Physics-Based Multi-Object Tracking for Non-Prehensile Robotic Manipulation

Zisong Xu

Department of Computer Science

University of Leeds

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy

19th August 2025

Dedicated to my parents who gifted me the miracle of life,
to my friends who accompanied me through life's night,
and to Xingxin who let me know the meaning of:

``Some of us get dipped in flat, some in satin, some in gloss.
But every once in a while, you find someone who's iridescent.
And when you do, nothing will ever compare.''

from ``Flipped''

Declaration

I confirm that the work submitted is my own, except where work which has formed part of jointly authored publications has been included. My contribution and that of the other authors of this work are explicitly indicated below. I confirm that appropriate credit has been given within the thesis where reference has been made to the work of others.
Some of the results and work presented in this thesis have been published in the following papers:

- Z. Xu, R. Papallas, and M. R. Dogar. "Real-Time Physics-Based Object Pose Tracking during Contact-Based Manipulation," in Embracing Contacts-Workshop at ICRA 2023.
- Z. Xu, R. Papallas, and M. R. Dogar. "Physics-based object 6d-pose estimation during non-prehensile manipulation," in International Symposium on Experimental Robotics, Springer, 2023, pp. 181–191.
- Z. Xu, R. Papallas, J. Modisett, M. Billeter and M. R. Dogar, "Tracking and Control of Multiple Objects during Non-Prehensile Manipulation in Clutter." in IEEE Transactions on Robotics, 2025, pp. 3929-3947.

The above publications are primarily my work. Dogar and Papallas helped in an advisory role and proofread the publications. Papallas also helped with the implementation of a controller. Modisett and Billeter helped with the implementation on GPUs. I performed the development, experimentation, and analysis entirely.

Acknowledgements

The completion of this research would not have been possible without deep appreciation, first and foremost, to the People's Republic of China—whose prosperity, development, and openness have provided a strong foundation for overseas students, giving us the confidence to face all challenges. I also sincerely thank the government and academic community of the United Kingdom of Great Britain and Northern Ireland, and the University of Leeds for their inclusive policies toward international students. This cross-cultural academic environment has offered invaluable support to this work.

I owe the completion of this thesis to the support of many people. First, I would like to express my heartfelt thanks to my supervisor, Professor Mehmet Dogar. Throughout my PhD journey, your patience, understanding, and encouragement helped me overcome countless challenges. I will always remember what you said when I first started: "Life is more than just study." That simple sentence helped me understand the true meaning of life. You've shown me what it means to be not only an excellent academic mentor but also a guide in life. It has truly been an honour and a blessing to have you as my supervisor.

Thank you to my thesis examiners, Qian Xie and Wasim Ahmad, for taking the time to attend my PhD viva. I am especially grateful to Wasim Ahmad for travelling from Glasgow to Leeds. Thank you both for your valuable comments and feedback.

I would also like to sincerely thank my colleagues from the Robotics Manipulation Lab: Shengyin Wang, David Mackenzie Charles Russell, Xiao Wang, Shaokang Wu, Longrui Chen, Yulei Qiu, Chen Liu, and Ga Jun Locher. I've learned so much from each of you. I'm truly grateful for your kindness, support, and understanding throughout our time together. It has been a great fortune to have met you all. I'm also very grateful to the many wonderful friends I met during my PhD, including but not limited to Linyan Han, Junlei Hu, Shenghao Qiu and Beite Yang.

I especially want to thank Rafael Papallas, our former postdoctoral colleague. You always encouraged me to reach out whenever I was stuck, no matter how silly the question might have seemed. Whenever I faced setbacks, you reminded me that they are part of life. Thank you so much—I will miss you greatly.

I would also like to thank my dear friends Alessandro Maria Laspina, Shuo Wang, and Yi Chen, who have supported me from afar in Italy and China. Your companionship across time zones has been a constant source of comfort and healing.

I want to thank my family for their constant support over the years. To my parents, Hui Xu and Shaojuan Yang—without your unconditional love and encouragement, I would not have been able to begin, let alone complete, my PhD journey. Every video call across time and distance has been a reminder of how deeply I miss you.

Finally, I want to thank my dear XingXin Lu. I know we are going through a very unique time—you are under academic pressure, and we are in a long-distance relationship, which makes every moment of care and connection even more precious. Every time we talk, you always ask about how I'm doing with such warmth and sincerity. Even when I'm stressed and sometimes show impatience or negativity, you respond with your cheerful smile. I've gained a lot during my PhD journey, but I truly believe the most meaningful part of it has been encountering you.

``愿我六根常寂静，心如宝月映琉璃。"

Abstract

This thesis presents an algorithm to track objects, particularly designed to address the challenge of "tracking objects in cluttered environments." The key point is that the robot can continuously track the target objects, even when they are heavily occluded. This method can be used to retrieve an object from inside a fridge or a crowded shelf, where the robot needs to know the 6D pose of each object in the scene to decide how to move obstacles and then locate and pick up the target item.

Accurately performing the task requires knowing the 6D pose of each object, which enables motion planning. However, obtaining such information in real-world environments is extremely difficult. Traditional motion planning usually assumes access to full 6D poses, often using tools like OptiTrack[1] with reflective markers attached to objects. Unfortunately, placing markers on every object and setting up an OptiTrack system is not feasible in practical situations.

Another option is to use RGB-based 6D pose estimation systems, but these become failures when objects are heavily occluded—something very common in real-world cluttered scenes.

This work addresses these challenges by proposing a tracking algorithm that combines physics predictions with vision information based on the particle filtering algorithm. The system uses the physics simulation and the robot's joint states as the motion model input and camera images as the observation model input. This setup helps recover the object's pose even when it becomes temporarily occluded.

Initially, the method used only RGB images to track a single object. Results showed that the physics-based approach worked well, outperforming baseline methods in tracking accuracy. However, using only RGB input struggled with occlusions and made it hard to handle multiple objects.

To overcome this, I proposed a new approach that uses depth images and a visibility score indicator. This enhanced method performs better than baseline systems when tracking multiple objects and dealing with heavy occlusions, while still maintaining good tracking accuracy.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Robotic manipulation, a cornerstone of modern robotics, aims to endow machines with the ability to physically interact with their environment to perform tasks. From automating complex assembly lines in manufacturing to assisting in household chores, the ambition is to create autonomous systems that can handle objects with human-like dexterity and intelligence. In contrast, significant progress has been made in structured environments like factories. A grand challenge remains: enabling robots to operate reliably in unstructured, cluttered, and dynamic settings, such as a crowded warehouse shelf or a messy kitchen refrigerator. In these real-world scenarios, simple pick-and-place actions are often insufficient. Robots must instead resort to more complex, non-prehensile manipulations—such as pushing, sliding, or poking—to rearrange obstacles and access target objects.

This necessity, however, exposes a critical vulnerability in current robotic perception systems. Consider the seemingly simple task of retrieving an egg from a full refrigerator, as illustrated in Fig. 1.1. The robot might need to push aside a butter dish and a banana to clear a path. During this process, the target egg, initially visible, can become temporarily occluded by the robot arm or other objects.. At this moment, most state-of-the-art, vision-based tracking systems fail because they lose sight of the target. The robot's "eyes" are blind to the object's state, causing the manipulation plan to halt and the task to fail. The issue of "tracking under occlusion" is a fundamental bottleneck preventing robots from achieving robust autonomy in complex manipulation tasks. How can a robot continue its task with confidence when it cannot see the object it intends to manipulate?

This thesis posits that the key to overcoming this challenge lies in physics-based perception. Humans intuitively understand that an occluded object does not simply vanish; it abides by the laws of physics. By integrating a physics simulator into the perception loop, a robot can reason about the hidden object's state, predicting its motion based on the robot's actions and physical principles like contact, friction, and gravity. This approach allows the system to maintain a probabilistic belief

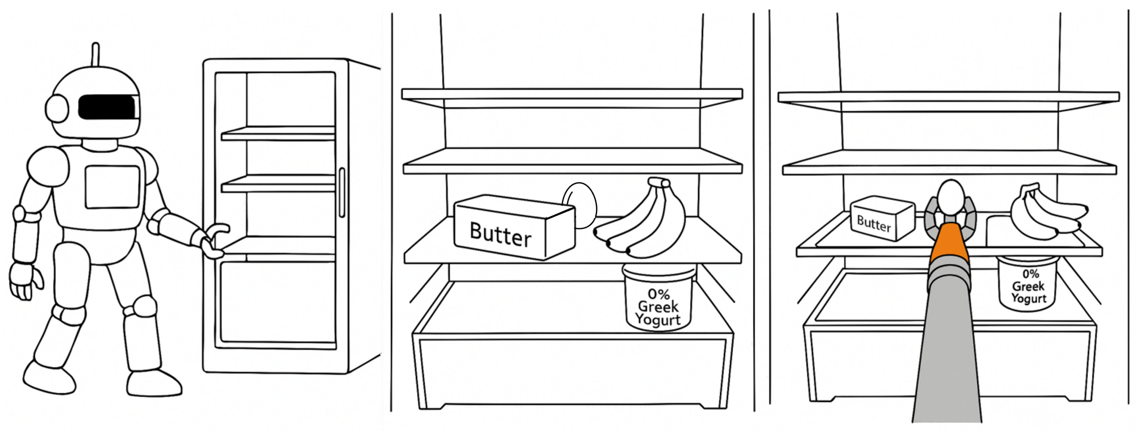


Figure 1.1: The robot needs to retrieve an egg from the fridge, but the egg is blocked by butter and a banana. To reach it, the robot must first push them aside.

of the object's 6D pose (position and orientation) even during complete occlusion, bridging the gap left by vision-only methods.

The central research question of this work is therefore: Can the integration of physical simulators into a tracking framework significantly improve a robot's ability to track multiple objects, especially those fully occluded, during non-prehensile manipulation in cluttered environments? This thesis aims to answer this question by developing and evaluating a series of novel tracking algorithms that leverage physics-based prediction to achieve robust and real-time performance.

1.1 Thesis Scope and Terminology

Consider the example in Fig. 1.1: A robot autonomously navigates to the fridge, opens the door, identifies the target object (an egg located deep inside the fridge) and obstructing obstacles (butter and bananas in front). It then formulates a manipulation plan to push aside the obstacles, safely grasp and retrieve the target object (the egg), close the fridge door, and depart.

While executing such actions is trivial for humans—who perform similar or even more complex tasks daily—it remains extraordinarily challenging for robots. The process involves numerous interdependent challenges: navigation, grasping, perception, motion planning, and handling uncertainties. My doctoral research focuses on one critical aspect of this process: tracking multiple objects subjected to non-prehensile manipulation by the robot during operational tasks. In the example of Fig. 1.1, my work specifically targets tracking the displaced butter and bananas, as well as the potentially affected egg, when the robot reaches into the fridge.

To maintain a clear focus on this core problem, this thesis makes the following simplifying assumptions, as these areas constitute independent and complex research fields on their own:

1. **Navigation:** This thesis assumes the robot is fixed on a tabletop, thus excluding challenges related to mobile navigation.
2. **Motion planning and control:** This thesis assumes the robot's actuators are either manually controlled or governed by well-established control algorithms, focusing on the perception aspect rather than the generation of motion trajectories.
3. **Grasping:** This thesis assumes the robot gripper can reliably manipulate/push target objects, enabling simplified force analysis while focusing on perception.

1.2 Main Themes and Challenges

The central theme of this thesis is the development of a physics-based perception system to robustly track object poses during robotic manipulation, especially under severe occlusion. Fig. 1.2 illustrates a real-world example of the task, where a robot manipulates four target objects in a cluttered scene. As shown, the objects frequently occlude one another or are occluded by the robot itself.

Addressing this problem requires tackling several significant challenges:

1. **High-dimensional state space:** The presence of multiple objects inherently raises the state space dimensionality. Tracking requires simultaneous estimation of both the robot's joint state and the 6D poses of all target objects.
2. **Indirect interaction:** In some scenarios, the robot does not directly control the target objects but influences them indirectly through contact with other objects, creating complex kinematic and dynamic couplings.
3. **Occlusion-related loss of visibility:** This is the core challenge. When a target object is occluded, it disappears entirely from the camera's view, causing all image-only based methods to fail.
4. **Computational cost of physics-based perception:** Physics-based perception necessitates computationally expensive physical simulations to infer object motions during robot pushing, limiting real-time applicability.

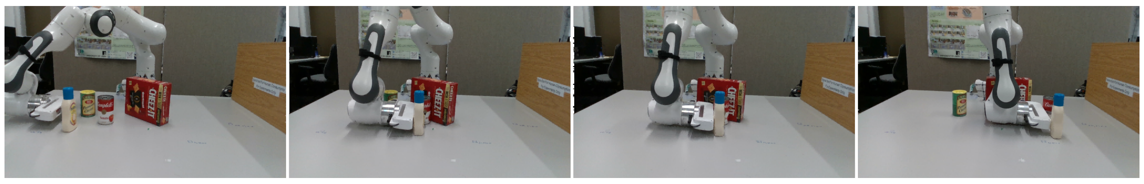


Figure 1.2: Robot is pushing 4 objects. The images are from the tracking camera, showing frequent occlusions among objects and by the robot arm.

5. **Learning-based physics simulation:** While neural network-based physics models can estimate object poses efficiently, they need to be retrained for different scenes or tasks, which limits their ability to generalise.

6. **Physical uncertainties:** Even state-of-the-art physics simulators cannot perfectly model real-world interactions. Consequently, the poses of manipulated objects inferred from simulators remain partially unreliable.

This thesis addresses these challenges by developing new physics-based tracking algorithms designed to overcome them, forming the core of my research contributions.

1.3 Aim

This work aims to investigate how physics-based perception can be integrated into tracking systems to effectively address the problem of "tracking occluded target objects during robotic manipulation" and tackle the challenges outlined in Sec. 1.2. A central question driving this work is: Can the integration of physical simulators significantly improve tracking accuracy for objects in cluttered environments? The proposed methodology emphasizes leveraging physics-based simulators. Consequently, I further explore how to scale the number of tracked objects while maintaining accuracy and real-time performance.

Advancing research on "tracking fully occluded target objects during robotic manipulation" is critical for enabling trajectory planning algorithms to generate more reliable motion plans. This problem holds significant short-term application potential in domains such as warehouse robotics (e.g., retrieving items from shelves) and domestic service robots (e.g., fetching objects from refrigerators). However, existing algorithms remain incapable of resolving the occluded target tracking" challenge in real-world settings with both speed and robustness. To the best of my knowledge, no prior work has systematically explored physics-based perception systems for this specific problem context.

1.4 Contributions

The main contributions of this thesis are as follows:

1. **Integration of physics-based prediction with RGB images for single-object tracking (Chapter 3).** This initial contribution establishes a foundational framework that fuses particle filter-based tracking with a physics engine. It serves as a proof-of-concept, demonstrating that even with a simple sensor modality (RGB), physics-based prediction can successfully maintain

the track of a single object during periods of full occlusion, validating the core hypothesis of this thesis.

2. **Extension to multi-object tracking by combining physics-based prediction with RGB images and visibility scoring (Chapter 4).** Building upon the first contribution, this work scales the problem to multiple objects. It introduces a novel visibility scoring mechanism that weights the contributions of visual data and physics predictions. This allows the system to robustly track multiple interacting objects.

3. **Enhanced accuracy for multi-object tracking using rendered depth and segmentation images, achieved through a CPU-GPU parallel computing framework that maintains system efficiency (Chapter 5).** This final contribution significantly enhances the system's accuracy and robustness by incorporating a depth sensor. To overcome the immense computational cost of processing this data and running physics simulations for multiple objects in real-time, a novel CPU-GPU parallel computing architecture is developed. This ensures the system remains efficient and practical for real-world robotic applications.

The results of this research demonstrate that physics-based perception significantly improves a robot's ability to track occluded objects.

1.5 Structure of the Thesis

This introductory chapter has explained the motivation behind this research, introduced the core research themes, and summarized its main contributions. The remainder of the thesis is organised as follows. Chapter 2 will provide detailed explanations of key concepts and review related research in robotic perception and physics-based modelling. The technical core of this work is presented in Chapters 3 to 5, where the proposed methods are progressively developed and analyzed. Finally, Chapter 6 concludes the thesis by summarising the findings and discussing potential future research directions.

1.6 Publication Note

1. Chapter 3 content appears in:
   (a) Published workshop paper: Embracing Contacts-Workshop at International Conference on Robotics and Automation (ICRA) 2023
   (b) Source code: https://github.com/ZisongXu/trackObjectWithPF
   (c) Demonstration video: https://youtu.be/srZZM_CKum4

2. Chapter 4 content appears in:
   (a) Published conference paper: International Symposium on Experimental Robotics
   (b) Source code: https://github.com/ZisongXu/trackObjectWithPF
   (c) Demonstration video: https://youtu.be/B9-5iTwgFnk
3. Chapter 5 content appears in:
   (a) Published journal paper: The IEEE Transactions on Robotics
   (b) Source code: https://github.com/ZisongXu/PBPF
   (c) Demonstration video: bhttps://youtu.be/7Y8KFVrvDhU

Chapter 2

Literature Review

This chapter will explore the literature review and background research essential for understanding and innovating in robotic non-prehensile manipulation and 6D pose estimation and tracking. Specifically, this chapter will cover a range of foundational and advanced knowledge that form the basis for the thesis:

- Section 2.1 will introduce the fundamentals of robotic manipulation, distinguishing between prehensile and non-prehensile manipulation techniques, and discussing their applications and challenges.
- Section 2.2 focuses on the methods used for object 6D pose estimation and tracking, a key element for precise object manipulation.
- Section 2.3 will discuss the probabilistic models and filtering methods to reduce uncertainty in robotic systems, and the critical role of filtering algorithms in enhancing the robustness of robot localization and pose estimation.
- Finally, Section 2.4 highlights the importance of tracking objects during non-prehensile manipulation, identifying industrial relevance, technological advancements, and current research gaps.

This chapter introduces the scope of the current challenges and highlights the methodologies and innovations that support my research efforts. It sets the theoretical foundation necessary for the subsequent exploration of new algorithms and systems designed to enhance the capability and efficiency of tracking multi-objects in clutter environments by robotic manipulation.

2.1 Fundamentals of Robotic Manipulation

The human hand has incredible abilities that have been essential for human evolution and progress [2]. While this dexterity was once critical for basic survival, its modern form is demonstrated in the complex, everyday challenge of navigating a

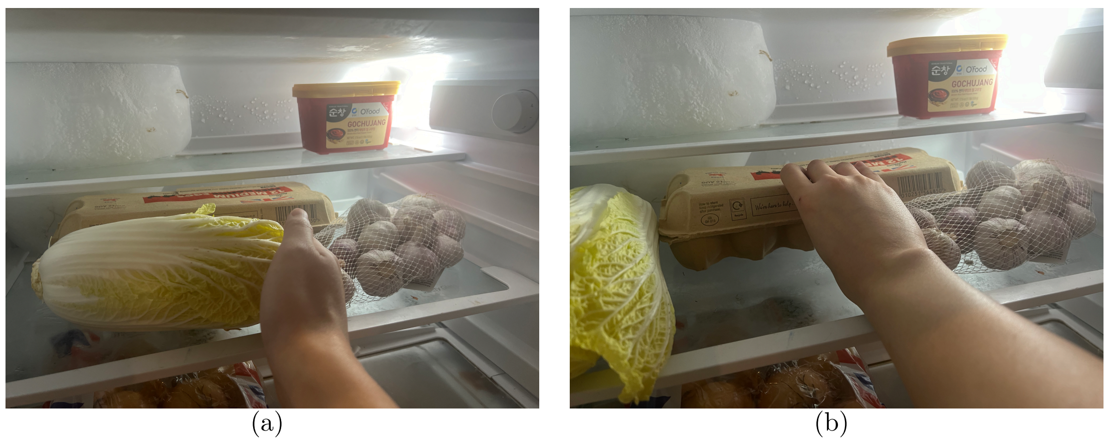
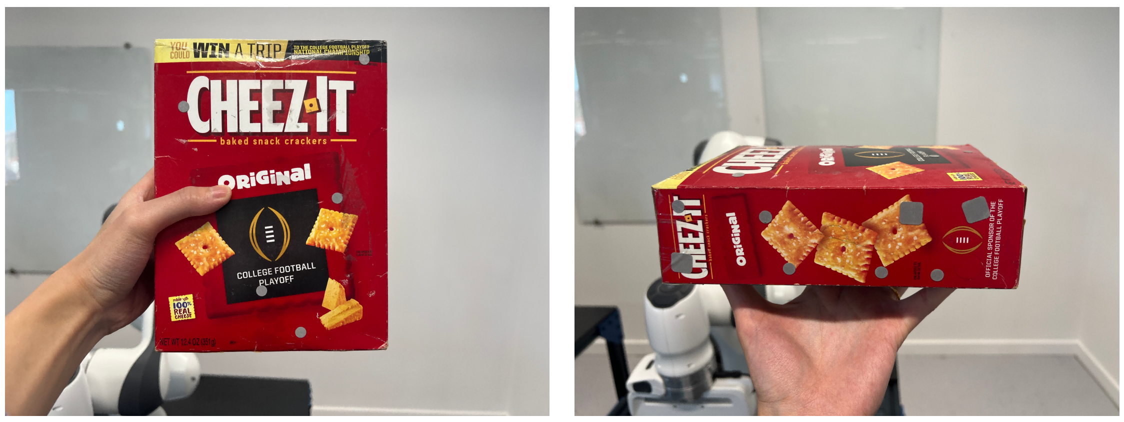


Figure 2.1: An example of complex human manipulation in a cluttered domestic environment. (a) A non-prehensile action is first required to push aside an obstacle (the Chinese lettuce). (b) The target object (the egg carton) can then be accessed and retrieved with a grasp.

cluttered space—for instance, by pushing aside an obstacle to retrieve a desired item from a full refrigerator (Fig. 2.1). This amazing ability to handle objects, adapting seamlessly between forceful interaction and delicate grasping, has inspired the field of robotics, especially robotic manipulation, where researchers aim to copy the precision and adaptability of human hands. Robotic manipulation focuses on allowing robots to handle and interact with objects in various and unpredictable environments, imitating the complex functions of the human hand. Using advanced control systems, sensory feedback, and mechanical design, robotic manipulation aims to close the gap between human hand skills and robotic automation, allowing robots to take on tasks like industrial assembly, healthcare, and helping in homes.

Inspired by the incredible capabilities of the human hand, robotic manipulation aims to address the key challenges of replicating its flexibility and precision. Although significant progress has been made, achieving human-like adaptability in robots is still a work in progress, especially in unpredictable and changing environments.

Robotic manipulation constitutes a core area of study within robotics [3], primarily concerned with the methods and mechanisms of robotic systems interacting with the physical world. This discipline is dedicated to equipping robots with the capability to execute tasks that involve direct physical contact with objects in their environment [4]. These tasks include actions such as moving, manipulating, and sensing these objects as part of their operational functions. According to Siciliano and Khatib [5], effective manipulation is crucial for the functionality and utility of robots, impacting a wide range of applications ranging from industrial automation to healthcare, and from service environments to exploration tasks.



(a) (b)

Figure 2.2: (a) Human prehensile manipulation: the cracker box is grasped by the human hand; (b) Human non-prehensile manipulation: the cracker box is placed on the human hand.

The efficacy of robotic manipulation relies on integrating advanced sensors, precise control systems, and adaptive algorithms that enable robots to perceive and interact with the environment in real time [3]. This integration facilitates the development of robots capable of complex and adaptive behaviours, which are necessary for operating in unstructured or dynamic environments [6]. For example, in Amazon's warehouses, robots equipped with these capabilities can assist in picking operations, such as selecting items from shelves and placing them into containers, or sorting items, thereby enhancing efficiency and safety. The Amazon Picking Challenge [7] was established to address these tasks, supporting advancements in robotic capabilities for warehouse operations.

In robotics, manipulation techniques are broadly divided into prehensile (Fig. 2.2(a)) and non-prehensile (Fig. 2.2(b)) categories. The choice of technique largely depends on the task requirements, the characteristics of the objects, and the surrounding environment. Prehensile manipulation [8], which involves gripping and holding, allows for exact control over an object's movement. Conversely, non-prehensile manipulation [9], such as pushing or sliding, is typically used when direct grasping is either infeasible or hazardous.

In the following section, I first introduce prehensile manipulation, covering key studies in this area. Subsequently, in Section 2.1.2, I focus on non-prehensile manipulation, providing an overview of related research.

2.1.1 Prehensile Manipulation

Prehensile manipulation refers to a robot's ability to securely grasp and control objects, mirroring the dexterity of the human hand (Fig. 2.2(a)). This method is foundational for tasks requiring high fidelity, as it allows a robot to establish a firm hold, ensuring the object's movement is directly coupled with the motion of the robot's end-effector. Its importance is well-established in industrial settings for precision tasks like component assembly [10]. Recent technological advancements have significantly enhanced these capabilities. These range from adaptive grippers that can dynamically modulate gripping force to suit different objects [11], to the advent of soft robotics, which has introduced compliant grippers capable of conforming to irregular object shapes, thereby increasing adaptability and safety [12].

This capability is the backbone of modern automated logistics and e-commerce fulfilment, where robots must handle a vast and unpredictable array of items for sorting, packing, and retrieving from shelves [7, 13]. However, executing a seemingly simple pick-and-place action is a multi-faceted challenge contingent on a cascade of successful steps. First, the system must perceive the scene to obtain an accurate initial pose estimation of the target object, as any error can lead to a failed grasp [14]. Following this, it requires sophisticated motion and grasp planning to determine a viable grasp configuration from countless possibilities, considering factors like stability and task constraints [15]. Furthermore, the physical properties of the object, such as its estimated weight, must be factored in to ensure the robot can lift it [2], while the application of appropriate gripping force is critical to secure the object without causing damage [16].

The complexity escalates dramatically in cluttered environments [17]. When a target object is surrounded by obstacles, a robot must find a collision-free path to perform the grasp [18]. The conventional prehensile-only solution—sequentially picking and placing each obstacle to clear a path—is often inefficient and dramatically increases overall task time. As discussed in the next section, attention has shifted towards non-prehensile manipulation strategies, where obstacles are pushed aside to more efficiently facilitate the subsequent handling of the target object.

2.1.2 Non-prehensile Manipulation

Non-prehensile manipulation involves controlling the motion of objects without grasping them firmly, typically through methods like pushing, sliding, rolling or tilting [19], as shown in Fig. 2.2(b). Non-prehensile manipulation requires complicated control strategies to manage the dynamics of indirect object interactions, where the robot must effectively influence object motion through limited contact points [20]. This manipulation style is crucial in environments where objects are impractical to grasp due to size, shape, or fragility constraints and where delicate environmental interaction is required. For example, Dogar and Srinivasa [21] propose a robotic manipulation framework that enables the robot to push aside large objects (do not conform to the hand's grasp), to access and grasp the desired object, as shown in Fig. 2.3.

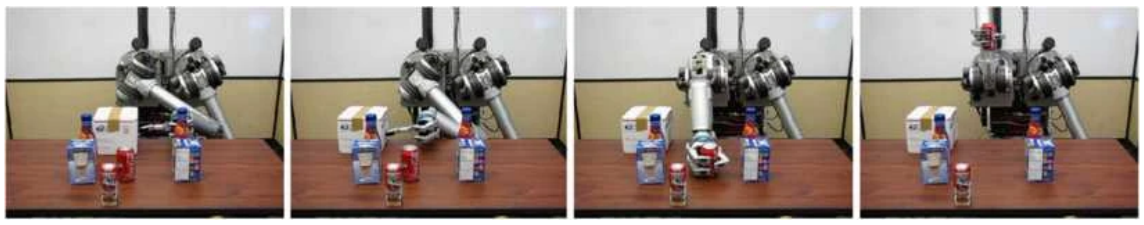


Figure 2.3: The robotic arm attempts to push the large box aside to grasp a red can positioned behind it [21] (figures, from left to right, represent sequential frame segments of the robot manipulation).

Non-prehensile manipulation relies heavily on understanding the dynamics and mechanics of object movement. Mason [22] provided the initial theoretical foundation by describing the mechanics of pushing objects, which includes analysis of the friction and contact points that influence motion. Lynch and Mason [23] developed a planner that explores paths for stably pushing large parts through the utilization of multiple contact points. Zhou et al. [24] discussed integrating feedback mechanisms and predictive modelling in non-prehensile manipulation, allowing for more refined control over object movement. Further, Yu et al. [25] provide a high-fidelity dataset of pushing interactions to understand better and model the physics of pushing, which is useful for developing and testing predictive models for non-prehensile manipulation.

In prehensile manipulation, once the target object is grasped, it remains affixed to the end-effector of the robot's arm, moving in conjunction with it until placed in a designated area. Thus, considerations regarding the object's 6D pose are unnecessary after grasping. In contrast, non-prehensile manipulation presents greater challenges. For example, if the robot makes a single contact point with an object to push it, the object might shift left or right unpredictably [26]. In such cases, developing reliable predictive models becomes important to accurately forecast the outcomes of interactions between the robot and the object. Completing subsequent non-prehensile manipulation tasks is facilitated by tracking the object's 6D pose.

