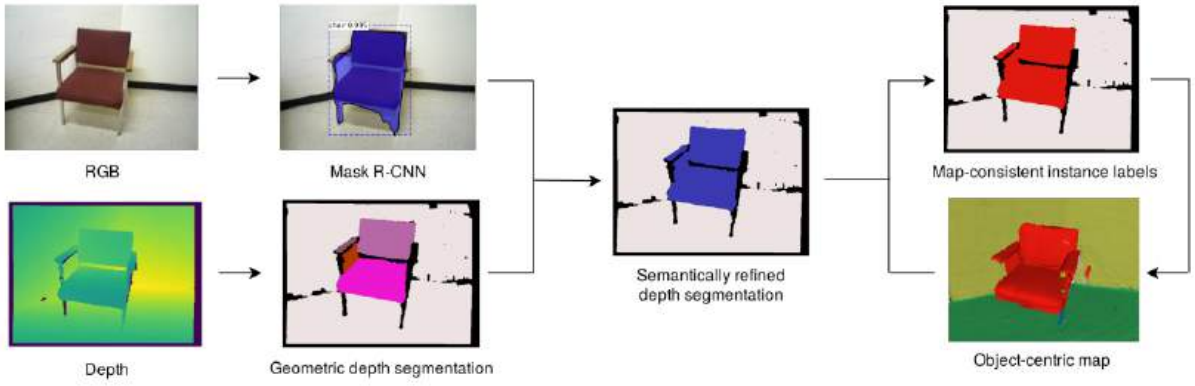


Figure 3.2: Voxblox++ [33] pipeline adapted in our work. The original pipeline used Mask R-CNN for instance segmentation, while we substitute it with YOLOv8 mask for faster segmentation. The process involves RGB and depth inputs, followed by depth segmentation and semantic refinement, leading to map-consistent instance labels and an object-centric map. (Image from [33]).

In our work, we employ the Voxblox++ [33] framework for volumetric 3D mapping. A brief explanation of the process illustrated in Figure 3.2 is given below:

- Detecting Objects: For every new frame, the system analyzes the RGB image using Mask R-CNN, which finds objects and creates masks for each object with labels [33].
- Breaking Down Depth Information: At the same time, the depth image is broken down into smaller 3D chunks (convex hull of segments), grouping parts of the objects that have a similar shape [33].
- Matching the Masks with Depth: The masks from the RGB image help label the 3D depth chunks and combine parts of the same object together, even if it is not a simple shape [33].
- Keeping Track of Objects: The system then matches the new 3D chunks it sees in the current frame with the global map of objects that has been built so far. This way, each object gets a consistent label across all frames [33].
- Updating the Map: Finally, the new details from the current frame (both shape and labels) are added to the global map, making it more detailed and accurate with every frame [33].

Algorithm 1 Grasp Pose Detection

Input: a viewpoint cloud, $\mathbb{C}$; a region of interest, $\mathcal{R}$; a hand, Θ; a positive integer, N
Output: a set of 6-DOF grasp candidates, $H \subset \mathcal{R}$

1: $\mathbb{C}' = \text{PreprocessCloud}(\mathbb{C})$
2: $\mathcal{R} = \text{GetROI}(\mathbb{C}')$
3: $S = \text{Sample}(\mathbb{C}', \mathcal{R}, \Theta, N)$
4: $I = \text{Encode}(S, \mathbb{C}', \mathcal{R}, \Theta)$
5: $H = \text{Score}(I)$
6: $g = \text{SelectGrasp}(S, H)$

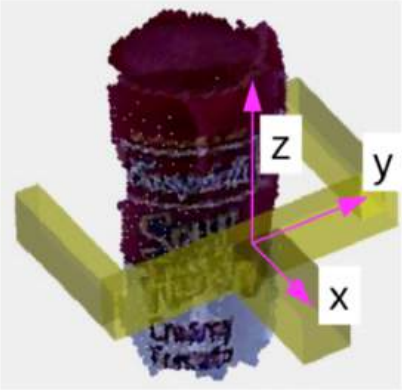


Figure 3.3: The left image is from the Grasp Pose Detection (GPD) by A. Ten Pas *et al.* [34], which identifies 6-DOF grasp candidates from a viewpoint cloud. The right image shows a sample grasp applied to an object, indicating the grasp orientation in the 3D space (X, Y, Z axes) by A. Ten Pas *et al.* [34].

3.3 Grasp Detector

In our work, we utilize the Grasp Pose Detector (GPD) [34] for grasp synthesis.
The working principle of the GPD is briefly explained below [34] and illustrated in Figure 3.3.

- Point Cloud Preprocessing: Cleaning of raw point cloud data through the removal of noise, outliers, and other errors reduces a bit of complexity in the data.
- Identify Region of Interest(ROI): A specific area or region of interest is selected in which the grasp will take place. Taking the target object's point cloud alone as an input to the GPD serves as a ROI. This step reduces the target space, even though it may just have a single object, it can include several objects or just an approximate location [34].
- Sample Grasp Candidates: Thousands of possible grasp positions are obtained from the obtained ROI. Each of these is defined by a 6-DOF hand pose. These are potential candidates for grasping the object [34].
- Encode Candidates: The encoding of each grasp candidate into a neural network-compatible representation is usually performed by projecting the data from candidates into the image multichannel format [34].
- Scoring the Grasp Candidates: Using a Convolutional Neural Network, a score is assigned to each grasp candidate based on how probable it is that the robot would be able to grasp the object with that particular grasp [34].
- Choose Best Grasp: The grasp with the maximum score and/or other relevant considerations is determined and chosen for execution. An example grasp proposal is given in the Figure 3.3

4

Methodology

This chapter gives the details of methodologies adopted to develop and integrate the CAPerMoMa system. We provide an overview of the CAPerMoMa system pipeline, highlighting its main functions and the flow of operations. Finally, each part of the pipeline is looked into in more detail, indicating its functions and interactions to achieve the system's purpose.

4.1 CAPerMoMa: System Overview

We integrate the CAPerMoMa framework in Robot Operating System (ROS1) [35], applying state machines and nodes. The structure of this pipeline is depicted in Figure 4.1; green serial nodes imply that key phases are executed sequentially, and these states dictate the general system flow, ensuring that a given step is completed before the next starts. In contrast, the yellow parallel nodes allow the system to execute several operations simultaneously for each phase. This parallelism increases the execution efficiency of the pipeline, through which, for example, the tasks of sensor data processing, object detection, and navigation planning do not block each other in their execution steps, reducing the overall execution time.

The execution of the state machine is as follows:

- **Exploration with Object Search:**
  - The robot begins by searching for objects using head movements, which is done by an entropy-based information gain approach. This method directs the head towards regions with the highest entropy, maximizing the likelihood of detecting objects. Further details on this technique are provided in Section 4.2.1.
  - In parallel, the YOLOv8 [36] object detection and next viewpoint planning for the mobile base are also running as shown in Figure 4.1. The robot can systematically explore the unknown environment by using the head movements with viewpoint planning. The system transitions to the next state as soon as the target object is detected.
- **Object Approach with Gaze Control:**
  - Upon detecting the target object, the robot locks its focus on the object using gaze control. At this point, head movement and base viewpoint planning are deactivated, while the YOLOv8

object detection node continues to run. Based on the YOLOv8 detection output, the system samples 10 navigation goals around the object, with the sampling radius manually set according to the table size. Once the sample points are generated, the robot selects and approaches the closest reachable navigation goal.

- **Target Object Mapping:**
  - This stage is closely coupled with the previous one. In our main implementation, gaze control and object mapping are combined into a single state, but here we separate them for simplicity. As the robot moves toward the sampled goal in the previous stage, the mapping process begins simultaneously.
  - Upon nearing the approach point, the robot maintains gaze control to further refine its focus on the object. During this time, it performs volumetric mapping to generate a 3D map of the object. The YOLOv8 detection node remains active, and its output is used for depth segmentation, which is then used for integrating the point cloud for the 3D mapping. More detailed information about the mapping process can be found in Section 4.2.6.
- **Target Object Manipulation:**
  - Once the 3D map is generated from the previous stage, it is immediately processed by the grasp pose estimation module. For grasp synthesis, we utilize the Grasp Pose Detector (GPD) [34], which identifies potential grasp poses based on the volumetric map.
  - As the grasp poses are generated, we calculate multiple base poses using the inverse reachability map. These base poses are filtered by checking for reachability, ensuring they are free of obstacles and within the robot's footprint. From the filtered options, we select the best base pose and its corresponding grasp pose.
  - Finally, the robot moves to the selected base pose and executes the grasp. For trajectory generation, we employ the MoveIt package in conjunction with the HSR's API. More details on the grasp execution and trajectory planning are provided in Section 4.2.9.

4.2 CAPerMoMa: Components Description

In this section, we provide a detailed discussion of each component within the CAPerMoMa system. Where necessary, we provide a brief background about the topic and explain about how we use the related concept/topic in our pipeline.

4.2.1 MoMa Head Movement

In this work, MoMa's head movements are based on entropy-based information gain. Here, the robot explores areas of the environment that are uncertain or have high information content. In this approach, we use the concept of entropy adapted from information theory [37] to quantify the uncertainty or randomness

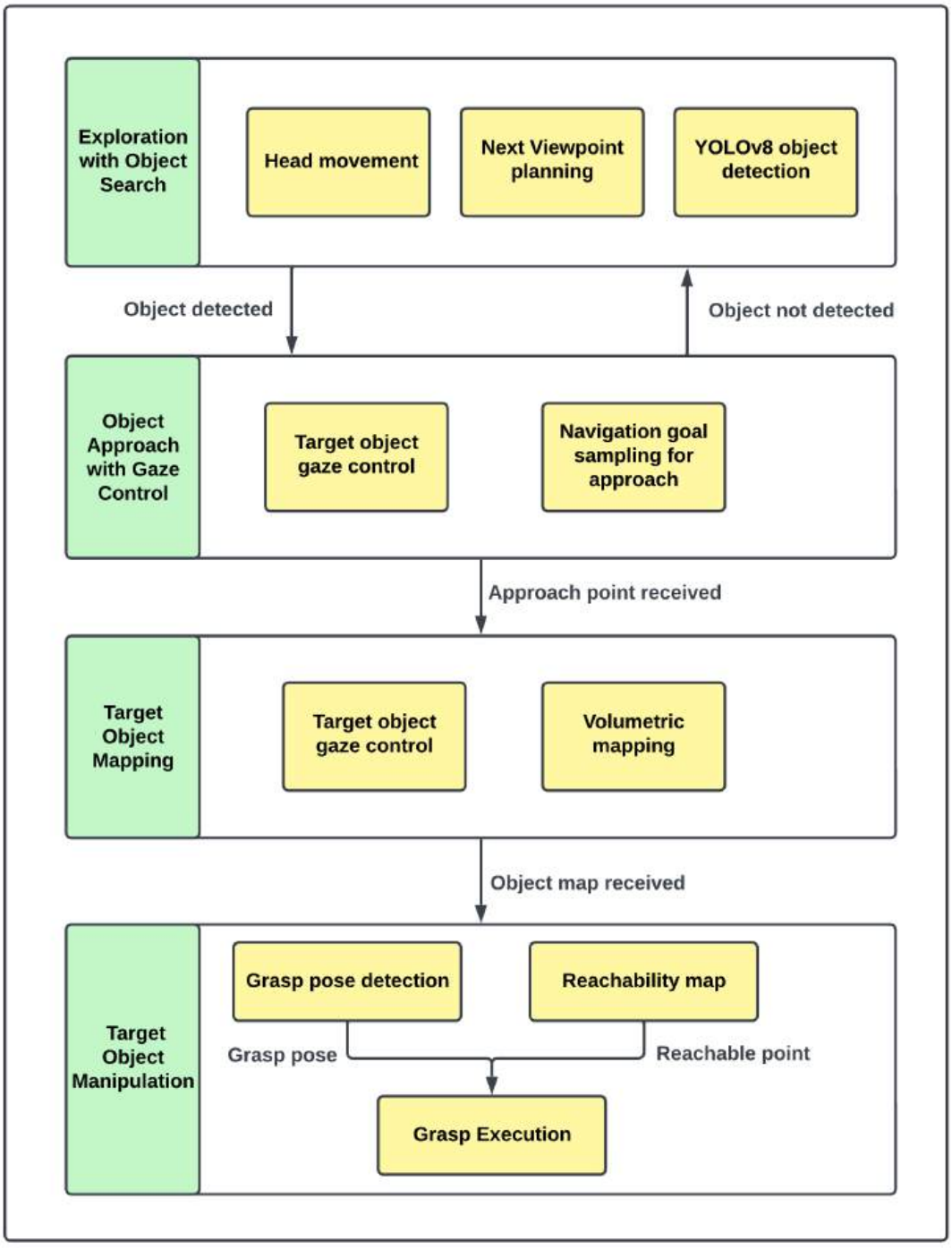


Figure 4.1: The diagram represents the CAPerMoMa system pipeline. The main stages, shown in green nodes, are executed in series and include exploration with object search, object approach with gaze control, target object mapping, and target object manipulation. Within each stage, the tasks represented by yellow nodes are performed in parallel, such as head movement, viewpoint planning, object detection, and volumetric mapping. The process transitions from exploration to manipulation, leading to successful grasp execution.

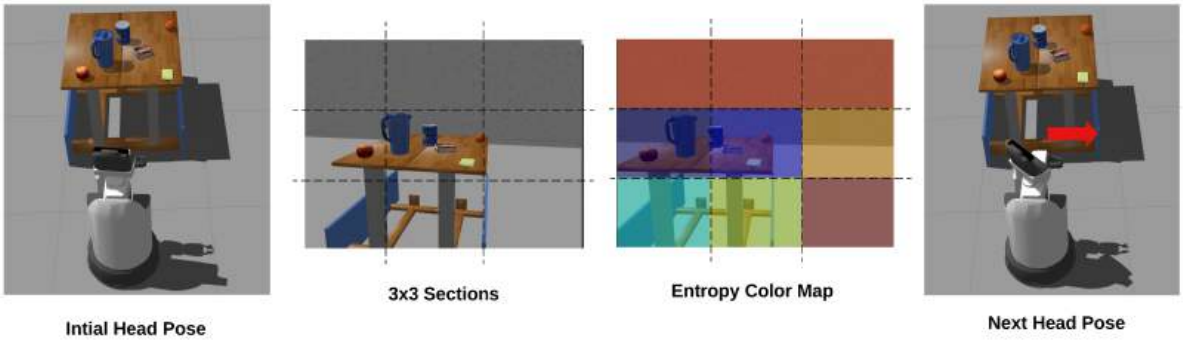


Figure 4.2: The robot's head movement is guided by entropy values in the environment. The scene is divided into a 3x3 grid, with higher entropy (red areas and blue for low values) indicating more uncertain or unexplored sections, prompting the robot to shift its gaze towards these regions. The initial head pose and the next head pose after entropy calculation are shown.

in the image data. This method is adapted to help MoMa during the exploration process also to plan next view point. We use information gain formulation instead of random movement to make MoMa focus where there is more information content (in our case walls). The robot will then move its head toward regions with higher entropy to maximize the information gain [37] (see Figure 4.2).

For a grayscale image, entropy H can be calculated using the formula [37]:

$$H = -\sum_{i=0}^{N} p_i \log_2(p_i) \tag{4.1}$$

Where:

- N is the number of unique pixel intensity values (for an 8-bit image, N=256).
- p_i is the probability calculated using histogram of pixel intensities and normalized.

The goal is to move the robot's head towards regions that maximize information gain. Information gain $\triangle H$ is defined as the reduction in entropy before and after observing a region:

$$\triangle H = H_{before} - H_{after} \tag{4.2}$$

In practice, this means identifying areas of the image with high entropy and focusing the robot's head on these regions.

Pan and Tilt Angles Calculation:

- Capture the current image from MoMa's head camera.
- Divide the image into smaller regions (i.e. a grid of 3x3 as shown in Figure 4.2). Each region will represent a potential area where the robot could direct its head.
- For each region, calculate the entropy using the formula 4.1.

