Model-Based Non-Prehensile Manipulation on a Tabletop under Uncertain Dynamics

by

Aydin Ahmadi

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

September 12, 2025

Model-Based Non-Prehensile Manipulation on a Tabletop under Uncertain Dynamics

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Aydin Ahmadi

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assist. Prof. Barış Akgün (Advisor)

_____

Prof. Mehmet Gönen

_____

Assoc. Prof. Emre Ugur

Date: _____

[Dedicated to my father and grandfather]

ABSTRACT

Model-Based Non-Prehensile Manipulation on a Tabletop under Uncertain Dynamics

Aydin Ahmadi

Master of Science in Computer Science and Engineering

September 12, 2025

Data-driven planar pushing methods have recently gained attention as they aim to reduce extensive engineering effort and increase generalization compared to purely analytical approaches. Existing work focus narrowly on specific capabilities (e.g. a subset of side switching, precision, model-free training, single task etc), limiting their applicability in broader manipulation scenarios. In this work, we present a model-based framework for non-prehensile tabletop pushing, motivated by the goal of using a single learned model to handle multiple tasks and objectives without retraining. Our approach is built on a novel recurrent architecture that implicitly captures object–environment interaction dynamics through a GRU-based model, enhanced with additional non-linear layers to improve expressivity and stability. To complement this architecture, we design a unique state–action representation tailored to pushing dynamics, which allows the model to generalize effectively across, uncertain dynamics, push lengths, and task requirements. For control, we employ a sampling-based model predictive controller, specifically Model Predictive Path Integral (MPPI), which leverages the learned dynamics model to generate adaptive, task-oriented actions. The proposed framework enables side-switching during pushes, supports variable push lengths, and incorporates diverse objectives such as precise positioning, trajectory following, and obstacle avoidance. The model is trained from simulation using domain randomization to facilitate sim-to-real transfer. We first conduct extensive studies to evaluate the architectural components, demonstrating that our design decisions lead to improved prediction accuracy and more stable rollouts. We then assess the full system in both simulation and real-world experiments, using a Franka Panda robot with markerless visual tracking. Our results show high success rates for precise positioning under strict position and orientation thresholds, and strong performance in trajectory tracking and obstacle-avoidance tasks. Importantly, the versatility of our framework is demonstrated by solving multiple tasks simply by modifying the objective function of the controller,

without requiring any model retraining. One drawback of our approach is that we focus on a single type of object. However, we further improve our framework based on this by training a model capable of handling wider push lengths and by designing a balanced controller that selects the most effective action, thereby reducing the number of steps required to achieve longer-horizon objectives.

ÖZETÇE

Belirsiz Dinamikler Altında Masa st
nde Model Tabanlı Kavramayan Manip
lasyon
Aydin Ahmadi
Bilgisayar Bilimi ve Mühendisliği, Yüksek Lisans
12 Eylül 2025

Veriye dayalı düzlemsel itme yöntemleri, analitik yaklaşımlara kıyasla, mühendislik çabasını azaltmayı ve genelleştirmeyi artırmayı hedefledikleri için takın zamanda ilgi görmeye başlamıştır. Mevcut çalışmalar genellikle dar kapsamlı olup belirli yeteneklere (ör. taraf değiştirme, hassasiyet, modelsiz öğrenme, tek görev vb.) odaklanmakta ve bu da daha geniş manipülasyon senaryolarındaki uygulanabilirliklerini sınırlamaktadır. Bu çalışmada, yeniden eğitime gerek duymadan tek bir öğrenilmiş model kullanarak birden fazla görevi ve amacı yerine getirmeyi hedefleyen model-tabanlı bir masaüstü itme çerçevesi sunulmaktadır. Yapılan çalışmaada, GRU-tabanlı bir model aracılığıyla nesne–çevre etkileşim dinamiklerini örtük biçimde yakalayan ve ek doğrusal olmayan katmanlar ile ifade gücü arttırılmış bir mimari tasarlanmıştır. Ek olarak, bu mimariye uygun özgün bir durum-hareket temsili geliştirilmiştir. Mimari ve temsil, itme becerisine özelleşmiş, ve belirsiz dinamiklere, farklı itme uzunluklarına, nesne tarafı değiştirilmesine ve çeşitli görev gereksinimlerine genelleyecek şekilde tasarlanmıştır. Kontrol için, öğrenilmiş dinamik modelinden yararlanarak uyarlanabilir ve görev odaklı eylemler üreten örnekleme-tabanlı bir model öngörülü denetleyici (Model Predictive Path Integral yönteminin özelleştirilmiş bir versiyonu), kullanılmaktadır. Önerilen yöntem, itme sırasında nesne tarafı değiştirmeyi mümkün kılmakta, değişken itme uzunluklarını desteklemekte ve hassas konumlama, yörünge takibi ve engelden kaçınma gibi çeşitli görevleri yerine getirme kapasitesine sahiptir. Model, simülasyondan gerçek ortama aktarımı kolaylaştırmak amacıyla alan rassallaştırılması kullanılarak elde edilen simülasyondan verilerinden eğitilmiştir. Öncelikle, mimari bileşenleri değerlendirmek için kapsamlı çalışmalar yürütülüp, tasarım kararlarının tahmin doğruluğunu artırdığını ve daha kararlı çıktılar sağladığını gösterilmektedir. Ardından, tam sistem hem simülasyonda hem de gerçek dünya deneylerinde, işaretsiz görsel takip donanımlı bir Franka Panda robotu kullanarak test edilmiştir. Sonuçlar, sıkı

pozisyon ve açı eşiklerinde hassas konumlama için yüksek başarı oranlarına ek olarak yörünge takibi ve engelden kaçınma görevlerinde güçlü performans ortaya koymaktadır. Ek olarak, önerilen yöntemin çok yönlülüğü, denetleyicinin amaç fonksiyonu değiştirilerek birden fazla görevin çözülmesiyle, herhangi bir model yeniden eğitimi gerektirmeden gösterilmektedir. Yaklaşımımızın bir dezavantajı tek bir nesne türüne odaklanmasıdır. Ancak, bu temelden yola çıkarak daha ileri bir yöntem daha geliştirilmiştir. Bu yöntem için daha geniş itme uzunluklarını tahminleyebilen bir model eğitilmiş ve uzak hedeflere ulaşmak için gereken adım sayısını azaltan ve en etkili itme davranışlarnını seçen dengeli bir denetleyici daha tasarlanmış ve hassas konumlama görevinde başarıyla test edilmiştir.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor and mentor, *Assist. Prof. Dr. Barış Akgün*, for his invaluable guidance throughout this journey. I am also profoundly thankful to my family—especially my mother, sister, and uncle—for their unwavering support, as well as to my friends, particularly Alireza and Amirhassan, for always being by my side.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Despite the remarkable progress achieved in the field of Artificial Intelligence (AI) over the past few decades, not all subdomains have experienced the same pace of advancement. One notable area that continues to face substantial challenges is robot learning. While considerable efforts in robotics have led to the development of systems capable of performing a wide range of tasks—such as assisting with domestic chores [Herzog et al., 2023], enhancing agricultural productivity through automation [Singh et al., 2024], and creating quadruped robots that demonstrate behaviors akin to those of animals and humans [Bledt et al., 2018]—numerous fundamental tasks still remain difficult to master in a generalized and robust manner.

Traditionally, many robotic tasks have been approached using modular control pipelines, where complex tasks are decomposed into subtasks, each governed by a carefully designed controller relying on analytical models and mathematical formulations. Although such methods can yield high precision and efficiency for specific scenarios, they often lack the flexibility to generalize beyond the narrowly defined conditions for which they were designed. This limitation becomes especially evident when robots encounter novel environments or slightly perturbed variations of the task. This gap between precision in controlled settings and adaptability in unstructured scenarios has motivated research into alternative approaches that can handle greater variability. In particular, recent advances in machine learning have opened the door to methods that move away from rigid, handcrafted control logic and instead enable robots to acquire flexible, generalizable behaviors through data-driven learning.

With the advent of deep learning and data-driven approaches, a paradigm shift has occurred in robot learning. Robots are increasingly being equipped to learn complex

behaviors directly from raw sensory inputs such as vision and proprioception, enabling the acquisition of skills through interaction and experience rather than explicit programming [Goodfellow et al., 2016]. By training models or policies that can map sensory observations to control actions, these methods have shown potential in bridging the gap between perception and control.

Nevertheless, despite these advancements, certain low-level manipulation tasks continue to pose significant research challenges. One such example is planar pushing, which involves the controlled manipulation of objects through sliding contacts across a surface [Stüber et al., 2020]. Unlike tasks that can be executed with grasping or rigid control, pushing requires an understanding of contact dynamics, frictional interactions, and the nonlinear effects introduced by surface geometry and mass distribution. As a result, planar pushing remains an active and vibrant area of study within the broader field of robot learning, demanding innovative methods that combine model-based reasoning with the adaptability of learning-based strategies.