2.2 6D Pose Estimation and Tracking of Objects

6D object pose estimation and tracking are fundamental to robotic manipulation, providing the spatial information necessary for interaction [27, 28]. Many advanced manipulation studies bypass this perception challenge by assuming known object poses, often relying on external motion capture systems like OptiTrack which require placing reflective markers on every object [29, 30, 31]. However, instrumenting every object in a general work environment is impractical. This necessitates markerless methods that can estimate and track object poses directly from sensor data.

Historically, 6D pose estimation relied on matching hand-crafted geometric features, such as point pairs or surface normals, between a 3D model and sensor data [32, 33]. While foundational, these methods often struggle with cluttered scenes and poor textures, and have now been largely superseded by more robust learning-based techniques.

The advent of deep learning has revolutionized pose perception. Many methods operate directly on RGB images, learning powerful features to overcome challenges like variable lighting and textureless surfaces. For instance, PoseCNN [34] first segments the object to isolate it from the background and then regresses its pose by analyzing features within the segmented region. Similarly, segmentation-based approaches like the one from Hu et al. [35] predict the pose of visible object parts and then vote to form a consensus on the complete object's pose, allowing for some robustness to partial occlusion. Other methods like DOPE [36] take a different, single-shot detector approach, predicting the 2D projections of an object's 3D bounding box corners and solving for the pose.

To further improve robustness, many systems incorporate depth information from RGB-D sensors, which provides direct geometric cues and helps resolve scale ambiguities inherent in RGB-only methods. DenseFusion [37] stands out as a seminal work in this area, introducing a novel architecture to fuse color and depth features at a pixel-wise level for a unified representation. Other works, like MegaPose [38], have explored using synthetic data and differentiable renderers to generalise to novel objects not seen during training, though this can be computationally intensive.

Building on single-frame estimation, tracking methods further enhance performance by leveraging temporal consistency. These approaches utilise information from previous frames to predict and constrain the pose in the current frame, showing greater resilience to partial occlusion and fast motion. One prominent line of work involves optimising a "bundle" of recent frames. For example, BundleTrack [39] and the recent state-of-the-art FoundationPose [40] track objects by matching keypoints and refining poses over a sliding window of keyframes.

Particle filters are another powerful and widely-used framework for pose tracking, as they can naturally represent the uncertainty in an object's state through a distribution of weighted samples. Early works demonstrated real-time performance by implementing the filtering process on a GPU [41]. More recent approaches, like PoseRBPF [42], use a Rao-Blackwellized particle filter to efficiently decompose the 6D pose space. Others have focused on improving the core filtering mechanism itself, for instance by introducing auxiliary particle sets to combat the common issue of particle deprivation [43].

However, despite these significant advances across different methodologies, a critical and shared limitation undermines all these vision-based approaches: they fundamentally depend on the object being continuously or frequently visible to the

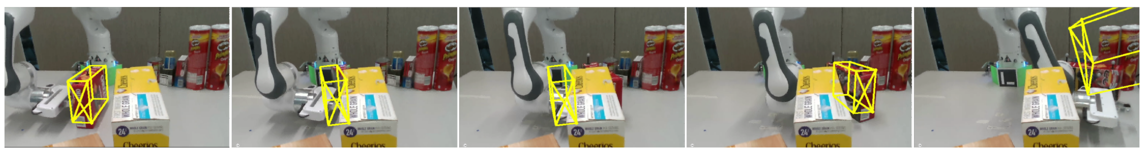


Figure 2.4: The goal of the task shown in the image is to track the red box being manipulated by the robot. The estimated pose of this red box is overlaid as a wireframe onto the image, during a push by a robot. Images show the estimated pose by a learning-based pose estimation system, DOPE [36] (figures, from left to right, represent sequential frame segments of the robot manipulation).

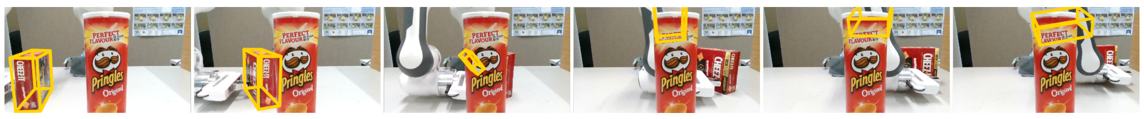


Figure 2.5: The estimated pose of an object is overlaid as a wireframe onto the image, during a push by a robot. Images show the tracking pose by a state-of-the-art camera-only RGB-D pose tracking system, FoundationPose [40] (figures, from left to right, represent sequential frame segments of the robot manipulation).

camera. As demonstrated in my experiments with state-of-the-art systems (Fig. 2.4 and Fig. 2.5), tracking fails catastrophically when an object is fully occluded during manipulation—a common and unavoidable event in cluttered environments. Once the track is lost, these systems often struggle to re-initialize correctly even after the object reappears. This reliance on an uninterrupted stream of visual features creates a major bottleneck for robust robotic manipulation in real-world scenarios.

Some research sidesteps this issue by using non-visual modalities. For example, Zhong et al. [44] use contact-based feedback and a particle filter to track objects without any visual aid. While effective in certain contexts, such methods may lack the precision offered by vision when objects are visible.

This review reveals a clear research gap: a need for a system that can robustly track objects even through periods of complete visual occlusion, yet still leverages visual data when available. This challenge motivates the core idea of my thesis: integrating physics-based prediction into the tracking loop to maintain a belief of an object's pose even when it is partially or entirely hidden from view.

2.3 Probabilistic State Estimation in Robotic Systems

My project aims to track objects manipulated by a robotic arm, using pose information from previous frames to improve tracking accuracy. This concept is based on previous research in robot navigation, particularly robot pose tracking. In this section, I will explore the probabilistic techniques for state estimation that form the foundation of my approach, highlighting why they are crucial for achieving accurate and reliable results in robotic systems.

2.3.1 The Bayesian Filtering Framework

The Bayes filter provides a general probabilistic framework for recursively estimating the state of a dynamic system [45]. This approach is foundational in robotics for handling the uncertainties inherent in sensor measurements and motion models. The core of the filter consists of a two-step cycle to update the belief of the system's state, $bel(x_t)$, from the previous state $bel(x_{t-1})$.

First, in the prediction (or control update) step, the system's motion model is used to predict the current state based on the previous state and any control inputs u_t. This generates a prior belief, $\bar{bel}(x_t)$:

$$\bar{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1} \tag{2.1}$$

Second, in the measurement update step, the new sensor measurement z_t is incorporated to correct the prior belief, resulting in the posterior belief $bel(x_t)$:

$$bel(x_t) = \eta p(z_t|x_t)\bar{bel}(x_t) \tag{2.2}$$

where η is a normalization constant. A well-known application of this framework is Markov Localisation, which specifically applies the Bayes filter to the problem of robot localization within a known map, assuming the current state only depends on the immediate previous state and action [46]. While the Bayes filter provides the theoretical basis, its direct implementation is often intractable. Therefore, various algorithms have been developed that make different assumptions to achieve practical solutions.

2.3.2 Gaussian Filters

For systems that linear models with Gaussian noise can describe, the Kalman Filter (KF) offers a highly efficient, closed-form solution to the Bayesian filtering problem [47]. The KF represents the belief of the state at all times as a Gaussian distribution, defined by a mean and a covariance Σ. The prediction and update steps are performed through simple matrix operations, making it computationally fast and optimal for linear systems. The main drawback, however, is that most real-world robotic systems, including their motion and sensor models, are inherently non-linear.

To address this, the Extended Kalman Filter (EKF) was developed [45]. The EKF extends the KF to handle non-linear systems by performing a local linearization at each time step. It uses a first-order Taylor series expansion (the Jacobian matrix) to approximate the non-linear motion and measurement models with linear ones around the current state estimate. While the EKF is a widely used and powerful

Algorithm 1 Particle Filter Algorithm

Input: $\mathcal{X}_{t-1}$: particle set at time $t-1$; u_t: robotic control at time t; z_t: robotic measurement at time t;

Output: $\mathcal{X}_t$: particle set at time t;

```
 1: function ALGORITHM PARTICLE-FILTER($\mathcal{X}_{t-1}$, $u_t$, $z_t$):
 2:     $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
 3:     for $m = 1$ to $M$ do
 4:         sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
 5:         $w_t^{[m]} = p(z_t|x_t^{[m]})$
 6:         $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + < x_t^{[m]}, w_t^{[m]} >$
 7:     for $m = 1$ to $M$ do
 8:         draw i with probability $\propto w_T^{[i]}$
 9:         add $x_t^{[t]}$ to $\mathcal{X}_t$
10:     return $\mathcal{X}_t$
```

tool, its reliance on linearization can lead to inaccuracies or even divergence if the system is highly non-linear or if the initial state uncertainty is large. The assumption that the belief remains Gaussian can also be violated in such cases.

2.3.3 Particle Filter

To overcome the limitations of Gaussian filters in highly non-linear and non-Gaussian scenarios, the Particle Filter offers a flexible, non-parametric alternative. It is particularly well-suited for global localisation and tracking problems where the probability distribution of the state might be multi-modal or of arbitrary shape [45].

In the particle filter, the samples of a posterior distribution are referred to as particles. This distribution approach involves knowing the outcome first and then estimating the probability distribution of the cause based on the outcome [48]. Here, each particle represents a possible state of the system, providing a discrete approximation of the probability distribution that describes the system's state. Therefore, I can have:

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]} \tag{2.3}$$

Each particle $x_t^{[m]}(1 \leq m \leq M)$ is a concrete instance of the state at time t. A particle represents a possible hypothesis based on the real-world state at the moment t. M is the number of particles in the particle set $\mathcal{X}_t$. I use a series of particles $\mathcal{X}_t$, to approximate the confidence $bel(x_t)$. The particle filter algorithm is stated in algorithm 1.

The particle $x_{t-1}^{[m]}$ generates a hypothetical state $x_t^{[m]}$ at the moment t according to the control u_t (these control data are derived from the number of revolutions of the robot's wheels; I call it odometry model) (Line 4), and then calculate a weight

$w_t^{[m]}$ for each particle $x_t^{[m]}$ based on the observation z_t (these observation data are derived from sensors such as cameras and laser sensors; I call it observation model) at the moment t (Line 5). Each particle is resampled according to the weight $w_t^{[m]}$ and finally the set of particles at moment t is obtained (Lines 8 to 11).

In my project, I plan to use the particle filter algorithm to facilitate completing my work, i.e. tracking objects during robotic non-prehensile manipulation. This choice is driven by the particle filter's ability to handle non-linear state-space models - the set of all possible states of the system [49], and its ability to accommodate any noise distribution [50]. This flexibility makes particle filtering particularly well-suited for my project, especially when obstacles or the robot arm significantly occludes target objects. The system leverages the algorithm to generate a probability distribution over the possible states of the target objects, rather than providing a single pose estimation.

2.4 Remarks

This chapter explores three main areas of this research: (1) types of robotic manipulation, (2) estimation and tracking of object 6D poses, and (3) methods for managing uncertainties in robotic systems. A detailed review of related work presents significant studies in these fields and their differences from my research. I also highlight the challenges in tracking objects under non-prehensile robotic manipulation, especially when objects are largely occluded in cluttered environments.

So far, no research has effectively solved the problem of tracking objects under non-prehensile robotic manipulations when they are largely or completely occluded. Most current methods use neural network-trained algorithms to estimate and track object poses. However, these methods fail when the objects are completely occluded, i.e. they are not visible in the camera's RGB or depth images. My research focuses on this specific but common situation in cluttered environments, such as Amazon warehouses' pick-and-pack operations. The goal is to figure out how to use the continuous motion of objects after a robot pushes them and the information provided when objects are occluded, to develop reliable tracking methods even when the camera can't see them.

In Chapter 3, I propose to use the PyBullet physics engine to represent each particle in the particle filter algorithm (representing possible environments of the target objects in the real world), and to use the features of the physics engine and particle filter to track objects. In Chapter 4, I extend the methods used in the previous chapter to track objects under non-prehensile manipulations using the information from occluded objects. Finally, in Chapter 5, I improve the tracking system by adding depth images from the camera and refining the methods that use information from the occluded object situation. My research results are encouraging, showing that my methods can effectively track occluded objects.

In the next sections, I will discuss using the PyBullet physics engine and RGB camera images to track target objects.

Chapter 3

Physics-Based Single Object 6D-Pose Tracking during Non-Prehensile Manipulation

Chapter Deliverables

Video: https://www.youtube.com/watch?v=7mNbwJCDdBc

Source Code: https://github.com/ZisongXu/trackObjectWithPF

This chapter provides an overview of the algorithm used to track a single object during non-prehensile robotic manipulation. It explains the algorithms, theory, and experimental results, preparing for further discussions on tracking multiple target objects in the upcoming chapter. The chapter is structured as follows:

- Section 3.1 will introduce the foundation of the problem I am trying to solve.
- Section 3.2 focuses on this work's assumptions, objectives and contributions.
- Section 3.3 provides a formal description of the problem to be addressed and introduces the key notations used throughout the chapter.
- Section 3.4 presents the main algorithm, physics-based particle filtering, I use, based on the particle filtering algorithm for tracking a single object.
- Section 3.5 will introduce two baseline methods: (i) Repetitive single-snapshot pose estimation, and (ii) Constant-velocity particle filtering.
- Section 3.6 will review the experiments and assess the algorithm's performance.
- Section 3.7 will conclude this single object tracking algorithm.

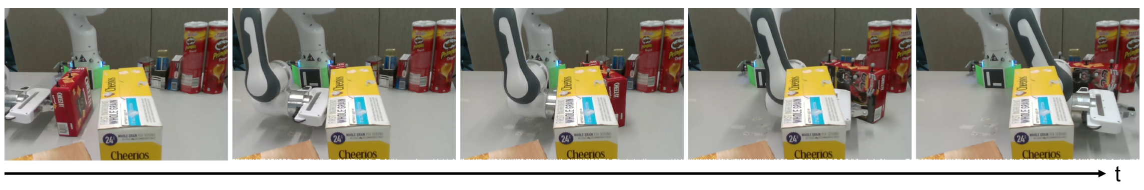


Figure 3.1: Robot pushing an object (red box) among cluttering objects. The images are from the tracking camera.

3.1 Introduction

In this chapter, I propose a framework for tracking a single target object based on a physics engine. Within this framework, the motion of the target objects is determined solely by the robot's movements and physical factors.

Fig. 3.1 demonstrates an example problem. The goal is to track a target object (shown as a red box) in real time during non-prehensile robotic manipulation. The robot moves the object through a cluttered space that obstacles or the robot's arm might obscure. The robot follows a predefined path controlled via a joystick. To achieve this, I start by initializing a particle set at the object's initial pose using a physics engine (each particle in the particle set represents the possible pose of the target object in the real world). As the robot moves, this particle set is updated with observation data from the camera's RGB images, helping us to select particles that most likely reflect the object's actual pose. The chosen particle set indicates the possible object's pose in the real world. A critical focus of Fig. 3.1 is on tracking the object when obstacles or the robot obscures it.

To clarify the terminology used throughout this thesis, a particle set is a data structure—specifically, a set of discrete, weighted samples used to represent a probability distribution. In the context of this work, each particle is a complete hypothesis of the object's 6D pose (position and orientation). The entire set, comprising many such particles, serves as a numerical approximation of my belief about the object's true pose at a single instant. Regions within the state space that have a higher density of particles are considered more probable. The Particle Filter is a recursive Bayesian algorithm that operates on the particle set. The Particle Filter is the complete mechanism that takes the particle set from the previous time step ($t - 1$) as input and produces an updated, more accurate particle set for the current time step (t). It achieves this through its distinct prediction, measurement update (weighting), and resampling steps. In essence, the particle set is the representation of the system's belief, while the Particle Filter is the algorithm that propagates and refines that belief over time.

In the situation shown in Fig. 3.1, when the target object is not visible to the camera, I do not use observation data (from RGB images taken by the camera) for updating particles. Instead, I rely more on how the particles move within the physics engine. Once the target object is visible again in the camera's view, I use visual information again to help update the particles. This method helps us keep tracking accurate, even when the object is not visible, by using the physics engine's predictions to fill in the gaps in observation data.

I compared the algorithm I developed for tracking target objects with learning-based methods that utilize neural networks trained to estimate the 6D pose of objects from single camera snapshots. To facilitate this comparison, experiments were conducted on the real robot using the same environment setting and the same robot movement trajectories. The results of these experiments demonstrated that the physics engine-based algorithm maintained stable tracking even when the target objects were occluded, whereas the learning-based methods failed under such conditions. This highlights the robustness of physics engine-driven approaches in scenes where observation data may be compromised.

3.2 Assumptions, Objectives and Contributions

This chapter outlines key assumptions, Objectives, and contributions to developing this single-object tracking algorithm for robotic non-prehensile manipulation.

3.2.1 Assumptions

In this chapter, I make the following assumptions:

- The target object undergoes various movements during robot manipulation, including translations and rotations in all directions, as well as tilting actions. Consequently, the pose of the object (its 6D pose) is defined within the $SE(3)$ space.
- The robot can manipulate the target object with ease, meaning that the robot is not hindered by excessive friction or the object's weight during the manipulation process.
- 3D CAD models of all objects are known (e.g., the table, static obstacles, robot and target objects).
- There is a perception system that can detect the 6D pose of a single object when it is clearly in the camera's field of view.
- During the pushing of the target object by the robot, the target object is obscured by the robot or other objects fixed on the table.

3.2.2 Objectives

The primary goal of this chapter is to develop a tracking algorithm that integrates a physics engine with a particle filter. This algorithm aims to effectively track the 6D pose of the single target object in cluttered environments during non-prehensile manipulation, even when the object is occluded. Occlusions present a significant challenge for current learning-based pose estimation algorithms, and the use of a physics engine and particle filter is expected to improve tracking performance significantly. This chapter seeks to introduce such a method, evaluate it, and compare it against existing state-of-the-art approaches.

The primary goal of this chapter is to develop and validate a tracking algorithm that produces a continuous 6D pose estimate for a single object undergoing non-prehensile manipulation. To clarify, the 6D pose track is the direct output and primary contribution of the algorithm developed here; the system does not assume a known pose but rather estimates the full 6D pose distribution at each time step.

To achieve this, the chapter pursues the following specific objectives:

- To design and implement a novel tracking framework that synergistically combines a physics-based motion model (for the prediction step) with visual observations from a camera (for the update step) within a particle filter structure.
- To demonstrate the framework's core capability to maintain a robust and continuous belief of the object's 6D pose even during periods of complete visual occlusion, a scenario where conventional vision-only pose estimation and tracking algorithms are known to fail.
- To quantitatively evaluate the proposed algorithm's tracking accuracy and compare its performance against a state-of-the-art, vision-only pose estimation baseline. This comparison serves to rigorously validate the tangible benefits of integrating a physics engine for tracking in cluttered environments.

By addressing these objectives, this chapter aims to provide a foundational proof-of-concept for the central hypothesis of this thesis: that physics-based prediction is a key enabler for robust robotic manipulation in dynamic, real-world settings.

3.2.3 Contribution

The chapter's contribution is a new algorithm for tracking a single object in cluttered environments during non-prehensile manipulations. This section formalises the problem this algorithm aims to solve.

3.3 Problem Formulation

I define the state of the object at time t by its full 6D pose, which I denote as x_t. This pose is an element of the Special Euclidean group, $x_t \in SE(3)$, and is mathematically represented as a 4×4 homogeneous transformation matrix. $SE(3)$ refers to the Special Euclidean group in three dimensions, which is the mathematical set of all possible rigid body transformations. Each such transformation consists of a rotation and a translation, and is commonly represented by a 4×4 homogeneous transformation matrix of the form $T = \left[\begin{smallmatrix} R & t \\ 0 & 1 \end{smallmatrix}\right]$. Here, R is a 3×3 rotation matrix from the Special Orthogonal group ($SO(3)$) that defines the object's orientation, and t is a 3×1 translation vector in $\mathbb{R}^3$ that specifies its position. Although represented by this matrix, a pose in $SE(3)$ has only six degrees of freedom (6-DoF)—three for translation and three for rotation.

My objective is to estimate the sequence of states $\{x_0, x_1, \ldots, x_t\}$ during manipulation, i.e. to track the object pose as it is being manipulated. At each time step t, I have access to two inputs:

- The control input, u_t. This represents the executed robot motion since the last time-step. For an N-joint robot arm, this is a vector of joint commands, $u_t \in \mathbb{R}^N$.
- The observation, z_t. This is an RGB image captured by a static camera looking at the scene. An image is a high-dimensional data structure that serves as the raw input to my perception system.

Therefore, my problem can be formalized as: At any time t, given the history of control inputs $\{u_0, u_1, \ldots, u_t\}$ and observations $\{z_0, z_1, \ldots, z_t\}$, predict an estimate of the object's pose matrix, $\tilde{x}_t$.

I use x_t^* to represent the ground truth pose of the object. I define the positional error as the Euclidean distance between the positional components of $\tilde{x}_t$ and x_t^*. I define the rotational error as the angle of the relative rotation between $\tilde{x}_t$ and x_t^*. Given $\tilde{q}_t$ and q_t^* as the quaternion components of the estimated and ground truth poses, respectively, the rotational error is given by the angle of the quaternion $q_t^* * \tilde{q}_t^{-1}$. My goal is to minimize both positional and rotational errors.

Below, I discuss three different methods (two baselines and my proposed method): (i) Physics-based particle filtering (proposed method, in Sec. 3.4), (ii) Repetitive single-snapshot pose estimation (baseline, in Sec. 3.5.1), and (iii) Constant-velocity particle filtering (baseline, in Sec. 3.5.2).

3.4 Physics-Based Particle Filtering (PBPF)

Instead of treating every time step as independent from the previous one, I propose to use a Bayesian filtering approach [45]. In this approach, at each time step, I estimate and update the probability distribution for x_t given all the previous controls and observations, $p(x_t \,|\, z_0, z_1, ..., z_t, u_0, u_1, ..., u_t)$, which is sometimes also called the *belief state*.

In particle filtering [45], the belief state at time t is represented using a set of *particles* sampled from this distribution:

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]} \tag{3.1}$$

where each particle $x_t^{[m]}$ $(1 \leq m \leq M)$ is a concrete instance of the state at time t. In my setting, each particle represents a possible pose of the manipulated object.

During particle filtering, at each time-step t, the previous set of particles $\mathcal{X}_{t-1}$ are updated using the current controls, u_t, and observation, z_t, to generate a new set of particles $\mathcal{X}_t$. This happens in two stages: the *motion update* (presented in Sec. 3.4.1), and the *observation update* (in Sec. 3.4.2).

3.4.1 Motion Update

During this first stage, for each particle $x_{t-1}^{[m]}$, I generate a new intermediate particle (shown as $\bar{x}_t^{[m]}$), by sampling:

$$\bar{x}_t^{[m]} \sim p(x_t \,|\, x_{t-1}^{[m]}, u_t) \tag{3.2}$$

The probability distribution $p(x_t \,|\, x_{t-1}^{[m]}, u_t)$ is called the *motion model*, and ideally it represents my uncertainty about the object's resulting pose, if the object started at pose $x_{t-1}^{[m]}$ and the robot moved with u_t. While I do not have direct access to this distribution, I estimate via a physics engine.

I assume access to a physics engine, represented as f:

$$x_t = f_\theta(x_{t-1}, u_t) \tag{3.3}$$

The physics engine includes a model of the robot, the environment, and the object, and predicts the resulting pose of the object x_t, given a previous state, x_{t-1}, and robot control, u_t, by simulating the robot motion inside the engine and finding the resulting motion of the object. Here, θ refers to physical parameters that affect the result of the physics engine, e.g. friction coefficient at the contacts, restitution of the contacts, the mass of the object etc. The physics engine is deterministic (i.e. outputs the same resulting state, if given the same inputs and parameters), and therefore cannot directly be used instead of the probabilistic motion model. However, I note that my uncertainty about the object's motion is due to two sources:

(i) my uncertainty about the exact physical parameters of the object, θ; and (ii) the discrepancy between the physics engine and real-world physics. Therefore, to address (i), I approximate the sampling from the motion model $\bar{x}_t^{[m]} \sim p(x_t \,|\, x_{t-1}^{[m]}, u_t)$, by first sampling θ from a distribution representing my uncertainty about the physical parameters of the object:

$$\theta_t^{[m]} \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2) \tag{3.4}$$

and then running the physics engine with the sampled parameter:

$$\bar{x}_t^{[m]} = f_{\theta_t^{[m]}}(x_{t-1}^{[m]}, u_t) + \epsilon \tag{3.5}$$

with the addition of Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_f^2)$ to address (ii) above. (In Eqs. 3.5, 3.11 and 3.12, for notational simplicity, I use the addition/subtraction operators over poses. In my actual implementation, I use quaternion algebra for the rotational components.)

In practice, I instantiate one physics engine per particle and use them to perform the motion update. Since each particle is independent of each other, these updates are parallelizable, which I make use of in my implementation. As such, the set of intermediate particles, $\bar{x}_t^{[m]}$ $(1 \leq m \leq M)$, are computed.

In this work, I limit the physical parameters I sample, θ, to the coefficients of friction, the restitution parameters, and the mass[1] of the object. However, uncertainty about other parameters (e.g. object shape, inertial parameters, robot hand shape, imperfections of the ground) can also be represented and integrated into this framework. In the above, $\mathcal{N}(\mu, \sigma^2)$ represents a normal (Gaussian) distribution[2]. The parameter μ_θ represents my best guess about the parameters θ, and σ_θ^2 represents my uncertainty (variance). The parameter σ_f^2 represents my estimated discrepancy between the physics engine and real-world physics.

3.4.2 Observation Update

During this second stage of particle filtering [45], for each intermediate particle $\bar{x}_t^{[m]}$, I calculate a *weight* $w_t^{[m]}$ using the observation z_t:

$$w_t^{[m]} = p(z_t \,|\, \bar{x}_t^{[m]}) \tag{3.6}$$

After the weight for all the intermediate particles is computed, they are used to *re-sample* the new set of particles $\mathcal{X}_t$, completing the particle filter update. During re-sampling, each intermediate particle $\bar{x}_t^{[m]}$ can be chosen (possibly multiple times) to be added to the new particle set $\mathcal{X}_t$, with a probability proportional to $w_t^{[m]}$.

[1] I assume the inertia tensor of an object to be diagonal and the object to have uniform density within its given 3-D model.

[2] Later in Eq. 3.9, I will also use the notation $\mathcal{N}(x;\, \mu, \sigma^2)$, which corresponds to the probability density at x, for mean μ and variance σ^2.

The expression $p(z_t \,|\, \bar{x}_t^{[m]})$ in Eq. 3.6 is called the *observation model*. In my setting, it ideally represents the probability of making the current observation (i.e. getting the current camera image) if the object was at pose $\bar{x}_t^{[m]}$. Since I do not have access to such a model directly, I again propose to use an approximation. Using the Bayes Theorem, I first re-write the observation model:

$$p(z_t|\bar{x}_t^{[m]}) = \frac{p(\bar{x}_t^{[m]}|z_t)\, p(z_t)}{p(\bar{x}_t^{[m]})} \tag{3.7}$$

Here, I note that $p(z_t)$ is the same for every particle since the current observation does not change between particles. Furthermore, I make a simplifying assumption that $p(\bar{x}_t^{[m]})$ are also similar for different particles. This assumption enables us to compute the weight using:

$$w_t^{[m]} \approx p(\bar{x}_t^{[m]}|z_t) \tag{3.8}$$

To compute $p(\bar{x}_t^{[m]}|z_t)$, I propose to use a single-snapshot pose estimation system to predict the pose of the object according to z_t, and then to use the distance of $\bar{x}_t^{[m]}$ to this predicted pose to compute a probability value. As the single-snapshot pose estimation system, I use DOPE [36], but other pose estimation methods can also be used. Using a notation similar to Eq. 3.10 where $DOPE(z_t)$ is the object pose predicted by DOPE given the camera image z_t, I compute the weight as:

$$p(\bar{x}_t^{[m]}|z_t) = \mathcal{N}(\bar{x}_t^{[m]};\, DOPE(z_t), \sigma^2_{DOPE}) \tag{3.9}$$

where the parameter σ^2_{DOPE} represents the variance of DOPE errors for the object. This can be estimated beforehand for an object by collecting DOPE estimates for the object and comparing it to a ground truth pose.

For a given camera image z_t, if the object is not visible in that image (perhaps because it is obstructed), DOPE may not output a pose estimate. At such time points, I skip the observation update.

3.4.3 Updating the Particle Filter

I update the particle filter at regular time intervals, Δt. Since the physics-based predictions, i.e. my motion update is the most computationally expensive part of the filter, I determine Δt as the smallest time duration within which I can perform the physics simulations for all particles. Note, however, that Δt also affects the motion model: Since, at each step of the particle filter, I want the motion model to move the simulated physical system to the current time, each of my physics simulations is integrated the same amount of simulated time, Δt.

3.4.4 Resampling

Multinomial resampling[51] is performed based on the weights of each particle to generate a new set of particles.

3.4.5 Calculating $\tilde{x}_t$

The particle filter keeps track of all particles $\mathcal{X}_t$ through the duration of manipulation. However, if a single estimate $\tilde{x}_t$ is required at any time t, then a statistic from the particles can be computed. In this work, I use the mean of all the particles in $\mathcal{X}_t$ to compute $\tilde{x}_t$. To find the mean of rotations of the particles, I use the method in Markley et al. [52].