- Once the entropy for each region is calculated, identify the region with the highest entropy. This region is where the robot's head should move to maximize information gain.
- Once the region with the highest entropy is identified, it must be mapped to the corresponding pan and tilt angles for the robot's head movement. The robot's head can rotate in two degrees of freedom:
  - Pan: Horizontal movement, denoted by θ_{pan} ranges from $-\theta_{pan}^{max}$ to θ_{pan}^{max}
  - Tilt: Vertical movement, denoted by θ_{tilt} ranges from $-\theta_{tilt}^{max}$ to θ_{tilt}^{max}
- The grid divides the viewing range into $g \times g$. These angles are computed using linear interpolation and finally sent to the HSR's head trajectory controller during the execution:

$$\theta_{pan} = \theta_{pan}^{min} + \frac{l}{g-1}(\theta_{pan}^{max} - \theta_{pan}^{min}) \tag{4.3}$$

$$\theta_{tilt} = \theta_{tilt}^{min} + \frac{k}{g-1}(\theta_{tilt}^{max} - \theta_{tilt}^{min}) \tag{4.4}$$

Where l, k are the row and column of the grid, respectively. g represents the grid size. The parameters $\theta_{\text{pan}}^{\text{min}}$ and $\theta_{\text{pan}}^{\text{max}}$ denote the minimum and maximum pan angles, typically ranging from -1.57 to 1.57 radians. The parameters $\theta_{\text{tilt}}^{\text{min}}$ and $\theta_{\text{tilt}}^{\text{max}}$ denote the minimum and maximum tilt angles, typically ranging from -0.17 to 0.17 radians.

4.2.2 Mobile Exploration

During the exploration phase, MoMa not only explores by moving its head but also navigates to different locations to search for the target object. In this work, we assume our MoMa operates solely in domestic home/office-like environments where the map is closed (as opposed to an open map with an increasing search space). We have already introduced works related to exploration in Section 2.2, and in our work, we adapt Open3DExplorer by Z. Wang *et al.* [14] for our basic mobile exploration task. In Section 2.2, we discussed different exploration methods based on unknown voxels size [14], but we utilize Open3DExplorer as it is based on hierarchical reactive planning [14]. The hierarchical reactive planning method simplifies the robot's exploration by dividing the complex 3D task into smaller, more feasible steps such as 2D exploration to cover space completely, creating a 3D map of the place, and localization and obstacles avoidance of the robot [14]. The robot initiates the exploring space with a 2D map for maximum space coverage and simultaneously verifies whether the important tasks are being carried out rightly—like adjusting its location and avoiding collisions (see left image in Figure 4.3). It does not explore every unknown spot in 3D but focuses first on the 2D map, and then processes other tasks (like path correction) as per the need [14]. It makes the exploration faster and more effective by maintaining dynamic priority of tasks. The right image in Figure 4.3 shows how we adapt the next best view from Open3DExplorer in our work (i.e. we use only the final exploration view). In our work we also assume that we have a 2D map for navigation utilizing the ROS navigation stack [38].

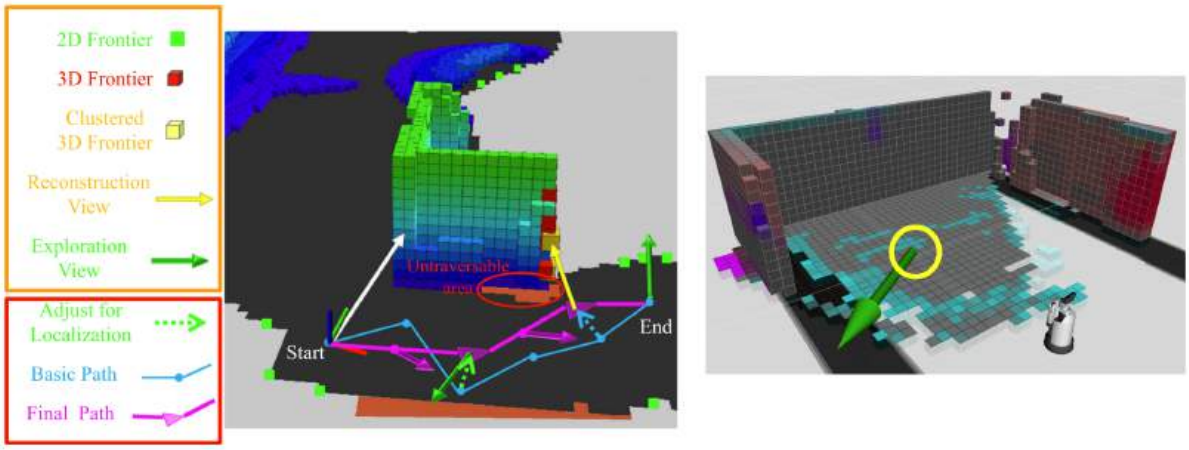


Figure 4.3: The left image, from Open3DExplorer, demonstrates hierarchical reactive planning for viewpoint exploration [14], incorporating various frontier types and path adjustments for localization (the items in red box are not relevant for us). The right image illustrates how this approach is adapted in our specific scenarios, showing MoMa interacting with the environment using a similar planning strategy

4.2.3 Object Detection

We use continuous object detection at various stages in our pipeline, leveraging the YOLOv8 model trained on COCO objects [39]. Instead of using a model tailored to a specific scenario, we employ the publicly available, generic YOLOv8 model [36] for the object detection task. Below we highlight the role of the object detector in our pipeline.

- The object detection node operates in parallel throughout the pipeline and is controlled via ROS service calls within the state machine.
- Continuous object detection starts in the exploration state and runs alongside the next viewpoint planning and head motion nodes, as depicted in Figure 4.1.
- The continuous nature of object detection is important as YOLOv8's output triggers the transition from the exploration state to the mapping state (see Figure 4.1).
- In the mapping state, the detection output is utilized to maintain gaze control, supporting the active perception system.
- Figure 4.4 shows a sample output from the YOLOv8 detector during exploration, where detecting the target object (e.g., an apple) triggers the transition to gaze control for mapping, which is discussed in the following Section 4.2.4.

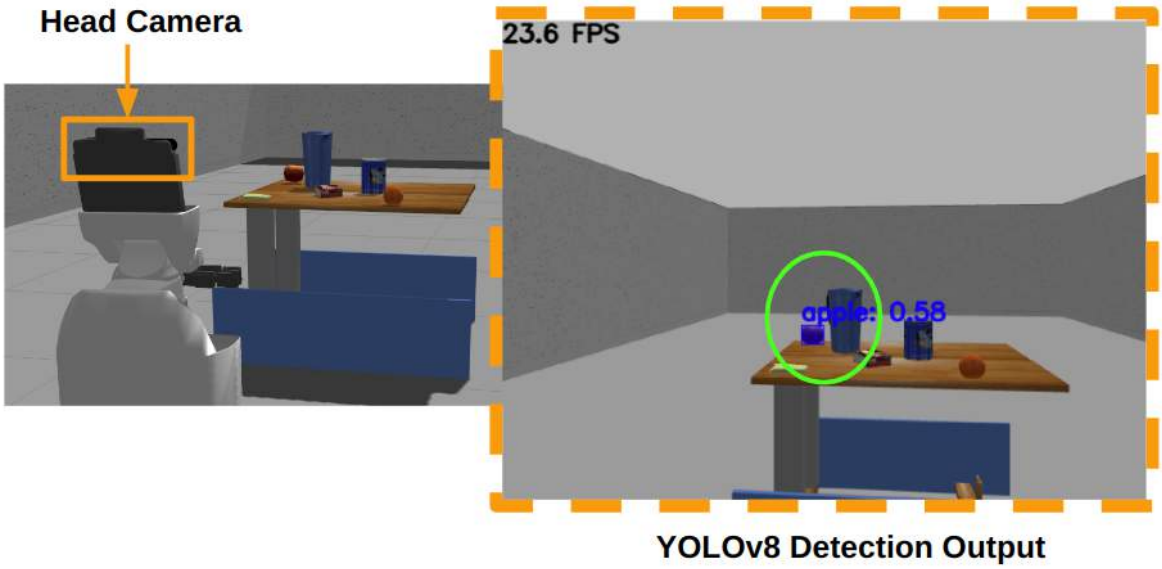


Figure 4.4: MoMa's Head-Mounted Camera View and online YOLOv8 Object Detection: MoMa's head camera captures a tabletop scene, while the YOLOv8 model identifies objects (e.g., apple) with confidence scores of 0.58, providing object localization and recognition.

4.2.4 MoMa Gaze Control

In our framework, gaze control is an important aspect in helping perception pipeline, between MoMa and its environment. It allows the robot to continuously adjust its view, maintain visual focus, and gather information about the object. This is particularly important for dynamic tasks like grasping, where the robot needs precise and up-to-date information about the object's position and orientation for mapping. Our gaze controller is depicted in Figure 4.5. The working of gaze controller is explained below:

- During the exploration phase, as soon as the target object is detected, MoMa transitions to the mapping phase. At this stage the YOLOv8 detection is still running.
- After an object is detected, the system computes the midpoint of the bounding box surrounding the object in the image. This midpoint represents the 2D position of the object in the image plane (see Figure 4.5).
- Along with the 2D RGB image used for detection, the system also has access to a depth image from the camera. This depth image provides distance information for each pixel.
- The camera's intrinsic parameters (focal length, principal point) are required to convert 2D pixel coordinates into 3D coordinates.

- Using the depth value at this midpoint and the camera's intrinsic parameters, the system converts the 2D position into a 3D pose (X, Y, Z) in the world frame. This process is known as "Pose 2D to 3D Conversion" (see Figure 4.5).
- The camera intrinsic matrix K relates pixel coordinates (u, v) to the 3D point (x, y, z) in the camera coordinate system. The intrinsic matrix is given by:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

  Where:

  - f_x and f_y are the focal lengths in the x and y directions.
  - c_x and c_y are the coordinates of the optical center (principal point).
  - The matrix K is extracted from the camera info topic in ROS.

- Given the 2D pixel coordinates (u, v) and the depth value z from the depth image, we can compute the corresponding 3D point (x, y, z) in the camera frame.

  The relationship is derived from the camera projection equation:

$$x = \frac{(u - c_x).z}{f_x} \tag{4.6}$$

$$y = \frac{(v - c_y).z}{f_y} \tag{4.7}$$

$$z = z \tag{4.8}$$

  where:

  - u and v are the pixel coordinates of the bounding box midpoint.
  - z is the depth value from the depth image at the pixel (u, v).
  - c_x, c_y, f_x, f_y are intrinsic parameters from the camera matrix K.

- Once the 3D point in the camera frame is obtained, it needs to be transformed into the world coordinate frame (denoted as "map" in our case).
- The 3D position of the object is then used to control the robot's gaze. The HSR's propriety viewpoint controller API is used to get the head's pan and tilt angles to center the detected object in the camera's field of view.
- This results in the robot "gazing" at the detected object (the apple in this case), which helps in tasks like object mapping or subsequent manipulation which we will discuss in the next sections.

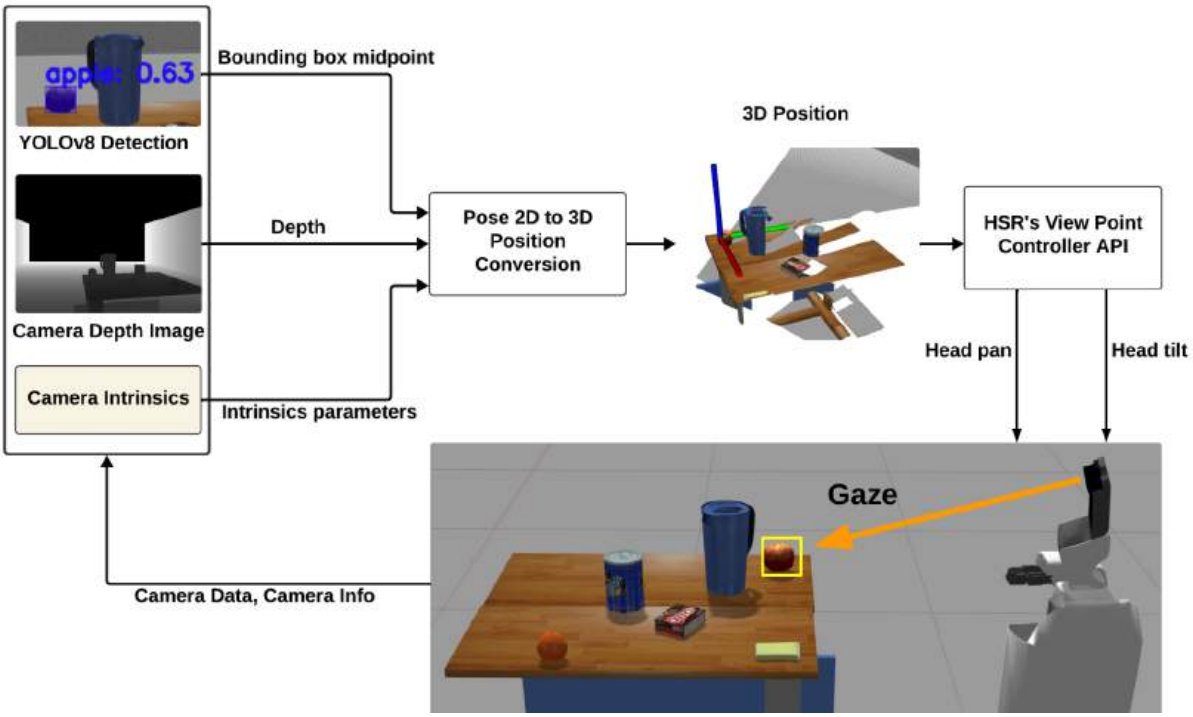


Figure 4.5: Gaze controller using continuous object detection: The system converts 2D bounding box data from YOLOv8 object detection into 3D coordinates using depth information and camera intrinsics. This information guides MoMa's head movements (pan and tilt) via the HSR's view point controller API, allowing it to focus (gaze) on the detected object in online.

4.2.5 Approach Sampling

Once the target object is detected, gaze is activated to focus on the object. The next step involves moving the robot closer to the object for grasping, as it is far from its current position. Thus we do approach sampling as described below:

- Navigation goals are sampled to approach the target object for successful manipulation.
- The environment is assumed to have a 2D map, typical for indoor/domestic scenarios.
- As shown in Figure 4.6, after object detection, **N** navigation goals (in this case **N=10**) are sampled, represented by red markers on the floor.
- The navigation goals are distributed in a circular manner around the object, with a radius based on the table's size.
- The orientation of each navigation goal is set to point towards the target object for an optimal approach.

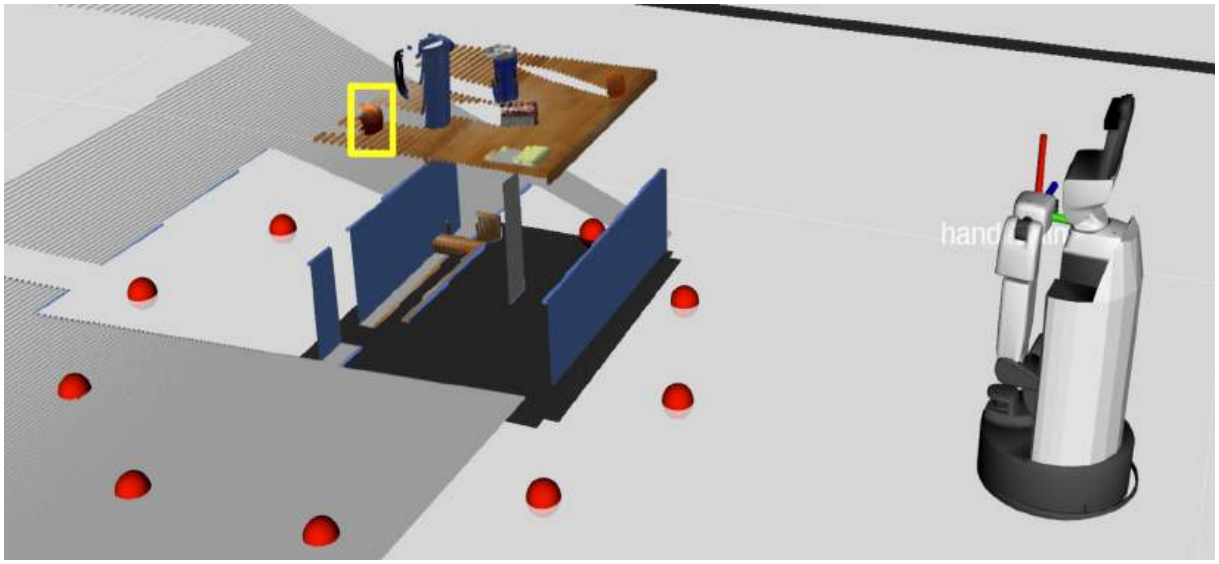


Figure 4.6: Approach Sampling: The red goals represent sampled approach positions for MoMa, while the target object (highlighted in the yellow box) is located on the table. MoMa uses one of these samples to determine optimal positioning for grasping or interacting with the object based on shortest distance.

- This approach sampling step is used solely for moving the robot towards the target object. As soon as MoMa begins its movement towards one of the navigation goals, it transitions to the mapping state.
- Volumetric mapping is initiated simultaneously with the movement, starting the integration of the point cloud for object mapping. More details about volumetric mapping will be covered in the next Section 4.2.6.