1.1 Problem Definition

Robotic manipulation can be broadly categorized into prehensile and non-prehensile techniques, each suited to different tasks and environmental conditions. Prehensile manipulation involves securely grasping objects using end-effectors such as parallel-jaw grippers, suction cups, or anthropomorphic hands. Once grasped, objects can be precisely positioned and oriented, enabling tasks that require high accuracy—such as placing components, inserting objects into fixtures, or tool-based assembly. This approach is widely used in structured settings where objects are presented in predictable configurations. However, grasping may be impractical in scenarios where the robot is already holding another item, or when object geometry (e.g., flat panels or deformable bags) prevents secure gripping. Additionally, cluttered or constrained environments may restrict access for effective grasping.

In such cases, non-prehensile manipulation offers a flexible alternative by allowing the robot to influence object motion without grasping. Robots may push, slide, roll, pivot, or even throw objects to achieve the desired outcome. These strategies are particularly

useful in dynamic or space-limited settings, such as warehouses or service environments. For example, a robot might push boxes onto a conveyor, slide plates across a table, or pivot a door open using a flat surface.

A widely studied form of non-prehensile manipulation is planar pushing, where a robot applies forces to an object on a flat surface to induce translation or rotation. This method supports a variety of tasks without the need for grasping and is effective in environments where direct control is infeasible. The resulting object motion depends on factors such as contact geometry, applied force direction and magnitude, object mass and inertia, and surface friction. As an underactuated form of manipulation, planar pushing involves indirect control, making it sensitive to uncertainties in friction, contact compliance, and limited sensor resolution. Various approaches have been explored to address this problem through planning, control, and reinforcement learning. [Stüber et al., 2020].

Traditional approaches to non-prehensile manipulation, such as planar pushing, rely heavily on analytical models of contact dynamics to generate appropriate control strategies. These methods typically incorporate physics-based formulations, including quasi-static or dynamic models, and employ dedicated controllers and/or planners to compute the sequence of actions required to achieve a manipulation goal [Mason, 1986, Goyal et al., 1989, Jia and Erdmann, 1999, Lee et al., 2015]. To function effectively, such models require precise knowledge of several object-specific and environmental parameters. These include the object's mass, center of mass (COM), geometry or shape, and the distribution and characteristics of surface friction. Additionally, external factors such as unexpected disturbances, surface irregularities, or sensor noise must often be considered to ensure accurate prediction of object motion. One of the primary advantages of traditional model-based methods lies in their interpretability and transparency. Because the behavior of the system is governed by deterministic equations rooted in mechanics, these methods allow for intuitive reasoning and in-depth analysis of system performance. The predictive nature of such approaches also makes them well-suited for scenarios that demand high precision and task-specific optimality, especially when the assumptions made by the models are closely aligned with the real-world conditions. Furthermore, from an engineering perspective, these methods often benefit from well-established theoretical guarantees regarding stability, feasibility, and convergence. However, despite their strengths, traditional

modeling and control approaches come with several significant limitations.

First, the process of constructing accurate models and designing corresponding controllers is often labor-intensive and requires a high level of domain expertise. Developing such systems for each new task or object can be time-consuming and prone to error. Second, these approaches typically exhibit limited generalization capability. That is, when there are deviations from the modeled parameters—such as changes in the object's mass, surface friction, or shape—performance may degrade significantly, and the controller may fail altogether. This lack of robustness often necessitates manual re-tuning of parameters or even re-engineering of the system to accommodate new conditions. Another critical limitation arises from the fact that many of the parameters required by the model are not directly observable in practice. For example, precise measurements of the center of mass or friction coefficients may not be available, especially in unstructured or dynamic environments. In such cases, system identification techniques must be employed to estimate these parameters, which introduces additional complexity and uncertainty into the control pipeline.

As a response to the limitations of traditional model-based methods, data-driven approaches have emerged to address some of these challenges [Bauza and Rodriguez, 2017, Li et al., 2018, Bauza et al., 2019, Kloss et al., 2020, Cong et al., 2020, Gao et al., 2020]. These methods are typically grounded in the paradigm of learning from experience, where a robot collects interaction data and uses it to learn models that capture the underlying dynamics of manipulation. Rather than relying on explicitly specified physical parameters such as mass or friction, these approaches embrace the inherent uncertainty and variability in real-world interactions by leveraging statistical learning techniques. A prominent subset of data-driven methods is learned model-based approaches, which aim to approximate the forward dynamics of pushing through supervised learning. In this framework, the robot first collects a dataset of interaction trajectories, often consisting of proprioceptive or visual observations and corresponding object motion outcomes. A predictive model—typically a neural network or a probabilistic regressor is then trained to map observations and actions to future object states. Once trained, this model is used in conjunction with a controller or planner to compute action sequences that achieve a desired goal [Bauza and Rodriguez, 2017, Li et al., 2018, Bauza et al.,

2019, Kloss et al., 2020, Cong et al., 2020, Gao et al., 2020]. One major advantage of this model-controller/planner decoupling is modularity: a well-learned dynamics model can be reused across multiple tasks, allowing practitioners to plug in different controllers or planners depending on the specific objective. Moreover, these approaches are capable of learning end-to-end mappings from raw sensory inputs—such as vision or proprioception—to predicted outcomes, effectively bypassing the need for handcrafted features or explicit modeling of object properties. However, learned models are not without their limitations. A central challenge is distribution shift; the discrepancy between the training data distribution and the scenarios encountered at test time. When the model is queried outside its training regime, it may produce highly inaccurate predictions, leading to compounding errors during execution. Additionally, these models often lack interpretability; unlike analytical models, their internal representations are typically opaque and difficult to diagnose or analyze. They also require careful design choices in terms of network architecture, loss functions, data preprocessing, and training procedures. Poorly chosen configurations or insufficient data diversity can severely degrade model performance. while data-driven methods offer increased flexibility and the potential for better generalization in complex environments, they present a trade-off: the benefits of end-to-end learning and reduced reliance on explicit modeling come at the cost of reduced transparency, increased sensitivity to data quality and coverage, and often greater engineering complexity in the learning pipeline. In practice, small changes in the environment or object properties can lead to significant performance drops if not properly accounted for in training. Consequently, the success of these methods hinges on robust data collection, careful model tuning, and thoughtful system design.

Another subset of data-driven approaches is model-free methods. These approaches aim to directly learn a pushing policy that maps sensory inputs to actions, without explicitly modeling the underlying object dynamics. Given a goal state, the agent learns through trial and error to generate actions that guide the object toward the desired configuration. Once a policy is trained, action inference is extremely fast, requiring only a forward pass through a neural network, making model-free methods attractive for real-time applications. A major advantage of model-free approaches lies in their ability to bypass the need for hand-crafted dynamics models or parameter identification like data-

driven approaches. Moreover, these policies can be trained end-to-end from raw observations, potentially learning features that are tailored to the manipulation task. Despite their strengths, model-free methods face several critical challenges. First is the issue of data efficiency. Unlike model-based approaches, which can plan using a learned or known dynamics model, model-free methods must acquire knowledge of the task exclusively through environment interaction. This typically requires millions to billions of transitions, which is particularly costly in real-world settings where data collection is expensive, time-consuming, or safety-critical. Like data-driven approaches, another significant limitation is the generalization gap. Because model-free policies are trained on a specific set of conditions, changes in task requirements—such as altering object geometry, target location, or frictional characteristics—often lead to degraded performance, unless the policy has been explicitly trained on such variations. This lack of adaptability limits their deployment in real-world applications that demand flexibility. Furthermore, model-free methods often also lack interpretability, which are desirable in high-precision manipulation tasks. Their performance can be sensitive to reward shaping, exploration strategies, and policy initialization. Careful tuning of these components is essential to achieving satisfactory results, yet adds complexity to the training pipeline. Lastly, model-free methods are only suitable for a single task and each new task either requires costly training or at least transfer learning, as opposed to model-based methods.

We take a model-based approach based on our main motivation of training economy; train a push model once and adapt the controller/planner for multiple tasks. Towards this end, we propose a novel GRU-based architecture for learning push dynamics and use a sampling-based Model Predictive Control (MPC) approach, specifically a modified version of Model Predictive Path Integral (MPPI [Williams et al., 2016]), for planar object manipulation on a table. We design our architecture to implicitly capture, potentially non-stationary, object and environmental dynamics by utilizing a recent set of previous state-action pairs, with an emphasis on the current state-action pair.