3.4.6 Computational Cost

A primary consideration for the practical application of my proposed method is its computational cost. The use of a physics engine for the prediction step, while providing a probable motion model, is computationally intensive. For each of the M ($M = 70$ in my experiments) particles in the filter, the physics engine must perform a full simulation step. This involves complex operations such as collision detection between the object, the robot, and the environment, followed by solving for contact dynamics (e.g., forces and friction) to predict the motion. This entire process must be repeated for every particle at every time step, making the prediction phase the main computational bottleneck of the filter.

This high computational demand places a hard limit on the filter's update frequency. In our implementation, the physics-based particle filter can achieve an update rate of approximately 5 Hz. In contrast, a filter using a simple analytical motion model, such as a constant velocity model, is computationally trivial and can run at a much higher frequency at 50 Hz, often limited only by the camera's frame rate. A high-frequency filter with a simple model can react quickly to new sensor data but may suffer from large prediction errors between steps, especially during complex interactions. Conversely, our physics-based model provides a high-fidelity prediction that is more accurate over longer time intervals but at a lower update rate, meaning the system is "blind" for periods between sensor updates.

To empirically investigate this trade-off and justify the use of a computationally expensive physics engine, we introduce our second baseline method in the next section: a constant-velocity particle filter (Sec. 3.5.2). This baseline is deliberately designed to be computationally lightweight, thereby maximising its update frequency. By comparing the tracking performance of our high-fidelity, low-frequency physics-based filter against this low-fidelity, high-frequency baseline, we can directly assess whether the significant improvement in motion prediction accuracy is worth the associated computational cost.

3.5 Baseline Methods

In this section, I will introduce two baseline methods that will be used as comparisons for the algorithm I have developed.

3.5.1 Repetitive Single-Snapshot Pose Estimation

One commonly used method in robotic manipulation is to use a system that can estimate the pose of the object from a single-snapshot, i.e. using only the current camera image z_t. There are multiple systems developed to perform object pose estimation given an image, as mentioned in Sec. 2.2. In this paper, I use DOPE [36] as a state-of-the-art deep-learning-based pose estimation system. DOPE has been trained on the YCB objects [53] that I also use, and is publicly available. Therefore, the repetitive single-snapshot pose estimation method, in my case, corresponds to running DOPE at every time-step t during manipulation:

$$\tilde{x}_t = DOPE(z_t) \tag{3.10}$$

This method treats every new observation as independent from the previous time steps, and therefore does not use temporal continuity. The performance particularly degrades when the object is partially or fully obstructed in the camera view. Moreover, it does not use the control information.

3.5.2 Constant-Velocity Particle Filtering

My first baseline method was presented in Sec. 3.5. My second baseline method, presented here, is a particle filter, similar to the one presented in Sec. 3.4, the only difference being in the *motion update* stage. While the particle filter presented in Sec. 3.4 uses a physics engine in its motion model, here, instead I use a computationally cheap motion model, which simply assumes that the object moves with a constant velocity. In other words, during the motion update, I update a particle at time step t using the estimated pose of the object from the previous updating time steps $t-1$ and $t-2$. I first calculate the "difference" between the estimated pose of the object at these previous time steps:

$$\mathrm{d}x = \tilde{x}_{t-1} - \tilde{x}_{t-2} \tag{3.11}$$

and then, I assume the object keeps moving with the same velocity, and use the motion update rule:

$$\bar{x}_t^{[m]} = x_{t-1}^{[m]} + \mathrm{d}x + \epsilon \tag{3.12}$$

where ϵ is an extra noise term, similar to Eq. 3.5. (Eqs. 3.11 and 3.12 again make an abuse of notation by using subtraction/addition over poses. In actual implementation, I use quaternion algebra to subtract/add rotational components.)

3.6 Experiments and Results

Here, I compare the three different methods I have presented on different non-prehensile manipulation scenes.

3.6.1 Scenes

I evaluated and compared the performance of these methods (PBPF, DOPE, and CVPF) in four different non-prehensile manipulation scenes.

Scene 1: Shown in Fig. 3.2(a). The robot pushes an object among the clutter, where the cluttering objects can obstruct the view of the camera.

Scene 2: Shown in Fig. 3.2(b). The robot pushes an object on a clear table, however the hand can obstruct the camera view during pushing.

Scene 3: Shown in Fig. 3.2(c). The robot pushes an object on a clear table, and the camera has a clear view of the object at all times.

Scene 4: Shown in Fig. 3.2(d). The robot tilts an object up and then brings it down. This scene is added to show the method has no limitation to planar pushing. The camera has a clear view at all times.

In the experiments, I used an object from the YCB dataset [53], particularly the CheezIt box. I picked this object because DOPE had a consistent and good estimation of its pose when the object was visible.

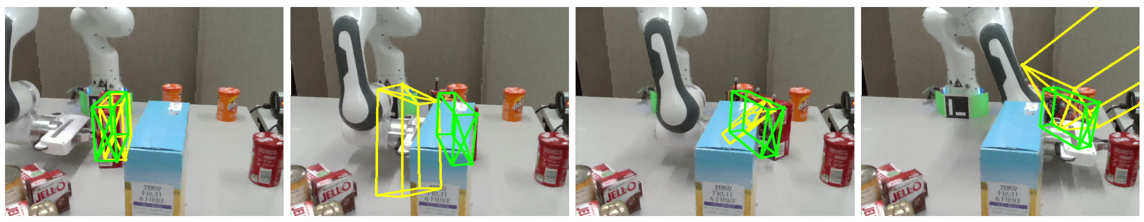
My experimental setup uses a static camera, positioned to maintain a comprehensive and unobstructed global view of the entire workspace. This fixed configuration was deliberately chosen over a wrist-mounted alternative. I found that the field of view of a wrist camera becomes severely restricted during close-contact manipulation. In contrast, my fixed-camera setup ensures a stable and continuous stream of visual data for the entire scene, which is essential for my framework to reliably track the poses of multiple interacting objects.

3.6.2 Implementation Details

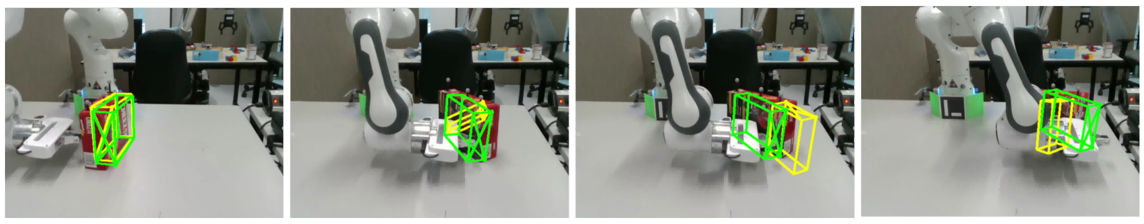
I have implemented the three methods as below[1].

PBPF: The physics-based particle filtering method (Sec.3.4). (a) *Motion model parameters.* μ_θ and σ_θ: Mean friction coefficient of 0.1 and standard deviation of 0.3, with minimum capped at 0.001. Mean restitution of 0.9 and standard deviation of 0.2. The mean mass of 0.38 kg, and the standard deviation of 0.5, with a minimum capped at 0.05 kg. σ_f: For position 0.005 m, for rotation 0.05 rad. As the physics model, f_θ, I used the PyBullet physics engine [54]. A PyBullet environment was initialised for each particle. (b) *Observation model parameters.* σ_{DOPE}: For position

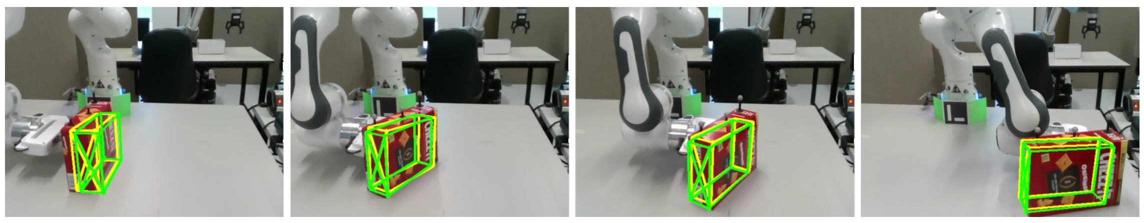
[1]Experiments were performed on CPU: 11th Gen Intel(R) Core(TM) i9-11900@2.50GHz; GPU: NVIDIA GeForce RTX 3090; RAM: 128580 Mb



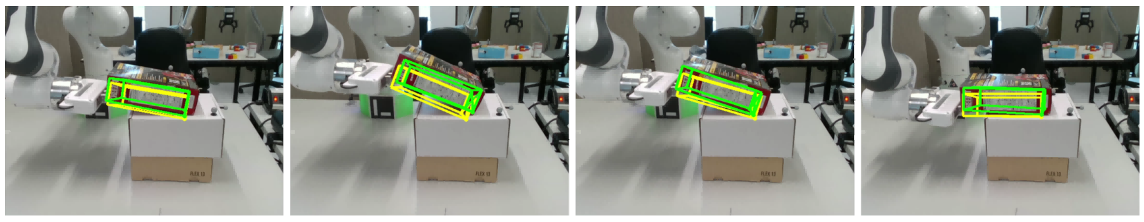
(a) Scene 1



(b) Scene 2



(c) Scene 3



(d) Scene 4

Figure 3.2: Robot manipulating an object (red box) among cluttering objects. Shown images are from the tracking camera. The estimated object pose of my proposed method (PBPF) is shown as a green wireframe box. The estimated object pose from a vision-only pose estimation system (DOPE) is shown as a yellow wireframe box. Please note wireframe boxes are overlayed on the images as post-processing. While the green box appears on top of the obstacle/robot hand in some images, this is only an artefact of the way I overlay these wireframe boxes. The pose estimates are in fact behind the obstacle/robot hand in these instances, close to the actual object pose. The physics-based method (green wireframe) performs better than others when the camera view is partially or completely obstructed, while it performs similarly to other methods when the camera view is always clear.

0.02 m and rotation 0.09 rad. (c) *Update frequency.* $\Delta t = 0.16s$. (d) *Number of particles.* $M = 70$. I tested other numbers of particles. Fig. 3.3 shows that the PBPF error decreases as the number of particles increases. Due to CPU/GPU limitations, the maximum number of particles that can allow the PBPF algorithm

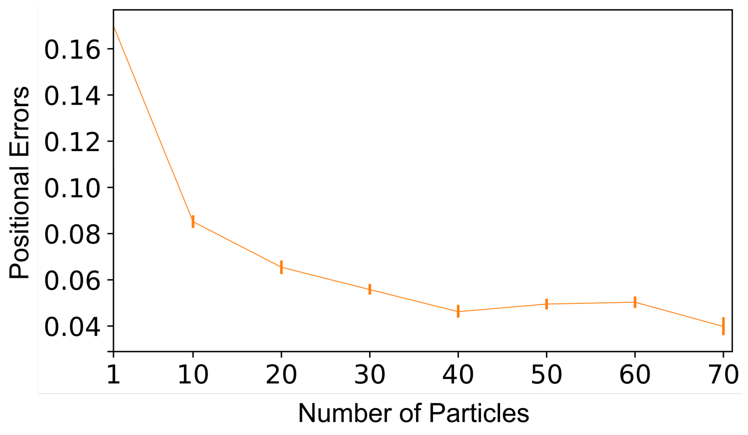


Figure 3.3: Effect of the number of particles on performance.

to run in real-time is 70. (e) *Initialization.* I use a Gaussian distribution to initialise 70 particles at $t = 0$. The mean is pose estimated by DOPE at $t = 0$. The standard dev. for particle initialisation for x, y, z axes and rotation are 0.07 m, 0.02 m, 0.01 m and 0.04 rad, respectively.

DOPE: The repetitive single-snapshot pose estimation method (Sec. 3.5.1). I used official DOPE implementation[1].

CVPF: The constant-velocity particle filtering method (Sec.3.5.2). All parameters were the same with PBPF, except the number of particles, M, and update interval Δt. Since CVPF is computationally much cheaper, for a fair comparison, I used more particles $M = 200$ and updated it faster, $\Delta t = 0.02s$, to allow it to integrate more information.

Ground truth: I used a marker-based OptiTrack [1] motion capture system to record "ground truth" poses of the object, x_t^*, as it is being manipulated, which uses reflective markers placed on the objects for precise pose estimation, as shown in Fig. 3.4.

3.6.3 Experimental Procedure

In each scene, I made 10 runs. During each run, I recorded the robot controls, u_t, and the camera images, z_t. I then evaluated the three methods on the same data recorded from the ten runs. I did this to be fair between the methods. Otherwise, all methods, including PBPF, are fast enough to be run in real-time, as Δt parameters also indicate.

At each time step of each run, I computed (as described in Sec. 3.3) the positional and rotational errors of PBPF, DOPE, and CVPF, using the ground truth data. When generating the positional and rotational errors for the DOPE method, if DOPE did not output any object pose at a certain time step (usually because the

[1]`https://github.com/NVlabs/Deep_Object_Pose`

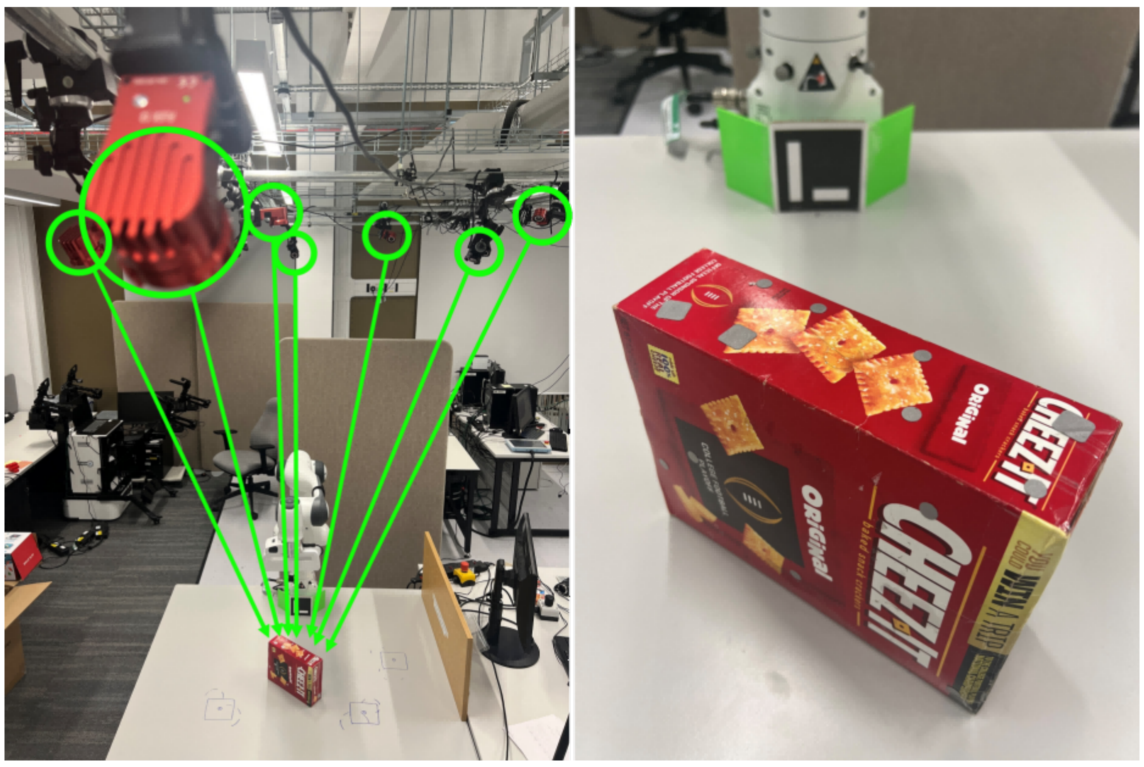


Figure 3.4: The OptiTrack system is used to determine the ground truth of the target objects. As shown in the left image, multiple cameras are mounted on the ceiling to track the reflective markers on the objects. The right image shows the reflective markers attached to the target object.

object is heavily obstructed), I used the last pose reported by DOPE before that time step.

3.6.4 Results

I present each method's positional and rotational errors over time (i.e. throughout the manipulation action) in Fig. 3.5. The plots show mean values together with the 95% confidence intervals in shaded regions, computed over the 10 runs in each scene.

In scenes with obstructions for the camera, the PBPF performs significantly better than both DOPE and CVPF. In Scene 1, the object is tracked well by all methods at the beginning, while the object is completely visible to the camera. As the surrounding clutter obstructs the object, the errors for DOPE and CVPF become quite high. While CVPF performs some smoothing, it cannot perform as well as the physics-informed PBPF. Similarly, in Scene 2, as the robot hand obstructs the camera view around $t = 8$, DOPE and CVPF perform significantly worse than PBPF. These results can also be observed in Fig. 3.2, where I overlay the PBPF estimation (as green wireframe box) and the DOPE estimation (as yellow wireframe box). (CVPF is not shown for visual clarity, but performs worse than PBPF in general.) I can see that while the green box is a good estimate of the actual object pose, the yellow box is not a good estimate when the camera view is not good. In Fig. 3.2, please note that the wireframe boxes are overlayed on the images as post-processing. For example, in Fig. 3.2(a) second and third images, and Fig. 3.2(b) final image, while the green box appears on top of the obstacle/robot hand, this is only an artefact of the way I overlay these boxes. The PBPF estimates are in fact behind the obstacle/robot hand in these instances, close to the actual object pose. In Scenes 3 and 4, where the object is always visible, DOPE performs very well, and therefore all methods perform similarly.

Fig. 3.5, a notable event occurs in Scene 1 around the 12-second mark. The plot shows a sharp, transient improvement in the accuracy of the DOPE baseline, where its error momentarily drops to a level close to the ground truth. This corresponds precisely to the moment when the target object, having been fully occluded, becomes briefly visible to the camera again. This allows the image-only method to re-acquire an accurate pose estimate. However, immediately after this point, the object is once more occluded by the robot arm, and the error for DOPE rapidly degrades, confirming its fundamental dependency on continuous visibility. A similar, more volatile pattern is observed in Scene 2 between the 8 and 12-second marks. The fluctuations in DOPE's error profile are attributable to intermittent occlusions, where the object's visibility flickers, causing instability in the purely vision-based estimates.

In stark contrast, our proposed PBPF method maintains a consistently low-error and stable track throughout these occlusion events in both Scenes 1 and 2. Because the physics engine continues to provide a reliable state prediction even when visual

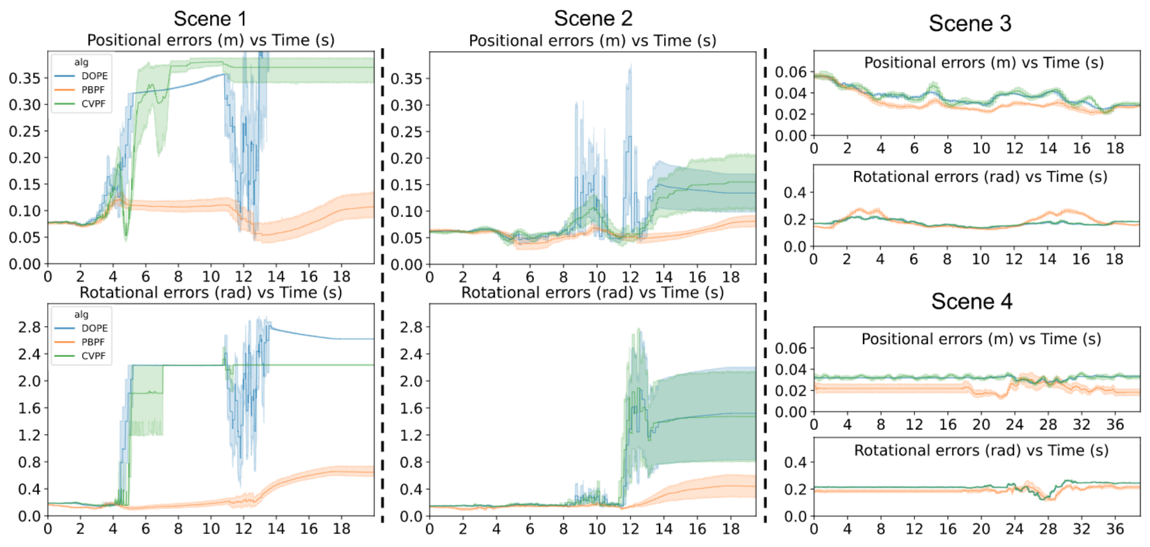


Figure 3.5: Comparison of different methods' positional and rotational errors in four scenes. Plots show mean values over 10 runs, with the shadows indicating the 95% Confidence Interval. In all plots, the horizontal axes show time, and the vertical axes show the positional/rotational error.

Table 3.1: Mean and standard deviation of the positional and rotational errors of different algorithms (“pos.err” means positional error and “rot.err” means rotational error)

	Scene 1				Scene 2				Scene 3				Scene 4			
	pos.err (m)		rot.err (rad)		pos.err (m)		rot.err (rad)		pos.err (m)		rot.err (rad)		pos.err (m)		rot.err (rad)	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
DOPE	0.384	0.244	1.855	1.019	0.117	0.090	1.059	1.088	0.036	0.007	**0.171**	0.023	0.031	0.002	0.218	0.033
PBPF	**0.091**	0.031	**0.300**	0.223	**0.056**	0.020	**0.284**	0.210	**0.030**	0.009	0.182	0.047	**0.022**	0.008	**0.194**	0.032
CVPF	0.116	0.074	0.438	0.690	0.077	0.044	0.553	0.853	0.037	0.008	0.173	0.022	0.032	0.004	0.217	0.035

data is absent, the filter does not lose track of the object and gracefully handles the transition between occluded and visible states.

Finally, the results for Scenes 3 and 4 serve as an important control case. In these scenarios, the target object remains fully visible throughout the manipulation. As expected, all methods perform robustly, yielding stable and low-error tracking. This confirms that under ideal conditions, the baseline methods are effective. However, it is the performance during the occlusion challenges in Scenes 1 and 2 that most effectively validates the central hypothesis of this chapter: the integration of a physics-based model is crucial for achieving robust tracking in cluttered and dynamic environments where objects are frequently occluded.

I also present the overall (averaged over all runs and all times-steps) positional and rotational errors in Table. 3.1. Again, I can see that PBPF performs significantly better in Scenes 1 and 2 because of the occlusion, and all methods perform similarly in Scenes 3 and 4 because the object is fully visible to the camera.

3.7 Conclusions

In this chapter, I introduced an algorithm designed for tracking a single target object under non-prehensile robotic manipulation. By incorporating a physics engine and using a particle filter algorithm, my tracking method maintains stable tracking even when the target object is completely occluded. Although using a physics engine significantly increases computational demands, experimental validation has demonstrated that these computational costs are justified given the improved tracking reliability.

Chapter 4

Physics-Based Multi-Objects 6D-Pose Tracking during Non-Prehensile Manipulation

Chapter Deliverables

Video: https://youtu.be/B9-5iTwgFnk

Source Code: https://github.com/ZisongXu/trackObjectWithPF

In this chapter, I extend the algorithm originally developed for tracking a single object during non-prehensile robotic manipulation to now track multiple objects. To improve accuracy, a visibility score has been introduced to better select particles from the particle set, which helps enable more accurate tracking of the 6D poses of multiple objects. This chapter explains the algorithm, theory, and experimental results, preparing for the enhanced version of this tracking algorithm for tracking and controlling multi-objects during non-prehensile manipulation. The layout of this chapter is outlined below:

- Section 4.1 will introduce the foundation of the problem I am trying to solve.
- Section 4.2 focuses on this work's assumptions, objectives and contributions.
- Section 4.3 formally describes the problem to be addressed and introduces the key notations used throughout the chapter.
- Section 4.4 presents the main algorithm, physics-based particle filtering, I use, based on the particle filtering algorithm for tracking multi-objects.
- Section 4.5 will mention the same two baseline methods described in tracking a single object algorithm: (i) Repetitive single-snapshot pose estimation, and (ii) Constant-velocity particle filtering.

- Section 4.6 will review the experiments and assess the algorithm's performance.
- Section 4.7 will conclude this multi-objects tracking algorithm.

4.1 Introduction

In this chapter, as the tracking algorithm has been expanded to accommodate multiple target objects, I have also broadened the initial problem description. This method retains its reliance on a physics engine and uses the particle filter algorithm. However, the movement of target objects is now influenced not solely by the robot's actions and physical factors, but also by their interactions with other target objects within the scene.

Fig. 4.1 depicts the movement of three target objects during a robot's non-prehensile manipulation. My objective is to track these objects in real-time. During their movement, initially, all target objects are visible to the camera. As the robot begins to push them, the Campbell Soup (a cylindrical can) is first obscured by the CheezIt box (a red box). Subsequently, the Jell-O (a small rectangular object) is also obscured by the CheezIt box, which in turn becomes partially obscured by the robot arm. Similar to the algorithm for tracking a single object, the robot arm's movement path is predetermined. To track multiple objects, each particle in the model represents the 6D pose of all objects in the scene. During particle resampling and updates, the poses of all target objects are updated.

Throughout the tracking process, in addition to using the physics engine and RGB images from the camera, I also compute a visibility score for each target object within every particle, which guides the resampling of particles. The core idea is that even when target objects are obscured, their obscured state is regarded as valuable information.

Similar to the algorithm for tracking a single target object, I conducted experiments applying both the multi-target tracking algorithm and learning-based algorithms across various real scenes. The results of these experiments demonstrate that incorporating a visibility score into the tracking algorithm enhances object tracking accuracy.

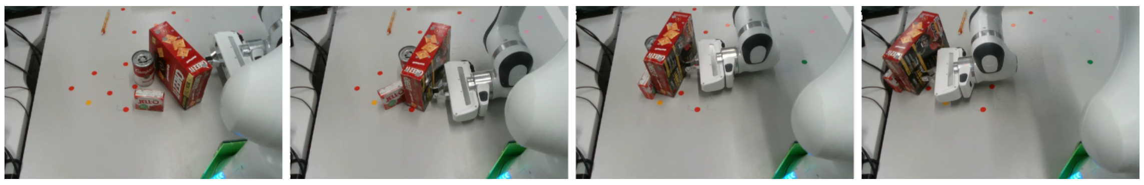


Figure 4.1: The robot arm manipulates the CheezIt box (a red box), causing it to move and subsequently push other target objects.

4.2 Assumptions, Objectives and Contributions

This chapter outlines key assumptions, Objectives, and contributions to developing this multi-objects tracking algorithm for robotic non-prehensile manipulation.

4.2.1 Assumptions

In this chapter, I make the following assumptions:

- Including all the assumptions from the previous methods.
- The robot can directly push multiple objects simultaneously or indirectly move one object via another.
- A perception system is available that can estimate the 6D poses of multiple target objects simultaneously when they are visible within the camera's field of view.
- During the manipulation process, the target objects may become obscured by the robot, other target objects (the algorithm is tracking), or fixed obstacles in the environment.

4.2.2 Objectives

The primary goal of this chapter is to develop a tracking algorithm that leverages a physics engine and a particle filter to effectively manage the tracking of multi-objects in non-prehensile manipulation within cluttered environments. This algorithm aims to maintain accurate tracking of the 6D poses of multi-objects even when they are obscured during manipulation.

4.2.3 Contributions

The contribution of this chapter is to propose an algorithm for tracking multi-objects in cluttered environments during non-prehensile manipulations.

4.3 Problem Formulation

I assume access to the sequence of joint-state inputs, u_t, and visual observations (z_t) from a camera, which is RGB images. I assume the camera's pose is known.

The problem is defined as follows: at any given time-step t, leveraging all prior control inputs $u_0, u_1, \ldots, u_t$ and visual observations $z_0, z_1, \ldots, z_t$, estimate the objects' current poses, denoted as $\{q_t^i\}_{i=1,\ldots,n} \in SE(3)^n$, where n is the number of objects that the robot is interacting with/I want to track, and q_t^i represents the pose of object i at time step t. Since inferring the exact poses of the objects is impossible (especially when they are occluded), I estimate a probability distribution over the object poses, instead of a single pose. Specifically, the problem is to estimate/track and continuously refine the probability distribution of the objects' poses $p(\{q_t^i\}_{i=1,...,n} | z_0, z_1, ..., z_t, u_0, u_1, ..., u_t)$, even when objects are partially or fully occluded. This conditional probability is commonly referred to as the *belief state*, which effectively integrates all prior information to predict the current poses of the objects. I describe a solution to the above tracking problem in Section 4.4.

4.4 Physics-Based Particle Filtering (PBPF)

This method adopts a Bayesian filtering approach, particularly *particle filtering* [45]. Fig. 4.2 shows the overall process of this algorithm.