4.2.6 Volumetric Mapping

Once an approach goal is received, the robot fully transitions into the mapping state. As the name suggests, the primary objective of this state is to generate a 3D representation of the target object, referred to as a volumetric map. Volumetric mapping is essential for obtaining an accurate surface representation of the target object by taking its geometry into account [33]. In this process, the target object's map is incrementally integrated to form a dense 3D model [33], which can then be utilized for grasp pose estimation. In our work, we employ the Voxblox++ [33] framework for volumetric mapping.

As discussed in Section 4.2.5, as soon as MoMa starts moving toward the approach point, we initiate the mapping process. The mapping process is triggered by querying the instance ID of the target object in the scene via a ROS service call. Once we obtain the instance ID of the target object, we begin the integration process to generate a dense representation of our global map. Figure 4.7 shows the final result of the volumetric mapping during the run. The reconstructed dense 3D map of the target object is then

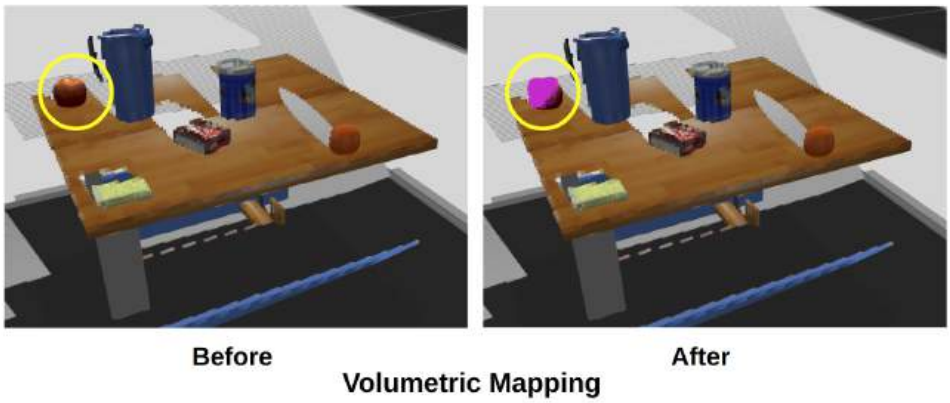


Figure 4.7: Before and after volumetric mapping using the Voxblox++ pipeline. The left image shows the scene prior to mapping, and the right image demonstrates the mapped surface portion in pink after volumetric integration, highlighting the mapped target object (circled).

sent as a point cloud (along with the complete point cloud of the scene) to a grasp pose estimator, which will be discussed in Section 4.2.7.

4.2.7 Grasp Synthesis

So far, we have completed the exploration and mapping stages. We have correctly identified our target object and performed the 3D mapping. Our next step is to transition to the manipulation state, where we identify the best graspable pose and determine the base placement. Before discussing the base placement, this section focuses on how we obtain the grasp pose. In our work, we utilize the Grasp Pose Detector (GPD) [34] for grasp synthesis. As mentioned in the previous section, once we have the map of our target object, we use the reconstructed map and the overall scene point cloud as inputs to the GPD. The GPD then provides multiple grasp proposals with associated scores.

From Figure 4.8, we can observe the final output of GPD's grasp proposals. Although GPD provides several possibilities, we select the top 10 proposals and perform a reachability analysis for each graspable pose to determine the best reachable and graspable pose. More details about reachability will be discussed in the next Section 4.2.8. At this stage, the GPD and the reachability ROS nodes are running in parallel. As soon as the 3D map is received, the GPD computation is triggered. In Figure 4.8, we can see that the robot is relatively close to the object, but in principle, using GPD, we can detect reasonably good grasps even at a distance, provided the object is sufficiently mapped. In Figure 3.3, the right image shows an example graspable position and orientation for a particular grasp.

4.2.8 Reachability and Inverse Reachability

At this stage, MoMa has finished exploring, detecting, approaching, mapping the object, and identifying a graspable pose. Now, the most important question is: is MoMa at the correct base placement to execute

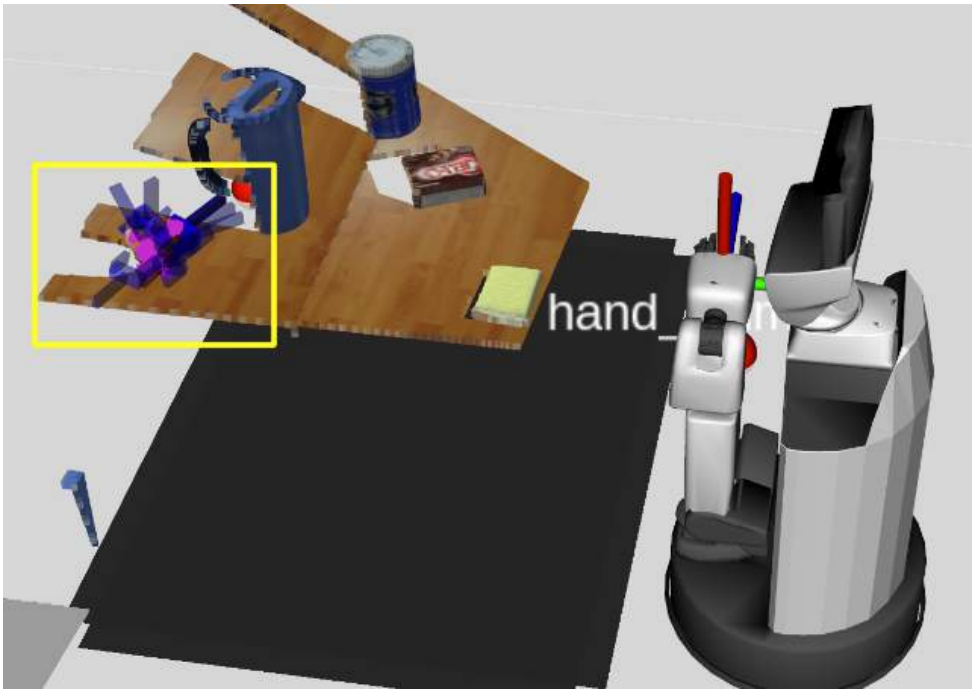


Figure 4.8: GPD output in our pipeline, showing grasp proposals generated for objects on the table (highlighted region).

the grasp? This is where we utilize the concepts of reachability and inverse reachability. We have already introduced the idea of reachability in Section 2.3. The main purpose of using reachability in our work is to check if the grasp proposal suggested by GPD falls within MoMa's reachable base workspace.

In our work, we generate reachability maps (based on the work of S. Jauhri *et al.* [10]) as shown in Figure 4.9 (left image). The right image in Figure 4.9 shows the inverse reachability maps, which provide all possible base placements for MoMa. These maps are generated using sampling-based methods by computing the forward kinematics (for reachability maps) of MoMa using its universal robot description file (URDF). The inverse reachability map is created by taking an inverse transform of the reachability map and projecting it onto the ground plane [40]. The benefit of using reachability maps is that they speed up the inverse kinematics computation, as the map serves as a lookup table, enabling quick computation of base placements [10].

As discussed in the previous Section 4.2.7, the grasp poses generated by the GPD are sent as input to the inverse reachability map, resulting in an output as shown in Figure 4.10. This computation of base placements is done for each GPD pose in online (i.e. during the run). As the number of grasp poses increases, the number of base placements also increases. To filter them, we use distance-based filtering, eliminating very distant base positions, as shown in Figure 4.10.

The final step in this stage is to select the best base pose by checking for static obstacles and the robot's footprint. Once the best base pose is identified, we select its corresponding GPD pose for grasping. Since we now have both the best base pose and the best grasp pose, the next immediate step is to navigate

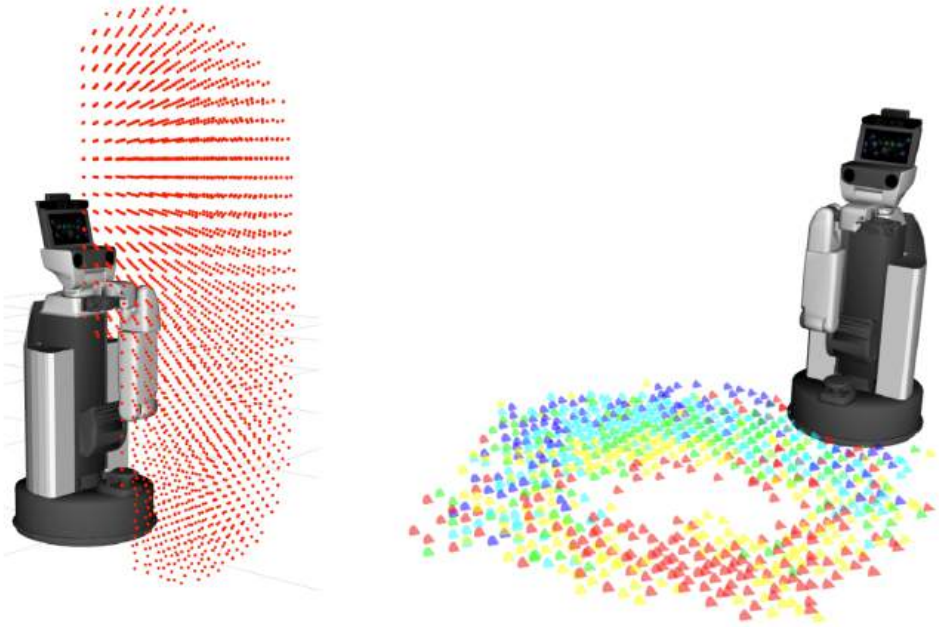


Figure 4.9: Left: Reachability map generated for the Toyota HSR, showing reachable poses in red. Right: Inverse reachability map illustrating regions where the robot can position itself to reach the target, based on the work by S. Jauhri *et al.* [10] (colors represent the base placements with scores and green being the most reachable).

to the best base pose and execute the grasp (grasp execution is discussed in the next Section 4.2.9).

4.2.9 Grasp Motion Generation

Finally, once we have identified the best grasp pose, MoMa proceeds with grasp execution using the MoveIt [41] motion planning ROS package. In our work, we utilize HSR's proprietary plugins for MoveIt, along with MoveIt's whole-body Cartesian motion capabilities. Figure 4.11 illustrates the various stages during grasping. Initially, the robot is brought to a pre-grasp state, followed by planning for the pose execution. The whole-body motion allows us to adjust the base during grasp execution.

During our initial testing, MoveIt's planning was not always successful due to difficult orientations. To address this, we use the HSR's end effector orientation during the pre-grasp phase, replacing the best pose's orientation with the pre-grasp orientation before executing the grasp. Once MoMa reaches the target object, we also lower the arm to ensure the grasp is complete, and then close the gripper. Finally, if we successfully lift the target object from its position (as shown in Figure 4.11), this indicates that we have successfully completed the task of grasping the object.

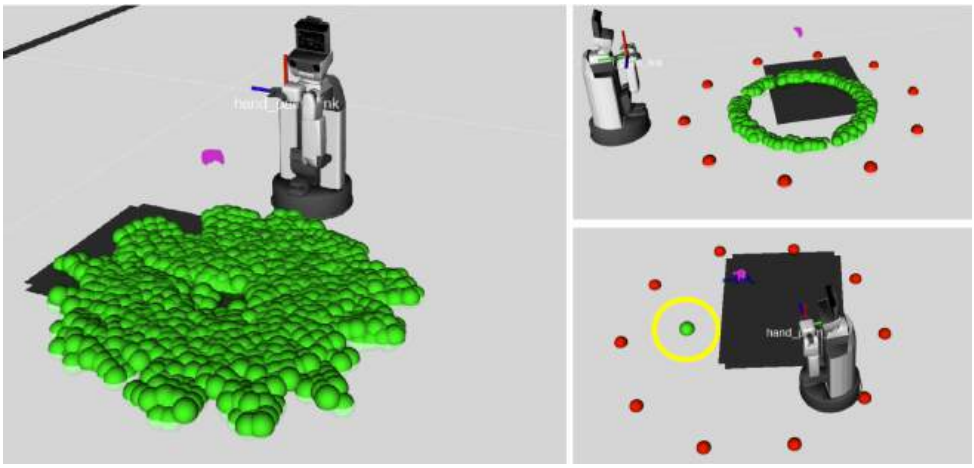


Figure 4.10: The usage of inverse reachability in our work for base placement. Left: Full inverse reachability map for a particular GPD pose. Top right: Filtered points based on distance from the target object. Bottom right: The best selected reachable point, shown in green and highlighted with a yellow circle.

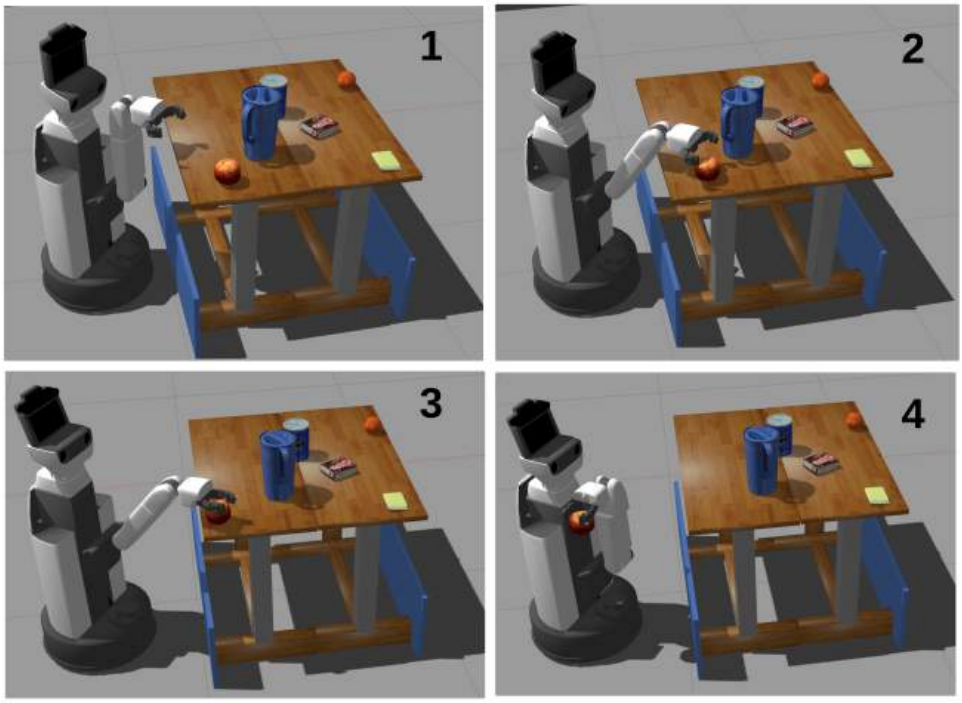


Figure 4.11: The image demonstrates the process of grasp motion generation using MoveIt. The robot begins in a pre-grasp pose, positioning its gripper near the target object. It then performs a whole-body Cartesian motion to move the gripper closer and align it properly. Once in position, the robot executes a successful grasp, lifting the object off the table.

5

Evaluation and Results

In this chapter, we outline the evaluation strategy for our CAPerMoMa system and describe the experiments designed to assess its performance. We provide detailed information about the experimental setup, data logging, and the metrics used to measure the system's effectiveness. The results from these experiments are then presented, followed by a comprehensive discussion that examines the system's behavior, success rates, and overall performance. Finally, we reflect on the implications of these results and what they reveal about the system's capabilities and limitations.

5.1 Experimental Setup

All experiments were conducted in simulation using the Gazebo 11 simulator on a high-performance PC. Below are the relevant technical specifications of the system used:

- **Model:** XMG NEO 16 (E23)
- **GPU:** NVIDIA GeForce RTX 4060 Laptop, 8 GB GDDR6
- **Processor:** Intel Core i9-13900HX
- **RAM:** 16 GB DDR5-5600 (2×8 GB)
- **Storage:** 1 TB M.2 Samsung 980, PCIe 3.0 x4 NVMe

5.2 Experiments Overview

All experiments are conducted in a simulated, closed domestic environment. One of the primary objectives of our CAPerMoMa system is to successfully grasp an object located on a table in the shortest time possible. To evaluate the system's performance, we designed three distinct sets of experiments.

- **Experiment 1:** The first experiment involves four different scenarios with progressively increasing search space. In each scenario, the target object is placed in five distinct locations on the table. Thus, this experiment consists of a total of 20 trials, with each scenario having five trials corresponding to different graspable object positions. The primary aim of this experiment is to assess the overall performance of the CAPerMoMa system, with particular focus on evaluating the time taken by each stage of the pipeline.