Our unique state-action representation includes the previous change of object pose, current object relative robot position (contact position) and the object relative push vector (both magnitude and direction). Our state-action representation and our choice of sampling-based MPC allows for changing the object sides during pushing. In addition,

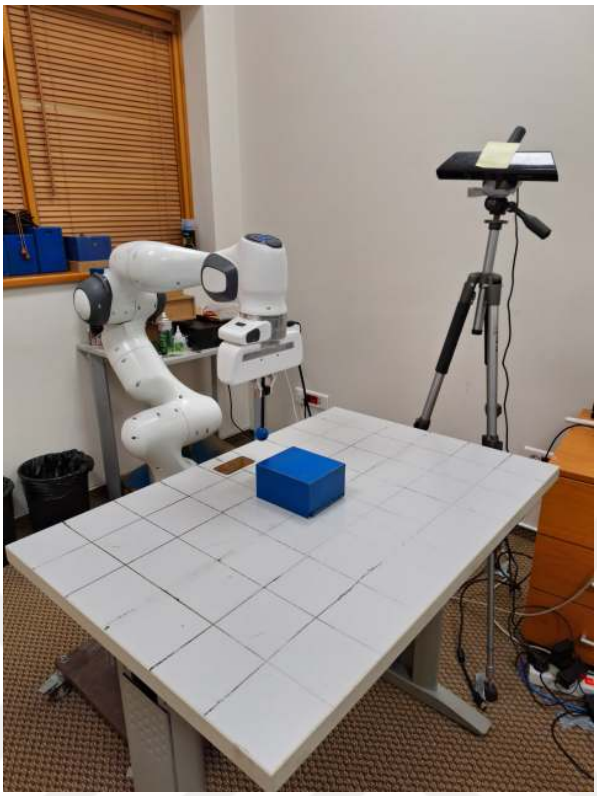


Figure 1.1: The real-robot setup. The robot holds a stick with a small ball at the end to interact with the objects. A RGBD camera is used for perception.

our sampling approach is goal-aware, which lets us generate different length pushes based on the current goal and adapt the samples for the task at hand. To the best of our knowledge, this is the first model-based approach that employs multi-length pushes. Our approach handles local optima for positioning, allows for additional objectives such as obstacle avoidance or different tasks such as trajectory following, and can trade-off task duration with high-precision. We train our model in simulation and collect different length and direction pushes with domain-randomization [Tobin et al., 2017, Peng et al., 2018] to facilitate sim-to-real transfer. We evaluate our approach both in simulation and on a real-world setup as shown in Fig. 1.1.

1.2 Background

In this section, we introduce the core terminology and formalism used throughout the thesis to describe planar pushing and to present a key architectural insight: the decoupling of

the learned model from the controller or planner. This modular structure enables flexible integration of learning and control components and facilitates generalization across tasks. Note that this formulation is general. We describe our specific formulation in

At a high level, our system consists of two primary components:

- A **learned dynamics model** that predicts the effect of a push action on an object, given the current sensory state.

- An **optimization-based controller** that utilizes the model to generate actions that move the object toward a specified goal.

Let the *sensory observation* at time step t be denoted by $\mathbf{I}_t$, which may include visual information (e.g., object pose from RGBD sensors) and proprioceptive data (e.g., end-effector position). The *goal state* is denoted by $\mathbf{g}_t$, typically representing the desired object pose.

The *control action* at time t is represented by $\mathbf{u}_t \in \mathbb{R}^d$ and is computed by a control function $\mathscr{S}$, which takes both the current observation and the goal as input:

$$\mathbf{u}_t = \mathscr{S}(\mathbf{I}_t, \mathbf{g}_t). \tag{1.1}$$

Central to this control function is the learned *push dynamics model*, represented by:

$$\hat{\mathbf{x}}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t; \theta), \tag{1.2}$$

where $\mathbf{x}_t \in \mathbb{R}^n$ is the current object state, $\mathbf{u}_t$ is the candidate action, and θ are the parameters of the model. The model may also take as input relevant object or end-effector attributes.

The *optimization loop* samples a set of candidate actions $\{\mathbf{u}_t^{(i)}\}_{i=1}^N$, evaluates their predicted outcomes using the learned model, and computes the associated costs $C(\hat{\mathbf{x}}_{t+1}, \mathbf{g}_t)$. The optimal action is then selected as:

$$\mathbf{u}_t^* = \arg\min_{\mathbf{u}_t \in \mathscr{U}} C\big(f(\mathbf{x}_t, \mathbf{u}_t), \mathbf{g}_t\big), \tag{1.3}$$

where $\mathscr{U} \subset \mathbb{R}^d$ denotes the admissible action space.

Alternatively, in sampling-based methods such as MPPI, the optimal action is computed as a *weighted average* of sampled actions:

$$\mathbf{u}_t^* = \sum_{i=1}^{N} w^{(i)} \mathbf{u}_t^{(i)}, \quad \text{where} \quad w^{(i)} = \frac{\exp\left(-\frac{1}{\lambda} C^{(i)}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{1}{\lambda} C^{(j)}\right)}, \tag{1.4}$$

with λ as a temperature parameter and $C^{(i)}$ the cost-to-go associated with the i-th trajectory. This probabilistic weighting allows the controller to smoothly interpolate between high-performing trajectories while retaining robustness to sampling noise.

1.3 Contributions

The key contributions of this work are as follows:

- **A novel sampling-based training method for pushing cubic objects.** We introduce a new data generation strategy that captures a wide range of contact interactions, including side-switching behavior and variable push lengths, which are critical for generalizing across diverse object configurations. The sampling process is coupled with a carefully designed state-action representation and a control loop that adaptively selects push actions based on the object's current orientation. Additionally, we define task-specific training metrics to evaluate prediction accuracy and assess long-horizon performance, enabling more effective model learning and controller evaluation.

- **A general-purpose planar pushing framework capable of solving multiple manipulation tasks.** Building on the learned dynamics model, we develop a planning and control architecture that supports a wide range of non-prehensile manipulation tasks. These include:

  - *High-precision positioning*, where the object must be pushed to a precise target pose;

  - *Trajectory following*, where the object is required to follow a continuous spatial path; and
  - *Rudimentary obstacle avoidance*, where planned pushes avoid static obstacles using model-based predictions.

  The decoupling of model and controller enables flexible task definitions by modifying the cost function or optimization routine.

- **Extensive experimental validation demonstrating successful sim-to-real transfer.** We conduct detailed evaluations of our method in both a high-fidelity simulation environment and on a real robotic platform. Experiments with the Franka Emika Panda robot show that models trained entirely in simulation can be effectively transferred to real-world pushing tasks, validating the robustness and generalization of our approach under real-world dynamics and sensor noise.

1.4 Organization

Chapter: Related Work In the next chapter, we discuss the related work on planar pushing. We examine traditional analytical methods in comparison to data-driven and model-free approaches. We also highlight the advantages of our method and its differences from the existing literature.

Chapter: Methodology This chapter presents our proposed approach. We describe the proposed model as well as the controller used in our data-driven, model-based framework, discussing them separately. In addition, we introduce the tasks designed to evaluate our approach.

Chapter: Experiments and Results In this chapter, we present our experimental results for both simulations and real-world experiments. We provide a detailed description of the experimental setup and compare our findings with related work. Specifically, we evaluate our method against a model-free baseline and demonstrate that our improved approach achieves significant gains over the primitive version, both in terms of the number of steps required and the overall motion-planning effort of the robot.

Chapter: Conclusion Finally, we conclude the thesis and summarize the key findings of our work.

Chapter 2

RELATED WORK

Pushing based manipulation can be divided into (i) analytical methods, (ii) model-based data-driven methods, and (iii) model-free data-drive methods. In the first two, models for push dynamics and controllers/planners are used to generate actions. The model is engineered in the first and learned in the second. In the third, a push policy is directly learned.

2.1 Analytical methods

[Mason, 1986] develop an analytical model for quasi-static planar pushing. [Goyal et al., 1989] expand this with the limit surface idea, which defines the relationship between sliding motion of the object and friction under the quasi-static assumption. This is used to determine the object velocity for a given pushing force. [Lynch, 1992] investigate planar pushing using sliding and sticking motions, and employ a search algorithm for planning. [Lee et al., 2015] introduce a hierarchical planning method for generating sequences of non-prehensile and prehensile actions, where contact points for pushing are derived as part of the planning process. This approach is designed to reduce the overall search space by decomposing the full manipulation task into smaller, more manageable sub-problems, allowing more efficient planning in complex environments. As outlined in Chapter 1, while analytical approaches offer interpretability and task-specific optimality under idealized conditions, their practical application is constrained by the need for accurate object-specific parameters such as mass, center of mass, and surface friction. These values are often difficult to measure in real-world scenarios, and even small deviations can lead to substantial performance degradation. Moreover, designing models and controllers for each new object or task is labor-intensive and typically lacks generalization to novel configurations or environmental changes.

2.2 Data-driven model based methods

Data-driven model-based methods aim to reduce engineering effort and better handle uncertainty. [Arruda et al., 2017] use Gaussian Process Regression and an Ensemble of Mixture Density Networks to evaluate uncertainty of object motion predictions. While their method demonstrates the ability to switch object sides during planning, it lacks evaluation in sim-to-real scenarios, limiting its generalizability to novel objects. Additionally, their approach does not support multi-length pushes and does not include planning-based task evaluations. Push-Net. [Li et al., 2018]leverages an LSTM module to encode temporal pushing interactions and incorporates an auxiliary objective to estimate the object's center of mass, implicitly embedding physical reasoning. The approach uses object-segmented masks as visual input and demonstrates the ability to generalize to novel objects. However, their method uses a relatively coarse goal threshold—5 cm for position and 10° for orientation—and employs a fixed push length of 2.5 cm, resulting in a success rate lower than ours.