Similar to the algorithm about tracking a single object, particle filtering represents the belief state at any given time t with a set of *particles*:

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]} \tag{4.1}$$

where each particle $x_t^{[m]} := \{q_t^i\}_{i=1,...,n}^{[m]}$ $(1 \leq m \leq M)$ is an instantiation of the pose state at time t and includes *all* the poses of the objects being tracked, where M is the number of particles. In the following sections, I also use the notation $q_t^{i,[m]}$ to

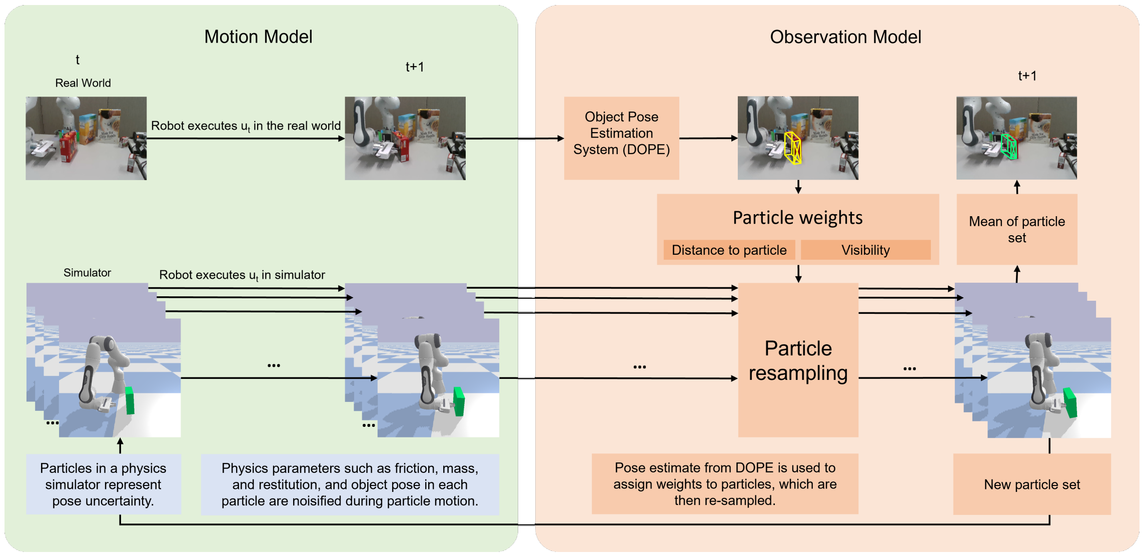

Motion Model
t
Real World
t+1
Robot executes u_t in the real world
Simulator
Robot executes u_t in simulator
...
Particles in a physics simulator represent pose uncertainty.
Physics parameters such as friction, mass, and restitution, and object pose in each particle are noisified during particle motion.
Observation Model
Object Pose Estimation System (DOPE)
t+1
Particle weights
Distance to particle
Visibility
Mean of particle set
Particle resampling
Pose estimate from DOPE is used to assign weights to particles, which are then re-sampled.
New particle set

Figure 4.2: This system for objects' pose estimation during robotic non-prehensile manipulation over two time steps, t and $t+1$. At t, the RGB image describes the scene, and particles represent possible objects' poses. As the system advances to $t+1$, these particles are updated by robot control, u_t, through physics simulation. A new RGB image is obtained from the camera. The RGB image is used to estimate objects' poses $DOPE(z_t)$, and are compared with poses of particles $\bar{x}_t^{[m]}$ to get the difference. Finally, particle resampling refines the objects poses estimations, and then gets a new particle set.

refer to the pose of a particular object i in particle $[m]$ at time t. I emphasize that, in this formulation, each particle contains *all* the tracked objects' poses (not individual separate objects), which enables the use of the physics-based interactions/constraints between objects.

The process of particle filtering involves a dual-stage update at each time step t: first, the *motion update* stage (discussed in Sec. 4.4.1), where particles are moved based on the latest robot control inputs u_t, followed by the *observation update* stage (discussed in Sec.4.4.2), where the particles are updated by the observational data z_t.

4.4.1 Motion Update

In the first stage of the particle filtering process, the propagation of each particle is addressed. Specifically, for a particle $x_{t-1}^{[m]}$ derived in the previous time step, an intermediate pose state $\bar{x}_t^{[m]} := \{\bar{q}_t^i\}_{i=1,\ldots,n}^{[m]}$ is formulated for the current time step, by the following probabilistic motion model:

$$\bar{x}_t^{[m]} \sim p(x_t \mid x_{t-1}^{[m]}, u_t) \tag{4.2}$$

where $p(x_t \mid x_{t-1}^{[m]}, u_t)$ represents the conditional probability distribution. This distribution characterizes the evolution of the system state from $x_{t-1}^{[m]}$ to x_t, influenced by the robot control u_t. Direct analytical derivation of this motion model is often impractical. I use a physics engine to approximate this distribution. The physics engine is modelled as f:

$$x_t = f_\theta(x_{t-1}, u_t) \tag{4.3}$$

which includes a model of the robot, the environment, and objects, and predicts the resulting poses of the objects x_t given their poses in the previous time step, x_{t-1}, and robot control, u_t, by simulating the robot motion inside the engine and finding the resulting motion of the objects. Here, θ represents physics parameters that impact the motion in the physics engine. Examples of such parameters include the friction coefficient at the contacts, contact restitutions, and the mass of the objects.

The physics engine is deterministic (i.e. outputs the same resulting state, if given the same inputs and parameters). It therefore cannot directly be used instead of the probabilistic motion model in Eq. 4.2. The uncertainty about the object's motion is due to (i) the uncertainty about the exact physics parameters, θ; and (ii) the discrepancy between the physics engine and the real-world physics.

To address point (i), I model these parameters for every object in each particle as random variables sampled from a known normal distribution. For example, for the mass of object i in particle $[m]$ at time t, I sample:

$$\text{mass}_t^{i,[m]} \sim \mathcal{N}(\mu_{\text{mass}_i}, \sigma^2_{\text{mass}_i}) \tag{4.4}$$

Similarly, I sample a value for each physics parameter of each object in the particle, and $\theta_t^{[m]}$ represents the collection of these sampled physics parameter values of each object in the particle m.

I then run the physics engine for that particle using the sampled parameters:

$$\bar{x}_t^{[m]} = f_{\theta_t^{[m]}}(x_{t-1}^{[m]}, u_t) + \epsilon \tag{4.5}$$

with the addition of Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_f^2)$ to address (ii) above. I assume that $f_{\theta_t^{[m]}}(x_{t-1}^{[m]}, u_t)$, i.e., the output of the physics engine, is always physically feasible, given a physically feasible input. When I add noise ϵ, if it puts the state into a physically impossible configuration (i.e., penetration between bodies), I simply draw a new ϵ value. If I cannot draw an ϵ that does not result in penetration, then I set ϵ to zero for that particular timestep and particle. The noise ϵ, is expanded to a higher-dimensional vector comprising independent 6D perturbations for each object. This perturbation is then applied to the predicted pose of each object within each particle to ensure the belief distribution maintains diversity.

As defined in Sec. 3.4.1. at each time step, I adjust the variable values to allow the physics engine to simulate the real-world environment as closely as possible. Below are explanations of the symbols used in this process: the parameter μ_{mass_i} represents the best guess about the mass of object i, and $\sigma^2_{\text{mass}_i}$ represents the uncertainty (variance). These mean and variance values for mass, friction, and restitution can be estimated offline beforehand for each object type. The parameter σ_f^2 represents the estimated discrepancy between the physics engine and real-world physics. In practice, σ_f needs to be estimated *per physics model*, f, for example by calibrating it using a dataset with ground truth object poses (such as the one I release), and σ_f can then be fixed. The mass and friction mean and variance values need to be provided per object. In this work, I use the best guess about these object values, but other methods can also be used, such as using a vision-based neural network to predict physics properties or extending the particle filter to also estimate these values.

4.4.2 Observation Update

During this second stage of particle filtering, for each intermediate particle $\bar{x}_t^{[m]}$, I calculate a *weight* $w_t^{[m]}$ using the observation, $w_t^{[m]} = p(z_t \mid \bar{x}_t^{[m]})$. After the weights for all the particles are computed, they are used to *re-sample* the new set of particles $\mathcal{X}_t$, completing the particle filter update. As I described in Sec. 3.4.2, since I do not have access to such a model directly, I propose to use an approximation. Using the Bayes Theorem, I can get:

$$p(z_t|\bar{x}_t^{[m]}) \approx p(\bar{x}_t^{[m]}|z_t) \tag{4.6}$$

To compute $p(\bar{x}_t^{[m]}|z_t)$, I propose to use the vision-only pose estimation system (e.g., DOPE) to predict the pose of the object according to z_t. However, as I described above each particle contains the pose state of all the objects $\bar{x}_t^{[m]} := \{\bar{q}_t^i\}_{i=1,\dots,n}^{[m]}$, and then I can assume that the pose of each object is independent of each other, so I write:

$$p(\bar{x}_t^{[m]}|z_t) = \prod_{i=1}^{n} p(\bar{q}_t^{i,[m]}|z_t) \tag{4.7}$$

use the distance of $\bar{q}_t^{i,[m]}$ to the predicted object pose to compute a probability value:

$$p(\bar{q}_t^{i,[m]}|z_t) = \mathcal{N}(\bar{q}_t^{i,[m]};\, DOPE(z_t), \sigma^2_{DOPE}) \tag{4.8}$$

where $DOPE(z_t)$ is the pose predicted by DOPE, and the parameter σ^2_{DOPE} represents the variance of DOPE errors for the object. This can be estimated beforehand for an object by collecting DOPE estimates for the object and comparing it to a ground truth pose.

4.4.3 Using Visibility Information

The observation model above assumes that a DOPE detection, $DOPE(z_t)$, is always available. However, for a given camera image z_t, DOPE may not output a pose estimate. At such time points, one option is to skip the observation update. I did this in the previous PBPF algorithm for tracking a single target object (3.4.2).

However, the lack of detection of the object is not a lack of information. The object is probably partially or fully occluded. Therefore, I also implemented an extension of this algorithm: *PBPF with Visibility* (PBPF-V).

For this method, I first compute a *visibility score*, $v_t^{i,[m]}$, for object i in particle m. To compute the visibility score, in the physics engine, I shoot R rays from the camera towards R points uniformly sampled on each object in each particle. Some of these rays hit the object, but others may hit other occluding objects or robot links before hitting the object. I define the visibility of an object as the ratio of rays hitting the object to the number of all rays:

$$v_t^{i,[m]} = \frac{\text{number of rays hitting the object } i \text{ in particle } m}{R} \tag{4.9}$$

A visibility score of 1 implies perfect visibility of the object by the camera, whereas a score of 0 implies the object is completely occluded. Then, I can relate the weight of a each object, $p(\bar{q}_t^{i,[m]}|z_t)$, to its visibility $v_t^{i,[m]}$ based on whether a DOPE estimate is available or not at time t. The particular method to relate these values depends on a model of DOPE behaviour under occlusion. Experiments with DOPE did **not** suggest that DOPE gradually worsens with more occlusion. Instead, after a certain amount of occlusion, DOPE loses the object completely. Therefore, instead of relating $p(\bar{q}_t^{i,[m]}|z_t)$ and $v_t^{i,[m]}$ (inversely) proportionally, I implemented a threshold-based model: If DOPE is not able to detect the object i at time t, then I assign a fixed high weight value (W_H) to these objects that have low visibility scores (i.e. $v_t^{i,[m]} < V_H$, where V_H is a fixed visibility threshold), and a fixed low weight value (W_L) to these objects that have high visibility scores (i.e. $v_t^{i,[m]} > V_H$). Conversely, if a DOPE estimate is available at time t, I compute the weight values described in Section 4.4.2. However, objects with low visibility ($v_t^{i,[m]} < V_L$, where V_L is a fixed low visibility threshold) are penalised by scaling their weight by $\alpha_W \in [0, 1]$.

4.5 Baseline Methods

I continue to employ the same baseline methods (Repetitive Single-Snapshot Pose Estimation in Sec. 3.5.1 and Constant-Velocity Particle Filtering in Sec. 3.5.2) for comparison that were used in the single object tracking algorithm.

4.6 Experiments and Results

I evaluated and compared the performance of this methods in eight different non-prehensile manipulation scenes. I used three objects from the YCB dataset [53], the CheezIt box, the Gelatin Jell-o and the Campbell Soup, a cylinder. DOPE had good performance of these objects when they were visible. All scenes can be seen in the video: https://youtu.be/B9-5iTwgFnk.

4.6.1 Scenes

Cheezit Scene 1 (shown in Fig. 4.3-first row) and **Soup Scene 1** (shown in Fig. 4.3-fifth row): The robot pushes the object among the clutter, where the cluttering object obstructs the view of the camera. The pose of the obstructing object is fixed and known.

Cheezit Scene 2 (shown in Fig. 4.3-second row) and **Soup Scene 2** (shown in Fig. 4.3-sixth row): The robot pushes the object on a clear table. The hand can obstruct the camera view during pushing.

Cheezit Scene 3 (shown in Fig. 4.3-third row) and **Soup Scene 3** (shown in Fig. 4.3-seventh row): The robot pushes the object on a clear table, and the camera has a clear view of the object.

Cheezit Scene 4 (shown in Fig. 4.3-fourth row): The robot tilts the object up and then brings it down. The camera has a clear view. The pose of the supporting object is fixed and known.

Scene 5 (shown in Fig. 4.3-eighth row): The robot pushes multiple objects. These objects occlude each other during the push. I track all objects' poses simultaneously.

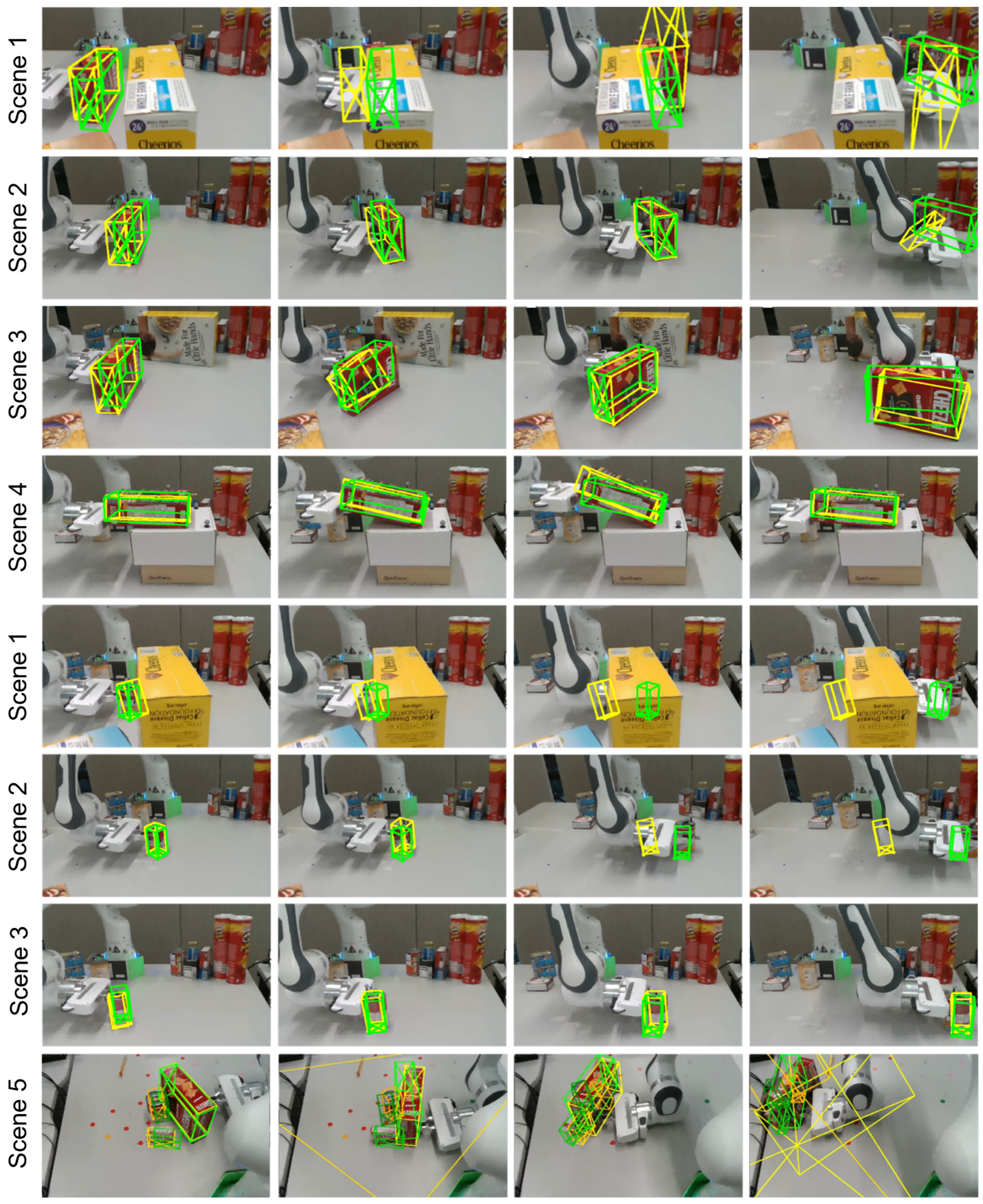


Figure 4.3: The estimated multi-objects poses of the method (PBPF-V) are shown as a green wireframe box. The poses from a vision-only system (DOPE) are shown as a yellow wireframe box.

In Scenes 1, 2, and 3, obstacles in the clutter do not physically interact with the robot and the tracked object.

4.6.2 Implementation Details

I have implemented the methods as below[1].

[1]Experiments were performed on CPU: 11th Gen Intel(R) Core(TM) i9-11900@2.50GHz; GPU: NVIDIA GeForce RTX 3090; RAM: 128580 Mb

PBPF: The physics-based particle filtering method (Sec.4.4). (a) *Motion model parameters.* μ_θ and σ_θ: Mean friction coefficient of 0.1 and standard deviation of 0.3, with minimum capped at 0.001. Mean restitution of 0.9 and standard deviation of 0.2. The mean mass of 0.38 kg and the standard deviation of 0.5 with a minimum cap of 0.05 kg. σ_f: For position 0.005 m, for rotation 0.05 rad. As the physics model, f_θ, I used the Pybullet physics engine [54]. A Pybullet environment was initialized for each particle. The Pybullet environments for particles were parallelized over the 8 (16 virtual) CPU cores (11th Gen Intel(R) Core(TM) i9-11900@2.50GHz) of the computer. (b) *Observation model parameters.* σ_{DOPE}: For position 0.02 m and rotation 0.09 rad. (c) *Update frequency.* $\Delta t = 0.16s$. (d) *Number of particles.* $M = 70$. (e) *Initialization.* I use a Gaussian distribution to initialize 70 particles at $t = 0$. DOPE estimates the mean pose at $t = 0$. The standard dev. for initialization is 0.16 m and 0.43 rad.

PBPF-V: An extension of PBPF which also considers visibility (as described in Sec. 4.4.3). All parameters are the same with PBPF except that the number of particles $M = 150$ in Scene 5. Additional visibility parameters: For Cheezit and Gelatin, $W_H = 0.75$, $W_L = 0.25$, $V_H = 0.95$, $V_L = 0.5$, $\alpha_W = 0.33$. For Soup, $W_H = 0.6$, $W_L = 0.45$, $V_H = 0.8$, $V_L = 0.9$, $\alpha_W = 0.33$.

DOPE: The repetitive single-snapshot pose estimation method (Sec. 3.5.1). I used official DOPE implementation[1].

CVPF: The constant-velocity particle filtering method. All parameters were the same with PBPF, except the number of particles, M. Since CVPF is computationally cheaper, it could handle more particles $M = 150$.

Ground truth: I used a marker-based OptiTrack system to record "ground truth" poses, x_t^*, during manipulation.

4.6.3 Experimental Procedure

In Scenes 1-4, I repeated 10 robot runs, and in Scene 5, I repeated 1 robot run (i.e. a total of 71 real robot executions). During each run, I recorded the robot controls, u_t, and the camera images, z_t. Since particle filtering is a sampling-based method, it can generate different results with the same input. For statistical accuracy, I evaluated each method 10 times on the data from each of the 10 runs in Scenes 1-4, giving 100 evaluations of each method for Scenes 1-4. In Scene 5, I evaluated each method 20 times on the data from the 1 real robot execution.

At each time step of each evaluation, I computed the positional and rotational errors of the mean estimate, against the ground truth. When computing errors for DOPE, if DOPE did not output any pose at a certain time step (e.g., because of occlusions), I used the latest pose reported before that time step.

[1]https://github.com/NVlabs/Deep_Object_Pose

Table 4.1: Mean and standard deviation of the errors of different methods

	Scene 1				**Scene 2**				**Scene 3**				**Scene 4**			
	pos.err(m)		rot.err(rad)		pos.err(m)		rot.err(rad)		pos.err(m)		rot.err(rad)		pos.err(m)		rot.err(rad)	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Cheezit																
PBPF	0.081	0.020	0.484	0.272	0.070	0.030	0.426	0.309	**0.019**	0.024	**0.367**	0.187	0.018	0.008	**0.092**	0.067
PBPF-V	**0.063**	0.020	**0.396**	0.295	**0.058**	0.021	**0.294**	0.204	-	-	-	-	-	-	-	-
DOPE	0.289	0.218	1.345	1.168	0.293	0.248	1.046	1.027	0.052	0.033	0.375	0.447	**0.017**	0.002	0.201	0.013
CVPF	2.359	7.421	1.472	0.951	0.407	0.678	0.878	0.952	25.14	152.8	0.879	0.952	0.029	0.012	0.207	0.022
Soup																
PBPF	**0.043**	0.023	1.133	0.364	0.049	0.032	0.851	0.249	**0.026**	0.021	**0.405**	0.455	-	-	-	-
PBPF-V	0.045	0.027	**1.077**	0.395	**0.046**	0.028	**0.724**	0.262	-	-	-	-	-	-	-	-
DOPE	0.215	0.117	1.174	0.372	0.206	0.112	0.776	0.222	0.071	0.140	0.438	0.482	-	-	-	-
CVPF	0.298	0.153	1.884	0.794	0.240	0.126	1.702	0.845	6.528	52.09	1.174	0.982	-	-	-	-

4.6.4 Results

I present the overall (averaged over all runs and all time-steps) positional and rotational errors from Scene 1-4 in Table. 4.1, and from Scene 5 in Table. 4.2. Compared to DOPE and CVPF, PBPF performs significantly better in Scenes 1, 2, and 3. In Scene 4, where DOPE has a good view and overall good detection, PBPF and DOPE perform similarly. CVPF performs worse out of all the methods. In Scene 5, I only ran the PBPF-V algorithm used for comparison with DOPE. PBPF-V performs significantly better than the baseline method.

When there is occlusion (Scenes 1 and 2), PBPF-V performs better than basic PBPF for the Cheezit object. For the Soup object, the difference between PBPF and PBPF-V is not significant. This is probably due to the Soup being a much smaller object, for which DOPE's behaviour under occlusion is much more difficult to model accurately.

It is important to note that results for the PBPF-V are not presented for Scenes 3 and 4. These two scenarios were designed as control cases in which the target object remains fully and continuously visible to the camera throughout the entire manipulation task. The core mechanism of the PBPF-V algorithm is to use a visibility score to intelligently weight the contribution of visual feedback, which is primarily

Table 4.2: Mean and standard deviation of the errors when tracking multi-object (Scene 5)

	Cheezit				**Gelatin**				**Soup**			
	pos.err(m)		rot.err(rad)		pos.err(m)		rot.err(rad)		pos.err(m)		rot.err(rad)	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
PBPF-V	**0.031**	0.011	**0.126**	0.087	**0.031**	0.023	**0.056**	0.030	**0.027**	0.013	**0.346**	0.139
DOPE	0.058	0.108	0.158	0.337	0.179	0.665	0.272	0.410	0.045	0.029	1.193	1.192

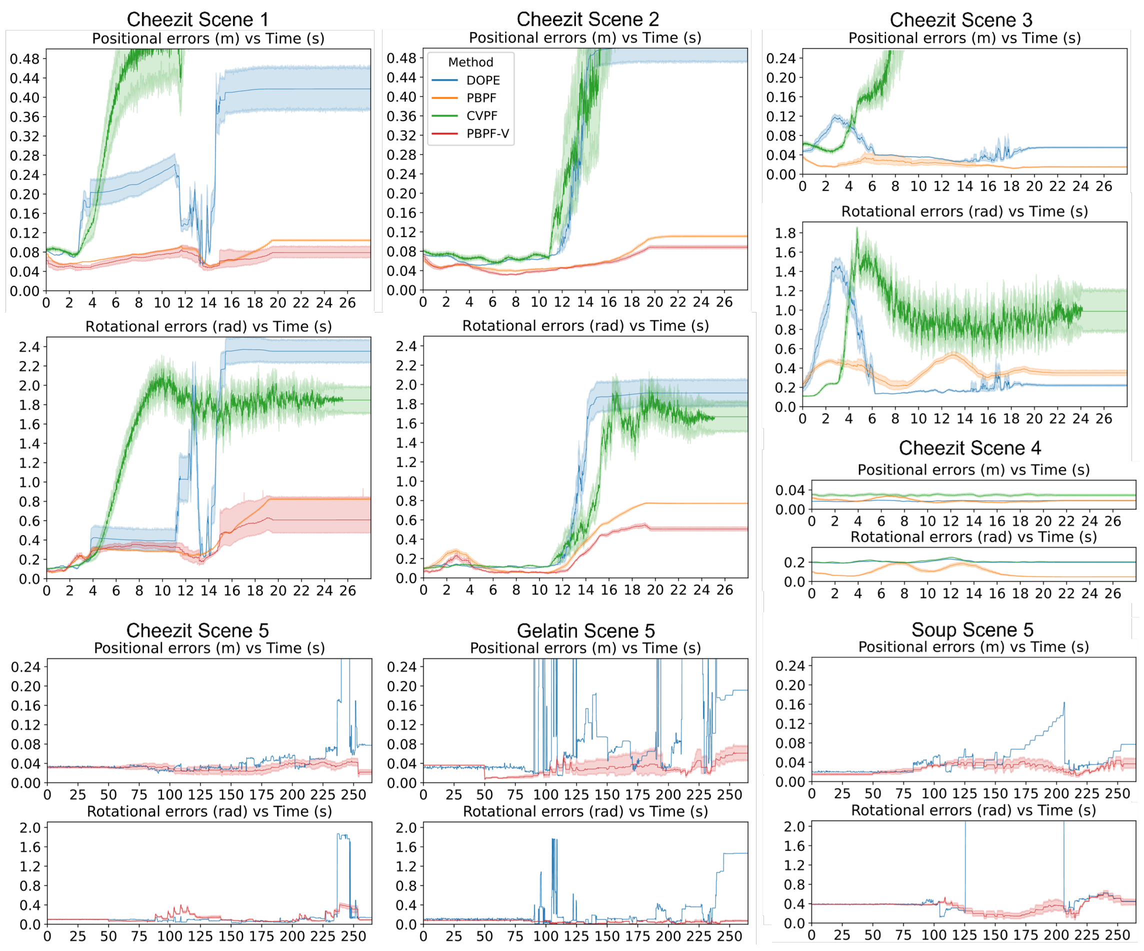


Figure 4.4: Positional and rotational errors of different methods in four different scenes with the Cheezit object and the fifth scene shows the tracking of three different target objects, each with its own positional and rotational errors in different methods. Plots show mean values with the shadows indicating the 95% Confidence Interval. In all plots, the horizontal axes show time, and the vertical axes show the positional/rotational error. Similar plots for all scenes and all objects can be seen in this repository.

advantageous under conditions of partial or total occlusion. In a scenario of constant and perfect visibility, the visibility score is perpetually equal to 1. Under this condition, the behavior of the PBPF-V algorithm becomes identical to that of the standard PBPF method. Therefore, conducting separate experiments for PBPF-V in these full-visibility scenes would yield redundant results and offer no additional insight into its specific contribution. For this reason, they were omitted from the comparison.

I also present the positional and rotational errors of each method throughout manipulation in Scenes 1-4 for Cheezit in Fig. 4.4. (Results for Soup are similar, and are in this repository.) I can see that PBPF and PBPF-V produce better and more stable estimates of the object position. In Fig. 4.4, I also show the results for each object in Scene 5. PBPF-V significantly outperforms DOPE.

It is worth noting that even with a good field of view, DOPE can be noisy, because it uses a single visual snapshot, which becomes unstable when the light, angle, and other factors change. This can lead to physically unrealistic poses, e.g. target object can be inside the robot arm, or two target objects can penetrate. In contrast, the method (PBPF) outputs physically feasible poses, effectively avoiding the unrealistic scenarios above.

4.7 Conclusions

In this chapter, I introduced an algorithm designed for tracking multi-objects under non-prehensile robotic manipulation. To maintain the efficiency of the algorithm and not reduce the number of particles, I had to slow down the robot's motion to give enough time for particle updates when performing multi-target object tracking. The computational demand significantly increases as I predict the 6D poses of multi-objects using a physics engine, compared to tracking a single object. Additionally, by incorporating visibility information into the method, the algorithm still outperforms the baseline approaches, though at the expense of efficiency. In the next chapter, I introduce an improved method for tracking multiple objects. This refinement enhances both the tracking accuracy and the computational efficiency of the algorithm, addressing the challenges identified with the initial approach.