- **Experiment 2:** The second experiment is an ablation study designed to analyze the importance of key components in the CAPerMoMa system. Specifically, we disable components such as gaze control and reachability analysis to observe the effect on system performance. This experiment aims to highlight the significance of these two components, which we hypothesized in Section [X] to be important for successful grasping.

- **Experiment 3:** The third experiment evaluates the CAPerMoMa system's performance with different objects. The goal is to determine the system's generalization capability to other object types.

The evaluation is based on two key metrics:

- **Total time taken** for each stage of the pipeline and for the entire mobile manipulation task.

- **Grasp success or failure**, where a successful grasp is defined as the robot's ability to successfully hold and lift the object from the table.

5.3 Data Logging

As discussed in the previous section, time is a key metric used to evaluate the performance of our CAPerMoMa system. To capture this, we implement a data logging mechanism during the experiment runs. The strategy employed for data logging is shown in the python pseudo-code provided in code listing A.1. Below, we describe the steps involved in the logging process:

- Each state of the system, including exploration, mapping, and manipulation, records the time required to complete its respective tasks. The start time for each state is recorded at the beginning of its execution, and the elapsed time is calculated once the state transitions to the next phase. This timing data is stored in the "userdata" variable for further processing.

- Within the manipulation state, additional subtasks such as 'placement', 'navigation', and 'grasp' are timed individually. This enables a more detailed breakdown of the time required for each stage of the manipulation process, allowing us to record data of specific components.

- The overall time taken for the entire mobile manipulation pipeline is recorded by logging the start time at the beginning of the pipeline execution and calculating the total elapsed time once the state machine completes all stages.

- Once the state machine has finished its execution, the recorded times for all states and subtasks are logged into a CSV file. If the file does not already exist, it is created and initialized with appropriate headers. Subsequent runs append the new time data as additional rows in the CSV file, providing a comprehensive log of the time performance metrics for each experiment run.

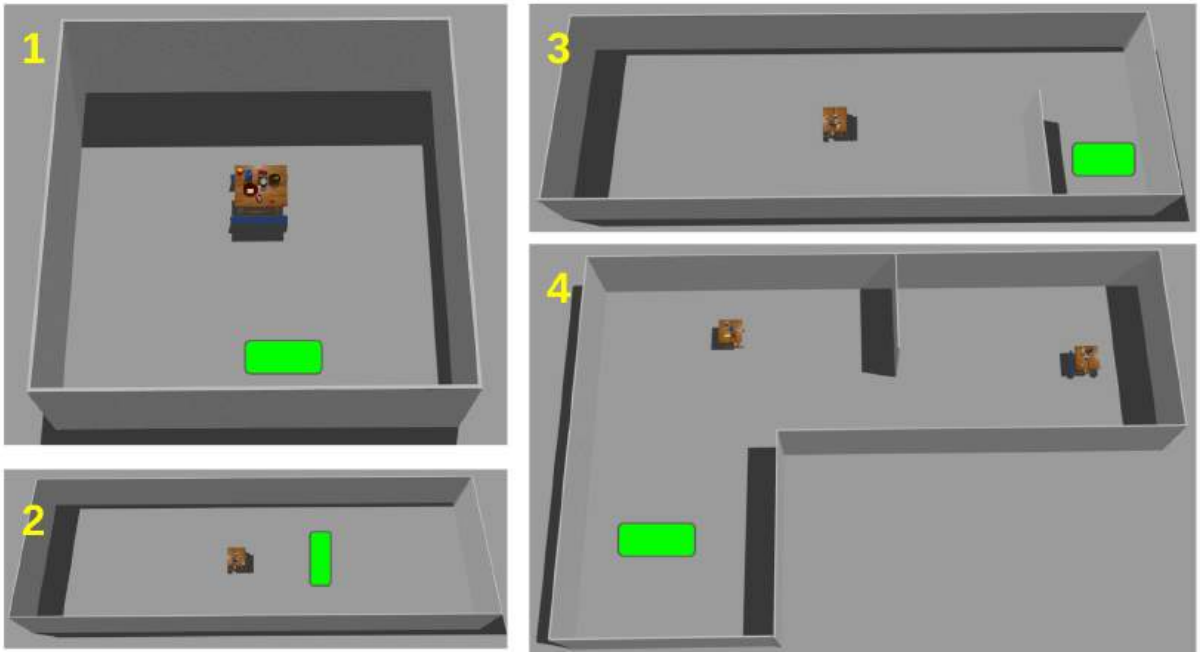


Figure 5.1: Four different experimental scenarios, each labeled with a number (1 to 4). In all scenarios, a small table with an apple and various objects on it is placed inside a walled environment. In scenario 4, we have an extra decoy table on the left. The green regions represent the start location for MoMa during an experiment.

5.4 Experiment-1

As outlined in Section 5.2, the primary objective of Experiment 1 is to evaluate the performance of the CAPerMoMa system. The experiment consists of four distinct scenarios, as depicted in Figure 5.1. In each scenario, the task involves searching for and grasping a target object (an apple, in this case) located on a table. The robot initiates each trial from a designated starting position in the green region, oriented away from the object.

Each scenario includes five trials, with each trial featuring a different object pose, as illustrated in Figure 5.2. Thus, the experiment comprises a total of 20 trials, combining the four scenarios. The following

Figure 5.2: Five different poses of an apple (highlighted in yellow circle and numbered), placed on a table alongside various objects.

sections present the results for each individual scenario and provide a detailed analysis. Subsequently, a comparative discussion of the results across all four scenarios is presented.

5.4.1 Scenario 1: Results and Discussion

For the results of scenario1 in Experiment 1, we evaluated the performance of the CAPerMoMa system in searching for and grasping the target object, an apple, placed on the table as shown in Figure 5.1. The results are presented in the Table 5.1 and focus on time data such as exploration time, mapping time, placement time, navigation time, grasp time, and the overall total execution time. Each trial corresponds to a different pose of the target object, leading to variation in the times recorded across the different stages of the manipulation pipeline.

Trial Number	**Exploration Time (s)**	**Mapping Time (s)**	**Placement Time (s)**	**Navigation Time (s)**	**Grasp Time (s)**	**Total Execution Time (s)**
1	1.02	24.29	9.77	17.68	**31.33**	84.10
2	1.10	20.46	2.78	39.90	60.92	125.16
3	1.02	**7.42**	7.01	14.01	53.27	82.73
4	**1.01**	16.42	**2.60**	32.57	64.37	116.97
5	1.02	15.34	6.89	**13.53**	41.36	**78.15**
Average	1.03	16.79	5.81	23.53	50.25	97.42
Standard Deviation	±0.04	±6.32	±3.07	±11.98	±13.78	±21.89

Table 5.1: Time data across five trials for scenario 1 along with average and standard deviation: smallest values in each column are highlighted in bold.

Key observations are given below:

- In Trial 1, the total execution time was 84.10 seconds, with the highest Grasp Time of 31.33 seconds, indicating that the grasping phase was a main contributor to the overall time, but at the same time it is the least time as compared to other trials (see Figure 5.4,5.5).
- Conversely, Trial 5 had the lowest total execution time of 78.15 seconds, with the fastest navigation time at 13.53 seconds.
- Exploration times were consistent across trials, with values around 1.01-1.10 seconds, this is because MoMa's initial pose was very near to the table and it could detect the target object quickly.
- Mapping time varied across trials, with Trial 3 having the shortest time of 7.42 seconds, significantly reducing the total execution time.
- Placement time showed notable variation, with Trial 2 achieving the least time at 2.78 seconds for positioning the robot's base.

The Figure 5.3 provides a visual summary of all the trials showing the averages with standard deviations (also see Table 5.1). It clearly shows that the grasping and navigation stages are the most time-consuming

and have the highest variability. This variability is due to the base placement and the graspable pose change for each trial. Exploration is the most consistent stage, contributing minimally to overall execution time due to shorter search space in scenario 1.

Grasp Outcome: For scenario 1, as shown in the grasp outcome Table 5.2, all five trials resulted in successful grasps with no reported failures (**100% success**). This consistent performance indicates that the CAPerMoMa system was able to handle all the tasks effectively in this controlled environment. This result demonstrates the system's capacity to reliably execute grasps in relatively straightforward scenarios, providing a good foundation for further testing in more complex environments.

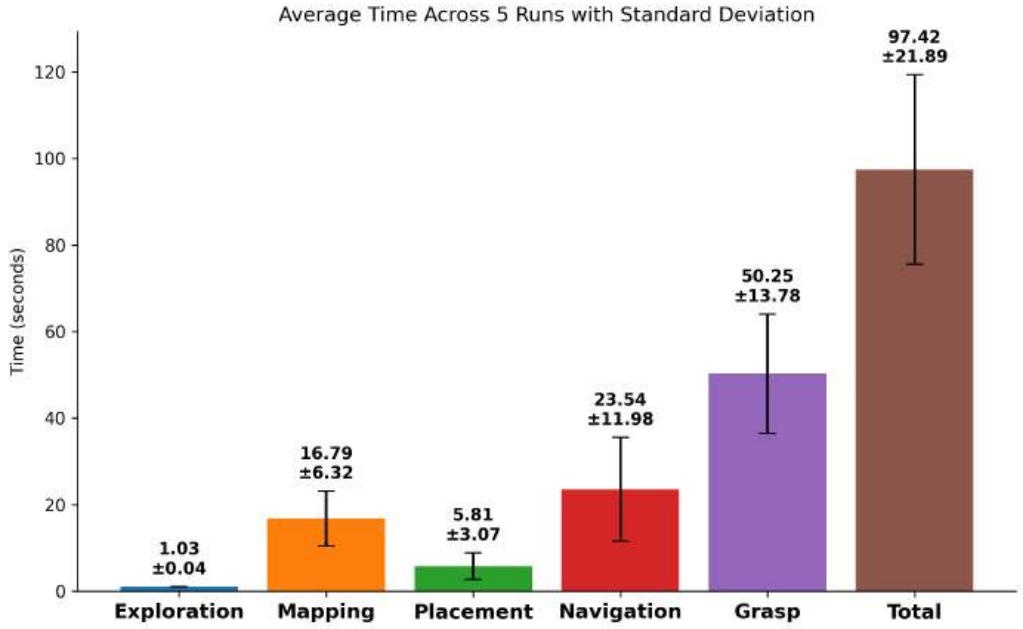
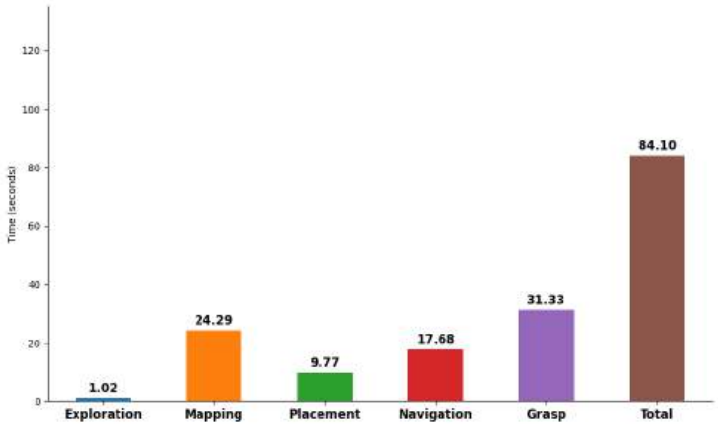


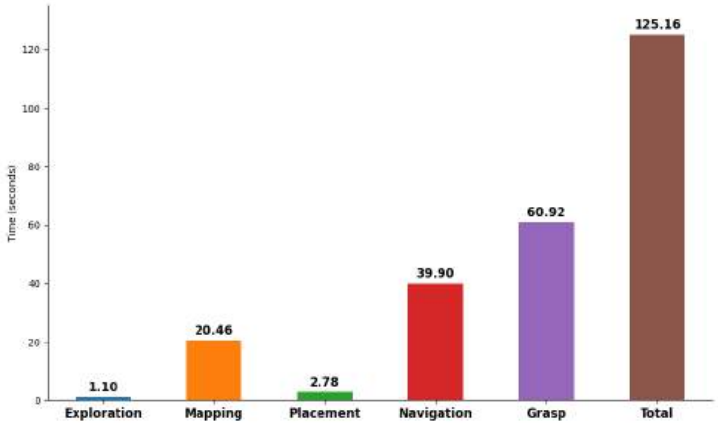
Figure 5.3: Average task completion times across various phases in scenario 1, including exploration, mapping, placement, navigation, and grasp, with error bars indicating the standard deviation.

Trial Number	Grasp Outcome	Reason for Failure
1	Success	No Failure
2	Success	No Failure
3	Success	No Failure
4	Success	No Failure
5	Success	No Failure

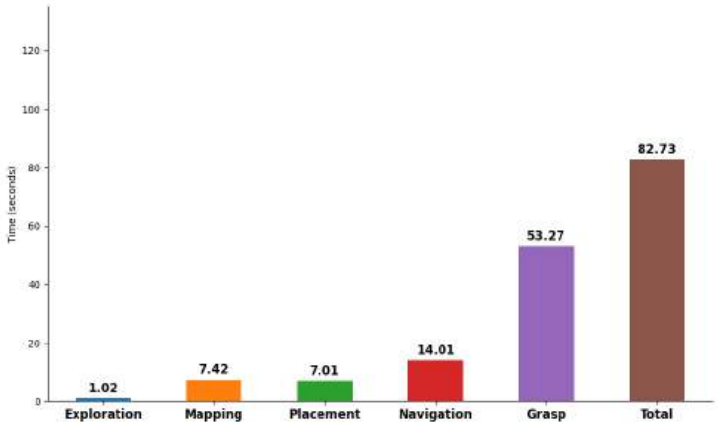
Table 5.2: Grasp outcome and failure analysis for scenario 1.



(a) Trial 1

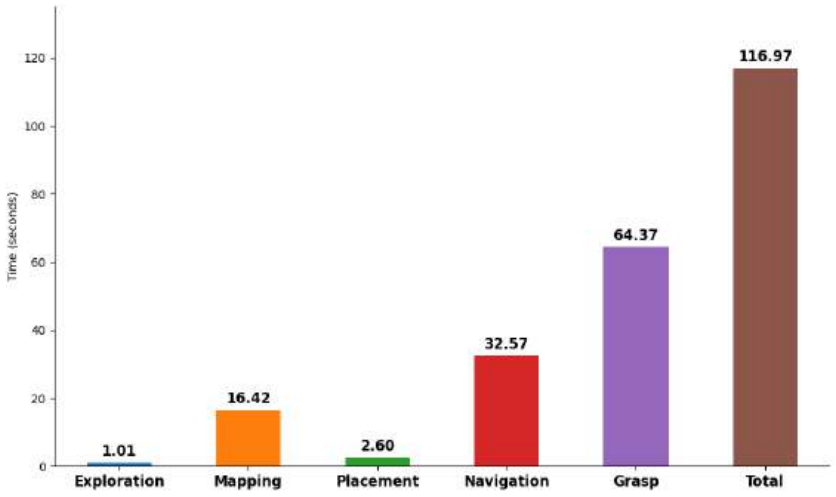


(b) Trial 2

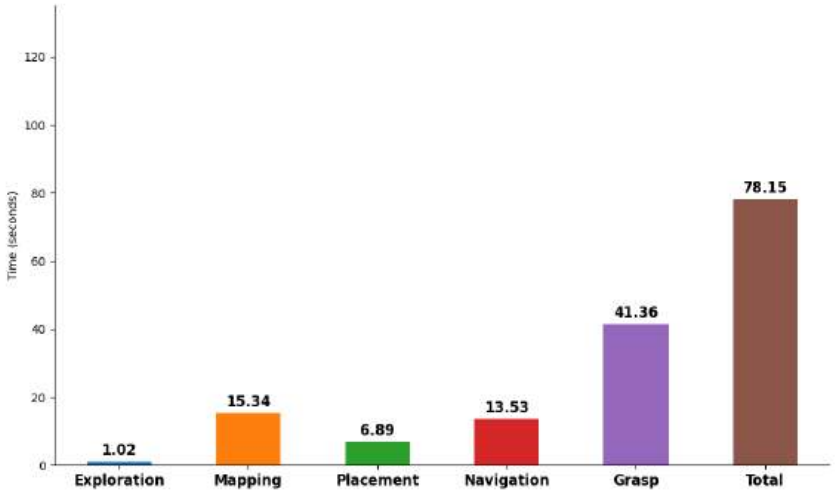


(c) Trial 3

Figure 5.4: Comparison of time spent on various stages of the task across trials 1-3 in scenario 1. The stages include exploration, mapping, placement, navigation, and grasping, with the total time shown in each bar plot.