[Yu et al., 2016, Bauza and Rodriguez, 2017, Bauza et al., 2019] demonstrate the effectiveness of learning models based on collected data that include shape, friction, mass distribution, and perceptual information. [Yu et al., 2016] present a large-scale dataset of planar pushing interactions, comprising over one million samples that include both positional data of the pusher and slider, as well as corresponding interaction forces. [Bauza and Rodriguez, 2017] used a Gaussian process-based model to predict both the outcome and variability of planar pushing, incorporating pusher velocity to relax the quasi-static assumption. While effective, it lacks generalization. More recently, [Bauza et al., 2019] presents a large dataset containing object pose information and visual data (RGB-D), and also introduces models trained on this dataset. However, their approach struggles with precise control of cubic objects, as it selects push points randomly within a 9–10 cm range and uses a fixed push length of 5 cm—limitations reflected in their reported results. [Kloss et al., 2020] introduce a hybrid model that integrates deep learning and analytical physics to predict planar object motion using RGB-D data. Their method estimates object pose and friction parameters through an analytical filter and learns an affordance model via CNNs. It uses object masks to classify motion outcomes, blending perception with

physical modeling. While it requires no prior object knowledge, its property estimations are limited and error-prone, and it relies on assumptions of uniform properties and known friction, which limits generalization.

There are model-based methods that do not rely on object-environment parameters. [Cong et al., 2020] propose a model-based strategy using a recurrent model and sampling-based MPC for control, which is similar to our approach. However, their approach ignores object orientation control, suffers from high sampling costs, does not consider multi-length pushes, and they do not include additional tasks. other drawbacks of their method include the fact that the direction and pose of the first actions did not affect the action selection for the next step. Additionally, it exhibited a low success rate within the threshold of 2.5 cm. [Gao et al., 2020] introduce a few-shot learning approach for predicting the motion of objects with varying shapes. While their method demonstrates promising results for translation prediction, it overlooks critical factors such as object orientation and the historical trajectory of object motion—both of which are essential for achieving smooth and reliable rearrangement. In contrast, our model addresses these limitations by incorporating both historical actions and environmental dynamics, enabling accurate predictions of object motion in terms of both position and orientation. Furthermore, our predictions demonstrate higher accuracy compared to the statistical performance reported in their work.

More recently, [Wang et al., 2024] proposed an online framework for trajectory-following tasks by leveraging a non-parametric model, showing promising results for both model learning and control. However, their approach is limited to trajectory tracking and employs a control strategy based on differential MPC and inverse-dynamics-based sampling, which restricts the ability to switch object sides—a crucial capability for more complex tasks like obstacle avoidance. Moreover, their method may not generalize well to such scenarios. In contrast, our data-driven approach not only avoids the need for retraining across different configurations but is also equipped with a system capable of effectively handling obstacle avoidance.

2.3 Model-free methods

Model-free reinforcement learning (RL) based methods overcome the online scalability challenges of MPC, at the expense of extensive training. [Cong et al., 2022] employ an RL method incorporating visuo-proprioception for objects of various shapes in planar pushing tasks. However, their performance is lacking and they do not incorporate the orientation of target as their objective. [Wang et al., 2023] propose a model-free RL method that uses contact force in their rewards. They only use a positioning task even though they utilize object and end-effector orientations. [Ferrandis et al., 2023] introduce a RL-based method for pushing cubic objects, using categorical exploration.

While their method shows promising results, there are several drawbacks. The required amount of transitions for proper training are in the order of billions. Since they use global coordinates for observations, their operational table size and target location are limited. A later version [Dengler et al., 2024] handles object collisions, however, as with any model-free method, the policies must be retrained for each task.

2.4 Discussion

Table 2.1 summarizes the main features and contributions of the most recent related papers and this thesis. It is difficult to fully perform a full apples-apples comparison since the underlying assumptions, setup and aims of the methods are different. However, the table still gives opportunity for qualitative comparison. As can be seen, most methods trade-off capabilities. In our case, we trade-off multiple types of objects but try to handle everything else (mainly tight positioning performance, handling uncertainty and ability to adapt to new tasks easily).

Table 2.1: Related Work

Paper	**Type**	**Switch side**	**Object variety (non-cubic)**	**Considering orientation**	**Different magnitudes**	**Success metric**	**Success rate**	**Perception**	**Domain Randomization**
[Cong et al., 2020]	MPC	No	No	No	No	5 cm	>90%	No	Yes
[Ferrandis et al., 2023]	RL	Yes	No	Yes	Yes	7 mm	>90%	No	Yes
[Wang et al., 2024]	MPC	Yes	Yes	Yes	Yes	Finish task	100%	No	Yes
[Kloss et al., 2020]	MPC	Yes	Yes	Yes	1 cm, 2 cm	5 mm–1 cm	≈100%	Yes	No
[Wang et al., 2023]	RL	Yes	No	No	Yes	2 cm	>90%	No	Yes (env.)
[Gao et al., 2020]	MPC	Yes	Yes	Yes	No	3 cm	Not stated	No	Yes
Ours	MPC	Yes	No	Yes	3mm-5cm	5mm	100%	No	Yes

Chapter 3

METHODOLOGY

3.1 Push Dynamics Model

3.1.1 Architecture

We introduce and evaluate our push dynamics model in this section. Our model takes object-robot configuration and robot action related inputs, without any explicit object-environment physical parameters, and predicts the change of object pose. We use a history of state-action pairs and Gated Recurrent Unit (GRU) layers to infer implicit dynamics while emphasizing the most recent pair for accuracy. The inputs to the model consist of the history of the previous state-action pairs. We take the object orientation into account in addition to the position. However, the pusher is only represented with a position since we assume a single contact point. Let t be the current time step, O_t denote the object pose in the plane (2D translation and orientation about the z-axis) and R_t be the 2D pusher position at t. Then, a single pair includes object motion of the previous time step ($\Delta O_t = O_{t-1}^{-1} O_t$), the relative position of the pusher with respect to the object ($RO_t = O_t^{-1} R_t$), and the current relative pusher motion ($\Delta RO_t = R_{t+1} - R_t$), indicating the direction and magnitude of the push. These are depicted geometrically in Fig. 3.2. Then current input is the history from time step $t - w + 1$ to t (where w denotes the window length), represented as:

$$H_t = \{(\Delta O_k, RO_k, \Delta RO_k)\}_{k=t-w+1}^{t} \tag{3.1}$$

Our model, denoted as f, takes the history as input and outputs the predicted object motion for the next time-step: $\Delta \hat{O}_{t+1} = f(H_t)$. The architecture of our model can be seen in Fig. 3.1. From the perspective of the model, the action part is the location, direction and magnitude of the push ($RO_t, \Delta RO_t$) and the previous pushes are part of the history. As

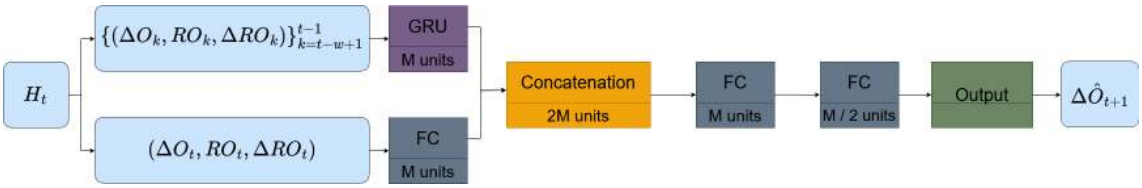


Figure 3.1: Push dynamics model architecture. The (RO_t, ΔRO_t) parts of the input constitute the action.

such, we emphasize this in our model along with the latest object motion which holds the latest object-environment interaction information.

These are separately given to a fully-connected (FC) layer. In parallel, the historical part from $t-w+1$ to $t-1$ is input to a GRU layer. The output of this layer encodes the object and environment properties along with pusher-object interactions. The FC and GRU output dimensions are the same to avoid one of them dominating the other.dominating the predictions. These outputs are concatenated and passed through 2 FC layers before the output layer. All FC layers have ReLU activations.

3.1.2 Data Collection

We collect training data through random pushing interactions in a simulated environment using PyBullet, building upon the framework provided by [Ferrandis et al., 2023]. A total of 10,000 episodes are recorded, each consisting of 100 pushes applied to a planar cubic object with randomized parameters. To ensure diversity in object poses and avoid bias in the training distribution, we incrementally rotate the object by one degree across episodes. The pushing magnitudes range from 2 mm to 3 cm, and push directions are sampled uniformly within a symmetric cone of angle $\pi/4$ oriented towards the object at the point of contact. This setup ensures a broad distribution of contact angles and motion outcomes. The pushes are applied from points sampled on the perimeter of a square surrounding the object, with a margin of 1 cm beyond its actual size, assuming the z-axis remains fixed. Figure 3.3 illustrates a sample pushing configuration from the dataset. To enhance the generalization of our learned model and promote sim-to-real transferability, we employ domain randomization during data collection. Following the principles established in [Tobin et al., 2017, Peng et al., 2018], we randomly vary several environmental and