Chapter 5

RGB-D Based Tracking and Control of Multiple Objects during Non-Prehensile Manipulation in Clutter

Chapter Deliverables

Video: https://www.youtube.com/watch?v=HxqMidaboqE

Source Code: https://github.com/ZisongXu/trackObjectWithPF

In this chapter, I present an update to the multi-object tracking algorithm discussed in the previous chapter, aiming to enhance both the accuracy of tracking multiple target objects and the computational efficiency of the algorithm. I employ CPU parallelism to update particles within the physics engine and incorporate depth images from the camera during the resampling process to refine particle selection. Additionally, GPU parallelism is used for depth image comparisons. This chapter explains the algorithm, underlying theories, and experimental outcomes, with the structure of the chapter outlined as follows:

- Section 5.1 will introduce the foundation of the problem I am trying to solve.
- Section 5.2 focuses on this work's assumptions, objectives and contributions.
- Section 5.3 formally describes the problem to be addressed and introduces the key notations used throughout the chapter.
- Section 5.4 presents the main algorithm, physics-based particle filtering combined with RGB-D images, I use, based on the particle filtering algorithm for tracking multi-objects.
- Section 5.5 will mention using the tracking algorithm as feedback in model predictive control (MPC).

- Section 5.6 will review the experiments and assess the algorithm's performance.
- Section 5.7 will further examine the potential factors that may have influenced the experimental results, as well as the limitations of my approach.
- Section 5.8 will conclude this multi-objects tracking algorithm.

5.1 Introduction

In this chapter, I describe methods to improve both the performance of tracking multiple target objects' 6D poses and the computational efficiency of the algorithm. Previously, I applied a physics engine-based particle filter algorithm to track multiple target objects, which significantly reduced the computational efficiency. Therefore, this chapter focuses on enhancing the computational efficiency of the algorithm while improving the accuracy of tracking the 6D poses of multiple target objects. Additionally, I will discuss the application of this multi-object tracking algorithm in robotic manipulation tasks, aiming to integrate it effectively into practical scenes.

Figure 5.1 illustrates the process of a robotic manipulation task, where the goal is to push a target object (a yellow-green cylinder) into the goal area (red circle). During the robotic manipulation task, the target object is initially fully visible but becomes completely obscured by objects such as Milk and CheezIt as the robot moves. As the robot continues to push, the target object reappears within the camera's field of view. A major challenge in executing robotic manipulation tasks is the occlusion of the target object by other items. Traditional tasks often employ specific methods, such as attaching AR markers or reflective markers to all objects, to determine their poses for manipulation. However, it is impractical to apply markers to every common object in real-world scenes. Therefore, I have applied a particle filter-based multi-objects tracking algorithm to robotic manipulation tasks. The objective is to provide accurate robotic trajectories for completing tasks, even when the target object is obscured.

In multi-object tracking tasks, the robot's movement trajectories are predetermined, while in robotic manipulation tasks, these trajectories are generated by the algorithm.

Building upon the tracking algorithm discussed in previous chapters, I have incorporated depth images as an additional criterion for particle resampling in the multi-object tracking algorithm. Extensive testing has demonstrated that using depth images greatly improves the performance of the multi-object tracking algorithm.

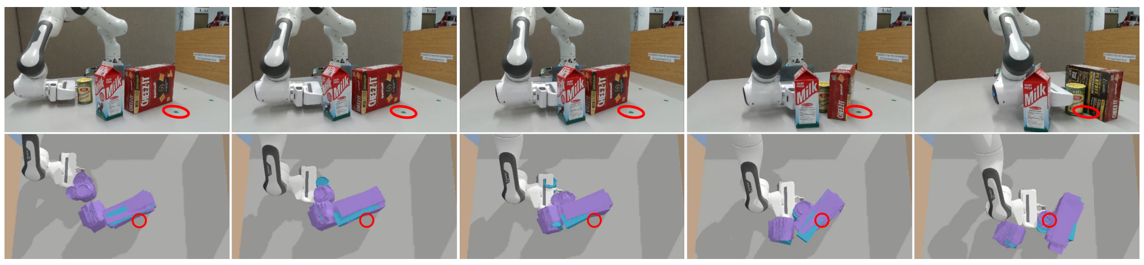


Figure 5.1: An experimental scene where a robot is pushing an object (the yellow-green cylinder) to a goal area (red circle) among other cluttering objects. The top row shows the camera image that is used for tracking the objects. Purple particles in the bottom row show the possible poses of the objects as estimated by the method. The blue particles show the estimated pose from a camera-only pose estimation system, DOPE [36].

5.2 Objectives and Contributions

This chapter outlines key objectives and contributions to developing this multi-objects tracking algorithm for robotic non-prehensile manipulation and using the algorithm in real robotic manipulation tasks.

5.2.1 Objectives

This chapter aims to develop a tracking algorithm that leverages a physics engine and particle filter to address the challenges of tracking multiple target objects' 6D poses under robotic non-prehensile manipulation, even when the objects are occluded during operations. The final objective is to apply this algorithm in real-world robotic systems, enabling the robotic manipulation algorithm to generate real-time motion trajectories for the robot to complete manipulation tasks effectively.

5.2.2 Contributions

The contribution of this chapter is to propose an algorithm for tracking multi-objects in cluttered environments during non-prehensile manipulations.

5.3 Problem Formulation

This chapter addresses the challenge of non-prehensile manipulation in cluttered environments, where a robot manipulates multiple objects simultaneously. To enable closed-loop control strategies for such tasks, my system requires a robust method for tracking the 6D pose of all objects at all times. To formally define the tracking problem, I first establish the key variables:

- **Control Input (u_t):** The control input, $u_t \in \mathbb{R}^N$, is the vector of commands sent to the robot's N joints (e.g., in the experiments, the joint states of a 7-DoF manipulator), representing the motion executed from time $t-1$ to t.
- **Observation (z_t):** The observation at time t, z_t, consists of synchronized data from a static camera: $z_t = ({}^{RGB}z_t, {}^{D}z_t)$. Here, ${}^{RGB}z_t$ is an RGB image and ${}^{D}z_t$ is a depth image. These images are high-dimensional data structures that provide the visual evidence for my tracking process.

The problem is similarly defined as I mentioned in the section 4.3, but it is worth noting that in the visual observations, the new algorithm uses both RGB images and depth images.

With robust object tracking, I can use the object pose estimates as feedback for closed-loop simultaneous control of multiple objects. Methods like Model Predictive Control (MPC) depend on such continuous and reliable feedback. The tracking solution I describe integrates well with an MPC framework. I present such an MPC approach in Section 5.5.

5.4 Physics-Based Particle Filtering (PBPF)

The process of particle filtering involves a dual-stage update at each time step t: first, the *motion update* stage (discussed in Sec. 5.4.1), where particles are moved based on the latest robot control inputs u_t, followed by the *observation update* stage (discussed in Sec.5.4.2), where the particles are updated by the observational data z_t. Fig. 5.2 shows the overall process of this algorithm.

5.4.1 Motion Update

The motion update part of the new algorithm hasn't changed much, using the same setup as mentioned before in Section 4.4.1 for updating the motion model. However, updating the motion model in a straightforward, step-by-step manner was slowing down the code. To fix this, I reorganized the code without changing the basic function of the motion update, moving the motion model calculations to a multi-processing method. This adjustment greatly improved the speed of the code.

5.4.2 Observation Update

In the observation update part, for each intermediate particle $\bar{x}_t^{[m]}$, I calculate an *weigth* $w_t^{[m]}$ using the observation z_t:

$$w_t^{[m]} = p(z_t \mid \bar{x}_t^{[m]}) \tag{5.1}$$

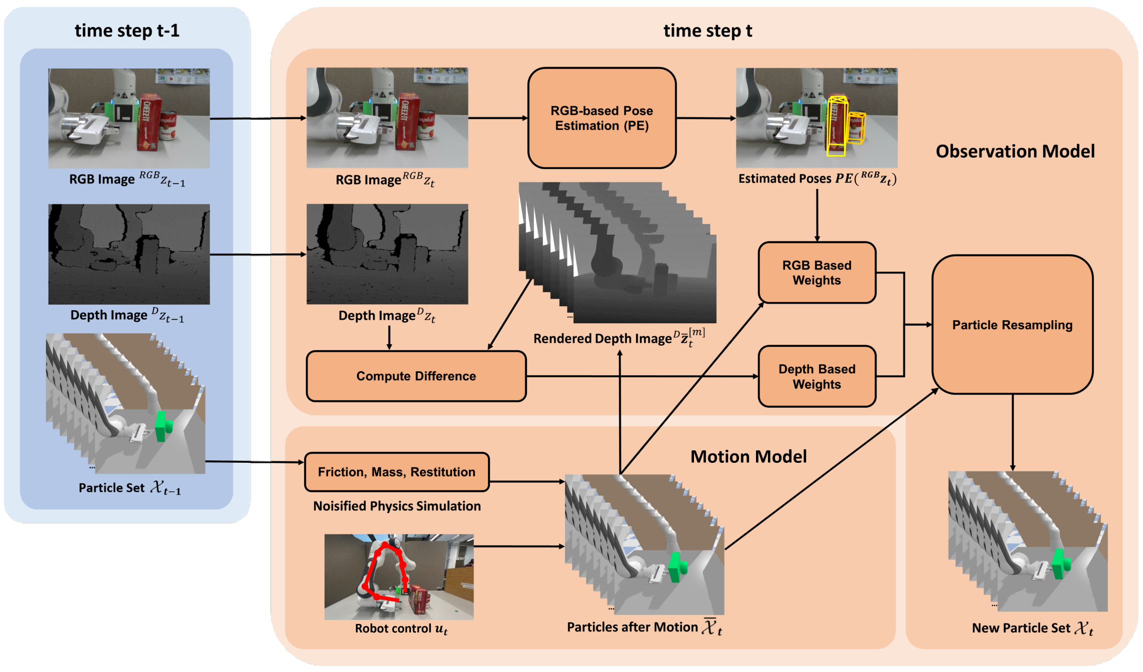

time step t-1
time step t
RGB Image $^{RGB}z_{t-1}$
Depth Image $^{D}z_{t-1}$
Particle Set $\mathcal{X}_{t-1}$
RGB Image $^{RGB}z_t$
RGB-based Pose Estimation (PE)
Estimated Poses $PE(^{RGB}z_t)$
Observation Model
Depth Image $^{D}z_t$
Rendered Depth Image $^{D}\bar{z}_t^{[m]}$
RGB Based Weights
Depth Based Weights
Particle Resampling
Compute Difference
Friction, Mass, Restitution
Noisified Physics Simulation
Motion Model
Robot control u_t
Particles after Motion $\bar{\mathcal{X}}_t$
New Particle Set $\mathcal{X}_t$

Figure 5.2: This system for objects' pose estimation during robotic non-prehensile manipulation over two time steps, $t-1$ and t. At $t-1$, RGB and depth images describe the scene, and particles represent possible objects' poses. As the system advances to t, these particles are updated by robot control, u_t, through physics simulation. New RGB and depth images are obtained from the camera. The RGB image is used to estimate objects' poses $PE(^{RGB}z_t)$, and are compared with poses of particles $\bar{x}_t^{[m]}$ to get the difference. The depth image is compared with the rendered depth images (according to the poses of objects in each particle in the simulation). Finally, particle resampling refines the objects poses estimations, and then gets a new particle set.

Upon calculating the weights for all intermediate particles, these weights are used to *re-sample* a new set of particles $\mathcal{X}_t$, thus concluding the particle filter update. During re-sampling, each intermediate particle $\bar{x}_t^{[m]}$ may be selected (potentially multiple times) for inclusion in the new set $\mathcal{X}_t$, with selection probabilities proportional to $w_t^{[m]}$.

To calculate the weights in Eq. 5.1 I use the RGB and depth images as observations:

$$w_t^{[m]} = p(^{RGB}z_t,\ ^{D}z_t \,|\, \bar{x}_t^{[m]}) \tag{5.2}$$

I make the simplifying assumption that the RGB and depth images captured by the camera are conditionally independent, given the true state of the world represented by the particle $\bar{x}_t^{[m]}$. Firstly, the two sensing modalities operate on different physical principles: the RGB sensor captures passive ambient light in the visible spectrum [55], while the depth sensor typically uses its own active infrared light source [56]. Consequently, their dominant sources of noise and error are largely decoupled. For instance, changes in ambient lighting drastically affect the RGB image but have minimal impact on an active depth sensor, whereas certain surface materials might disrupt depth readings without significantly altering their colour appearance.

This assumption of conditional independence allows us to decompose the joint probability distribution from Equation 5.2 into the product of two separate likelihood models:

$$w_t^{[m]} = p({}^{RGB}z_t \mid \bar{x}_t^{[m]}) \cdot p({}^{D}z_t \mid \bar{x}_t^{[m]}) \tag{5.3}$$

This decomposition is highly advantageous as it significantly simplifies the problem. It enables us to design and compute two independent likelihood functions: one that evaluates the color-based similarity of the observation ($p({}^{RGB}z_t \mid \bar{x}_t^{[m]})$), and another that evaluates the geometric or shape-based similarity ($p({}^{D}z_t \mid \bar{x}_t^{[m]})$).

RGB Images

The expression $p({}^{RGB}z_t \mid \bar{x}_t^{[m]})$ in Eq. 5.3 ideally represents the probability of making the current observation (i.e. getting the current camera RGB image) if objects were at pose $\bar{x}_t^{[m]}$. Since I do not have direct access to such a model, using the Bayes Theorem, I first re-write the observation model:

$$p({}^{RGB}z_t|\bar{x}_t^{[m]}) = \frac{p(\bar{x}_t^{[m]}|{}^{RGB}z_t)\, p({}^{RGB}z_t)}{p(\bar{x}_t^{[m]})} \tag{5.4}$$

Here, I note that $p({}^{RGB}z_t)$ is the same for every particle since the current observation does not change between particles. Furthermore, I make a simplifying assumption that $p(\bar{x}_t^{[m]})$ is also similar for different particles. This enables us to compute the weight using:

$$p({}^{RGB}z_t|\bar{x}_t^{[m]}) \approx p(\bar{x}_t^{[m]}|{}^{RGB}z_t) \tag{5.5}$$

Given that each particle contains the pose state of all the objects $\bar{x}_t^{[m]} := \{\bar{q}_t^i\}_{i=1,\dots,n}^{[m]}$, and assuming that the pose of each object is independent of each other, I write:

$$p(\bar{x}_t^{[m]}|{}^{RGB}z_t) = \prod_{i=1}^{n} p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t) \tag{5.6}$$

Since my particles, $\bar{x}_t^{[m]}$, are feasible (e.g., non-penetrating) outputs of the motion model, none of the simplifications above result in assigning probabilities to infeasible states.

I compute $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ for each object, based on the results of two computations: a *Distance comparison* and a *Visibility score.*

Distance Comparison: I first use an off-the-shelf RGB-based single-snapshot pose estimation (PE) system to predict the pose of each object according to ${}^{RGB}z_t$ and then use the distance of $\bar{q}_t^{i,[m]}$ to the predicted object pose to compute a probability value. Using $PE^i({}^{RGB}z_t)$ to represent the output of the pose estimation system for object i, i.e., the predicted pose of the object i given the camera image ${}^{RGB}z_t$, I compute a probability, p_{PE}, for each object:

$$p_{PE}(\bar{q}_t^{i,[m]}|{}^{RGB}z_t) = \mathcal{N}(\bar{q}_t^{i,[m]};\, PE^i({}^{RGB}z_t), \sigma^2_{PE^i}) \tag{5.7}$$

where the parameter $\sigma^2_{PE^i}$ represents the variance of PE system errors for the object. The variance can be estimated beforehand by collecting pose estimates for each object and comparing them to the ground truth pose. In practice, I use two normal distributions to compute the probability in Eq. 5.7. One distribution accounts for the positional Euclidean distance between $\bar{q}_t^{i,[m]}$ and $PE^i({}^{RGB}z_t)$. The second distribution represents the rotational distance between $\bar{q}_t^{i,[m]}$ and $PE^i({}^{RGB}z_t)$, computed as the minimum rotation around a single axis required to align the two orientations. The final probability in Eq. 5.7 is obtained by multiplying the probabilities from these two normal distributions. I use Normal distribution as the most generic model. Alternatively, given a particular PE system, the distribution for that system can be empirically modelled (either as a different parametric distribution or as a statistical/learned model) for a given object.

While $p_{PE}(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ can be used as the value of $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ in Eq. 5.6, this requires that an output from the PE system for each object, $PE^i({}^{RGB}z_t)$, is present. However, situations exist where a PE system fails to detect an object, which means $PE^i({}^{RGB}z_t)$ does not exist. For example, this happens when the object is occluded. To address these situations which occur when the object is occluded, I also evaluate the visibility of an object in a particle.

Visibility Score: Note that, given an image ${}^{RGB}z_t$, the absence of object detection is also useful information. It often indicates that the object is either partially or completely occluded, which can provide significant information about its pose. To leverage this information, I use rendered artificial images of each particle from the perspective of the camera, which I call a "simulated camera" below. I then establish the following four conditions:

1. If the object i at pose $\bar{q}_t^{i,[m]}$ in the particle $\bar{x}_t^{[m]}$ is visible to the simulated camera (i.e. camera's view to the object is not occluded by other objects in $\bar{x}_t^{[m]}$, the robot, or the environment), and the PE system detects it in the actual image (i.e., $PE^i({}^{RGB}z_t)$ exists) then $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ is equal to the probability estimated using the PE system, $p_{PE}(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$.

2. If the object i at pose $\bar{q}_t^{i,[m]}$ in the particle $\bar{x}_t^{[m]}$ is visible to the simulated camera, but the PE system does not detect it in the actual image (i.e., $PE^i({}^{RGB}z_t)$ does not exist) then $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ is given a low probability value, $\alpha_l \in [0, 1]$, which is a predefined parameter. It represents the probability that the PE system cannot detect a visible (non-occluded) object.

3. If the object i at pose $\bar{q}_t^{i,[m]}$ in the particle $\bar{x}_t^{[m]}$ is not visible to the simulated camera (i.e. camera's view to the object is mostly occluded by other objects in $\bar{x}_t^{[m]}$, the robot, or the environment), but the PE system detects it in the actual image (i.e., $PE^i({}^{RGB}z_t)$ exists) then $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ is given a low value, $p_{PE}(\bar{q}_t^{i,[m]}|{}^{RGB}z_t) \cdot \alpha_w$, where α_w ranges between 0 and 1. In rare cases, even mostly occluded objects can be detected by PE systems, and this probability reflects that case.

4. If the object i at pose $\bar{q}_t^{i,[m]}$ in the particle $\bar{x}_t^{[m]}$ is not visible to the simulated camera, and the PE system also does not detect it in the actual image (i.e., $PE^i({}^{RGB}z_t)$ does not exist) then $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ is given a high probability value $\alpha_h \in [0,1]$. This represents the probability that the PE system does not detect an object when it is mostly occluded, which is high.

Based on the conditions outlined above, the value of each $p(\bar{q}_t^{i,[m]}|{}^{RGB}z_t)$ in Eq. 5.6 is determined.

The method I presented above requires deciding whether an object is visible or occluded in a particle. I determine this by computing a visibility score for each object in each particle $Visible^i(x_t^{[m]})$, which represents the proportion of the part of the object i in particle $[m]$ that is not occluded.

To compute $Visible^i(x_t^{[m]})$ the system uses the simulated camera to render one segmented image with all objects in the particle, as well as individual segmented images for each object separately. I show an example in Fig. 5.3 for a given particle. The top-left image shows the rendered segmented image for all objects in the particle (including the robot and the environment). I count the number of pixels belonging to each object in this image, as shown top-right in Fig. 5.3. This count reflects the portion/footprint of the object that is not occluded and thus visible to the camera. The bottom three rows on the left show the individual segmented images rendered for each object separately. I also count the number of pixels belonging to the objects in these images, as shown in the bottom three rows in the middle. These counts reflect the total possible footprint of each object at that pose. I define the visibility score of each object as $Visible^i(x_t^{[m]})$ as the ratio of the unoccluded footprint to the total possible footprint, as shown in the bottom three rows on the right of Fig. 5.3. A visibility score of 1 indicates perfect visibility, where the entire object is visible to the camera. Conversely, a score of 0 indicates that the object is entirely occluded. To simulate the camera, i.e., to render the segmented images above, and to determine the pixel counts, I use GPU hardware queries. The details of GPU-based rendering are explained in the paper that is submitted to the TRO journal[57], and this part is developed by my co-authors Jaina Modisett and Markus Billeter.

I decide whether an object in a particle is visible to the camera, by comparing $Visible^i(x_t^{[m]})$ to a threshold, V, which is a predefined parameter of the system. If

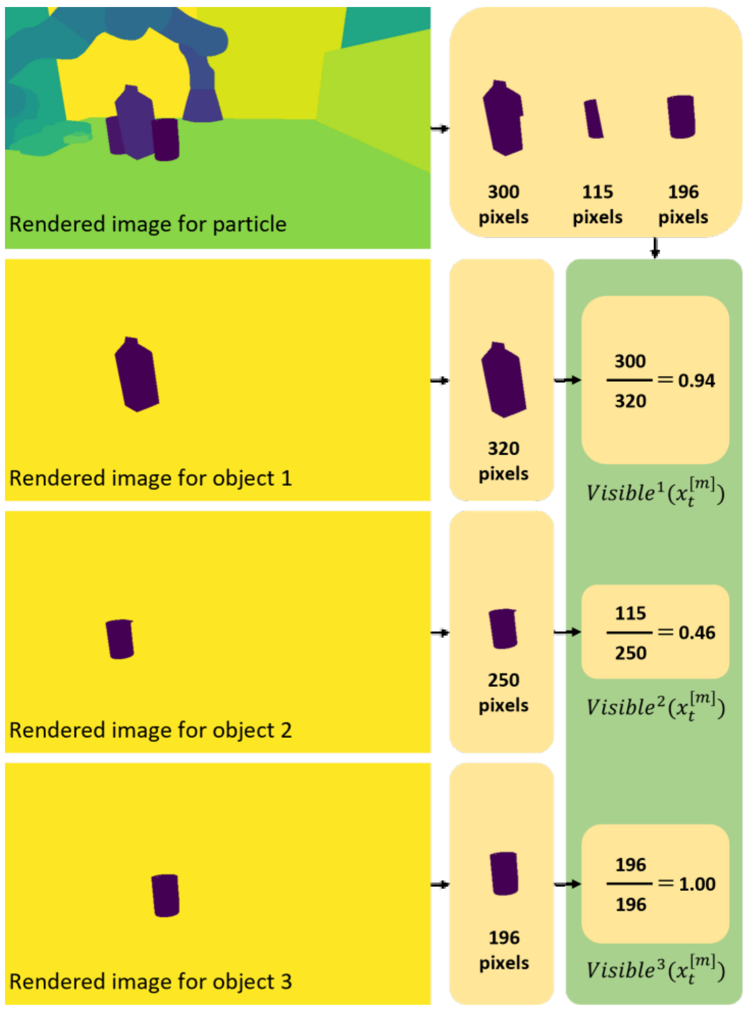


Figure 5.3: Visibility score calculation for three objects in an example particle. A segmented/footprint image is rendered for each object jointly in the particle (top-left) and individually (bottom-three-leftmost) at their corresponding poses in the particle. The pixels belonging to each object are counted in each of these images (top-right and bottom-three-middle). A visibility score for each object is calculated as the ratio of the unoccluded footprint to the total possible footprint (bottom-three-rightmost).

$Visible^i(x_t^{[m]})$ is higher than a V, then I consider the object in the particle to be visible, in the four conditions listed above. If it is lower, I consider it occluded in the particle. I determine V for each object type beforehand, by increasingly occluding it with an obstacle and identifying the degree of occlusion where the PE system starts to fail.

Depth Images

To compute $p({}^D z_t \mid \bar{x}_t^{[m]})$ in Eq. 5.3, this approach again involves using the simulated camera to render a depth image for each particle at every time step. Specifically, after updating each particle by motion model (Sec. 5.4.1), I render a depth image based on the poses of objects inside the intermediate particle $\bar{x}_t^{[m]}$, and then compare the rendered depth image ${}^{D}\bar{z}_t^{[m]}$ with the real depth image ${}^{D}z_t$ captured by the camera. Each particle needs to render only one depth image encapsulating the pose state information of all objects. Again, I use GPU parallelization to render the depth images, which markedly improves efficiency.

An error score for each particle is obtained by comparing the rendered depth images with the real depth image. I use the error score, $e_t^{[m]}$, using a visible surface discrepancy error function similar to the one proposed by Lee et al. [58]. However, Lee et al. use depth information on a per-object basis and therefore require segmentation of the real depth image. Instead, my joint multi-object treatment of the scene enables us to render the full scene directly, instead of relying on potentially inaccurate segmentation of individual objects.

I define my error score as:

$$e_t^{[m]} = \underset{\substack{\bar{\mathbf{p}} \in \, {}^{D}\bar{z}_t^{[m]} \\ \mathbf{p} \in \, {}^{D}z_t}}{\operatorname{avg}} \begin{cases} 0 & \text{if } |{}^{D}\bar{z}_t^{[m]}(\bar{\mathbf{p}}) - {}^{D}z_t(\mathbf{p})| < \beta \\ 1 & \text{otherwise} \end{cases} \tag{5.8}$$

where $\bar{\mathbf{p}}$ and $\mathbf{p}$ represent the pixels in ${}^{D}\bar{z}_t^{[m]}$ and ${}^{D}z_t$ respectively, β represents the threshold value used to assess the degree of correspondence between pixel values. If the absolute depth difference between corresponding pixels is less than the threshold β, these pixels are assigned to 0. Otherwise, the pixels are assigned 1, as shown in Fig. 5.4. This binary classification of pixel distances results in a new set of pixel values. Subsequently, these values are averaged to calculate the $e_t^{[m]}$. Using this method, I compute error scores, $\{e_t^{[m]}\}_{m=1,\dots,M}$, for all particles at time step t. Upon calculating the $\{e_t^{[m]}\}_{m=1,\dots,M}$, a normalization operation is performed to get $p({}^{D}z_t \mid \bar{x}_t^{[m]})$:

$$p({}^{D}z_t \mid \bar{x}_t^{[m]}) = \frac{e_t^{[m]} - \min(e_t^{[1]}, \dots, e_t^{[M]})}{\sum_{m=1}^{M} \left(e_t^{[m]} - \min(e_t^{[1]}, \dots, e_t^{[M]}) \right)} \tag{5.9}$$

The selection of the threshold β in Equation 5.8 is critical to the performance of the likelihood model. This parameter acts as a tolerance margin, defining how much discrepancy between the rendered and real depth values is considered negligible. The choice of β involves a direct trade-off: a value that is too small would make the error function overly sensitive to sensor noise and minor inaccuracies in the CAD models, potentially penalising even near-perfect poses. Conversely, a value that is too large would reduce the model's discriminative power, making it too permissive and unable to effectively distinguish between correct and incorrect poses. In my implementation, I adopt the value for β directly from the work of Lee et al. [58], upon which my error function is based. This approach aligns my methodology with a comparable state-of-the-art method, providing a well-established and validated starting point for this parameter. While more advanced techniques, such as using a trained neural

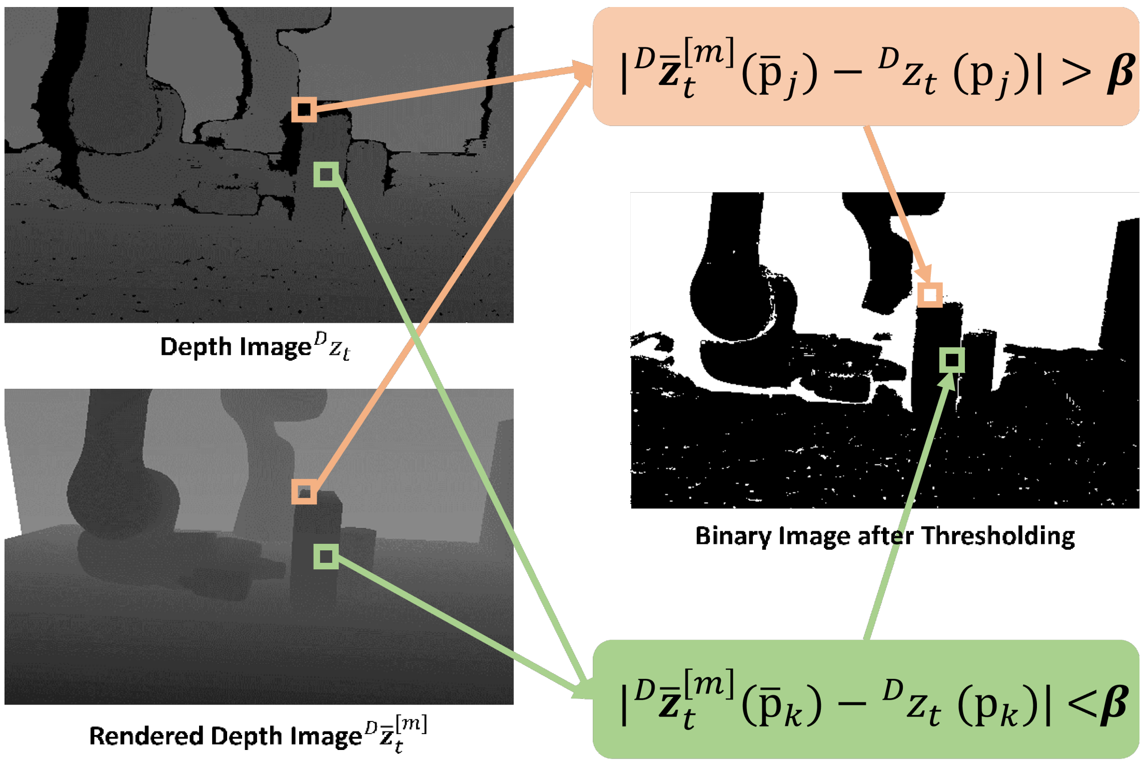

Depth Image $^{D}z_t$
Rendered Depth Image $^{D}\bar{z}_t^{[m]}$
$|^{D}\bar{z}_t^{[m]}(\bar{p}_j) - {}^{D}z_t(p_j)| > \beta$
Binary Image after Thresholding
$|^{D}\bar{z}_t^{[m]}(\bar{p}_k) - {}^{D}z_t(p_k)| < \beta$

Figure 5.4: I subtract the pixel values at corresponding locations between the two depth images. The absolute differences are then compared against a predefined threshold, β. For each pixel, if the absolute difference is less than β, the pixel in the resulting image is assigned a value of 0. Conversely, if the difference exceeds β, the pixel is assigned a value of 1.

network, could potentially determine an optimal or even adaptive β value, adopting a proven value from prior work ensures a robust and reproducible baseline for my experiments. The selected value was held constant across all experiments to ensure a fair comparison.