(a) Trial 4



(b) Trial 5

Figure 5.5: Comparison of time spent on various stages of the task across trials 4-5 in scenario 1. The stages include exploration, mapping, placement, navigation, and grasping, with the total time shown in each bar plot.

5.4.2 Scenario 2: Results and Discussion

For Scenario 2, the results in the Table 5.3 highlight the system's performance in a more complex environment, with increased search space compared to Scenario 1. MoMa is placed in the green region facing away (opposite direction) from the target object as this results in more search operation.

Trial Number	Exploration Time (s)	Mapping Time (s)	Placement Time (s)	Navigation Time (s)	Grasp Time (s)	Total Execution Time (s)
1	101.80	**2.45**	2.93	14.48	43.82	165.47
2	73.62	23.33	7.54	**0.44**	**39.14**	144.07
3	124.96	11.66	3.51	26.66	54.18	220.98
4	**39.71**	17.43	3.71	26.12	41.11	**128.09**
5	63.97	22.34	**3.04**	30.77	46.48	166.60
Average	80.81	15.44	4.15	19.69	44.94	165.04
Standard Deviation	±33.22	±8.61	±1.92	±12.35	±5.86	±35.13

Table 5.3: Time data across five trials for scenario 2: smallest values in each column are highlighted along with average and standard deviation.

Key observations are given below:

- Exploration Time: The average exploration time across the five trials was 80.81 seconds, with significant variability as indicated by a standard deviation of 32.12 seconds. Trial 4 achieved the lowest exploration time of 39.71 seconds, while Trial 3 took the longest, at 124.96 seconds. The variation can be attributed to differences in object positions and the complexity of the environment, suggesting that the system's exploration performance is sensitive to initial object visibility and placement.

- Mapping Time: The average mapping time was 15.44 seconds, with a relatively moderate standard deviation of 8.08 seconds. Trial 1 had the fastest mapping time of 2.45 seconds, while Trial 2 required more time for mapping (23.33 seconds). The system performed best in Trial 1, where MoMa might have found the target object quickly from the far and initiate the mapping early.

- Placement Time: Placement times were more consistent, with an average of 4.15 seconds and a standard deviation of 1.89 seconds. The lowest placement time was 3.04 seconds in Trial 5, and the highest was 7.54 seconds in Trial 2. The variation in placement times indicates that some object locations required more precise positioning, possibly due to the table obstacle or the robot's footprint needing additional adjustments.

- Navigation Time: Navigation time showed significant variability, with an average of 19.69 seconds and a large standard deviation of 11.98 seconds. Trial 2 had the fastest navigation time, taking only 0.44 seconds, indicating that MoMa was very close to the best placement location. In contrast, Trial 5 took the longest (30.77 seconds) as it was far from MoMa.

- Grasp Time: Grasping remained relatively stable across trials, with an average time of 44.94 seconds and a standard deviation of 6.00 seconds. The best grasp time was recorded in Trial 2 (39.14 seconds), while Trial 3 had the highest (54.18 seconds) (see Figure 5.7,5.8).

- Total Execution Time: The overall execution time for Scenario 2 averaged 165.04 seconds, with the lowest total time recorded in Trial 4 (128.09 seconds) and the highest in Trial 3 (220.98 seconds).

The Figure 5.6 provides a visual summary of all the trials showing the averages with standard deviations (also see Table 5.3). It clearly shows that the grasping and navigation stages are the most time-consuming and have the highest variability. This variability is due to the fact that the base placement and the graspable pose change for each trial. Exploration is the most consistent stage, contributing minimally to overall execution time due to shorter search space in scenario 1.

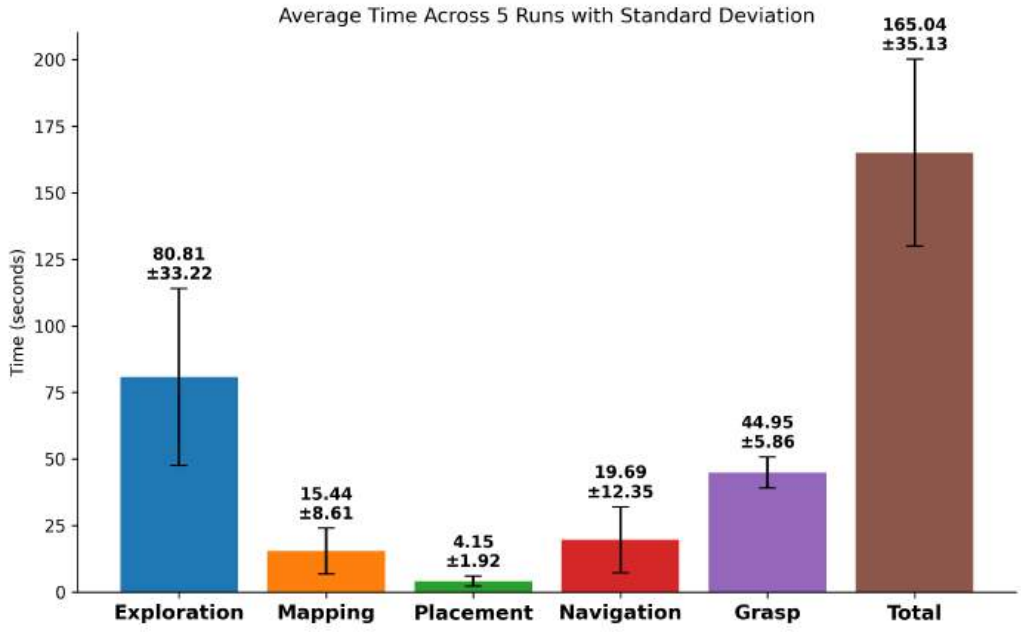
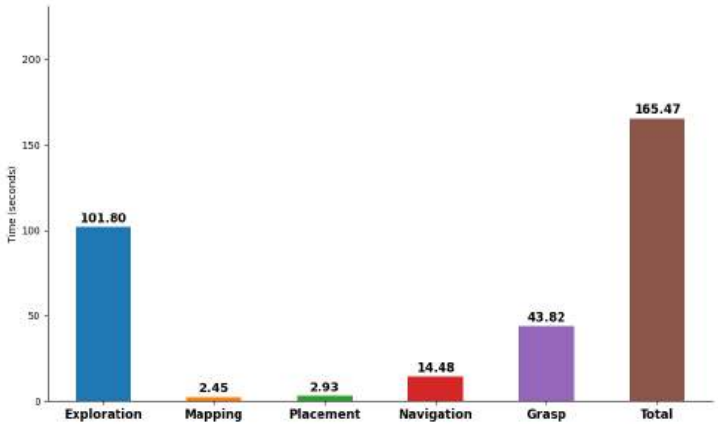
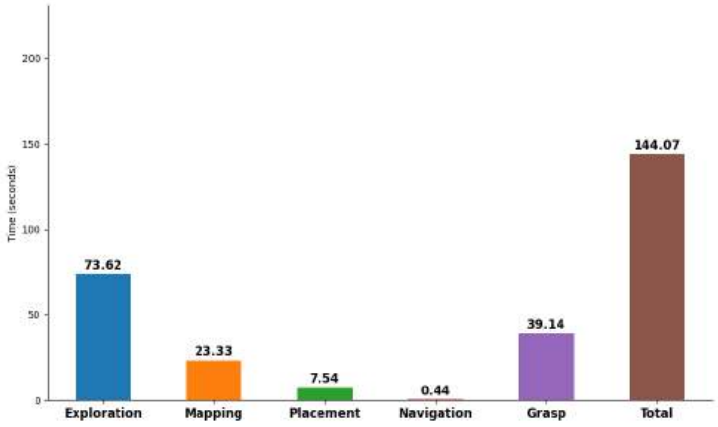


Figure 5.6: Average task completion times across various phases in scenario 2, including exploration, mapping, placement, navigation, and grasp, with error bars indicating the standard deviation.

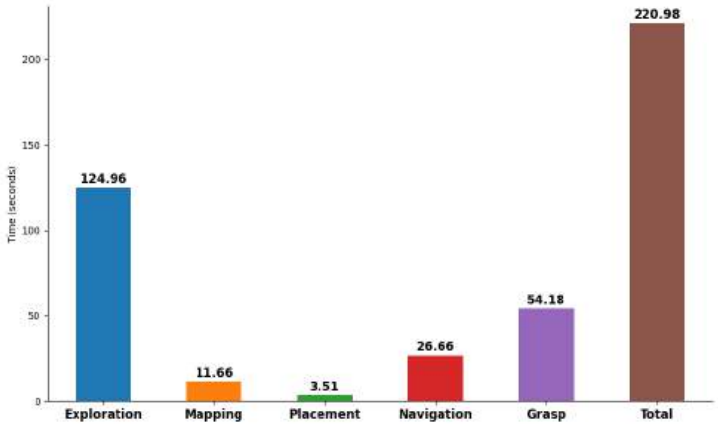
Grasp Outcome: In Scenario 2, the system experienced two grasp failures out of five trials as shown in Table 5.4 (**60% success**). In Trial 2, the failure occurred due to the object rolling over during the grasp attempt, highlighting a need for better object stabilization, especially for objects prone to movement (see Figure 5.9). In Trial 3, the failure was caused by the early closing of the gripper, which indicates a timing issue between object detection and gripper actuation (see Figure 5.10).



(a) Trial 1

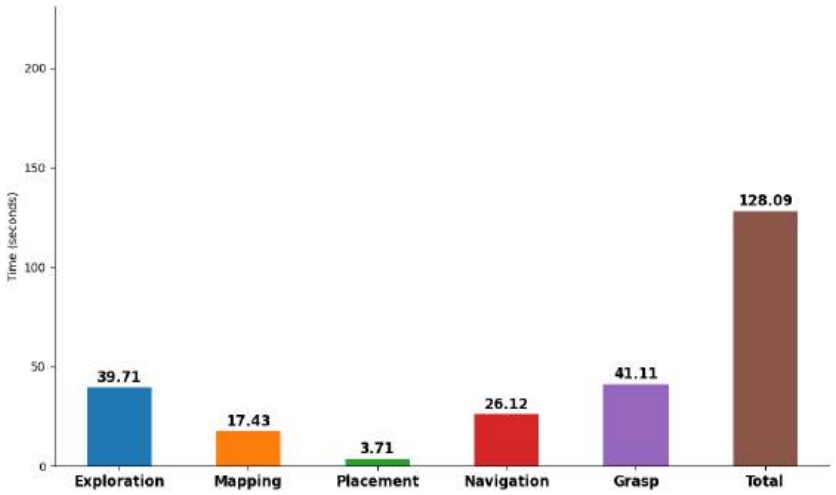


(b) Trial 2

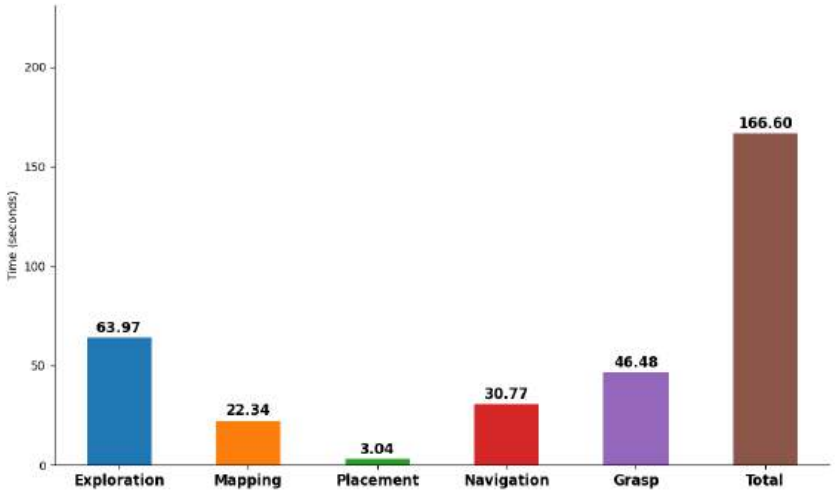


(c) Trial 3

Figure 5.7: Comparison of time spent on various stages of the task across trials 1-3 in scenario 2. The stages include exploration, mapping, placement, navigation, and grasping, with the total time shown in each bar plot.



(a) Trial 4



(b) Trial 5

Figure 5.8: Comparison of time spent on various stages of the task across trials 4-5 in scenario 2. The stages include exploration, mapping, placement, navigation, and grasping, with the total time shown in each bar plot.

Trial Number	Grasp Outcome	Reason for Failure
1	Success	No Failure
2	Fail	Object roll over
3	Fail	Early gripper closing
4	Success	No Failure
5	Success	No Failure

Table 5.4: Grasp outcome and failure analysis for scenario 2.

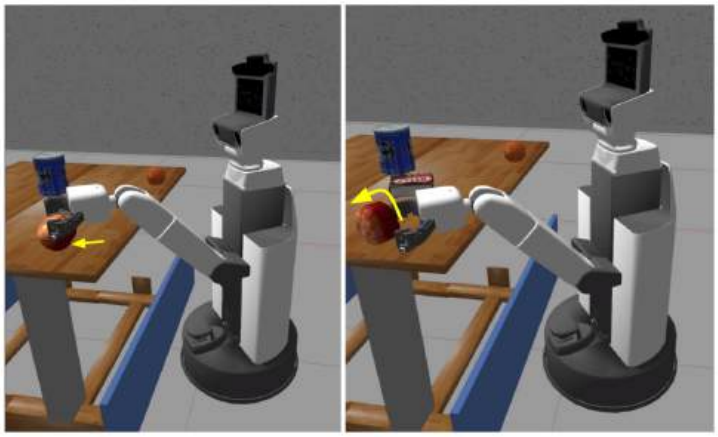


Figure 5.9: The object roll-over problem during grasping. Left: the gripper approaches the apple, but due to the pose-offset the palm moves extra ahead and the round shape of the object, the apple starts to roll as shown by the yellow arrow. Right: the apple rolls out of the gripper's range, causing a grasp failure.

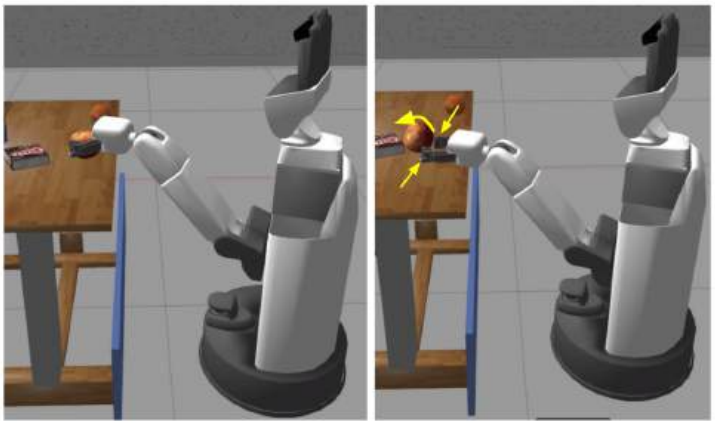


Figure 5.10: The early gripper closing problem. Left: the robot approaches the apple in preparation for a grasp. Right: the gripper begins to close prematurely, as indicated by the yellow arrows, before making full contact with the object. This results in a failure to secure the apple properly.

5.4.3 Scenario 3: Results and Discussion

For Scenario 3, the results in the Table 5.5 highlight the system's performance in an even more complex environment compared to previous scenarios. With increased search space and further away from the table. MoMa is placed in the green region which is like a separate room.

Trial Number	Exploration Time (s)	Mapping Time (s)	Placement Time (s)	Navigation Time (s)	Grasp Time (s)	Total Execution Time (s)
1	89.60	**10.90**	3.14	23.13	**38.22**	**165.00**
2	75.77	54.76	3.31	**10.71**	38.61	183.16
3	210.49	119.05	15.47	31.86	57.26	434.14
4	**58.79**	103.92	3.80	16.50	38.18	221.20
5	71.11	59.35	**3.09**	20.30	40.94	194.80
Average	101.15	69.60	5.76	20.50	42.64	239.66
Standard Deviation	±62.11	±42.99	±5.43	±7.87	±8.25	±110.61

Table 5.5: Time data across five trials for scenario 3: smallest values in each column are highlighted along with average and standard deviation

Key observations are given below:

- Exploration Time: The average exploration time across the five trials was 101.15 seconds, with significant variability as indicated by a standard deviation of 58.56 seconds. The fastest exploration time was 58.79 seconds in Trial 4, while Trial 3 took the longest at 210.49 seconds. The long exploration is because the frontiers were focused towards the walls most of the time.

- Mapping Time: The average mapping time was 69.60 seconds, with a standard deviation of 41.57 seconds. The fastest mapping time was recorded in Trial 1 at 10.90 seconds, while Trial 3 took the longest, requiring 119.05 seconds. The longer mapping time is due to longer time to query for an instance during the run.