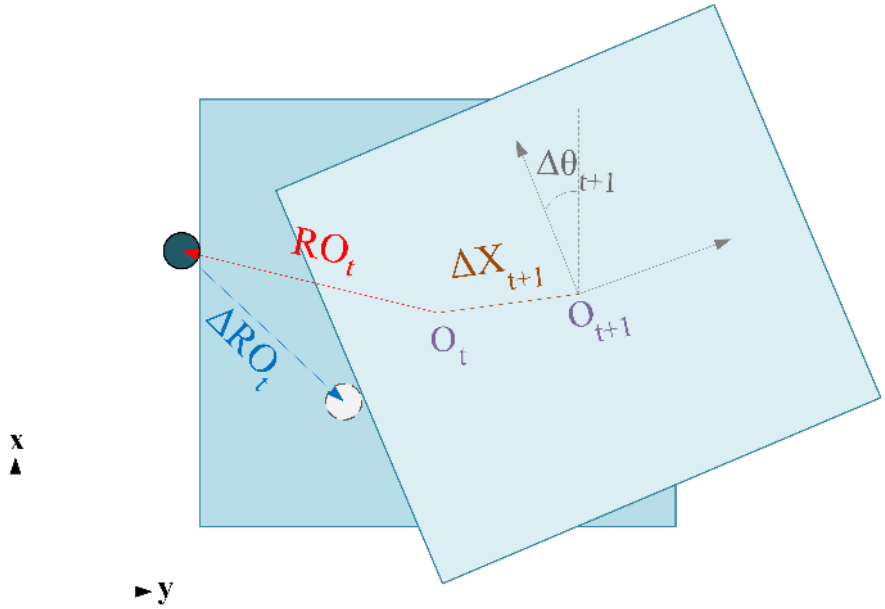


Figure 3.2: State-action representation. O_t represents the 2D object pose before the push and O_{t+1} represents it after the push. RO_t represents the position of the pusher with respect to object before the push and ΔRO_t represents the pusher motion. ΔX_{t+1} and $\Delta\theta_{t+1}$ denote the changes in the object's position and orientation, respectively, after the push, and $\Delta O_{t+1} = (\Delta X_{t+1}, \Delta\theta_{t+1})$.

Table 3.1: Domain randomization parameters

Parameter	Sampling Distribution
Friction	$[0.5, 0.7]$
Restitution	$[0.4, 0.6]$
Object Length	$[0.11, 0.13]$ m
Object Width	$[0.09, 0.11]$ m
Object Mass	$[0.3, 0.7]$ kg

object parameters at the beginning of each episode. These parameters include the object's mass, dimensions, friction coefficient, and the restitution of the supporting surface. The exact ranges and details of these parameters are listed in Table 3.1. This strategy enables the model to learn robust contact dynamics across a wide variety of scenarios, reducing overfitting to a narrow domain. Notably, we observed that the model trained entirely on simulated data generalized effectively to real-world robot experiments without requiring any additional fine-tuning. This highlights the effectiveness of combining fully random pushing actions with domain randomization for robust and scalable data-driven manipulation learning. For the improved version of our framework, which we discuss later in Section 3.2, we sample pushing magnitudes between 3 mm and 5 cm.

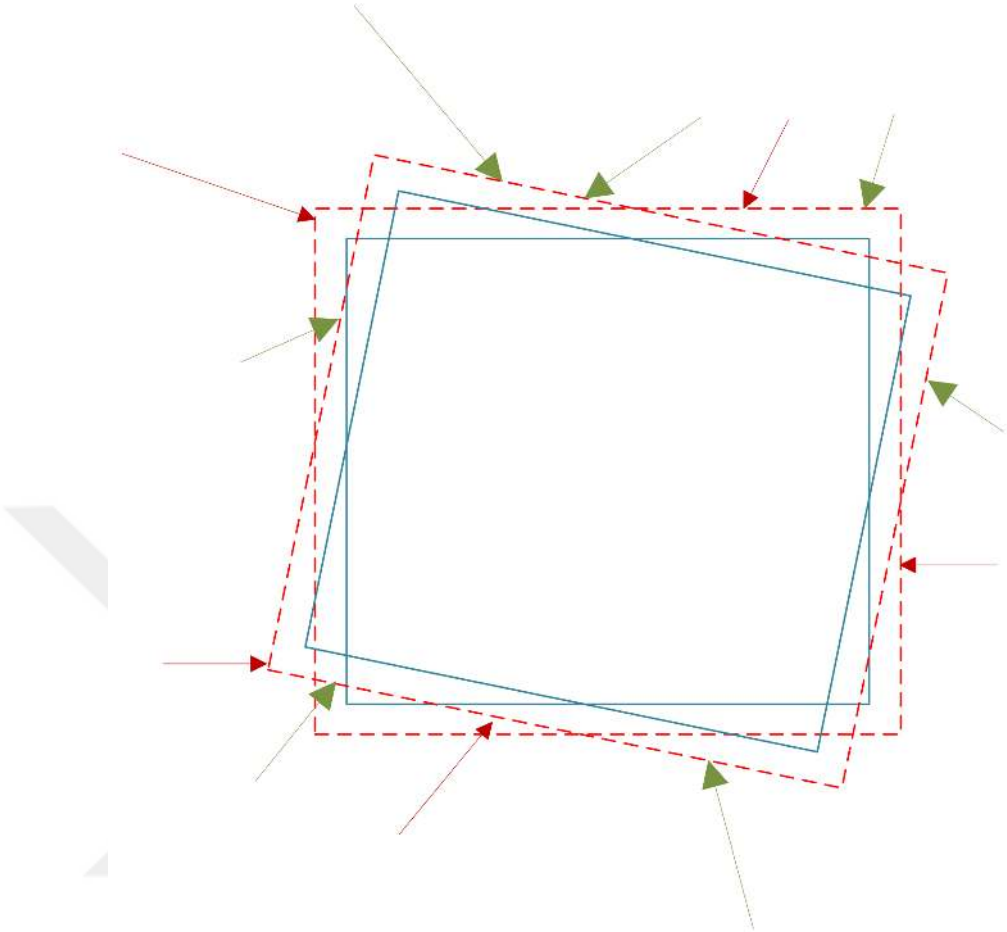


Figure 3.3: Data collection process.This figure illustrates the data collection process over 10,000 episodes. Actions are sampled with directions uniformly distributed between -45° and 45°, and push magnitudes ranging from 2 mm to 3 cm. The red dots indicate the boundary of relative poses where actions are applied, designed to prevent premature contact (i.e., touching) before the action is executed.

3.1.3 Training

We use two models in practice: one for predicting the 2D translational motion and another for predicting the rotational motion about the plane-normal. The loss for the translation prediction model is defined as the mean squared distance between the target and the predicted translations. The loss for the orientation prediction model is defined as the mean non-oriented angular difference between the target and the predicted rotations. The non-oriented angle is calculated as follows:

$$\mathrm{D}_\theta(\theta_1, \theta_2) = \min(|\theta_1 - \theta_2|,\ 2\pi - |\theta_1 - \theta_2|) \tag{3.2}$$

Two models performed better in practice, mainly due to removing the need for tuning the relative weights of the position and the orientation losses. These models use the exact same architecture other than the output size and we use $M = 64$ (see Fig. 3.1) to instantiate both of them. We perform a 0.7-0.2-0.1 train-validation-test set splits. The models were trained using the Adam optimizer [Kingma and Ba, 2014] with a learning rate scheduler. A total of 40 epochs were enough for convergence in all cases and we take the model with the best validation performance.

3.1.4 Model Evaluation

We evaluated our model on one-step predictions from the test set. We pick two baselines to demonstrate the effectiveness utilizing the history while focusing on the current time-step in our architecture. The first one is a 2-layer GRU model inspired from [Cong et al., 2020] (We replace the LSTM layers with GRU layers and even reduce the number of hidden units in our model) and (ii) a model without a history. Notably, our model has approximately 40% fewer learnable parameters than the 2-layer GRU baseline.

Table 3.2 presents the prediction performance of the three baseline models, along with our model used for the improved framework. To better capture model performance under different push lengths, we divided the test set into two subsets: the ”small” set with pushes ranging from 2 mm to 8 mm, and the ”big” set with pushes ranging from 8 mm to 3 cm. For the improved framework, pushes range from 3 mm to 5 cm, and in this case we evaluate the model on the entire dataset without splitting it into subsets.

Our model outperforms both of the baselines. The performance compared to the no history baseline is expected. The no history model cannot infer the object properties with just looking at the current state. On the other hand, 2-Layer GRU baseline performed the worst - particularly bad for position prediction - even though it was taking the history into account. Overfitting is the main culprit. The best model had good training and validation performances. One additional potential reason is that the action cannot influence the

Table 3.2: Object motion prediction performance of our proposed model and baselines. Positions are in millimeters and orientations are in radians. The values in parenthesis denote the standard deviations. The last row shows the results for the improved version model.

Model / Set	**Avg. Position Err.**	**Avg. Orientation Err.**
No History - Small Set	0.6 (0.4)	0.004 (0.004)
2-Layer GRU - Small Set	1 (0.5)	0.006 (0.005)
Ours - Small Set	**0.3 (0.3)**	**0.003 (0.003)**
No History - Big Set	1 (1)	0.01 (0.015)
2-Layer GRU - Big Set	2.4 (1.7)	0.018 (0.02)
Ours - Big Set	**0.8 (0.8)**	**0.007 (0.009)**
Model for Improved Version of Framework	1.5 (1.7)	0.016 (0.023)

prediction enough and it gets dwarfed by the history. Since our dataset is heavily randomized, this particular architecture (limited inductive bias compared to ours) arguably did not have enough data points to generalize. As such, we conclude that our architecture provides a good balance between historical information to infer pushing properties and the current action to perform accurate predictions. Furthermore, each step prediction takes only 2 ms, simultaneously predicting both translation and orientation motion.