The motivation behind my model is to *quickly* compute a probability value without performing object-level segmentation or reasoning on the cluttered real-depth images. If object-level segmentations/pose estimates are available on the real-depth image, then more informative, and more symmetric, models can be built. However, object-level segmentation/pose-estimation on depth images (e.g., ICP-like schemes) are time-consuming, and given my particular focus on highly occluded scenes, are not reliable.

Computational Cost

To ensure efficient performance, the particle filter is updated (i.e., motion and observation updates are performed) at fixed time intervals, denoted as Δt. The most computationally demanding part of this process is the motion update, which relies on physics-based predictions. Consequently, Δt is selected based on the minimum time interval necessary to complete the physics simulations for all particles within the system.

5.5 Model Predictive Control (MPC)

In the previous section, I described a method for tracking the 6-D pose of objects. This section proposes a Model Predictive Control (MPC) framework that uses feedback from the particle filtering algorithm to push an object to a goal region. The tracking system enables the MPC to control object poses even when they are occluded. Figure 5.1 shows example manipulation tasks, with varying numbers of objects. The MPC approach was mainly developed by my co-author Rafael Papallas and I integrated it with my pose tracking system and evaluated the performance of the integrated system.

5.5.1 MPC Framework

Alg. 2 presents a typical MPC framework. It starts by getting an initial state of the system, such as the robot joints and the poses of objects, using a pose estimation system (line 2). In line 3, it generates a straight line trajectory from the end-effector's position to the goal region. The MPC algorithm then repeatedly, until successful or timeout (line 4), call the optimizer on line 5, passing in the current state (x_{current}) and a trajectory (τ). The operational principles of this optimiser are detailed in Section 5.5.2. If the optimization was unsuccessful, the MPC (lines 6 and 7) will be terminated. If optimization was successful, it executes n_u controls of the optimized trajectory in the real-world in line 8. In line 9, it updates the trajectory by removing the executed controls. Finally, The MPC algorithm updates the simulator with feedback from the real-world in line 10.

A particle is selected from the PBPF algorithm to update the simulation state in line 10. Recall that a particle represents the poses of all the objects, and then

Algorithm 2 Model Predictive Control (MPC) Framework

```
 1: procedure MPC
 2:   $x_{\text{current}}$ ← initialise simulation state from real-world
 3:   $\tau$ ← initialise trajectory
 4:   while not successful and not timeout do
 5:     $\tau$, optimizationSuccessful ← OPTIMIZE($x_{\text{current}}, \tau$)
 6:     if not optimizationSuccessful then
 7:       return
 8:     execute $n_u$ controls from $\tau$ in the real-world
 9:     $\tau$ ← remove the $n_u$ executed controls from $\tau$
10:     $x_{\text{current}}$ ← update simulation state from real-world
```

select $x_t^{[m*]}$, the particle closest to the mean of the particle set, as follows:

$$x_t^{[m*]} := \arg\min_m d(x_t^{[m]}, x_t^{[\mu]}) \tag{5.10}$$

where $x_t^{[\mu]}$ is the mean of the particle set, and $d(x_t^{[m]}, x_t^{[\mu]})$ is the distance of a particle, $x_t^{[m]}$, to the mean. Calculating the distance of a particle by summing up the distances of all the object poses to their corresponding mean pose. The distance for an object combines the Euclidean positional distance (δ_p) and the rotational distance (δ_θ; minimum angle between two orientations): $w_{\delta_p} \cdot \delta_p + w_{\delta_\theta} \cdot \delta_\theta$. This MPC variant is called, MPC-PBPF.

A simple MPC baseline can use feedback from a pose estimation system, like DOPE, in line 10 instead. Such an MPC baseline is implemented and called, MPC-DOPE. MPC-PBPF, MPC-DOPE, and an open-loop system without feedback are compared in Section 5.6.

5.5.2 Stochastic Trajectory Optimizer

Algorithm 3 presents the stochastic optimizer [59, 60] that is used in Algorithm 2 and line 5. It starts by rolling out the trajectory, τ (controls in the joint-space), using a physics simulator from the current state, x_{current}, to get a state sequence S (line 2). The cost is computed over that state sequence using a weighted cost function, $\mathcal{C}(S) : S \to \mathbb{R}$, in line 3, as follows:

$$\mathcal{C} = \sum_{i=1}^{|S|} w_1 \cdot d_{\text{g}} + w_2 \cdot d_{\text{o}} + w_3 \cdot d_{\text{z}} + w_4 \cdot \dot{q}_{\text{ee}} + w_5 \cdot c_s \tag{5.11}$$

where d_{g} is the Euclidean distance from the object to the goal region, d_{o} is the Euclidean distance from the end-effector to the object, d_{z} is the deviation of the end-effector along the z-axis from its initial state (keeping the robot gripper parallel to the table), $\dot{q}_{\text{ee}}$ is the end-effector's linear velocity, and c_s is an indicator function for collision with static obstacles. If successful, the MPC algorithm just returns the

Algorithm 3 Stochastic Trajectory Optimization

```
1: function OPTIMIZE($x_{\text{current}}, \tau$)
2:   $S \leftarrow$ rollout $\tau$ from $x_{\text{current}}$
3:   obtain the cost of rollout using $\mathcal{C}(S)$
4:   while not successful and not convergent do
5:     sample $k$ noisy trajectories from $\tau$
6:     rollout each noisy trajectory
7:     obtain cost for each rollout using $\mathcal{C}$
8:     $\tau \leftarrow$ best trajectory among the $k$
9:   return $\tau$, optimizationSuccessful
```

solution without any optimization (line 9). If not, it starts by sampling k trajectories from τ by introducing Gaussian noise to the controls (i.e., adding noise to each of the joints over the full trajectory horizon) (line 5). Then, the MPC algorithm rollouts each of the k trajectories (line 6). It computes a cost for each rollout (line 7). Note that the MPC algorithm parallelises lines 5-7. In line 8, it picks the trajectory with the lowest cost. The MPC algorithm repeats until it finds a successful trajectory or converges to a local minimum. If it hits a local minimum, the task is declared as a failure and I return false for `optimizationSuccessful`. While susceptible to local minima, local trajectory optimizers remain effective for generating contact-based manipulation trajectories [60, 59], balancing computational efficiency with the ability to escape some local minima by carefully tuning noise injection.

Please note that the input trajectory, τ, can be an initial rough trajectory (for example, from line 3 of Algorithm 2) or a warm-start with a good trajectory from a previous optimization process (as interleaving planning and execution in the MPC framework).

5.6 Experiments and Results

I present two sets of experiments evaluating the performance of my methods.

First, I evaluate the *tracking performance*. In these experiments, the robot executes a pre-determined motion (i.e., without MPC) interacting with the objects. I evaluate how accurate the estimated poses of the objects are (as compared to ground truth poses) during the robot motion. These experiments evaluating the tracking performance are presented in Sec. 5.6.1 to Sec. 5.6.3.

Second, I evaluate the *control performance*, where an MPC controller (as discussed in Sec. 5.5) pushes an object to a goal region. In these experiments, I evaluate how successfully the object is pushed into a goal region when the MPC controller uses my physics-based pose tracking methods as opposed to other baseline pose estimation methods. These experiments evaluating the control performance are presented in Sec. 5.6.4 to Sec. 5.6.6.

5.6.1 Tracking Tasks and Metrics

To verify the performance of tracking methods, I create different non-prehensile manipulation scenes. In these scenes, the robot performs an open-loop pre-defined motion. Examples can be seen in Table 5.1. I create 50 similar scenes:

- **One-object-push:** An example scene is shown in the top two rows Table 5.1. I create 20 similar scenes with different objects.
- **Two-objects-push:** An example scene is shown in the middle two rows Table 5.1. I create 20 similar scenes with different objects.

- **Three-objects-push:** An example scene is shown in the bottom two rows of Table 5.1. I create 10 similar scenes with different objects.

The above experiments are configured such that, during the manipulation, the objects can be partially or fully obscured by the robotic arm, obstacles, or other objects. This creates significant challenges to the tracking accuracy and effectiveness of manipulation tasks. I use a variety of objects from two datasets, YCB-dataset [53] and HOPE-dataset [61], as shown in Table 5.2. These objects include symmetric objects, and of varying sizes, demonstrating the applicability of the method to a wide range of objects. In all scenes, tracked objects only interact with the robot and other tracked objects. For the ground truth object poses, I used the OptiTrack system [1], which uses reflective markers placed on the objects for precise pose estimation, as shown in Fig. 3.4. I use an ar-marker on the robot to find the camera's pose with respect to the robot, i.e., for extrinsic camera calibration.

Each of runs contains an average of 8630 time-steps, i.e., data points at which I have an RGB-D image, robot joint information, and object pose ground truth information. Over 50 runs, I have a total of 431500 data points. The results I present below are averaged over these data points.

To evaluate the accuracy of pose tracking, I employ two metrics: the Average Distance of Discrepancy (ADD) [34] and the Symmetric Average Distance of Discrepancy (ADD-S). ADD measures the mean Euclidean distance between corresponding

Table 5.1: Example Tasks for Single and Multi-Object 6D Pose Tracking Experiments and Comparison of Performance of FOUN and PBPF-RGBD Methods

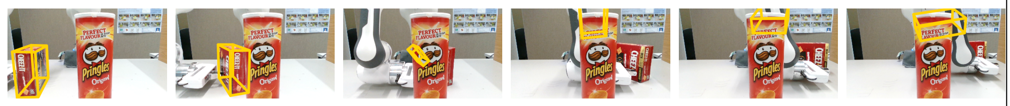
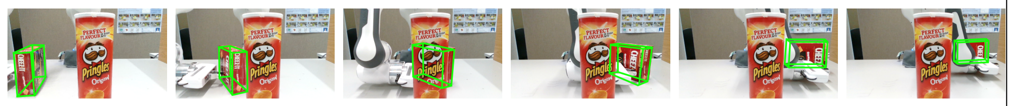
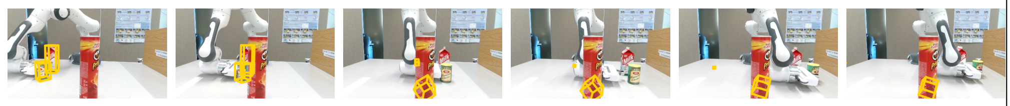
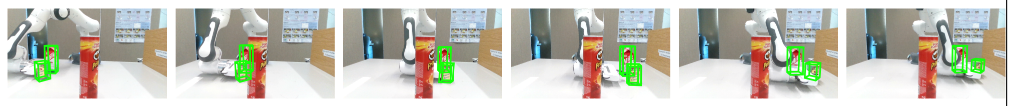
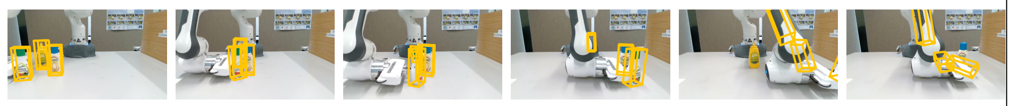
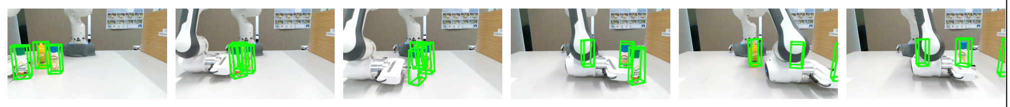
	Method	*time* →
Single object push	FOUN	
	PBPF-RGBD	
Two objects push	FOUN	
	PBPF-RGBD	
Three objects push	FOUN	
	PBPF-RGBD	

Table 5.2: Objects Used in Tracking Experiments

YCB Objects [53]		HOPE Objects [61]					
Cracker	**Soup**	**Ketchup**	**Mayo**	**Milk**	**SaladDressing**	**Parmesan**	**Mustard**

points on the tracked object model and the object model at the ground truth pose. ADD-S extends this by considering the mean distance under the best permutation of model points, making it robust to symmetrical objects where orientation discrepancies might otherwise be misleading. I specifically report the *area under curve* (AUC) [34] of ADD and ADD-S. This is the metric most commonly used to evaluate pose estimation systems' accuracy for varying accuracy thresholds [34, 14, 40, 36, 62].

I release this dataset of 50 experiments (431500 data points) including timestamped RGB-D observations, robot joint values, object pose ground truth values, and object/robot/environment 3D models, for others in the community to be able to use not only the RGB-D information (which is often provided in other similar datasets) but also physics-based inference (enabled by timestamped robot joint values and 3D models).

5.6.2 Implementation of Tracking Methods

I evaluated four different methods for tracking objects[1].

1) **Diff-DOPE:** Diff-DOPE [62] refines the object's pose at every time step, using prior estimates provided by DOPE [36], a single-snapshot object pose estimation system. I used the official Diff-DOPE implementation[2].

2) **Diff-DOPE (Tracking):** I modified Diff-DOPE such that it uses a different prior estimate at each time step. Instead of using DOPE, I use the pose estimated by Diff-DOPE in the previous time step as the prior estimate. This makes Diff-DOPE capable of using information from previous time-steps, similar to the methods. For the very first frame of a run, I still use DOPE as the prior.

3) **FoundationPose:** FoundationPose [40] uses a combination of deep learning and geometric optimization to track an object. It is one of the leading methods listed on the worldwide BOP leaderboard[3] (at the time of writing). I used the official FoundationPose implementation[4].

4) **PBPF:** This Method as explained in Sec. 5.4. Particularly, I implemented three versions of the method: **PBPF-RGB**, **PBPF-D** and **PBPF-RGBD**. The

[1]Experiments were performed on CPU: Intel(R) Xeon(R) W-2295 CPU@3.00GHz; GPU: NVIDIA GeForce RTX 3080; RAM: 192906 Mb

[2]https://github.com/NVlabs/diff-dope

[3]https://bop.felk.cvut.cz/leaderboards

[4]https://github.com/NVlabs/FoundationPose

distinction among the three versions lies solely in their observation model. Specifically, PBPF-RGB uses only RGB images (Section 5.4.2), PBPF-D relies only on depth images (Section 5.4.2)[1], and PBPF-RGBD integrates both RGB and depth images (Section 5.4.2 and 5.4.2) as its observation model. **Implementation details:** The implementation details and the parameter values I used in all PBPF versions are: (a) *Motion model parameters.* I used the same coarse friction and restitution parameters for all objects: $\mu_{\text{friction}} = 0.1$ and $\sigma_{\text{friction}} = 0.3$, with the minimum sampled value capped at 0.001. $\mu_{\text{restitution}} = 0.9$ and $\sigma_{\text{restitution}} = 0.2$. I used different mass estimates for objects: μ_{mass} of Cracker, Soup, Milk, Parmesan and Mustard are 0.45 kg, 0.35 kg, 0.04 kg, 0.035 kg and 0.05 kg, respectively. μ_{mass} of Ketchup, Mayo and SaladDressing are the same at 0.06 kg. The σ_{mass} for Cracker and Soup is 0.5, and 0.1 for the remaining objects, with a minimum sampled value capped at 0.02 kg. σ_f: For position 0.005 m, for rotation 0.05 radians. I used the Pybullet physics engine [54] as the physics model, f_θ. I employ a multi-processing approach to initialize and manage each particle within individual PyBullet environments, using all 18 (36 virtual) CPU cores available on the computer. (b) *Observation model parameters.* As the PE system (used to estimate p_{PE} as explained in Sec. 5.4.2), I use DOPE since it is a single-pass neural network that was fastest among the ones I have tested, with relatively accurate pose estimation for known objects. σ_{PE}: For position 0.1 m and rotation 0.2 radians. Visibility parameters: the threshold V used for comparing $Visible^i(x_t^{[m]})$ to decide object visibility is set to 0.55 for all other objects (except 0.45 for the large Cracker object) when the PE system detects the object. If the PE system does not detect the object, the threshold V is set to 0.6 for all objects (0.5 for the Cracker); $\alpha_w = 0.33, \alpha_h = 0.6, \alpha_l = 0.55$ for all objects ($\alpha_h = 0.75, \alpha_l = 0.45$ for Cracker). Depth parameters: $\beta = 0.03$ m. (c) *Update frequency.* $\Delta t = 0.25s$. (d) *Number of particles.* These tests with different numbers of particles showed accuracy to plateau around 50 particles. On the other hand, due to computational cost, the number of particles needs to be reduced as more objects are tracked to allow the PBPF algorithm to run at the same update frequency. I use the following particle counts: $M = 70$ for one object, $M = 50$ for two objects, and $M = 40$ for three objects. (e) *Initialization.* I initialize particles at $t = 0$ by sampling from a Gaussian distribution. I use the pose from the PE system at $t = 0$ as the mean pose, and the standard deviations for initialization are 0.03 m and 0.2 radians. During initialization, if objects are penetrating each other or the robot, I move them in the direction of the contact normal until they are not penetrating.

5.6.3 Pose Tracking Results

Table 5.3 presents the overall average tracking performance of different methods. (Please note lower values are better for ADD and ADD-S, whereas higher values are

[1]PBPF-D uses the RGB-based DOPE at t = 0 to initialize the objects' poses.

Table 5.3: Overall average tracking accuracy for different methods (↓ represents lower is better, ↑ represents higher is better)

Method	ADD ↓	AUC-ADD ↑	ADD-S ↓	AUC-ADDS ↑
Diff-DOPE	0.125	42.8	0.101	53.8
Diff-DOPE (Tracking)	0.187	30.9	0.153	39.9
FoundationPose	0.118	52.3	0.100	59.8
PBPF-D	0.065	58.8	0.049	70.0
PBPF-RGB	0.088	46.0	0.068	59.6
PBPF-RGBD	**0.030**	**70.1**	**0.021**	**79.2**
PBPF-RGBD (Best Particle)	0.016	83.7	0.012	87.6

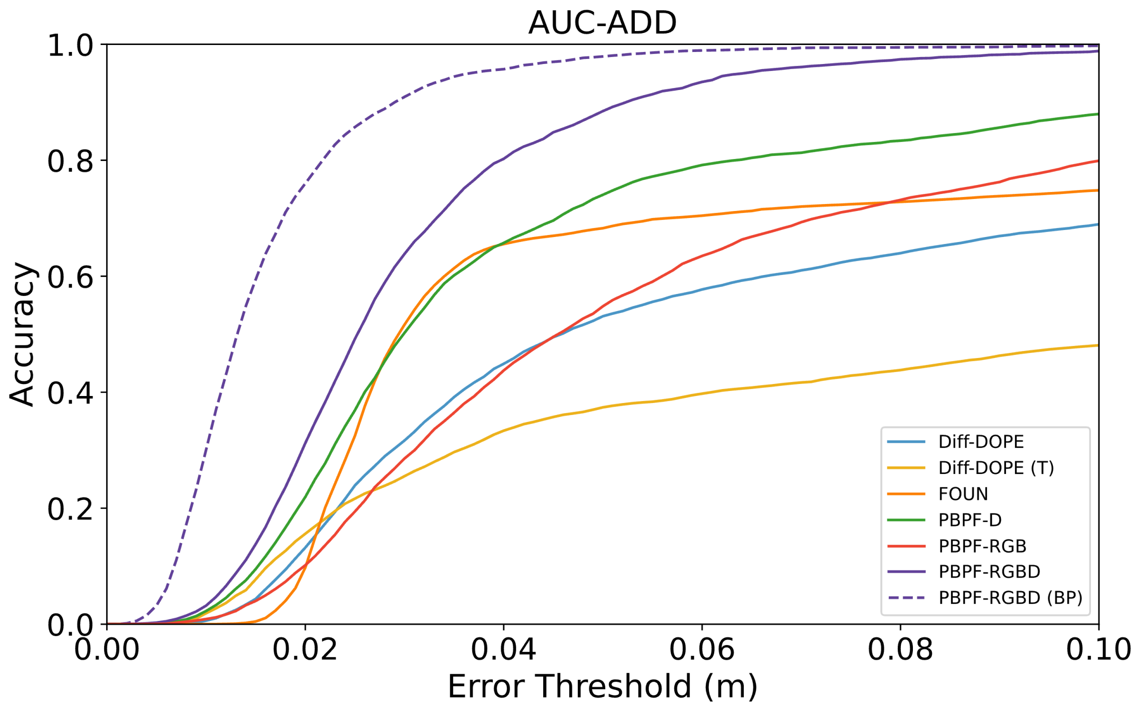


Figure 5.5: Accuracy versus Error threshold plots for different methods, used to compute Area Under the Curve (AUC). The horizontal axis represents the increasing threshold of error allowed, ranging from 0.00 to 0.10 meters, and the vertical axis shows the accuracy of the methods, ranging from 0 to 1, for that given threshold. (BP means Best Particle and T means Tracking.)

better for AUC.) I evaluated each method 100 times[1] in each of the 50 runs (giving us 100 different estimates for each of the 431500 data points) to accommodate the inherent randomness in some of these methods (e.g., the probabilistic nature of particle filtering). If a method failed to output a pose estimation at a certain time-point, I used its most recent output. PBPF outputs a set of particles, not a single

[1]All robot/camera/ground truth data were recorded in a ROS bag file during actual robot manipulation. These bag files were then replayed, respecting timestamps, 100 times for each different method.

Table 5.4: Object-specific accuracy of tracking for different methods

Method	Diff-DOPE		Diff-DOPE (T)		FoundationPose		PBPF-RGB		PBPF-D		PBPF-RGBD	
Metric	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S
Cracker	0.152	0.109	0.170	0.105	0.049	0.039	0.093	0.062	0.070	0.055	**0.032**	**0.025**
Soup	0.075	0.052	0.162	0.133	0.071	0.054	0.065	0.043	0.030	0.018	**0.029**	**0.017**
Ketchup	0.076	0.061	0.207	0.176	0.137	0.124	0.057	0.043	0.037	0.026	**0.028**	**0.020**
Mayo	0.155	0.137	0.191	0.167	0.080	0.065	0.071	0.054	0.070	0.051	**0.028**	**0.020**
Milk	0.118	0.092	0.150	0.117	0.223	0.208	0.084	0.066	0.037	0.025	**0.027**	**0.019**
Mustard	0.177	0.152	0.191	0.162	0.133	0.109	0.071	0.051	0.132	0.104	**0.035**	**0.024**
Parmesan	0.118	0.100	0.178	0.151	0.115	0.093	0.076	0.055	0.042	0.028	**0.031**	**0.020**
SaladDressing	0.093	0.075	0.299	0.268	0.148	0.121	0.240	0.230	0.122	0.107	**0.032**	**0.021**
Mean	0.125	0.101	0.187	0.153	0.118	0.100	0.088	0.068	0.065	0.049	**0.030**	**0.021**

pose estimate. To compute the accuracy values, I used the particle that is closest to the mean of the particles. I also present the AUC curves in Fig. 5.5 (which show accuracy vs error threshold and is used to compute the area under the curve)

The results presented in the table show that PBPF-RGBD performs significantly better, compared to baselines. This is possible because this method uses the additional robot control, and the physics-rollout, information while estimating objects' poses.

Among these results, I also show the accuracy of the "best particle" from the particle set (the last row of Table 5.3). The "best particle" is the particle that is closest to the ground truth pose. While it is impossible to know which particle is the best particle when one does not have access to ground truth, I report this, since it can be useful to put a bound on the expected error for conservative/worst-case manipulation planning systems, i.e., systems that plan robust robot motion that take into account all particles in the particle set.

Table 5.4 presents a more detailed view, showing ADD and ADD-S results for different objects, including data from all my 50 scenes (i.e., including when these objects may be in the scene together with multiple other objects). These results show that the PBPF-RGBD algorithm performs better consistently over all objects of various shapes and properties I have tested.

It is also clear from the results that using both RGB and depth information is useful and necessary to achieve the best performance. Table 5.3 demonstrates that PBPF-RGBD significantly outperforms PBPF-D (implying the value of RGB information) and PBPF-RGB (implying the value of depth information). I also note that PBPF-D outperforms PBPF-RGB. I think there are two reasons for this. First, PBPF-D in fact uses RGB information, even if only at initialization (at $t = 0$), which puts it at an advantage when compared to PBPF-RGB. Second, in my scenes, where significant occlusions frequently occur, the accuracy of RGB-only pose estimation systems, e.g., DOPE, suffers drastically.

However, RGB is particularly useful when objects have similar shapes. For example, the error results of the Mustard and SaladDressing objects in Table 5.4 reveal that PBPF-RGBD performs notably better than PBPF-D. This is due to the algorithm's difficulty distinguishing between the similarly sized Mustard and SaladDressing objects when only depth images are used, leading to decreased accuracy. This also applies to other target objects with similar size and shape.

I show some visual examples of tracking performance in Table 5.1. I use wireframes overlaid on the images to show the estimated poses of different methods (orange for FoundationPose and green for my method). (For my method, I draw the wireframe at the pose of the particle closest to the mean of the particles.) As can be seen in the figures, when all objects are visible, both systems perform well in estimating object poses. However, when there are occlusions (either behind other objects or behind the robot), FoundationPose estimates diverge significantly, whereas PBPF can estimate physically plausible poses for objects.

An advantage of my method is that it provides consistent tracking of objects even when there are heavy occlusions. To analyze this better, in Fig. 5.6, I evaluate each method's performance when the target object is at different levels of visibility to the camera. Here, the visibility values are computed similar to the process shown in Fig. 5.3, but this time using the ground truth poses of the objects. In Fig. 5.6, the solid lines show the mean ADD scores for all methods, computed over all objects at all time-steps in all of my 50 scenes. PBPF-RGBD demonstrates impressive accuracy across all visibility levels. Diff-DOPE's performance improves as visibility of the object improves, but stays well above PBPF-RGBD performance. FoundationPose performs well when the object is fully visible (i.e., visibility is 1.0), however suffers significantly, performing even worse than Diff-DOPE, at visibility values around 0.6. This is because, when FoundationPose loses track of the object behind occlusion, its accuracy suffers drastically, even after the object re-appears in the scene and becomes visible. (This can also be observed in Table 5.1, and the attached video.) To understand FoundationPose performance when I remove the effects of such highly erroneous estimates on the mean, in Fig. 5.6, I also present the median ADD performance with the dashed line. The median performance of FoundationPose improves as visibility improves, as expected. PBPF-RGBD's mean and median performances are almost identical, showing robustness, due to the stabilizing effect of the physics constraints.

5.6.4 Control Tasks and Metrics

In addition to the experiments I performed above, I also performed experiments to show how the real-time tracking of objects can be used for goal-directed manipulation in clutter. To evaluate the effectiveness of my method in manipulation tasks using MPC (Sec.5.5), I designed four different tasks as shown in Fig. 5.7:

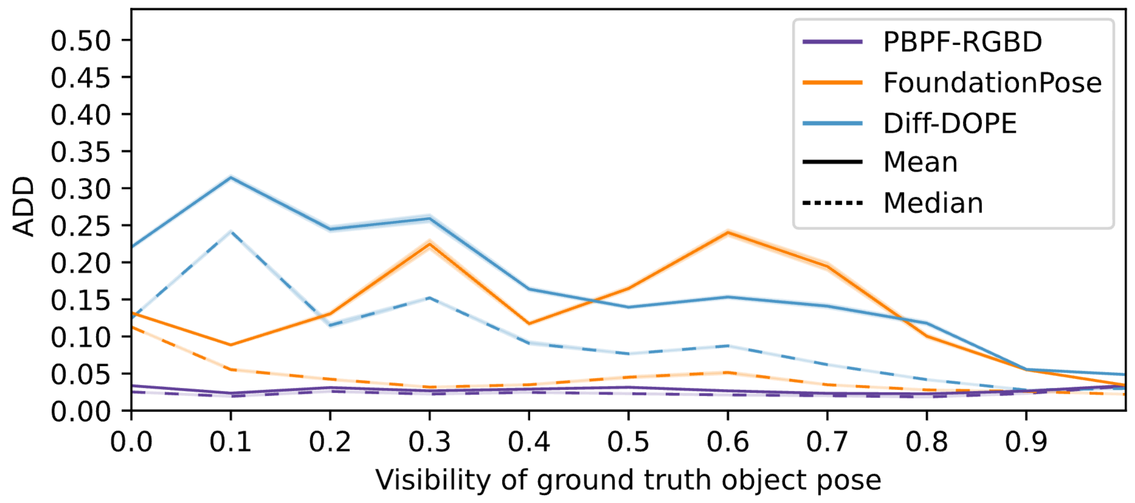


Figure 5.6: Estimation error (ADD) vs. visibility of ground truth object poses.