- Placement Time: Placement times were relatively consistent, with an average of 5.76 seconds and a lower standard deviation of 5.37 seconds. Trial 5 had the quickest placement time of 3.09 seconds, while Trial 3 took significantly longer at 15.47 seconds (see Figure 5.12,5.13).

- Navigation Time: The average navigation time was 20.50 seconds, with a standard deviation of 7.88 seconds. The fastest navigation was in Trial 2 at 10.71 seconds, while Trial 3 took the longest at 31.86 seconds.

- Grasp Time: Grasp times were fairly consistent across trials, with an average of 42.64 seconds and a standard deviation of 8.04 seconds. The fastest grasp was achieved in Trial 1 at 38.22 seconds, while Trial 3 had the slowest grasp at 57.26 seconds.

- Total Execution Time: The overall execution time varied significantly, with an average of 239.66 seconds and a standard deviation of 101.80 seconds. Trial 1 had the fastest total execution time at 165 seconds, while Trial 3 had the longest at 434.14 seconds.

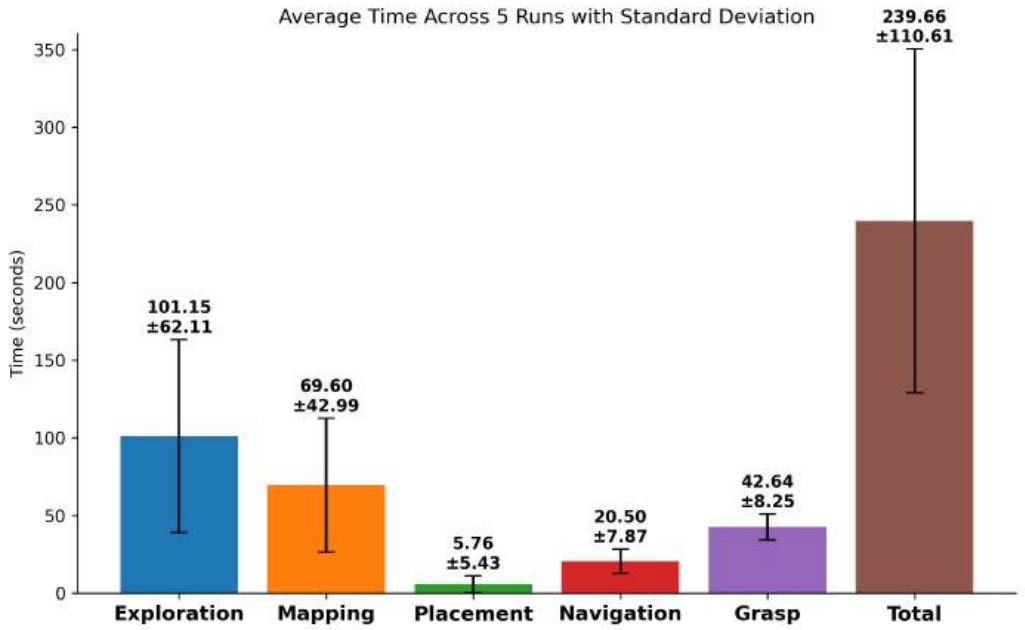


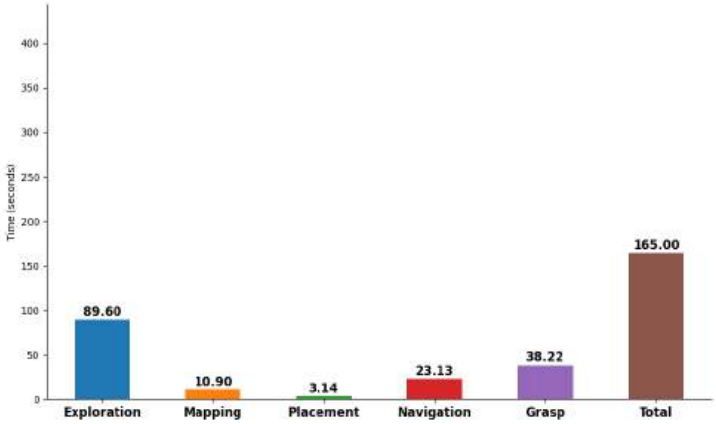
Figure 5.11: Average task completion times across various phases in scenario 3, including exploration, mapping, placement, navigation, and grasp, with error bars indicating the standard deviation.

The Figure 5.11 provides a visual summary of all the trials showing the averages with standard deviations (also see Table 5.5).

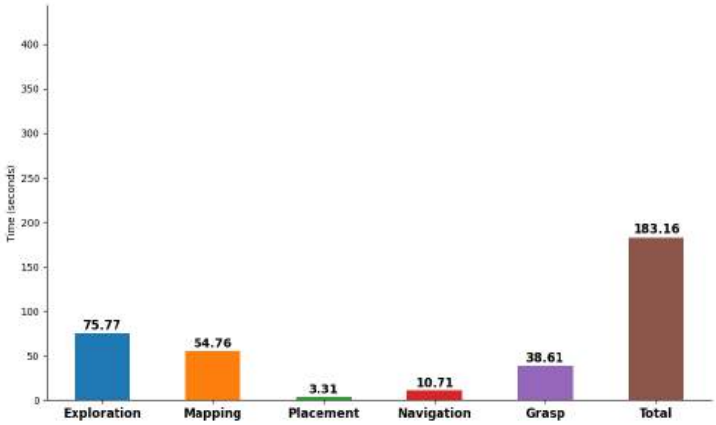
Trial Number	Grasp Outcome	Reason for Failure
1	Success	No Failure
2	Success	No Failure
3	Fail	Wrong base placement
4	Success	No Failure
5	Success	No Failure

Table 5.6: Grasp outcome and failure analysis for scenario 3.

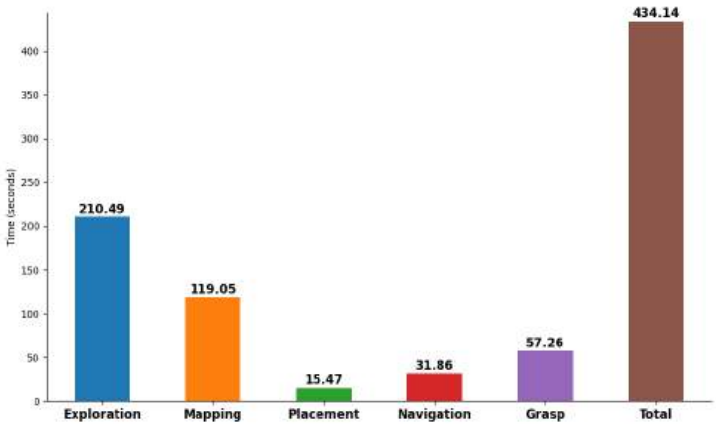
Grasp Outcome: In Scenario 3, the system experienced one grasp failure out of five trials as shown in Table 5.6 (**80% success**). The only failure occurred in Trial 3, where the system experienced a grasp failure due to wrong base placement. This indicates that the robot was not positioned optimally for a successful grasp, leading to a misalignment with the object (see Figure 5.14).



(a) Trial 1

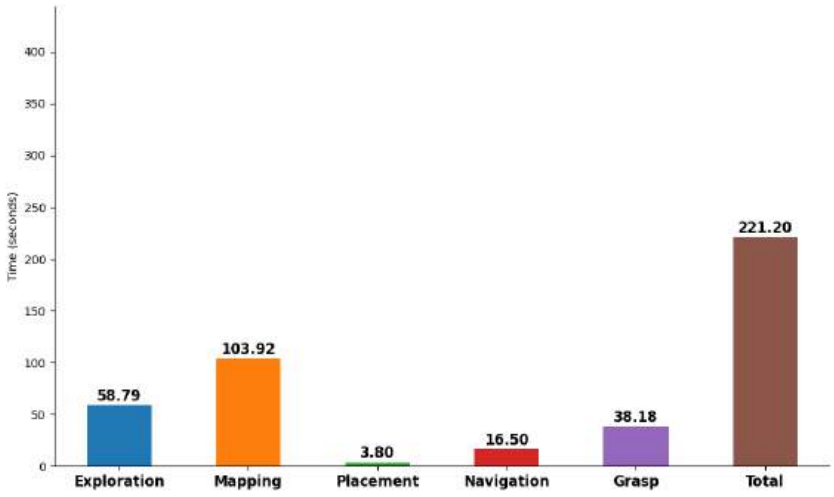


(b) Trial 2

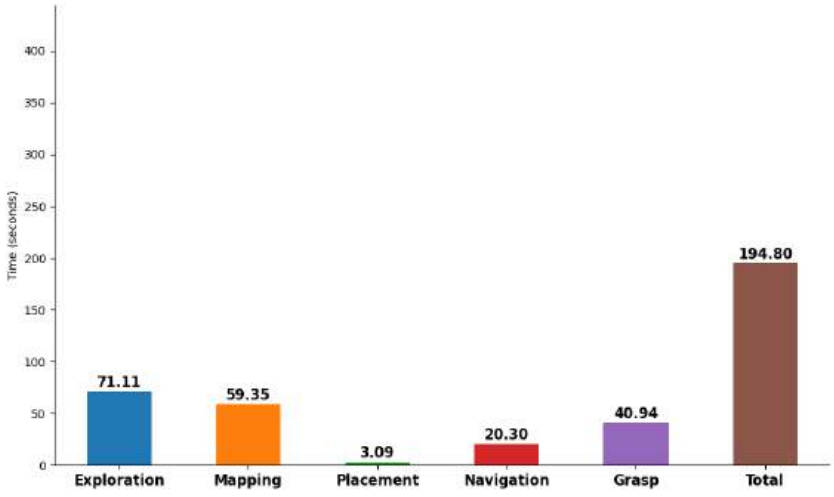


(c) Trial 3

Figure 5.12: Comparison of time spent on various stages of the task across trials 1-3 in scenario 3. The stages include exploration, mapping, placement, navigation, and grasping, with the total time shown in each bar plot.



(a) Trial 4



(b) Trial 5

Figure 5.13: Comparison of time spent on various stages of the task across trials 4-5 in scenario 3. The stages include exploration, mapping, placement, navigation, and grasping, with the total time shown in each bar plot.

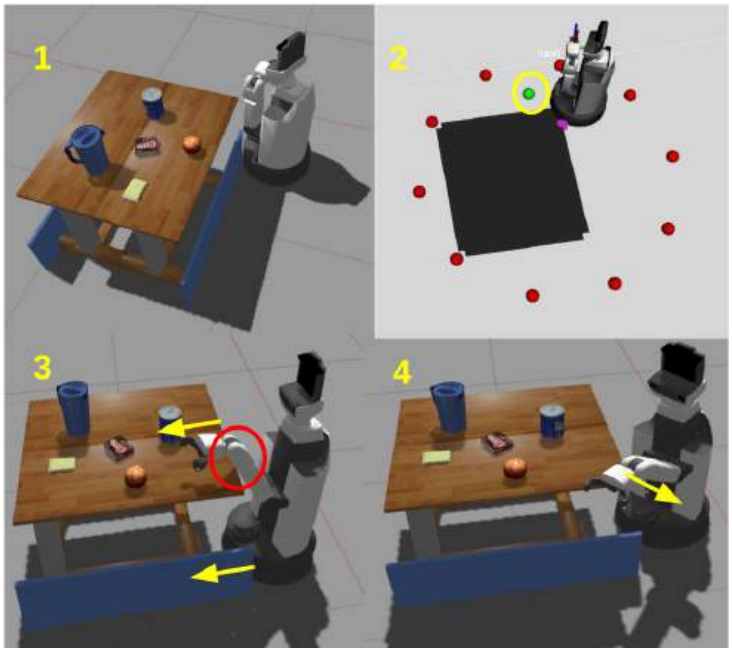


Figure 5.14: Wrong base placement problem leading to a failed grasp. The robot's position obstructs its arm movement, causing a collision with the table and preventing a successful grasp.

5.4.4 Scenario 4: Results and Discussion

For Scenario 4, the results in the Table 5.7 highlight the system's performance in an even more complex environment compared to previous scenarios. With increased search space and very far away from the table. MoMa is placed in the green region which is like a separate room.

Trial Number	Exploration Time (s)	Mapping Time (s)	Placement Time (s)	Navigation Time (s)	Grasp Time (s)	Total Execution Time (s)
1	**91.79**	98.72	6.17	**8.19**	208.71	413.58
2	121.58	**17.83**	6.79	12.57	39.94	**198.72**
3	99.25	102.54	4.49	14.54	**38.33**	259.15
4	119.30	107.51	**2.33**	14.87	36.67	280.69
5	142.04	34.75	4.77	4.47	38.78	224.82
Average	114.79	72.27	4.91	10.93	72.48	275.39
Standard Deviation	±19.87	±42.51	±1.73	±4.49	±76.16	±83.40

Table 5.7: Time data across five trials for scenario 4: smallest values in each column are highlighted along with average and standard deviation

Key observations are given below:

- Exploration Time: The average exploration time across the five trials was 114.79 seconds, with a

standard deviation of 19.33 seconds. The fastest exploration time was recorded in Trial 1 at 91.79 seconds, while Trial 5 took the longest, at 142.04 seconds.

- Mapping Time: The average mapping time was 72.27 seconds, with a larger standard deviation of 38.37 seconds. Trial 2 had the fastest mapping time, at 17.83 seconds, while Trial 4 took the longest, at 107.51 seconds (see Figure 5.16,5.17).

- Placement Time: Placement times were relatively consistent across all trials, with an average of 4.91 seconds and a small standard deviation of 1.80 seconds. The fastest placement time was recorded in Trial 4 at 2.33 seconds, while Trial 1 took the longest at 6.17 seconds.

- Navigation Time: The average navigation time was 10.93 seconds, with a standard deviation of 4.56 seconds. The fastest navigation was observed in Trial 1, which took 8.19 seconds, while Trial 3 required the longest navigation time at 14.54 seconds.

- Grasp Time: Grasp times showed significant variability, with an average of 72.48 seconds and a large standard deviation of 77.94 seconds. The longest grasp time was recorded in Trial 1 at 208.71 seconds, while the shortest time was in Trial 3 at 38.33 seconds.

- Total Execution Time: The overall execution time across the trials averaged 275.39 seconds, with a large standard deviation of 80.82 seconds. Trial 2 had the fastest execution time at 198.72 seconds, while Trial 1 took the longest at 413.58 seconds.

The Figure 5.15 provides a visual summary of all the trials showing the averages with standard deviations (also see Table 5.7).

Trial Number	**Grasp Outcome**	**Reason for Failure**
1	Fail	Wrong pose and grasp planning failed
2	Success	No Failure
3	Fail	Instance ID problem and mapping fail
4	Success	No Failure
5	Fail	Instance ID problem and mapping fail

Table 5.8: Grasp outcome and failure analysis for scenario 4.

Grasp Outcome: In Scenario 4, the grasp outcomes indicate a higher failure rate compared to previous scenarios, with three out of five trials resulting in failure, as shown in Table 5.8 (**40% success**). The only failure occurred in Trial 3, where the system experienced a grasp failure due to wrong base placement. Trial 1 resulted in failure due to wrong pose and grasp planning failure, indicating that the robot was unable to generate an appropriate grasp plan or approach angle for the object. Trials 3 and 5 failed due to an instance ID problem and mapping failure, where GPD is unable to receive correct point cloud and suggests random poses leading to a failure.