3.2 Control

We use a modified MPPI as our controller which is a sampling-based method for efficiently generating actions. The basic idea is to sample a number of actions, predict corresponding roll-outs using a model, calculate their objective value (e.g. cost, reward), calculate the next action based on these, apply the first action, then repeat. Note that the objective values are defined for a given task and we assume they are to be minimized. For example, distance of the last position of a roll-out to the target can be an objective function for positioning. The single time-step of this control method is depicted in Alg. 1.

We use a goal-aware sampling procedure. In our initial approach, the relative pusher position (*RO*) and the push magnitude were sampled simultaneously from the dataset. For a single sample, a random location on the periphery of the object (*RO*) is picked, followed

Algorithm 1 Action Selection in our MPPI implementation

1: **Fixed Input:** D_O, N, T, $c(\cdot)$, $f(\cdot)$, λ
2: **Input:** The state-action history of the previous $w-1$ steps and latest object motion ΔO_t ($\bar{H}_t$), the current object pose O_t
3: **Output:** The next action ($RO_t, \Delta RO_t$) and the predicted object motion ($\Delta\hat{O}_{t+1}$)
4: $A = a_1, a_2, \ldots, a_N \leftarrow$ sampleActions(D_O, O_t, N)
5: $\tau = \tau_1, \tau_2, \ldots, \tau_N \leftarrow$ generateRollouts(f, A, T, $\bar{H}_t$)
6: $C = c_1, c_2, \ldots, c_N \leftarrow$ calculateObjectiveValues(c, τ)
7: $\Omega = \omega_1, \omega_2, \ldots, \omega_N \leftarrow$ calculateActionWeights(C) (Eq. 3.3)
8: $a_t = \leftarrow$ combineActions(A, Ω) (Eq. 3.4)
9: $\Delta\hat{O}_{t+1} \leftarrow f(H_t)$, ($H_t$ includes a_t)
10: **return** $a_t, \Delta\hat{O}_{t+1}$

by a random push direction within a symmetric $\pi/4$ cone towards the object. We then randomly pick a push length but the range is selected based on the goal. The direction and length is used to calculate ΔRO. The object dimensions (D_O) are required for some these steps. This approach allows us to switch object sides and perform different length pushes. We use $a_i = [RO^i, \Delta RO^i]$ to represent the i^{th} sampled action as a 4 dimensional vector.

In a single control step, we sample N actions (line 4) and generate N rollouts by applying each sampled action repeatedly for a fixed horizon length, T. We use the model ($f(\cdot)$) in an autoregressive way to generate an individual rollout (line 5) and denote the i^{th} rollout as τ_i. Then the objective values (c_i) of the roll-outs are calculated (line 5), based on the given objective function $c(\tau)$. The action selection is done by taking the weighted combination of the sampled actions. Individual action weights are calculated with the Eq. 3.3. Then, the actions are combined using the Eq. 3.4 where $a_t = [RO_t, \Delta RO_t]$ is the selected action. These steps are in (line 7-8).

$$\omega_i = \frac{\exp\left(\frac{-c_i}{\lambda}\right)}{\sum_{j=1}^{N} \exp\left(\frac{-c_j}{\lambda}\right)} \tag{3.3}$$

$$a_t = \frac{\sum_{i=1}^{N} (\omega_i \times a_i)}{\sum_{i=1}^{N} \omega_i} \tag{3.4}$$

This approach prioritizes actions that are more effective in reducing the objective value (e.g. distance) while still allowing other actions to affect the outcome to avoid being overly greedy. The trade-off is set by the parameter λ. However, the resulting relative pusher position may be in collision with the object. In that case, we restart the sampling process. If we cannot find a feasible solution in 20 restarts, we select the action with the lowest cost.

The next step is to apply the action (not shown in the algorithm). The robot end-effector pose is calculated based on the current object pose O_t and the selected relative pusher position RO_t. We use a motion-planner to reach this pose. Then, the robot is moved based on the selected ΔRO_t, using a Cartesian controller. After the robot motion, the object pose is recalculated using an external camera. This process repeats until an objective (e.g., object within a threshold) or a maximum number of iterations are reached.

In the rest of this thesis, as a primitive method, we adopt a two-stage control approach. In the first stage, the primary aim is to rapidly bring the object close to the target which is achieved by sampling longer pushes. We switch to the second stage once the object is within 1 cm of the object. In the second stage, the strategy changes to sampling shorter pushes. The idea is to perform precise adjustments and accurately pose the object. In the first stage, the pushes are sampled in the range 8mm-3cm, and in the second stage, from 1 mm to 8 mm.

To improve the performance and robustness of this method, we introduce several modifications aimed at enhancing both the diversity of sampled actions and the precision of the action selection process. First, we increase the number of sampled actions per control iteration, enabling a denser exploration of the action space and significantly increasing the likelihood of identifying high-quality actions. Instead of sampling RO_t and ΔRO_t jointly, we now sample each component separately. A balanced number of RO_t values are sampled from all four sides of the cube, which encourages more diverse and symmetric interaction strategies. We also expand the sampling range of push magnitudes to span from 3 mm to 5 cm, thereby accommodating both coarse repositioning and fine-tuning within a unified framework.

Another key enhancement concerns the temporal structure of actions during the rollout. Rather than applying the same action across all steps in a rollout, we introduce

Algorithm 2 Improved Action Selection in our MPPI Implementation

Fixed Input: $D_O, N, T, c(\cdot), f(\cdot)$
Input: State-action history $\bar{H}_t$ of previous $w-1$ steps, current object pose O_t, latest object motion ΔO_t
Output: Next action $(RO_t, \Delta RO_t)$ and predicted object motion $\Delta\hat{O}_{t+1}$

$RO \leftarrow$ sampleBalancedROsides(O_t, N) ▷ Sample RO_t from all sides of object
$\Delta RO \leftarrow$ samplePushMagnitudesAndDirections(N) ▷ Magnitudes in [3 mm, 5 cm], varied directions
$A = a_1, a_2, \ldots, a_N \leftarrow$ combineActions(RO, ΔRO)
$\tau_1, \ldots, \tau_N \leftarrow$ generateDirectionalRollouts(f, A, T, $\bar{H}_t$) ▷ Each rollout perturbs direction within $\pm 5^\circ$ per step
$C = c_1, c_2, \ldots, c_N \leftarrow$ calculateObjectiveValues(c, τ)
$i^* \leftarrow \arg\min_i c_i$
$a_t \leftarrow a_{i^*}$
$\Delta\hat{O}_{t+1} \leftarrow f(H_t)$ ▷ H_t includes a_t
return $a_t, \Delta\hat{O}_{t+1}$

directional variation at each step. Specifically, while the push magnitude remains constant, the direction changes within a range of $\pm 5^\circ$ relative to the previous direction. The first action's direction in the rollout is sampled, and subsequent actions are perturbed based on this initial value. This change allows the controller to explore more realistic and continuous action trajectories.

Finally, instead of selecting the action based on a weighted average over sampled rollouts , we adopt a more decisive strategy by directly choosing the action with the minimum rollout cost. This avoids the dilution effect of averaging, which previously led to RO_t values occasionally falling inside the object boundary. These modifications retain the overall structure of the controller while significantly improving sample efficiency, control accuracy, and robustness across a wide range of pushing scenarios. Algorithm 2 reflects the improved version of our action selection strategy, incorporating enhancements in sampling, rollout generation, and action selection.

3.3 Tasks

3.3.1 Tasks and Their Objective Functions

In our evaluations, we use precise posing of an object as our main task. The aim is to push the object to a target pose (position and orientation). We additionally use two minor tasks; (i) trajectory following and (ii) obstacle avoidance. Trajectory following involves pushing the object along a series of waypoints. Obstacle avoidance, requires the system to reach a target pose while circumventing obstacles that may be on the direct path between the initial and target poses. Each task has its own objective function.