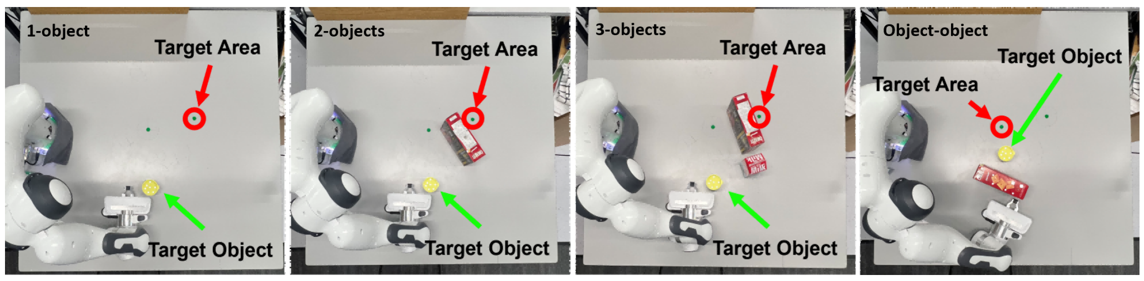


Figure 5.7: Examples of four different manipulation tasks, where an MPC controller aims to push a target object to a target area among other objects, either directly (first three tasks on the left), or indirectly by pushing another object (rightmost task).

- **1-Object**: The robot pushes an object to a target area. Two different target areas are used for different instances of this task.
- **2-Objects:** The robot pushes an object to a target area, while there is a second object in the way. Two different target areas are used for different instances of this task.
- **3-Objects:** The robot pushes an object to a target area, while there are two other objects in the way. Two different target areas are used for different instances of this task.
- **Object-Object:** The robot pushes an object to a target area, but indirectly, by pushing another object.

I ran each method 3 times for the same task and the same target area; i.e., in total, I ran each method 21 times (6 times on 1-Object tasks, 6 times on 2-Objects tasks, 6 times on 3-Object tasks, and 3 times on Object-Object tasks).

In executing control tasks, the primary consideration is whether the target object has reached the target area. The target area is defined as a circle with a radius of 5

Table 5.5: Success rates of controllers on different manipulation tasks

	OPENLOOP ↑	MPC-DOPE ↑	MPC-PBPF ↑
1-Object	6/6	6/6	6/6
2-Objects	2/6	1/6	6/6
3-Objects	1/6	0/6	6/6
Object-Object	0/3	0/3	3/3
Total	9/21	7/21	21/21

cm. Upon completion of a run, I assess success by measuring whether the object's centroid lies within this predefined target area. I run each control method for a maximum of 1000 controls, and if the target area is not reached by then, the method reports failure. The parameters for my MPC algorithm are: $w_{\delta_p} = 0.7$, $w_{\delta_\theta} = 0.3$, $n_u = 100$. The parameters for the stochastic trajectory optimiser are: $|\tau| = 1000$, $w_1 = 60000$, $w_2 = 20000$, $w_3 = 45000$, $w_4 = 3000$, $w_5 = 10000$, $k = 40$, I sample noise for each of the 7-DOF from a Normal distribution with mean 0 and $\sigma = 0.3$.

5.6.5 Control Methods

I use three different methods to perform the manipulation tasks:

- **OPENLOOP:** This method operates without any feedback during the manipulation task except the initial time-step. At the initial time-step, the object poses are estimated using DOPE, and Alg. 3 is used to optimize one trajectory to the goal. This trajectory is then executed open-loop, without any feedback.
- **MPC-DOPE:** This method uses Alg. 2. As state feedback, it uses the pose estimate of the objects provided by the DOPE algorithm at each time-step.
- **MPC-PBPF:** This method uses Alg. 2. As state feedback, it uses the pose estimate of the objects provided by the PBPF-RGBD algorithm at each time-step.

5.6.6 Results of Control Experiments

Table 5.5 shows the success rates of different methods in different tasks. In all 21 experiments, MPC-PBPF successfully moved the target object to the target area, even when there were significant occlusions of the target and other objects. While OPENLOOP and MPC-DOPE were also successful for the easier 1-Object task, they had significantly lower success rates for tasks with more objects, where the lack of accurate knowledge of object poses was detrimental.

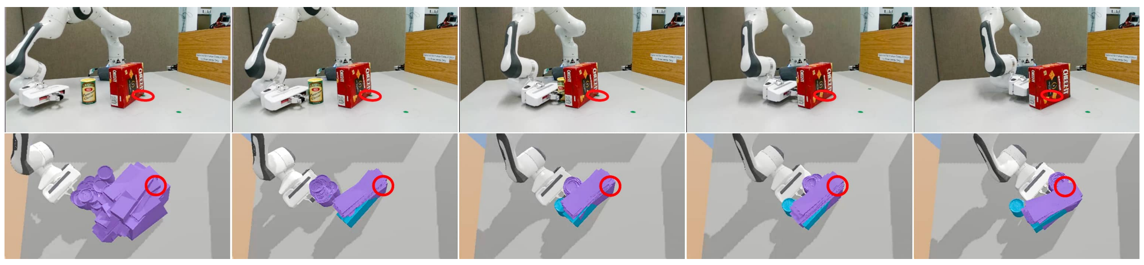


Figure 5.8: Example MPC-PBPF execution, where the task is to push the cylindrical Parmesan object to the target area (shown as a red circle), while there is another object in the way (Cheezit object). In the second row, the particles of PBPF-RGBD are shown in purple at each timestep. The blue particles show the poses reported by DOPE at those time-steps.

Fig. 5.8 shows MPC-PBPF performance in an instance of 2-Objects task. The second row shows the particles in purple, while it also shows the DOPE detections of the objects in blue. The red circle shows the target region. Even when the target object (Parmesan) is completely occluded, MPC-PBPF can still accurately push the object to the target area. In contrast, when the DOPE method is employed, occlusion of the object leads to incorrect predictions, resulting in manipulation task failures. Similarly, Fig. 5.1 shows MPC-PBPF performance in an instance of 3-Objects task. Corresponding videos of these experiments can be seen in the multimedia attachment.

5.7 Further Analysis & Limitations

In this section, I perform additional experiments and analysis to understand the limitations of my approach.

5.7.1 Improvements through CPU and GPU parallelism

To achieve real-time performance, I used both CPU and GPU parallel computing. CPU multi-threading was used to run physics predictions for multiple particles concurrently, while the GPU handled the rendering computations.

CPU parallel computing

To accelerate the computational physics prediction step of my particle filter, I employ a parallel framework built upon Python's multiprocessing library. The architecture use a process-per-particle model, establishing a persistent worker pool where a dedicated operating system process manages each of the M particles. This design is highly effective for the CPU-bound task of running independent physics simulations, as it fully leverages multi-core processors by circumventing Python's Global

Interpreter Lock. Communication is handled asynchronously via a command pattern: the main process dispatches tasks to each worker through a dedicated queue. The resulting predicted poses from each simulation are then written to a synchronised dictionary that is shared among all processes, allowing the main thread to gather results efficiently. This persistent worker architecture minimises process creation overhead, significantly improving the computational efficiency of the filter's prediction stage. Fig 5.9(a) shows the workflow of CPU parallel computing.

GPU parallel computing

The GPU pipeline was developed and implemented by my co-authors Jaina Moddisett and Markus Billeter. The GPU-based rendering pipeline, built using the Vulkan API, is optimised to efficiently compute the per-particle visibility and error scores required by my tracking framework. The system leverages hardware occlusion queries to count the visible pixels for each object, using a sophisticated multi-pass approach with manipulated depth tests to avoid costly buffer clears and image readbacks. For the final error calculation, a compute shader pipeline evaluates the pixel-wise discrepancy between the rendered synthetic depth image and the real observation. A highly efficient parallel reduction algorithm is then employed to sum these per-pixel results directly on the GPU. This entire process is designed to minimise data transfer across the PCIe bus; only the final scalar results—occlusion counts and error scores—are returned to the CPU, enabling the high-throughput evaluation necessary for real-time performance. Fig 5.9(b) shows the workflow of GPU parallel computing [63].

As quantified in Table 5.6, using these parallelisation techniques provides a significant performance enhancement. This reduction in latency is crucial for making my tracking framework viable for real-time, closed-loop manipulation tasks.

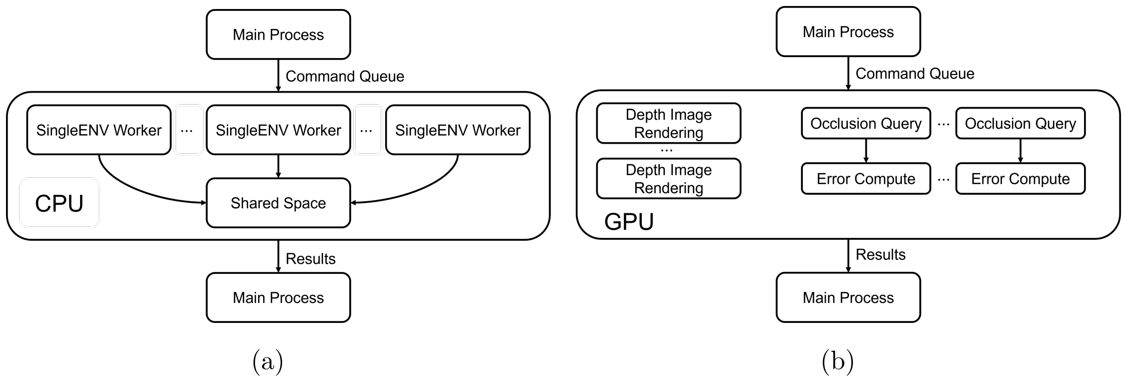


Figure 5.9: (a) CPU parallel computing workflow; (b) GPU parallel computing workflow.

Table 5.6: Ablation Study of Parallelism on Component-wise Processing Time (s)

	1-obj M=70	2-obj M=50	3-obj M=40
Before using parallelism			
Physics Update	0.132	0.365	0.749
Depth Update	42.35	40.61	41.93
After using parallelism			
Physics Update (CPU)	0.102	0.096	0.106
Depth Update (GPU)	0.101	0.073	0.073

5.7.2 Robustness to uncertainty in object friction and mass

My method requires mean and variance values for the mass (μ_{mass} and σ^2_{mass}) and friction ($\mu_{friction}$ and $\sigma^2_{friction}$) parameters of the objects. I performed experiments, analyzing how sensitive my method's performance is to the accuracy and uncertainty of these physics parameters of objects. I present these results in Table 5.7, where the top sub-table (the first three rows of the table) shows the results of experiments where I varied only the μ_{mass} of objects and measured the performance of my method PBPF-RGBD. As shown in the table, for these experiments, as μ_{mass} I used the values 0.01, TM, 0.5, 1.00, and 5.00. (TM stands for the "true mass" of an object, as reported in Sec. 5.6.2.) Since my complete dataset includes scenes with objects of a wide range of true mass values (some ten times heavier than the others), to make the effect of varying mass values clear, in these experiments I only used the subset of the dataset that includes scenes with objects of similar true mass between 0.03 kg and 0.06 kg, corresponding to Parmesan, Mustard, Ketchup, Mayo, SaladDressing. (Hence, the TM results are slightly different than the results I report for my method in Table 5.3 using the complete dataset). The results show that, while the method with the true mass, TM, performs best, other μ_{mass} values did not result in drastic drops in the performance. I hypothesize that there are two reasons for this. (1) My method is capable of showing some robustness due to the intentional uncertainty introduced into the system using the sampling variances. To test this, I ran my method again with different μ_{mass} values, but this time with the three variance parameters set to zero ($\sigma_{friction}, \sigma_{mass}, \sigma_f = 0$), as shown in the second sub-table. As expected, the overall performance significantly suffered, but also the difference between the TM performance and other mass values was more significant, confirming my intuition. (2) Mass of an object is less consequential in slower interaction tasks, and my dataset heavily includes slower pushing tasks. This agrees with quasi-static analysis of sliding/pushing [64], where it is shown that, when object accelerations are

Table 5.7: Tracking accuracy for different parametrizations

μ_{mass}	0.01	TM	0.50	1.00	5.00
ADD ↓	0.028	**0.025**	0.028	0.029	0.029
AUC-ADD ↑	71.9	**74.2**	71.6	71.1	71.0
$\mu_{mass}(\sigma_{fri}, \sigma_{mass}, \sigma_f = 0)$	0.01	TM	0.50	1.00	5.00
ADD ↓	0.093	**0.069**	0.101	0.094	0.103
AUC-ADD ↑	37.5	**46.1**	35.8	34.2	32.4
μ_{mass}(Poking)	0.01	TM	0.50	1.00	5.00
ADD ↓	0.058	**0.046**	0.065	0.060	0.060
AUC-ADD ↑	51.8	**55.7**	50.1	48.9	48.8
$\mu_{friction}$	0.01	0.10	0.25	0.75	1.00
ADD ↓	0.043	**0.028**	0.031	0.041	0.052
AUC-ADD ↑	63.4	**72.2**	69.8	65.7	63.2
$\mu_{friction}(\sigma_{fri}, \sigma_{mass}, \sigma_f = 0)$	0.01	0.10	0.25	0.75	1.00
ADD ↓	0.087	**0.067**	0.069	0.088	0.119
AUC-ADD ↑	41.3	**56.2**	50.0	41.5	32.5
$\sigma_{friction/mass}(\sigma_f = 0)$	0.1×	1.0×	2.0×	5.0×	10×
ADD ↓	0.138	**0.053**	0.057	0.056	0.069
AUC-ADD ↑	31.7	**54.2**	52.6	51.2	48.6
σ_f pos:	0.000	0.005	0.010	0.020	0.050
σ_f rot:	0.00	0.05	0.10	0.20	0.50
ADD ↓	0.063	**0.028**	0.080	0.170	0.693
AUC-ADD ↑	53.8	**72.2**	50.0	34.6	9.70

negligible, (i.e., when the energy induced into the object through the robot finger is small enough that it is *immediately* dissipated by the object-ground frictional forces, and hence the pushed object does not lose contact with the finger during pushing) the mass of the object can mostly be ignored in predicting its motion. To test this, I performed new experiments with the robot, where I poked an object with high impact, so that the object accelerates and keeps sliding after contact. (I present more details about this experiment in Sec. 5.7.5.) The third sub-table (μ_{mass}(Poking)) shows the results, where the effects of mass are more significant, again confirming my insight.

Similarly, I performed experiments using different values for the $\mu_{friction}$ parameter of my method. While measuring the correct coefficient of friction is more difficult than measuring mass, I estimate it to be between 0.1 and 0.25 for my objects, estimated using a protractor and sloped surface. The results in the sub-tables show that the method performs best with correct friction values, and again that setting $(\sigma_{friction}, \sigma_{mass}, \sigma_f = 0)$ makes this more pronounced.

I also measured the performance under varying variances of the physics parameters, shown in sub-table "$\sigma_{friction/mass}(\sigma_f = 0)$". For these experiments, I set $\sigma_f = 0$, since it would interfere with assessing the impact of variance values for friction and mass. In this sub-table, a value of 0.1× indicates scaling the original values reported in Sec. 5.6.2 by 0.1, and similarly for other columns. As expected, setting the variance values too low or too high degraded performance, since too low values make the filtering less robust to uncertainty, but too high variance values diminishes the information provided by the physics model.

5.7.3 Effect of motion noise, σ_f

As shown in the bottom sub-table of Table 5.7, I varied the positional and rotational motion noise, σ_f, to investigate the effect of this parameters on the performance of my system. The results show that tuning this parameter is important, as the performance of the system is significantly effected. Note, however, that the motion model noise σ_f reflects the discrepancy between the physics model and the real-world physics, and therefore should be estimated only once for the physics model (e.g., physics engine) and can then be fixed. Therefore, given a dataset with a ground truth (such as the dataset I are providing), if a new physics engine is used with my method, σ_f should be calibrated to the best performing value, and can then be fixed for later use of the method.

5.7.4 Effect of number of objects

An important limitation of my system is the computational cost of physics simulations, which increase with the number of physically interacting objects. In my original dataset, in Sec. 5.6.1), my experiments were limited to 3 simultaneously interacting objects. Up to three objects, my system did not show a degradation in performance (the first three sub-tables of Table 5.8). Performing experiments involving four or more objects presented significant challenges, particularly in acquiring accurate ground truth pose information. Furthermore, as more objects are tracked, the update time increases, making it difficult to update my filter frequently.

However, I still wanted to evaluate the performance of my method when the number of objects are increased. Therefore, I set up a scene with 4 objects and another with 5 objects, and performed pushing on these objects, carefully making sure that I still have ground truth readings (i.e., objects are visible to the OptiTrack cameras). I show one of these scenes in Fig. 5.10-top-left and also in the attached video. I show the performances of my method and baselines in the "4-Objects" and "5-Objects" sub-tables of Table 5.8. For these two scenes, I used only 40 particles, similar to my 3-object scenes. As expected, my method's performance is significantly degraded compared to its performance on my original dataset. Still, PBPF-RGBD performs better compared to baselines, since occlusions (which happen frequently in such crowded scenes) are much more of an issue for the baseline methods. In

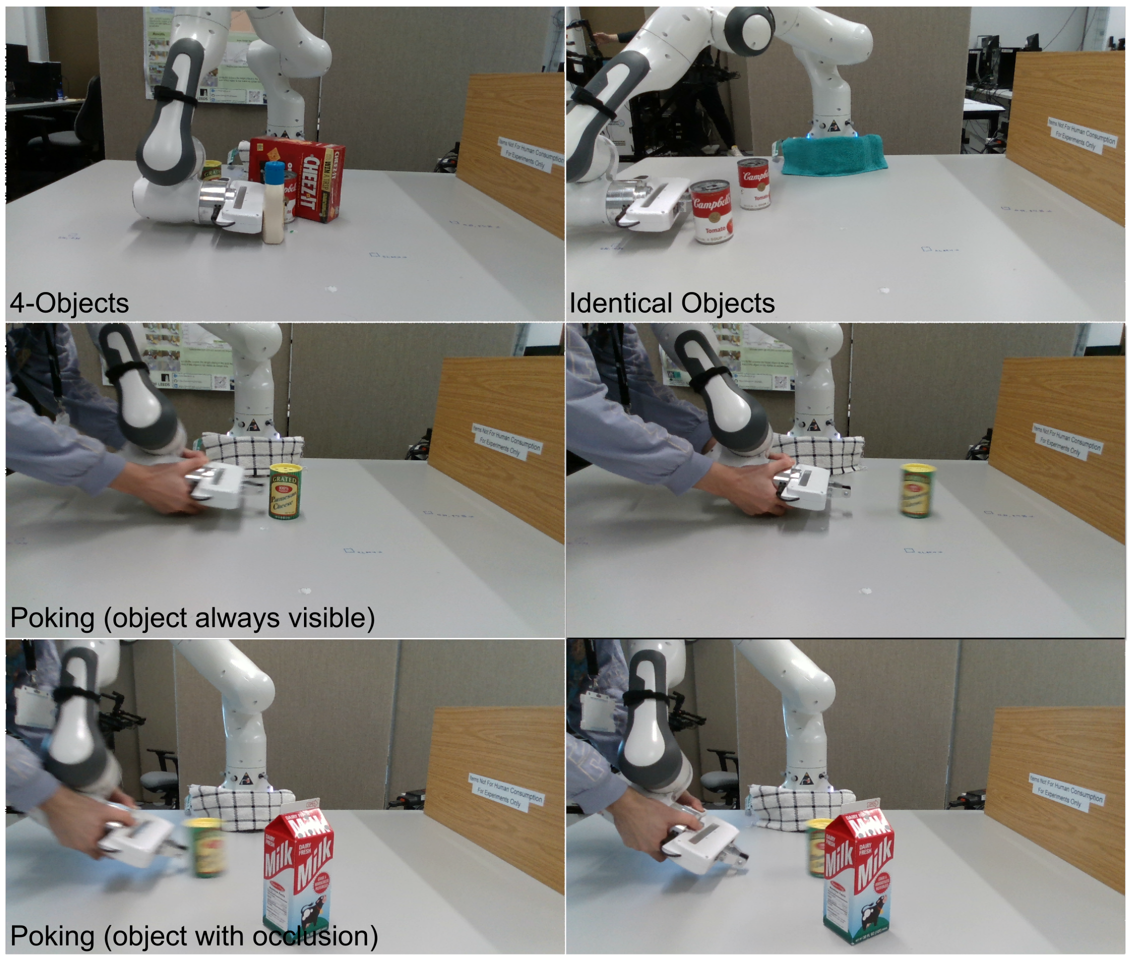

4-Objects
Identical Objects
Poking (object always visible)
Poking (object with occlusion)

Figure 5.10: Other interesting scenes. 1. The top-left shows robot manipulating 4 objects simultaneously. 2. Top-right shows robot manipulating two identical objects simultaneously. 3. The second row images show the robot poking/hitting an object to slide it away. 4. The bottom row also shows the robot poking/hitting an object to slide it away, but with occlusions to the camera. (Please also see attached video for video versions.)

Table 5.9 I present the computational time each component of my method takes for scenes with different number of objects, and when different number of particles, M, is used. These results are averaged over all runs with those number of objects in them. (Standard deviations are not shown, but are negligible.) As shown, the main bottleneck is clearly the physics update.

A realistic manipulation scenario is unlikely to involve tens of interacting objects that needs to be tracked. Therefore, in realistic settings, my system can still be beneficial.

5.7.5 High-impact interactions

Through my experiments, I also discovered that my system's performance degrades with the highly dynamic motion of the objects, for example when an object drops

Table 5.8: Overall tracking accuracy for other scenes

1-Object	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.132	0.126	**0.029**
AUC-ADD ↑	43.4	52.1	**71.1**
2-Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.098	0.191	**0.029**
AUC-ADD ↑	46.3	41.5	**70.7**
3-Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.147	0.043	**0.031**
AUC-ADD ↑	38.9	63.0	**69.0**
4-Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.218	0.054	**0.038**
AUC-ADD ↑	25.0	52.7	**61.9**
5-Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.195	0.055	**0.049**
AUC-ADD ↑	23.2	49.2	**59.7**
Poking (object always visible)	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.029	**0.028**	0.048
AUC-ADD ↑	71.4	**72.3**	56.2
Poking (object with occlusion)	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.090	0.191	**0.049**
AUC-ADD ↑	40.9	22.4	**52.9**
Identical Objects	Diff-DOPE	FoundationPose	PBPF-RGBD
ADD ↓	0.137	0.037	**0.026**
AUC-ADD ↑	25.4	63.1	**72.9**

down to the table, as shown in Fig. 5.11. In such cases where objects went through high-impact contact interactions, the inaccuracy of the physics predictions and the fast motion of the objects degraded the performance. While it proved difficult to collect ground pose truth information in scenes such as Fig. 5.11, to explore this issue further, I conducted experiments where the robot performed a fast planar hit

Table 5.9: Time (s) Consumed by Each Component

	1-obj	2-obj	3-obj	4-obj	5-obj
	M=70	M=50	M=40	M=40	M=40
Physics Update	0.102	0.096	0.106	0.141	0.167
Depth Update	0.101	0.073	0.073	0.089	0.092
RGB Update	0.031	0.029	0.040	0.051	0.055
Total Time	0.239	0.204	0.227	0.295	0.329

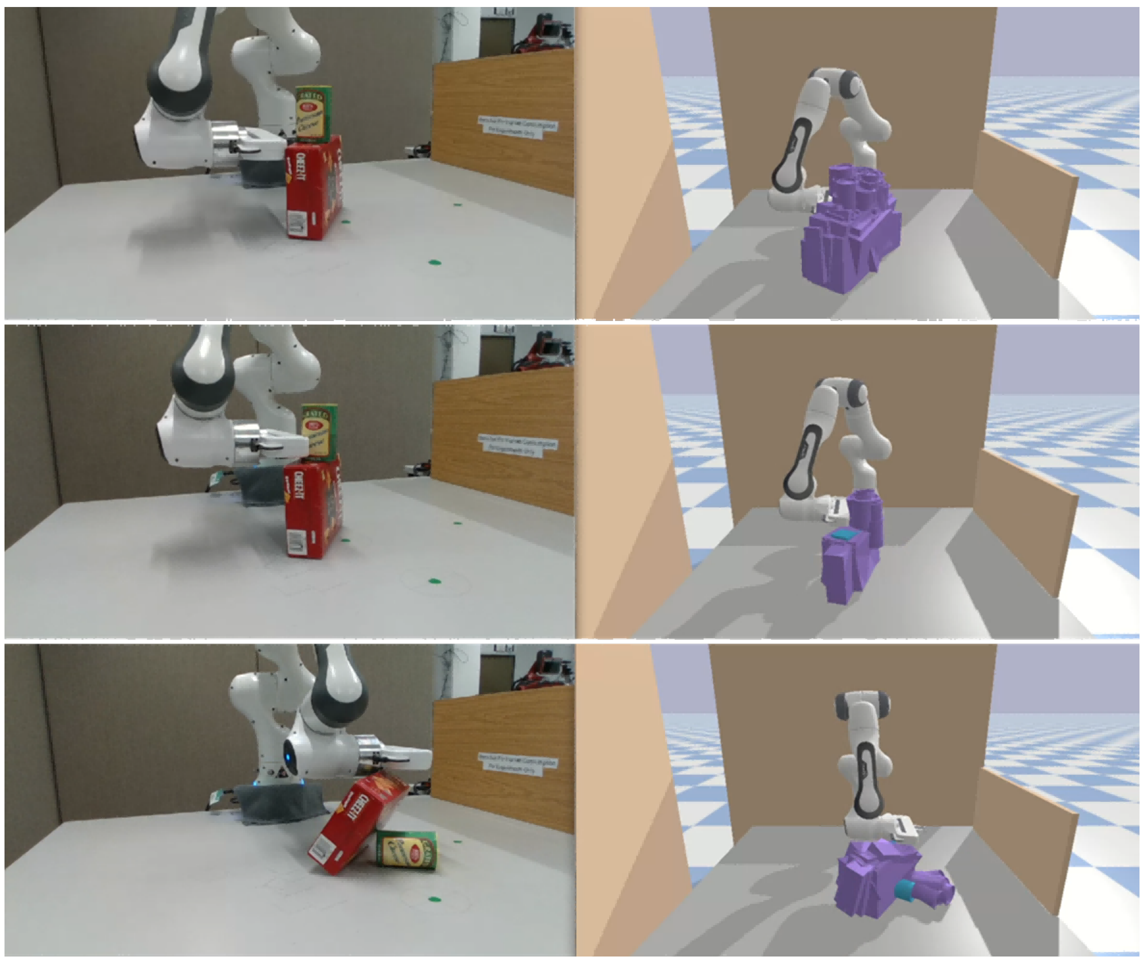


Figure 5.11: An example with a high-impact drop of an object, where my system's tracking performance is degraded.

on the object to create an impact interaction, resulting in the object sliding away from the hand, as shown in Fig. 5.10 bottom-two rows, and in the attached video. I call this "Poking". The performance of different methods for Poking is shown in Table 5.8. When the poked object is completely visible, my method performs significantly worse. Here, the inaccurate physics predictions hinder, rather than help, tracking. Still, when I experimented with a scene where the poked object was occluded for part of its slide on the table (bottom row of 5.10), PBPF-RGBD performed better than the baselines, as shown in the table.

I point out that my system is not limited to planar motion. In Fig. 5.12, I show an example tracking result of system, the robot lifts an object up and then down. The results from the experiment, which involved vertical movements, demonstrated that the system can effectively track such motions, as long as they are not highly dynamic.

5.7.6 Tracking scenes with identical objects

I also tested whether PBPF-RGBD can track scenes with identical objects. A scene with two identical objects can be seen in Fig. 5.10-top-right, and the attached video. The results are presented in the bottom rows of Table 5.8, which again shows better performance for PBPF-RGBD. To work under this setting, given multiple PE system estimates of the same type of object at the same time-point, PBPF-RGBD needs to identify which particular object (of that type) in a particle these different estimates belong to. To do this, I simply assign the nearest object of the correct type in a particle to the first PE estimate. If an object has already been assigned to a previous PE estimate, I assign the next nearest one.