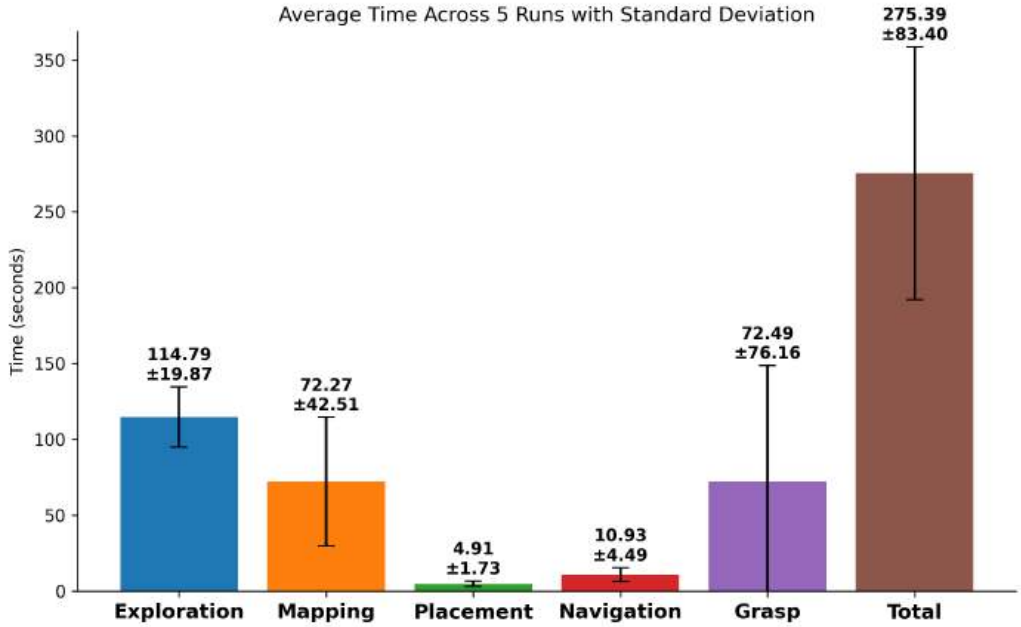
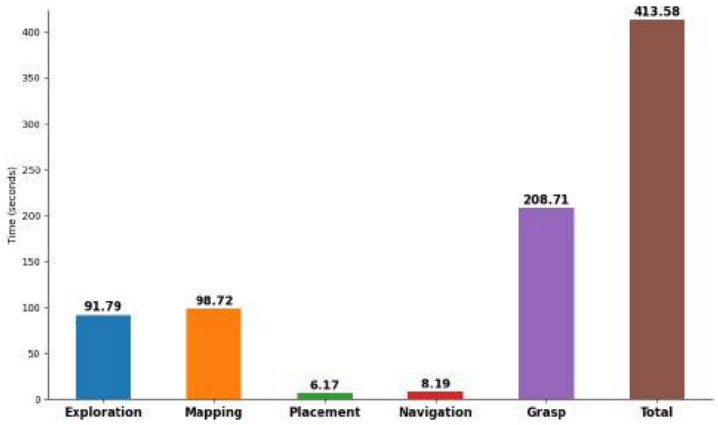


Figure 5.15: Average task completion times across various phases in scenario 4, including exploration, mapping, placement, navigation, and grasp, with error bars indicating the standard deviation.

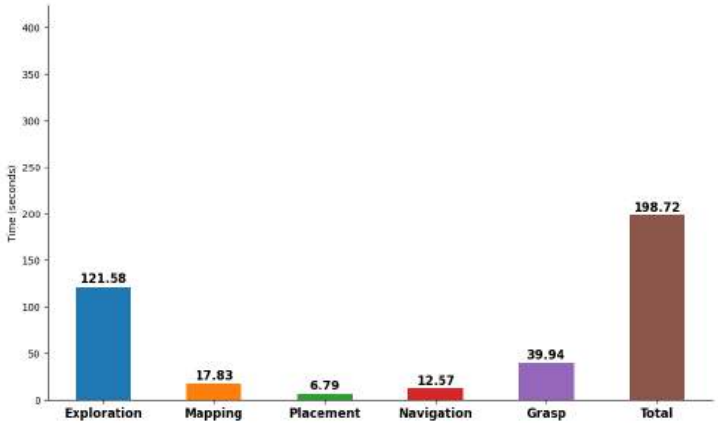
5.4.5 Common Discussion

This section presents the key observations derived from comparing all four scenarios:

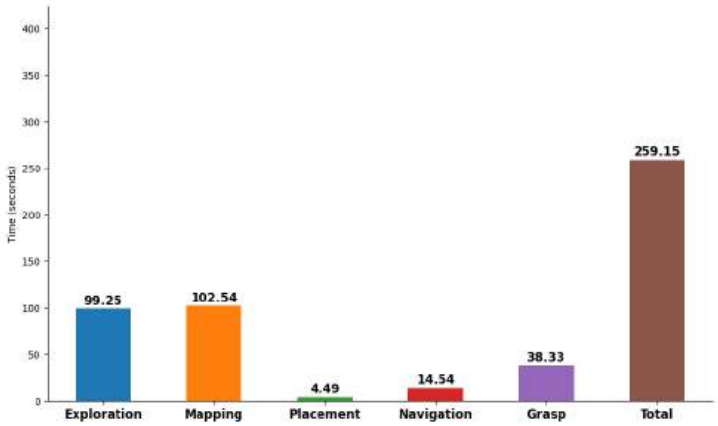
- Grasp Success Rate: Across all 20 trials, a total of 14 successful grasp attempts were recorded, resulting in an overall **grasp success rate of 70%**.
- Mapping Efficiency: In Scenarios 1 and 2, the mapping pipeline performed optimally, completing the mapping process in **under 25 seconds**. However, in Scenarios 3 and 4, the extended object search times caused the mapping nodes to run for an extended period, leading to an additional computational load. Once the object was detected, additional mapping queries were required to refine the mapping, thereby increasing the total mapping time.
- Exploration Time: The average exploration time increased proportionally from Scenario 1 to Scenario 4. This behavior was anticipated, as the map size expanded with each scenario, thereby increasing the time required to locate the target object.
- Base Placement Computation: The base placement strategy generally functioned efficiently across the trials, with the exception of Trial 3 in Scenario 3, where the base placement time was noticeably



(a) Trial 1

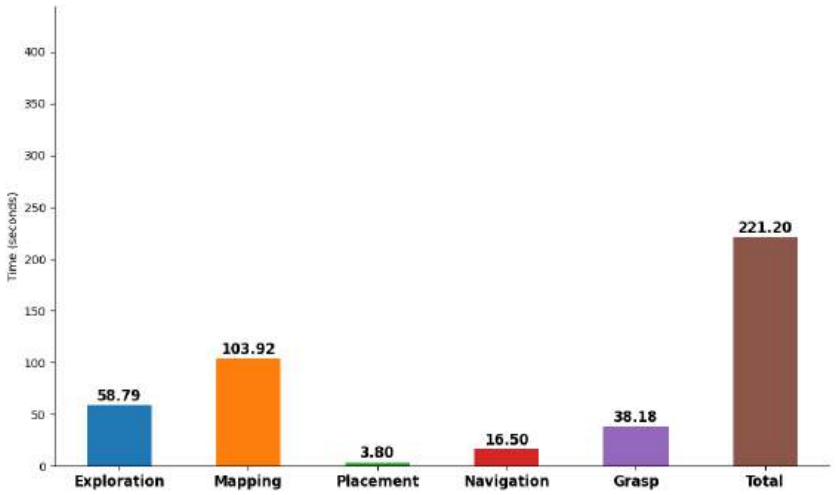


(b) Trial 2

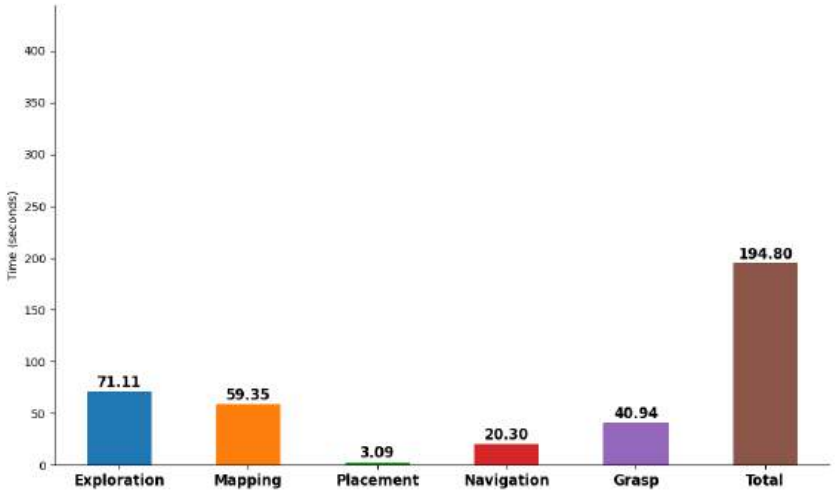


(c) Trial 3

Figure 5.16: Comparison of time spent on various stages of the task across trials 1-3 in scenario 4. The stages include exploration, mapping, placement, navigation, and grasping, with the total time shown in each bar plot.



(a) Trial 4



(b) Trial 5

Figure 5.17: Comparison of time spent on various stages of the task across trials 4-5 in scenario 4. The stages include exploration, mapping, placement, navigation, and grasping, with the total time shown in each bar plot.

longer. This delay can likely be attributed to the presence of obstacles, such as the table, or the robot's footprint, which required additional adjustments to find an optimal placement.

- Navigation and Grasp Pipelines: The navigation and grasping pipelines, primarily utilizing standard ROS components, operated as expected with respect to execution times.

5.5 Experiment-2

As outlined in Section 5.2, this experiment involves conducting an ablation study to evaluate the significance of key components in the CAPerMoMa system. Specifically, we disable components such as gaze control and reachability analysis to observe their impact on system performance. We use Scenario 2 as the standard testing environment for this study, as it offers a balanced level of difficulty, being neither too complex nor too simple. The experiment is divided into two distinct ablation cases:

- **Ablation 1 (Disabling Gaze Control):** In this case, we deactivate the gaze control mechanism and examine its effect on the system's performance, particularly during the mapping phase. Our initial hypothesis posits that disabling gaze control will result in an increase in mapping time. The idea is that continuous object detection, facilitated by gaze control, helps depth segmentation and volumetric mapping. By disabling this component, the object detector may struggle to consistently identify the target object, leading to slower mapping completion.

- **Ablation 2 (Disabling Reachability Analysis):** In this case, we disable the reachability analysis module and assess its effect on the system's manipulation performance. We hypothesize that without reachability analysis, the system will fail to manipulate objects positioned far from the robot's current location.

5.5.1 Ablation 1: Results and Discussion

The results of ablation study 1 are given in Table 5.9.

Trial Number	Exploration Time (s)	Mapping Time (s)	Placement Time (s)	Navigation Time (s)	Grasp Time (s)	Total Execution Time (s)
1	66.89	81.02	5.40	22.81	41.06	217.18
2	61.06	152.65	5.47	13.25	**38.80**	271.24
3	46.17	**20.19**	**2.80**	11.94	46.88	**127.98**
4	**36.02**	128.52	3.26	**9.93**	44.97	222.70
5	72.98	47.14	7.05	27.53	56.79	211.51
Average	56.62	85.90	4.80	17.09	45.70	210.12
Standard Deviation	±15.19	±51.26	±1.71	±6.70	±6.65	±52.02

Table 5.9: Time across five trials for ablation 1: smallest values in each column are highlighted along with average and standard deviation

In this study, our main focus is on comparing the mapping time between two configurations: with and without gaze control. Figure 5.18 presents a comparison of the mapping times for both cases, using the data obtained from Scenario 2 discussed in Section 5.4.2.

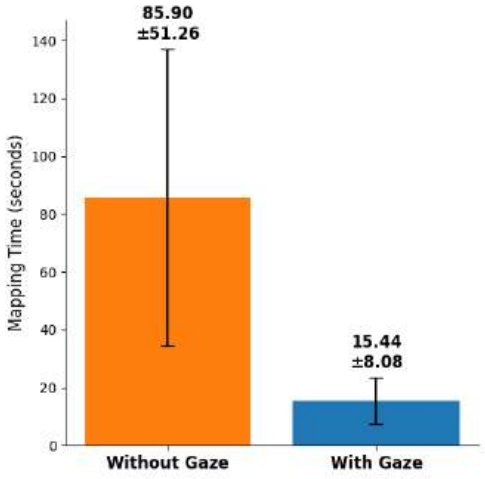


Figure 5.18: Comparison of mapping time with and without Gaze control. The system without gaze control exhibits significantly longer mapping times.

The Figure 5.18 demonstrates a significant reduction in mapping time—approximately **82%**—when gaze control is utilized, compared to the system operating without gaze control. The extended mapping time in the absence of gaze control can be attributed to the fact that, with gaze control, the robot's head remains consistently focused on the target object from the moment it is detected. This continuous focus facilitates uninterrupted object tracking and mapping. In contrast, without gaze control, MoMa system may intermittently lose sight of the object, causing delays in the mapping process, especially during the transition from exploration to mapping.

These findings support our initial hypothesis that gaze control significantly reduces mapping time. While the results presented are specific to scenario 2, we expect similar behaviour across other scenarios.

5.5.2 Ablation 2: Results and Discussion

The Table 5.10 summarizes the grasp outcomes for five trials in ablation 2, where reachability analysis was disabled. The results indicate that in four out of five trials, the robot failed to grasp the target object (**20% grasp success**).

- Trial 1 and 5 both failed due to the object being unreachable, likely because the robot's base was not positioned optimally to extend its manipulator fully toward the target as shown in Figure 5.19 and 5.20.

- In Trial 2 and 4, the failure occurred due to MoMa's base hitting the table, causing obstructions that prevented the robot from executing the grasp.
- Trial 3 was the only successful grasp, where the robot was able to position itself and manipulate the object without any problems. We see that, with out reachability only such cases succeed because the objects location is positioned on the table where MoMa is approaching from after exploration is done. In other cases if the object is on the other end we expect a similar behaviour as shown in Trail 1 and 5.

Trial Number	Grasp Outcome	Comments
1	Fail	Target object unreachable
2	Fail	Base hitting the table
3	Success	No Failure
4	Fail	Base hitting the table
5	Fail	Target object unreachable

Table 5.10: Grasp outcomes and associated comments for ablation 2, where reachability analysis is disabled.

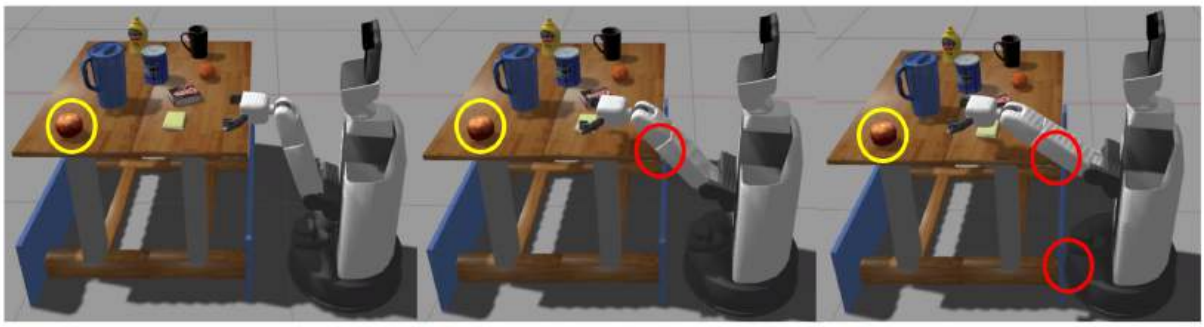


Figure 5.19: Trial 1 - Target object unreachable. MoMa attempts to grasp the apple (highlighted in yellow), where the base of MoMa is blocked by the table (highlighted in red), resulting in a failed grasp attempt.

These results suggest that disabling reachability analysis has a significant negative impact on MoMa's ability to grasp objects, particularly in situations where MoMa's base placement is important to avoid collisions with nearby obstacles and also during other mobile manipulation scenarios.

5.6 Experiment-3

In the third experiment, we evaluate the CAPerMoMa system's performance using various objects to assess its generalization capabilities. The objective is to determine how well the system can handle different object types. We chose scenario 2 from experiment 2 as the testing environment to maintain consistency.

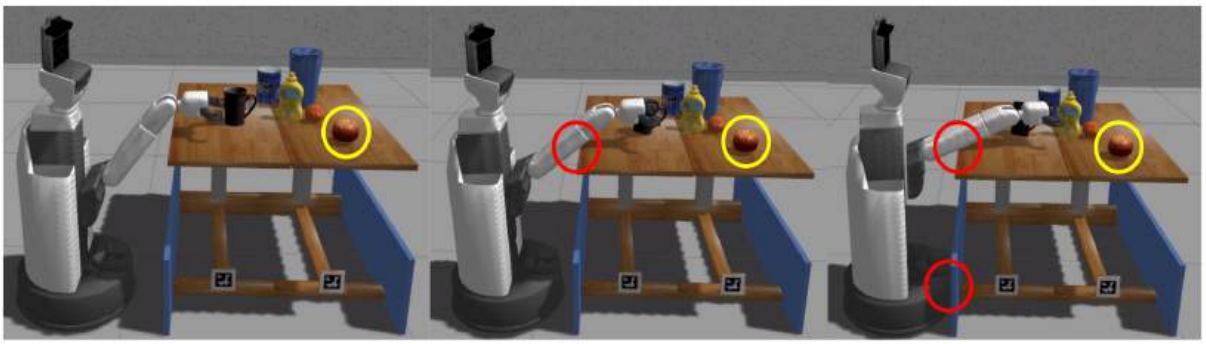


Figure 5.20: Trial 5 - Target object unreachable. MoMa attempts to grasp the apple (highlighted in yellow), where the base of MoMa is blocked by the table (highlighted in red), resulting in a failed grasp attempt.