We define the pose difference in Eq. 3.5, where $P_i = [X_i, \theta_i]$ is the object pose, X_i is the associated 2D position vector, θ_i is the associated orientation around the 2D plane normal, D_θ is from Eq. 3.2, and w_θ is the orientation weight.

$$D_{pose}(P_1, P_2) = \|X_1 - X_2\|^2 + w_\theta D_\theta(\theta_1, \theta_2)^2 \tag{3.5}$$

Precise Posing: In this task, we define the objective function of a roll-out as the pose difference between the predicted object pose of the last roll-out time step and the target pose, as defined in Eq. 3.6, where $\hat{P}_{\text{last}}$ is the predicted object pose at the last roll-out time step.

$$c_{posing}(\tau_i) = D_{pose}(P_{\text{target}}, \hat{P}^i_{\text{last}}) \tag{3.6}$$

We have an alternative formulation which penalizes the robot travel between pushes, to prevent frequent side-switching for small gains. This is given in Eq. 3.7, where ϕ is the motion planning penalty weight, R_{current} is the last pusher position and R_{sampled} is the sampled pusher position.

$$c_{penalty}(\tau_i) = c_{posing}(\tau_i) + \phi \left\| R_{\text{current}} - R_{\text{sampled}} \right\|^2 \tag{3.7}$$

Trajectory Following: Trajectory following tasks can have a variety of objective functions. Trajectory following tasks can be formulated using a variety of objective functions, depending on how the trajectory is represented. In our case, we adopt a dense

trajectory representation, meaning that waypoints are placed closely along the desired path of the object. Since we repeat the same push during rollouts, it is enough for us to look only at the first step. We also do not expect frequent side-switching due to dense waypoints. As such, we use the simple objective defined in Eq. 3.8, where P_{p} is the next waypoint and $\hat{P}_{\text{first}}$ is the predicted object pose at the first roll-out time step.

$$c_{tf}(\tau_i) = D_{pose}(P_{\text{p}}, \hat{P}^i_{\text{first}}) \tag{3.8}$$

Obstacle Avoidance: This task is similar to precise positioning with the added requirement of avoiding obstacles. There are multiple objective functions in the literature. We chose a function that exponentially penalizes the distance between the object and the obstacles and the distance between the pusher and the object as defined in Eq. 3.9, where d_o/d_p represents the closest distance between the predicted object surface/pusher position during the roll-out steps and the closest obstacle surface.

$$c_{oa} = w_1 \exp(-\alpha_o d_o) + w_2 \exp(-\alpha_p d_p) \tag{3.9}$$

Here, w_1 and w_2 are weighting factors that determine the relative importance of penalizing proximity between the object and the obstacles (w_1) and between the pusher and the object (w_2). The parameters α_o and α_p control the sensitivity of the exponential penalty to the distances d_o and d_p, respectively. In our experiments, $w_1 = 10$, $w_2 = 10$, and $\alpha_o = \alpha_p = 100\ln(10)$. During this task, the obstacle avoidance objective is added to the positioning objective only if d_o or d_p is less than 1 cm. This formulation penalizes actions that bring the object or the pusher closer than 1 cm to any obstacle, encouraging the selection of safer paths while still making progress towards the target.

This function penalizes actions that bring the object or the pusher closer than 1 cm to any obstacle, encouraging the selection of safer paths that still make progress towards the target.

Chapter 4

EXPERIMENTS AND RESULTS

4.1 Simulation

We use the same PyBullet simulation environment to collect model data to evaluate our approach. The environment consists of a cubic object to be pushed toward a specified goal pose using a spherical pusher of radius 1.25 cm. The horizon length, T, for precise posing and obstacle avoidance is 5, and it is 1 for trajectory following. The number of samples, N, is 20 per control step for precise posing and 100 for the other tasks. The orientation weight, w_θ, is 0.025 for precise posing, 0 for trajectory following, and 0.3 for obstacle avoidance when the object is within 5 cm of the target pose and 0 elsewhere. The motion planning weight ϕ is dynamic and is the quarter of the distance to the target position in the precise posing task until the object is within 2 cm of the target, at which point it becomes zero. For obstacle avoidance, the robot weight remains zero but for trajectory following we both tested zero and 0.01 weights. In the improved version of our controller, we introduce several key modifications. Specifically, for trajectory following, we use a rollout horizon of $T = 1$ and increase the number of sampled actions to $N = 100$ per control step. For precision positioning, we also use a rollout horizon of $T = 4$ and $T = 3$ and sample $N = 100$ actions, distributing them equally across all four sides of the object (25 samples per side).

For benchmarking purposes, we adopt the model-free policy introduced by [Ferrandis et al., 2023] as a baseline for the precise posing task. Specifically, we use the policy trained and released by the original authors without further fine-tuning. This policy was developed using reinforcement learning in simulation and is designed to directly control the pusher to move the object to a desired pose. The observation space for this model-free policy includes both the current pose of the object and the pose of the pusher, providing the agent with sufficient spatial context to infer appropriate actions. The action space

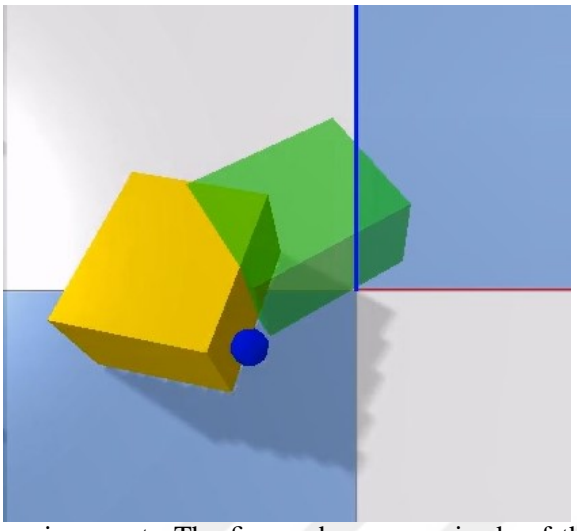


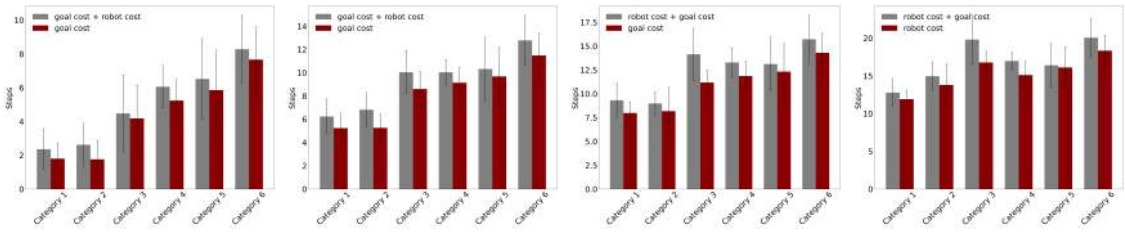
Figure 4.1: Simulation environment. The figure shows an episode of the precision positioning task, where the object must be pushed to its goal pose using the pusher. In the simulation, the yellowish cube represents the current object pose, the greenish marker indicates the target pose, and the blue sphere represents the pusher.

consists of velocity commands for the pusher in the 2D plane, allowing for continuous control over its motion.

In addition to this baseline, we evaluate both of our proposed precise posing objectives—based on Eq. 3.6 and Eq. 3.7—within our model-based control framework. These objectives are designed to explicitly guide the pushing behavior by incorporating goal pose alignment and, optionally, penalizing excessive pusher movement.

Beyond precise posing, we also apply our approach to other manipulation tasks, including obstacle avoidance and trajectory following. This broader evaluation demonstrates the generalizability and robustness of our model-based method across diverse pushing scenarios, where each task imposes different constraints on the controller and relies on distinct cost formulations.

Precise Manipulation: There are six categories of experiments, each comprising 10 cases. The experiments are organized into two primary categories based on the distance between the initial pose and the target pose. The first category involves target distances ranging from 5 cm to 15 cm, and the second category from 15 cm to 20 cm. Within each distance category, the experiments are further subdivided based on the the target and the initial orientation differences into three intervals: 0°–30°, 30°–60°, and 60°–90°. Each



(a) (b) (c) (d)

Figure 4.2: Number of steps taken by the primitive method across different categories. The subfigures correspond to different success thresholds: 5 cm, 2 cm, 1 cm, and 3 mm + 2 degrees from left to right. The vertical axes show the steps to reach these thresholds in simulation. The categories in the horizontal axes correspond to the Cartesian product of $\{(5cm-15cm),(15cm-20cm)\}$ distances and $\{(0°–30°),(30°–60°),(60°–90°)\}$ orientation differences between the initial pose and the target pose (e.g. Category 1 is (5cm-15cm) and (0°–30°), 6 is (15cm-20cm) and (60°–90°)). The crimson and gray bars correspond to the objective functions defined in Eq. 3.6 and Eq. 3.7 respectively.

individual experiment was repeated 5 times, for a total of $2 \times 3 \times 10 \times 5 = 300$ episodes.

The main results are shown in Fig. 4.2. The trends within each objective are expected. The number of required steps increase with threshold strictness, with target distance and with orientation difference. However, the model performance did not vary much between the (0°–30°) and (30°–60°) orientation difference categories. Overall, fewer than 20 steps are required for the object to reach within 1 cm of the target pose.These results also demonstrate the advantage of decoupling the learned model from the controller: greater precision can be achieved simply by allowing more time, without necessitating retraining or altering the policy structure. Although a direct comparison with existing methods is challenging due to variations in evaluation setups and implementation details, our method achieves a favorable trade-off between efficiency and accuracy. It consistently requires fewer steps while maintaining high success rates under stringent error thresholds, attributable to the robustness of the learned dynamics and our approach to generating variable-length push actions.

Including the robot motion penalty increased the required steps as expected, since not switching sides may prolong posing. Fig. 4.1 demonstrates that incorporating the motion planning cost reduces the robot travel which suggests that trading off number of steps with robot travel is feasible. The average time to finish the pushing episodes is 5.96 seconds.