5.8 Conclusions

This work addresses the critical challenge of tracking multi-objects during robotic non-prehensile manipulation under occlusions — scenes where conventional vision-based tracking methods often fail. By combining physics-based predictions within a filtering framework, I perform relatively robust multi-object tracking even when targets are significantly occluded by obstacles, robot arms, or other target objects. My key contribution is integrating physical prediction (robot joint states) with vision and depth information to resolve problems caused by occlusions. Moreover, I demonstrated the practical utility of my method by integrating it with a Model Predictive Control (MPC) framework. This integration facilitates tasks such as pushing objects to target areas; a key capability for applications where robots need to retrieve items from cluttered environments like baskets or shelves. While my experiments confirm the effectiveness of the proposed method, several limitations remain. Notably, the current implementation requires an initial estimate of the objects' 6D poses for particle initialization. The reliance on a physics engine introduces considerable computational costs, which pose challenges in maintaining both high tracking accuracy and speed when scaling to a large number of objects or dealing with high-impact scenarios. Future work will focus on eliminating the dependency

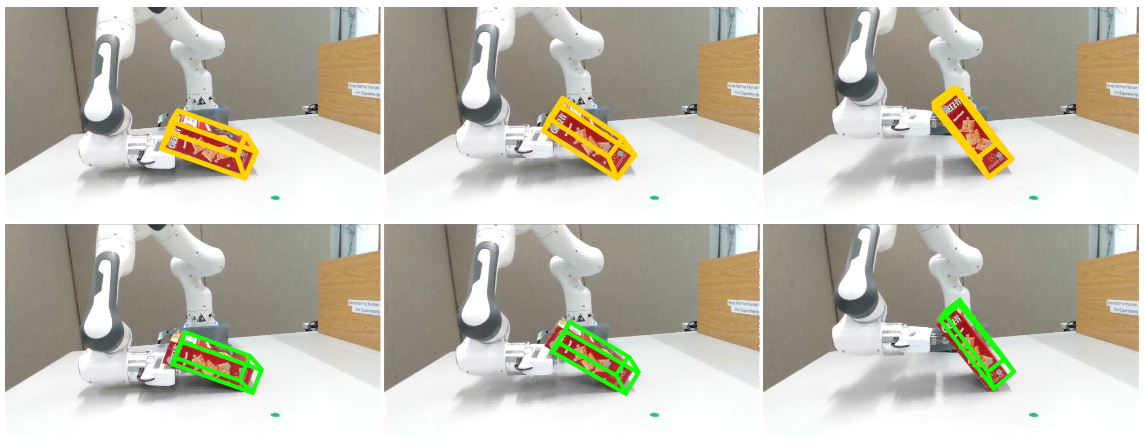


Figure 5.12: Robot lifting and object up and then down. Top row: FoundationPose. Bottom row: PBPF-RGBD.

on precise initial pose estimates. I also investigate the use of faster physics engines (e.g., [65, 66]) to improve real-time performance without decreasing tracking precision.

Chapter 6

Conclusions & Future Work

The final chapter summarizes the thesis, highlighting the contributions to knowledge and suggesting areas for further research.

- Section 6.1 will introduce the summary of contributions.
- Section 6.2 will focus on limitations and directions for future research.
- Section 6.3 presents final remarks.

6.1 Conclusions

This thesis focuses on the 6D pose tracking of target objects in cluttered environments, specifically when manipulated by a robot through non-prehensile actions and under physics-based perception. Such problems commonly arise in various scenarios, as shown in Figure 6.1: for example, a humanoid robot retrieving an egg from a fridge or picking items from a crowded shelf in a warehouse. In these cases, to reach the target in a cluttered setting, the robot must continuously track the poses of moved objects to avoid planning errors and accurately locate the target in real-time.

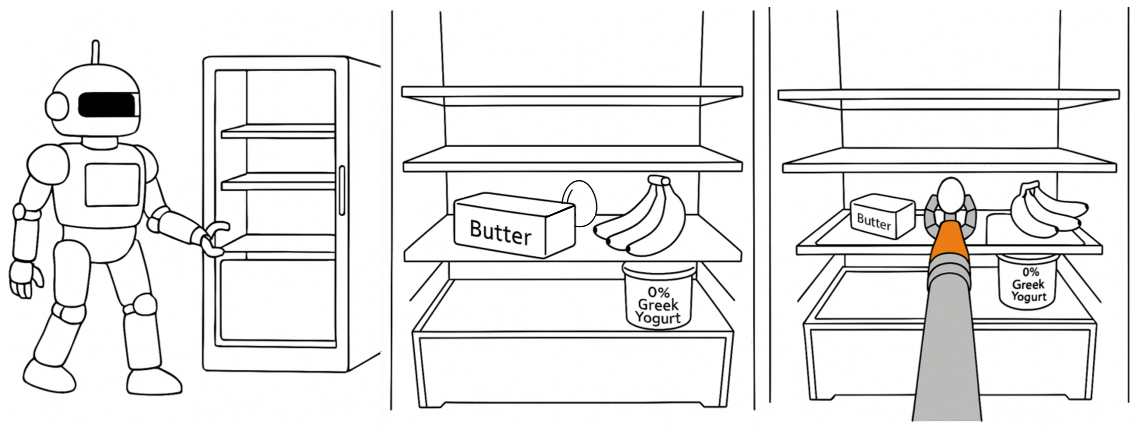


Figure 6.1: Which are some of the remaining challenges concerning this task?

In this thesis, I explore how to combine physics-based perception and camera vision information to address these challenges. The main focus is on these two sources of information and their integration with the particle filtering algorithm.

In Chapter 3, I first introduce a method that integrates physics-based perception with camera vision information into a particle filter algorithm. The physics engine serves as the motion model of the particle filter, providing the 6D poses of particles along with the robot's joint states. The vision information from the camera is used as input to the observation model, offering an estimate of the target object's 6D pose in the real environment. With these inputs, the system can infer physically plausible object poses, avoiding unrealistic results such as the object floating in the air or penetrating the robot or other obstacles. I compare this system with a baseline that relies entirely on RGB image-based neural network models for 6D pose estimation. Experimental results show that when the target object is occluded by obstacles, the RGB-based baseline completely fails. In contrast, my physics-based method maintains particles behind the obstacles, allowing it to continue estimating the object's 6D pose.

In Chapter 4, I extend the tracking from a single target object to multiple objects and propose a new evaluation metric based on a visibility score to select better-performing particles, thereby improving tracking accuracy. The visibility score helps filter out particles that remain in visible regions when the target is occluded, allowing the system to keep those hidden behind obstacles. Extending the algorithm to multi-object tracking increases the system's degrees of freedom, and the use of the visibility score significantly raises the computational cost. As a result, the algorithm runs more slowly. Chapter 5 presents my solution to these challenges.

Finally, the contribution of Chapter 5 is to move the entire physics-based perception computation to parallel processing on the CPU, which greatly improves efficiency. The visibility score calculation is also updated by using the GPU to render segmentation images for each object in every particle, making it possible to compute visibility scores more efficiently. Compared to calculating visibility scores directly within the physics simulation, this method significantly reduces both computation time and cost. Additionally, real-depth images from the camera are used. By comparing these with the depth images rendered in parallel for each particle, the system selects more accurate particles. Although this adds a step to the tracking process, the use of GPU parallel rendering ensures that it does not introduce significant overhead. As a result, the tracking algorithm maintains both accuracy and a fast update rate, even when tracking multiple target objects.

Despite the contributions, this study has several limitations:

- The method assumes all tracked objects are initially visible, as the particle initialization depends on their starting poses.

- It also assumes all objects in the scene are known and have 3D models; the current approach cannot handle unknown objects.
- Motion planning is not addressed; the robot's movements are controlled either manually or by existing planning algorithms.
- The system assumes the robot is the only agent interacting with objects. For instance, if a human moves objects in the scene, tracking may fail.
- While the method is designed to handle multiple objects, in practice, the number is limited by the computational cost of physics simulation as object count increases.
- In this work when objects go through high-impact contact interactions, the inaccuracy of the physics predictions and the fast motion of the objects degraded the performance.

In the next section, I will outline possible future research directions based on these limitations.

6.2 Future Work

In this section, I discuss several future directions and ways to improve this work within the context of non-prehensile manipulation in cluttered environments.

6.2.1 Partially Observable Initial Scenes

As discussed in earlier chapters, the robot interacts with different target objects in each particle. A key step in this process is initializing the particles based on the initial poses of the objects. In the main part of this thesis, I deliberately excluded this challenge by assuming that all target objects are fully visible at the initial time step ($t = 0$), allowing the focus to remain on the core tracking problem.

However, in real-world cases like the one shown in Figure 6.1, the egg may not be visible to the camera at the beginning, meaning some objects start in an unobservable state.

One possible solution is to initialize only the visible objects in each particle using their observed 6D poses. For the objects that are not visible, their poses can be initialized randomly. As the robot moves through the fridge or shelf during the retrieval task, the range of possible poses for the initially occluded objects can be gradually narrowed using the RGB-D images from the camera. Once a hidden object becomes visible again, the particles representing that object will converge toward its actual pose.

6.2.2 Handling unknown objects

In this thesis, all tracked objects are assumed to be "known," meaning their CAD models are available. The method cannot handle "unknown" objects, and so far, no effective solution has been found for tracking them. One possible direction is to train a large neural network model that can include features from a wider range of objects.

6.2.3 Handling deformable objects

In this thesis, the focus is on standard rigid bodies, such as cylinders and box-shaped objects, which are assumed to be easily pushed by the robot. However, in real-world settings—especially in everyday environments—many objects are deformable. These deformable objects introduce additional challenges in computer vision, pose tracking, and physics simulation. From a practical standpoint, exploring how to track deformable objects during manipulation would be a very meaningful direction for future research.

6.2.4 Handling more complex non-prehensile interactions and tracking more target objects

As discussed in Section 5.7.5, my method tends to perform less effectively in scenarios involving high-impact contact interactions. In real-world situations, many factors can influence how an object moves after strong contact—such as friction between objects, mass, inertia, and material properties. It is often difficult to accurately obtain these parameters in practice, and without precise values, the simulation may become unreliable. Tracking more target objects in parallel within the physics engine greatly increases the computational cost, which slows down the system. This is also the main reason why my method shows reduced accuracy in the poking task.

One possible solution is to adopt faster physics simulators, such as Isaac Sim or Gym[65] or Genesis[66]. Alternatively, learning-based physics engines could be explored to better model such interactions.

6.3 Final Remarks

In this chapter, I summarize the work and findings of this study. The results show that incorporating physics-based perception provides clear advantages in tracking objects manipulated by a robot, especially when those objects are occluded by obstacles or the robot itself. This research introduces reasonable simplifications and assumptions to focus on the problem of tracking non-prehensile manipulated objects using physics-based perception. I also highlight the limitations of the current work and suggest possible directions for future improvement.

Appendix A

Preliminary Results of Partially Visible Scene

I designed the scene shown in Fig. A.1, where two objects are placed in a basket, with the ketchup on top of the milk. The camera looks down into the basket from above, which causes the milk to be occluded at the start. As a result, the system cannot detect the milk, making accurate 6D pose estimation impossible.

This situation led me to think: what happens if, in the initial state, the system can only detect one object and then run my algorithm (in this case, the ketchup)? Following the standard algorithm, I would initialize particles only for the ketchup.

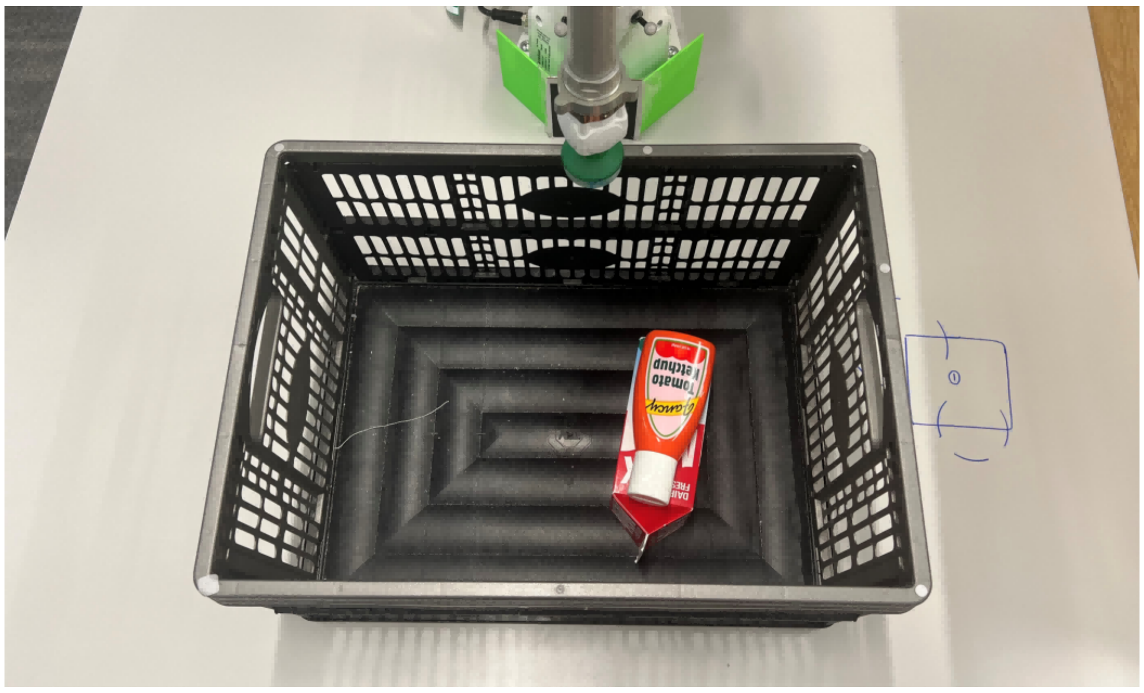


Figure A.1: Stacked objects in a basket scene. As shown in the figure, the ketchup is placed on top of the milk, and the camera looks down into the basket from above. The milk is occluded by the ketchup, making it impossible for the system to determine the milk's initial 6D pose.

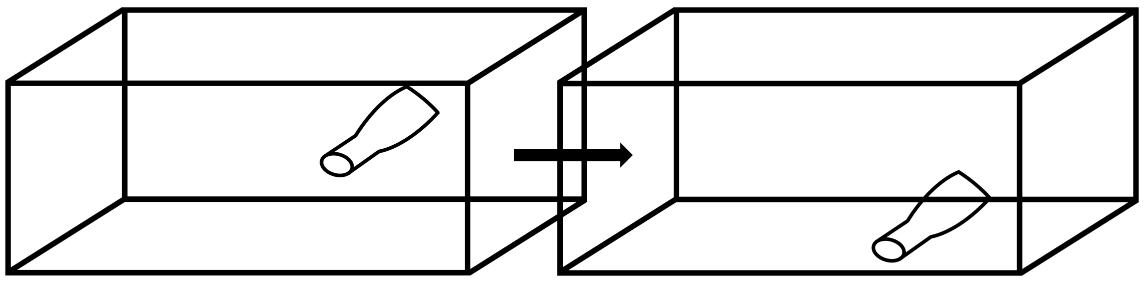


Figure A.2: Objects in the physics simulation. When there is no support beneath a particle, it will fall to the bottom of the basket.

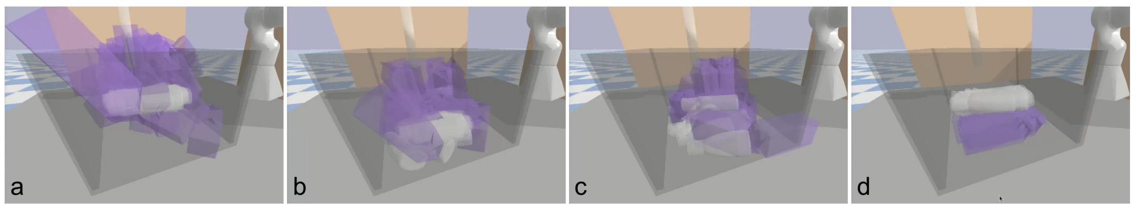


Figure A.3: A scene where only part of the target objects is visible at the initial state. (a) shows ketchup particles initialized based on the observed pose, while milk particles are randomly initialized. (b) shows that after running the algorithm, all ketchup particles fall to the bottom of the basket. (c) shows that after reinitializing and rerunning the algorithm, at least one ketchup particle remains in a stable state. (d) shows the result after reinitializing all particles based on the stable ones.

Since my method relies on a physics engine, if only the ketchup particles are initialized, they will fall to the bottom of the basket during execution, as shown in Fig. A.2.

So, I reconsidered the system design based on this situation. The algorithm can first initialize particles for the objects that are visible in the initial state using their estimated 6D poses. For the occluded objects, particles can be randomly initialized across the scene, as shown in Fig. A.3-a. Then, the system begins running. At this point, the system may encounter a case like the one in Fig. A.3-b, where all ketchup particles fall to the bottom of the basket. If this happens, the algorithm restarts the initialization for all target object particles until a case like Fig. A.3-c is reached—where at least one ketchup particle remains in a stable state, meaning its 6D pose is close to the observed pose. Using these stable particles, the algorithm reinitializes all target object particles in the scene, resulting in the final configuration as shown in Fig. A.3-d. Then, the system proceeds with motion planning and tracking of the target objects.

Bibliography

[1] *Motion Capture Systems -OptiTrack Webpage.* 1996. URL: `https : / / optitrack.com` (visited on 08/06/2024).

[2] Aude Billard and Danica Kragic. "Trends and challenges in robot manipulation". In: *Science* 364.6446 (2019), eaat8414.

[3] Frank Merat. "Introduction to robotics: Mechanics and control". In: *IEEE Journal on Robotics and Automation* 3.2 (1987), pp. 166–166.

[4] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Force control.* Springer, 2009.

[5] Bruno Siciliano and Oussama Khatib. "Robotics and the Handbook". In: *Springer Handbook of Robotics.* Springer, 2016, pp. 1–6.

[6] Nikolaus Correll, Bradley Hayes, Christoffer Heckman, and Alessandro Roncone. *Introduction to autonomous robots: mechanisms, sensors, actuators, and algorithms.* Mit Press, 2022.

[7] Nikolaus Correll, Kostas E Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M Romano, and Peter R Wurman. "Analysis and observations from the first amazon picking challenge". In: *IEEE Transactions on Automation Science and Engineering* 15.1 (2016), pp. 172–188.

[8] Antonio Bicchi. "On the closure properties of robotic grasping". In: *The International Journal of Robotics Research* 14.4 (1995), pp. 319–334.

[9] Matthew T Mason. *Mechanics of robotic manipulation.* MIT press, 2001.

[10] Mark Cutkosky and Paul Wright. "Modeling manufacturing grips and correlations with the design of robotic hands". In: *Proceedings. 1986 IEEE international conference on robotics and automation.* Vol. 3. IEEE. 1986, pp. 1533–1539.

[11] Manuel G Catalano, Giorgio Grioli, Edoardo Farnioli, Alessandro Serio, Cristina Piazza, and Antonio Bicchi. "Adaptive synergies for the design and control of the Pisa/IIT SoftHand". In: *The International Journal of Robotics Research* 33.5 (2014), pp. 768–782.

[12] Daniela Rus and Michael T Tolley. "Design, fabrication and control of soft robots". In: *Nature* 521.7553 (2015), pp. 467–475.

[13] Yu Sun, Joe Falco, Máximo A Roa, and Berk Calli. "Research challenges and progress in robotic grasping and manipulation competitions". In: *IEEE robotics and automation letters* 7.2 (2021), pp. 874–881.

[14] Bowen Wen, Chaitanya Mitash, Baozhang Ren, and Kostas E Bekris. "se (3)-tracknet: Data-driven 6d pose tracking by calibrating image residuals in synthetic domains". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 10367–10373.

[15] Muhammad Babar Imtiaz, Yuansong Qiao, and Brian Lee. "Prehensile and non-prehensile robotic pick-and-place of objects in clutter using deep reinforcement learning". In: *Sensors* 23.3 (2023), p. 1513.

[16] Zhen Xie, Xinquan Liang, and Canale Roberto. "Learning-based robotic grasping: A review". In: *Frontiers in Robotics and AI* 10 (2023), p. 1038658.

[17] Andrew Kimmel, Rahul Shome, Zakary Littlefield, and Kostas Bekris. "Fast, anytime motion planning for prehensile manipulation in clutter". In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2018, pp. 1–9.

[18] Changjoo Nam, Jinhwi Lee, Sang Hun Cheong, Brian Y Cho, and ChangHwan Kim. "Fast and resilient manipulation planning for target retrieval in clutter". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 3777–3783.

[19] Mukund Kumar Menon, VI George, and Shashank Goyal. "Non-prehensile Modes of Object Manipulation: A Comprehensive Review". In: *National Conference on CONTROL INSTRUMENTATION SYSTEM CONFERENCE*. Springer. 2018, pp. 449–462.

[20] Kevin M Lynch and Matthew T Mason. "Dynamic nonprehensile manipulation: Controllability, planning, and experiments". In: *The International Journal of Robotics Research* 18.1 (1999), pp. 64–92.

[21] Mehmet R Dogar and Siddhartha S Srinivasa. "A planning framework for non-prehensile manipulation under clutter and uncertainty". In: *Autonomous Robots* 33 (2012), pp. 217–236.

[22] Matthew T Mason. "Mechanics and planning of manipulator pushing operations". In: *The International Journal of Robotics Research* 5.3 (1986), pp. 53–71.

[23] Kevin M Lynch and Matthew T Mason. "Stable pushing: Mechanics, controllability, and planning". In: *The international journal of robotics research* 15.6 (1996), pp. 533–556.

[24] Misha Denil, Pulkit Agrawal, Tejas D Kulkarni, Tom Erez, Peter Battaglia, and Nando de Freitas. "Learning to Perform Physics Experiments via Deep Reinforcement Learning". In: *International Conference on Learning Representations*. 2022.

[25] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing". In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2016, pp. 30–37.

[26] Gaotian Wang, Kejia Ren, and Kaiyu Hang. "UNO Push: Unified Nonprehensile Object Pushing via Non-Parametric Estimation and Model Predictive Control". In: *arXiv preprint arXiv:2403.13274* (2024).

[27] P Aivaliotis, A Zampetis, G Michalos, and S Makris. "A machine learning approach for visual recognition of complex parts in robotic manipulation". In: *Procedia Manufacturing* 11 (2017), pp. 423–430.

[28] HM Ravindu T Bandara, MMSN Edirisighe, BLPM Balasooriya, and AG Buddhika P Jayasekara. "Development of an interactive service robot arm for object manipulation". In: *2017 IEEE International Conference on Industrial and Information Systems (ICIIS)*. IEEE. 2017, pp. 1–6.

[29] Wissam Bejjani, Rafael Papallas, Matteo Leonetti, and Mehmet R Dogar. "Learning a Value Function Based Heuristic for Physics Based Manipulation Planning in Clutter". In: *IROS workshop: Machine Learning in Robot Motion Planning*. 2018.

[30] Wisdom C Agboh and Mehmet R Dogar. "Pushing fast and slow: Task-adaptive planning for non-prehensile manipulation under uncertainty". In: *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*. Springer. 2020, pp. 160–176.

[31] Mehmet Dogar, Ross A Knepper, Andrew Spielberg, Changhyun Choi, Henrik I Christensen, and Daniela Rus. "Multi-scale assembly with robot teams". In: *The International Journal of Robotics Research* 34.13 (2015), pp. 1645–1659.

[32] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. "Model globally, match locally: Efficient and robust 3D object recognition". In: *2010 IEEE computer society conference on computer vision and pattern recognition*. Ieee. 2010, pp. 998–1005.

[33] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes". In: *2011 international conference on computer vision*. IEEE. 2011, pp. 858–865.

[34] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes". In: *Robotics: Science and Systems XIV* (2018).

[35] Yinlin Hu, Joachim Hugonot, Pascal Fua, and Mathieu Salzmann. "Segmentation-driven 6d object pose estimation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 3385–3394.

[36] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. "Deep object pose estimation for semantic robotic grasping of household objects". In: *arXiv preprint arXiv:1809.10790* (2018).

[37] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. "Densefusion: 6d object pose estimation by iterative dense fusion". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 3343–3352.

[38] Yann Labbé, Lucas Manuelli, Arsalan Mousavian, Stephen Tyree, Stan Birchfield, Jonathan Tremblay, Justin Carpentier, Mathieu Aubry, Dieter Fox, and Josef Sivic. "Megapose: 6d pose estimation of novel objects via render & compare". In: *arXiv preprint arXiv:2212.06870* (2022).

[39] Bowen Wen and Kostas Bekris. "Bundletrack: 6d pose tracking for novel objects without instance or category-level 3d models". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 8067–8074.

[40] Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. "Foundationpose: Unified 6d pose estimation and tracking of novel objects". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 17868–17879.

[41] Changhyun Choi and Henrik I Christensen. "RGB-D object tracking: A particle filter approach on GPU". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 1084–1091.

[42] Xinke Deng, Arsalan Mousavian, Yu Xiang, Fei Xia, Timothy Bretl, and Dieter Fox. "PoseRBPF: A Rao–Blackwellized particle filter for 6-D object pose tracking". In: *IEEE Transactions on Robotics* 37.5 (2021), pp. 1328–1342.

[43] Elizabeth A Olson, Jana Pavlasek, Jasmine A Berry, and Odest Chadwicke Jenkins. "Counter-Hypothetical Particle Filters for Single Object Pose Tracking". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 3853–3859.

[44] Sheng Zhong, Nima Fazeli, and Dmitry Berenson. "Soft tracking using contacts for cluttered objects to perform blind object retrieval". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 3507–3514.

[45] Sebastian Thrun. "Probabilistic robotics". In: *Communications of the ACM* 45.3 (2002), pp. 52–57.

[46] V Fox, Jeffrey Hightower, Lin Liao, Dirk Schulz, and Gaetano Borriello. “Bayesian filtering for location estimation”. In: *IEEE pervasive computing* 2.3 (2003), pp. 24–33.

[47] Gary Bishop, Greg Welch, et al. “An introduction to the kalman filter”. In: *Proc of SIGGRAPH, Course* 8.27599-23175 (2001), p. 41.

[48] *Posterior Distribution.* Agenter. 2017. URL: `https://www.zhihu.com/question/24261751` (visited on 01/15/2022).

[49] IJ Nagrath. *Control systems engineering.* New Age International, 2006.

[50] Petar M Djuric, Jayesh H Kotecha, Jianqui Zhang, Yufei Huang, Tadesse Ghirmai, Mónica F Bugallo, and Joaquin Miguez. “Particle filtering”. In: *IEEE signal processing magazine* 20.5 (2003), pp. 19–38.

[51] Randal Douc and Olivier Cappé. “Comparison of resampling schemes for particle filtering”. In: *4th Intl. Symposium on Image and Signal Processing and Analysis.* 2005.

[52] F Landis Markley, Yang Cheng, John L Crassidis, and Yaakov Oshman. “Averaging quaternions”. In: *Journal of Guidance, Control, and Dynamics* 30.4 (2007), pp. 1193–1197.

[53] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. “Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols”. In: *arXiv preprint arXiv:1502.03143* (2015).

[54] Erwin Coumans and Yunfei Bai. “Pybullet, a python module for physics simulation for games, robotics and machine learning (2016–2019)”. In: *URL http://pybullet. org* (2019).

[55] Eric R Fossum. “CMOS image sensors: Electronic camera-on-a-chip”. In: *IEEE transactions on electron devices* 44.10 (2002), pp. 1689–1698.

[56] Robert Lange and Peter Seitz. “Solid-state time-of-flight range camera”. In: *IEEE Journal of quantum electronics* 37.3 (2001), pp. 390–397.

[57] Zisong Xu, Rafael Papallas, Jaina Modisett, Markus Billeter, and Mehmet R Dogar. “Tracking and Control of Multiple Objects during Non-Prehensile Manipulation in Clutter”. In: *Under Review* (2025).

[58] Gijae Lee, Jun-Sik Kim, Seungryong Kim, and Kanggeon Kim. “6D Object Pose Estimation Using a Particle Filter With Better Initialization”. In: *IEEE Access* 11 (2023), pp. 11451–11462.

[59] Rafael Papallas, Anthony G Cohn, and Mehmet R Dogar. “Online replanning with human-in-the-loop for non-prehensile manipulation in clutter—a trajectory optimization based approach”. In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5377–5384.

[60] Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. "Predictive sampling: Real-time behaviour synthesis with mujoco". In: *arXiv preprint arXiv:2212.00541* (2022).

[61] Stephen Tyree, Jonathan Tremblay, Thang To, Jia Cheng, Terry Mosier, Jeffrey Smith, and Stan Birchfield. "6-DoF pose estimation of household objects for robotic manipulation: An accessible dataset and benchmark". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 13081–13088.

[62] Jonathan Tremblay, Bowen Wen, Valts Blukis, Balakumar Sundaralingam, Stephen Tyree, and Stan Birchfield. "Diff-dope: Differentiable deep object pose estimation". In: *arXiv preprint arXiv:2310.00463* (2023).

[63] Zisong Xu, Rafael Papallas, Jaina Modisett, Markus Billeter, and Mehmet R Dogar. "Tracking and Control of Multiple Objects during Non-Prehensile Manipulation in Clutter". In: *IEEE Transactions on Robotics* 41 (2025), pp. 3929–3947.

[64] Robert D Howe and Mark R Cutkosky. "Practical force-motion models for sliding manipulation". In: *The International Journal of Robotics Research* 15.6 (1996), pp. 557–572.

[65] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. "Isaac gym: High performance gpu-based physics simulation for robot learning". In: *arXiv preprint arXiv:2108.10470* (2021).

[66] *Genesis: A Universal and Generative Physics Engine for Robotics and Beyond.* Dec. 2024. URL: `https://github.com/Genesis-Embodied-AI/Genesis`.