Figure 5.21 depicts the two new objects utilized in this experiment. These objects serve as the new targets, and they are positioned according to Pose 3 from Figure 5.2.

Figure 5.21: Experiment 3 objects: left image mustard bottle and right image coffee cup.

5.6.1 Experiment-3: Results and Discussion

The Table 5.11 presents the results from experiment 3, comparing the performance of the CAPerMoMa system with two different objects, a mustard bottle and a coffee cup.

Object	Exploration Time (s)	Mapping Time (s)	Placement Time (s)	Navigation Time (s)	Grasp Time (s)	Total Execution Time (s)
Mustard Bottle	1.02	50.02	7.82	16.59	76.7	152.15
Coffee Cup	1.02	42.41	3.92	11.03	43.29	101.67

Table 5.11: Time data for different objects in Experiment 3: smallest values in each column are highlighted.

The exploration time remains consistent across both objects at 1.02 seconds. However, the mapping time is notably higher for the mustard bottle compared to the coffee cup. Overall, the grasp execution was also consistent as compared to experiments so far.

The Figure 5.22 and 5.23 show the successful runs for the mustard bottle and the coffee cup respectively. With this experiment we also prove that our CAPerMoMa system can also generalize to other objects as well.

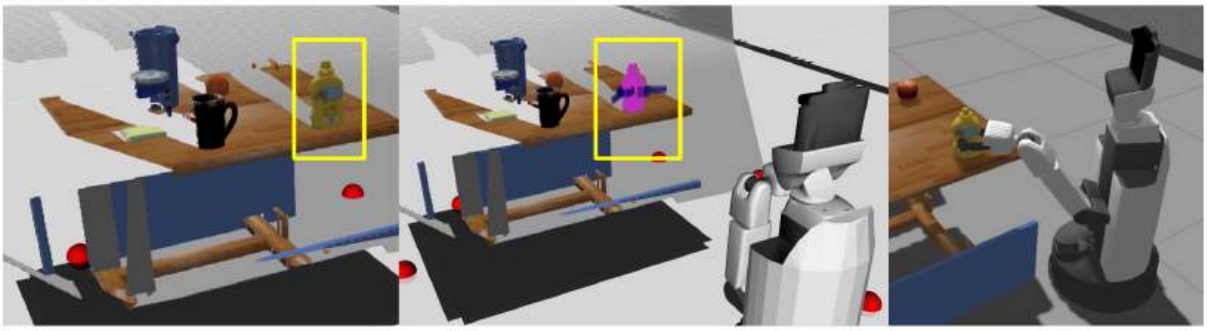


Figure 5.22: Grasping sequence for the mustard bottle: The left image shows the target object, the mustard bottle. In the middle image, MoMa approaches the object for grasping, with the object mapped and grasp pose detected. The right image shows the successful grasp of the object by MoMa's gripper, completing the task.

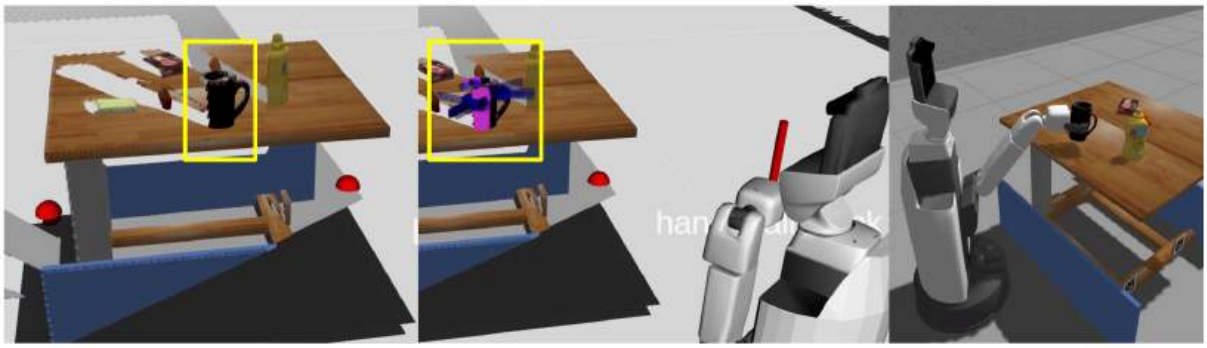


Figure 5.23: Grasping sequence for the coffee cup: The left image shows the target object, the coffee cup. In the middle image, MoMa approaches the object for grasping, with the object mapped and grasp pose detected. The right image shows the successful grasp of the object by MoMa's gripper, completing the task.

6

Conclusions

In this work, we presented CAPerMoMa, a system designed to improve MoMa's ability to explore unknown environments, detect objects, and execute precise manipulations. We integrated our system using active perception—where MoMa's camera sensors actively adjust to gather the most relevant information—which we termed coupled active perception. The key objective is to locate and successfully grasp the object in the shortest amount of time, with a high success rate, rather than constructing a comprehensive map of the surroundings and consuming more time.

We conducted an evaluation of CAPerMoMa through three sets of experiments. In experiment 1, the system demonstrated the best average mapping time under 20 seconds, achieving a grasp success rate of 70% across 20 trials. Experiment 2, which involved an ablation study, highlighted that gaze control significantly reduced mapping time (more than half the time) compared to passive systems. Additionally, integrating reachability maps improved grasp success by optimizing base placement. Finally, experiment 3 demonstrated that CAPerMoMa generalizes well to new objects, further validating its flexibility in handling different and new objects for manipulation tasks.

6.1 Contributions

Our contributions in this thesis are summarized as follows:

- CAPerMoMa System: we introduced CAPerMoMa, a system capable of searching, mapping, navigating, and grasping objects in a tabletop mobile manipulation scenario. The system is fully modular, designed using state machines and ROS nodes, allowing for easy reconfiguration. Simple Python-based state machines are used to compose the system's operational states.

- Continuous Gaze Control: we integrated a continuous gaze control mechanism into the system, enabling MoMa to actively perceive its environment in online. This allows for more accurate object detection and tracking during manipulation tasks.

- Reachability with Navigability: we combined navigability with reachability maps, incorporating MoMa's footprint and obstacles into the planning process. This ensures that MoMa can not only reach its target but also navigate safely around obstacles in the environment.

- Entropy-Based Head Movement: an entropy-based head movement strategy was introduced during the exploration phase. This method directs MoMa's head toward areas with high information gain, aiding the search process.
- Evaluation and Study: a comprehensive evaluation and ablation study of the CAPerMoMa system was conducted. The results demonstrated that the integration of gaze control improved (by reducing) mapping time, which in turn reduced overall execution time.

6.2 Future Work

- Future work can focus on several promising directions to enhance the CAPerMoMa system. One possible extension is to adopt a receding horizon formulation. In our current implementation, we rely on state machines, which may occasionally require transition delays between states. A receding horizon approach would allow for the dynamic computation of task priorities, enabling the system to compute scores for each task in online and prioritize them accordingly, potentially improving the system's responsiveness and reducing overall execution time.
- Another direction for improvement lies in the exploration strategy. While our current setup is effective in indoor environments, developing simpler and more targeted methods for object search could enhance performance. For example, using an object-based information gain approach, where the information gain is computed with the target object in mind rather than the entire scene, could speed up object detection and mapping.
- Additionally, in experiment 1, scenario 3, we observed instances where the robot collided with the table, despite using reachability maps. This issue stemmed from incorrect pose proposals by the grasp detector. A potential research direction would be to incorporate safety and fault-tolerant mechanisms into the system, ensuring that the robot can detect and avoid such issues during grasp execution, ultimately leading to more safe and reliable manipulation.

A

Data Logging

The data logging pseudo code is given below:

```
# Pseudo-code for State Machine with Logging Logic
 import smach
 import time

 # Exploration state logs the time taken for exploring and detecting objects
 class Exploration(smach.State):
     def execute(self, userdata):
         start_time = time.time()
         # Perform exploration tasks here
         elapsed_time = time.time() - start_time
         userdata.exploration_time += elapsed_time
         return 'mapping'  # Transition to mapping

 # Mapping state logs the time taken to map the environment and identify graspable
 objects
 class Mapping(smach.State):
     def execute(self, userdata):
         start_time = time.time()
         # Perform mapping tasks here
         elapsed_time = time.time() - start_time
         userdata.mapping_time += elapsed_time
         return 'manipulation'  # Transition to manipulation

 # Manipulation state logs the time taken for grasping and placing the object
 class Manipulation(smach.State):
     def execute(self, userdata):
         placement_start = time.time()
         # Perform best placement tasks computation
         userdata.placement_time = time.time() - placement_start

         navigation_start = time.time()
         # Goto the  navigation point
         userdata.navigation_time = time.time() - navigation_start

         grasp_start = time.time()
```

```
            # Perform grasp action tasks
            userdata.placement_time = time.time() - placement_start

            return 'done'

    # Main function to execute the state machine and log all time data
    def main():
        pipeline_start = time.time()
        # Initialize the state machine and userdata
        sm = smach.StateMachine(outcomes=['DONE'])
        sm.userdata = {
            'exploration_time': 0,
            'mapping_time': 0,
            'manipulation_time': 0,
            'placement_time': 0,
            'nav_time': 0,
            'grasp_time': 0
        }

        with sm:
            # Add states to the state machine
                # Exploration state
                # Mapping state logs
                # Manipulation state

        sm.execute()
        total_pipeline_time = time.time() - pipeline_start

        # Log the time data into a CSV file
        log_data_to_csv(run_number=1,  # Increment this for multiple runs
                        exploration_time=sm.userdata['exploration_time'],
                        mapping_time=sm.userdata['mapping_time'],
                        placement_time=sm.userdata['placement_time'],
                        navigation_time=sm.userdata['navigation_time'],
                        grasp_time=sm.userdata['grasp_time'],
                        total_time=total_pipeline_time)

    if __name__ == '__main__':
        main()

```

Code Listing A.1: Data logging strategy pseudo code.

B

Description of AI-Generated Content

We use AI-generated content in all of our chapters. All the AI content is carefully reviewed before including it in our work. The AI tools used in this work are:

- Grammarly: We use Grammarly mainly to check grammatical errors, spell checks, and sentence corrections. The paid version includes a built-in AI model, which we use.
- GitHub Copilot: We use Copilot for automatic code completion and suggestion generation. We also use Copilot for debugging our code.
- ChatGPT: is an AI chatbot. It is used for paraphrasing in several chapters. First, we input the paragraph we wish to write and give it as a query to correct grammatical errors and paraphrase. Over-emphasised sentences are discarded. We also use it to generate captions for our images.

Use of AI-generated content in our code:

- We used ChatGPT, along with Copilot to develop and debug issues for few sections of our code described in Section 4.1, Section 4.2.1, and Section 4.2.4.

References

[1] iRobot Corporation, "Roomba," https://www.irobot.com/, 2024, accessed: 2024-09-23.

[2] C. Müller, W. Kraus, B. Graf, and K. Bregler, "World robotics 2023 – service robots," IFR Statistical Department, VDMA Services GmbH, Frankfurt am Main, Germany, Tech. Rep., 2023.

[3] J. A. Gonzalez-Aguirre, R. Osorio-Oliveros, K. L. Rodriguez-Hernandez, J. Lizárraga-Iturralde, R. Morales Menendez, R. A. Ramirez-Mendoza, M. A. Ramirez-Moreno, and J. d. J. Lozoya-Santos, "Service robots: Trends and technology," *Applied Sciences*, vol. 11, no. 22, p. 10702, 2021.

[4] S. Jauhri, S. Lueth, and G. Chalvatzaki, "Active-perceptive motion generation for mobile manipulation," *arXiv preprint arXiv:2310.00433*, 2023.

[5] J. M. Garcia-Haro, E. D. Oña, J. Hernandez-Vicen, S. Martinez, and C. Balaguer, "Service robots in catering applications: A review and future challenges," *Electronics*, vol. 10, no. 1, p. 47, 2020.

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

[7] HBRS, "Lucy's simple domestic trials," YouTube video, 2020, accessed: 2024-09-29. [Online]. Available: https://www.youtube.com/watch?v=i6VbOCcqcFk

[8] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos, "Revisiting active perception," *Autonomous Robots*, vol. 42, pp. 177–196, 2018.

[9] S. Chitta, E. G. Jones, M. Ciocarlie, and K. Hsiao, "Mobile manipulation in unstructured environments: Perception, planning, and execution," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 58–71, 2012.

[10] S. Jauhri, J. Peters, and G. Chalvatzaki, "Robot learning of mobile manipulation with reachability behavior priors," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 8399–8406, 2022.

[11] B. Hamner, S. Koterba, J. Shi, R. Simmons, and S. Singh, "An autonomous mobile manipulator for assembly tasks," *Autonomous Robots*, vol. 28, pp. 131–149, 2010.

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

[13] F. Reister, M. Grotz, and T. Asfour, "Combining navigation and manipulation costs for time-efficient robot placement in mobile manipulation tasks," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9913–9920, 2022.

[14] Z. Wang, H. Chen, and M. Fu, "Whole-body motion planning and tracking of a mobile robot with a gimbal rgb-d camera for outdoor 3d exploration," *Journal of Field Robotics*, vol. 41, no. 3, pp. 604–623, 2024.

[15] A. Batinovic, A. Ivanovic, T. Petrovic, and S. Bogdan, "A shadowcasting-based next-best-view planner for autonomous 3d exploration," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2969–2976, 2022.

[16] D. Duberg and P. Jensfelt, "Ufoexplorer: Fast and scalable sampling-based exploration with a graph-based planning structure," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2487–2494, 2022.

[17] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, "Efficient autonomous exploration planning of large-scale 3-d environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1699–1706, 2019.

[18] D. Deng, R. Duan, J. Liu, K. Sheng, and K. Shimada, "Robotic exploration of unknown 2d environment using a frontier-based automatic-differentiable information gain measure," in *International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2020, pp. 1497–1503.

[19] C. Connolly, "The determination of next best views," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 1985, pp. 432–435.

[20] J. Huang, B. Zhou, Z. Fan, Y. Zhu, Y. Jie, L. Li, and H. Cheng, "Fael: fast autonomous exploration for large-scale environments with a mobile robot," *IEEE Robotics and Automation Letters*, vol. 8, no. 3, pp. 1667–1674, 2023.

[21] O. Porges, "Analysis and applications of reachability and capability maps for robotic manipulators," Lulea Technical University - Julius-Maximilians University Wurzburg, Tech. Rep., April 2014. [Online]. Available: https://elib.dlr.de/93767/

[22] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, USA, 2010.

[23] M. A. Roa, K. Hertkorn, F. Zacharias, C. Borst, and G. Hirzinger, "Graspability map: A tool for evaluating grasp capabilities," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 1768–1774.

[24] 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.

[25] R. Bajcsy, "Active perception," *Proceedings of the IEEE*, vol. 76, no. 8, pp. 966–1005, 1988.

[26] O. Brock, J. Park, and M. Toussaint, "Mobility and manipulation," *Springer Handbook of Robotics*, pp. 1007–1036, 2016.

[27] R. B. Rusu, A. Holzbach, R. Diankov, G. Bradski, and M. Beetz, "Perception for mobile manipulation and grasping using active stereo," in *IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2009, pp. 632–638.

[28] S. Xie, C. Hu, D. Wang, J. Johnson, M. Bagavathiannan, and D. Song, "Coupled active perception and manipulation planning for a mobile manipulator in precision agriculture applications," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 12 665–12 671.

[29] A.-D. Mezei and L. Tamas, "Active perception for object manipulation," in *International Conference on Intelligent Computer Communication and Processing (ICCP)*. IEEE, 2016, pp. 269–274.

[30] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.

[31] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, "Development of the research platform of a domestic mobile manipulator utilized for international competition and field test," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7675–7682.

[32] Toyota Motor Corporation, "Human support robot (hsr)," https://hsr.io/, 2024, accessed: 2024-09-09.

[33] M. Grinvald, F. Furrer, T. Novkovic, J. J. Chung, C. Cadena, R. Siegwart, and J. Nieto, "Volumetric instance-aware semantic mapping and 3d object discovery," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3037–3044, 2019.

[34] 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.

[35] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[36] Ultralytics, "Ultralytics yolov8 github repository," https://github.com/ultralytics/ultralytics, 2023, accessed: 2024-09-22.

[37] S. Otte, J. Kulick, M. Toussaint, and O. Brock, "Entropy-based strategies for physical exploration of the environment's degrees of freedom," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 615–622.

[38] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *International Conference on Robotics and Automation*. IEEE, 2010, pp. 300–307.

[39] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

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

[41] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.