We evaluated the improved controller on the same six categories described above, as

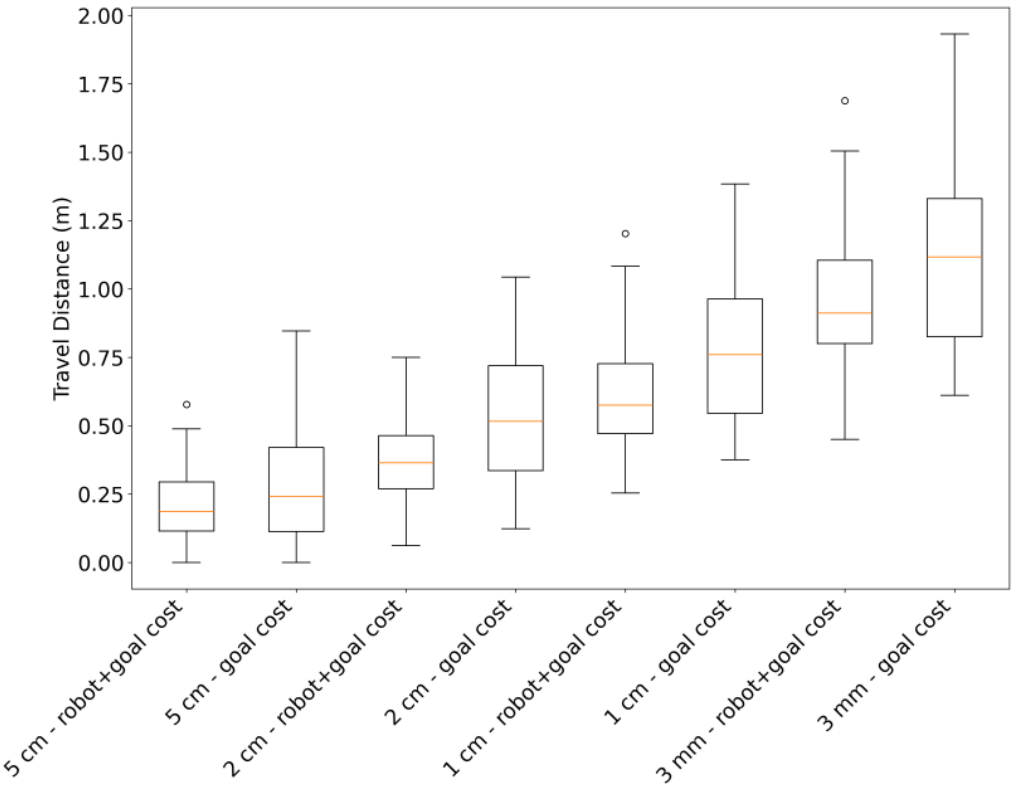


Figure 4.3: The robot travel for our primitive method. The "goal cost" corresponds to Eq. 3.6 and "robot cost + goal cost" corresponds to Eq. 3.7. he figure illustrates that pushing the object toward a distant goal with larger orientation changes results in increased robot travel.

well as on an additional set of experiments involving larger target distances. Specifically, the new set consists of two distance ranges: 10–20 cm and 20–30 cm. Each distance range is further subdivided into three orientation intervals: 0°–30°, 30°–60°, and 60°–90°, resulting in six new categories. For all categories, we tested two sampling strategies for the initial push direction: a narrow range of $[-5°, 5°]$ and a wide range of $[-45°, 45°]$. The results, presented in Figure 4.4 and Figure 4.5, show that the number of steps required to reach each success threshold is reduced by approximately half. Furthermore, Figure 4.6 illustrates the substantial reduction in robot travel distance achieved by the improved controller. We conducted experiments with horizons of 3 and 4: a larger horizon tends to avoid selecting unnecessarily long pushes, while a smaller horizon may increase the number of steps. On average, the episode completion time was 13.65 s for horizon 4 and 5.80 s for horizon 3.

We have also compared our method with a model-free baseline. We show the ag-

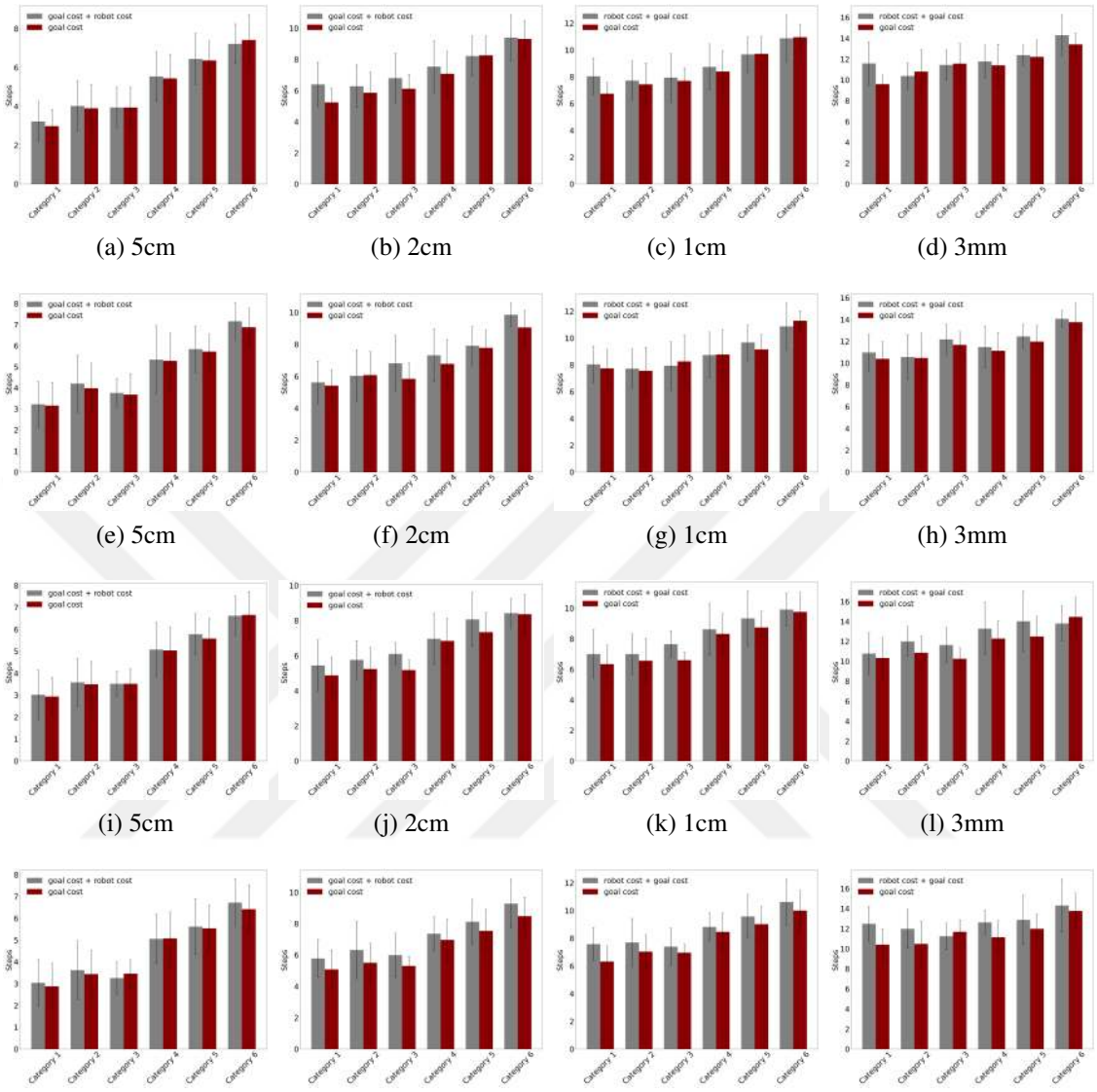
(a) 5cm (b) 2cm (c) 1cm (d) 3mm

(e) 5cm (f) 2cm (g) 1cm (h) 3mm

(i) 5cm (j) 2cm (k) 1cm (l) 3mm

(m) 5cm (n) 2cm (o) 1cm (p) 3mm

Figure 4.4: Number of steps taken by the improved method across different categories in short-distance pushing scenarios. Results are evaluated at four success thresholds—5 cm, 2 cm, 1 cm, and 3 mm + 2°—shown from left to right in each group. Each subfigure shows the number of simulation steps required to meet the specified success criterion. Rows correspond to different configurations: **Row 1** (row 5 overall) uses Horizon 4 with wide orientation sampling $[-45°, 45°]$; **Row 2** uses Horizon 4 with narrow sampling $[-5°, 5°]$; **Row 3** uses Horizon 3 with wide sampling; and **Row 4** uses Horizon 3 with narrow sampling. The horizontal axis within each plot categorizes the tasks based on the Cartesian product of translation distance intervals $\{(5\,\text{cm–}15\,\text{cm}),\ (15\,\text{cm–}20\,\text{cm})\}$ and orientation differences $\{(0°–30°),\ (30°–60°),\ (60°–90°)\}$. For example, Category 1 denotes 5–15 cm distance and 0°–30° rotation, while Category 6 denotes 15–20 cm and 60°–90°.



(a) 5cm (b) 2cm (c) 1cm (d) 3mm

(e) 5cm (f) 2cm (g) 1cm (h) 3mm

(i) 5cm (j) 2cm (k) 1cm (l) 3mm

(m) 5cm (n) 2cm (o) 1cm (p) 3mm

Figure 4.5: Number of steps taken by the improved method across different categories in long-distance pushing scenarios. Results are evaluated at four success thresholds—5 cm, 2 cm, 1 cm, and 3 mm + 2°—shown from left to right in each group. Each subfigure reports the number of simulation steps required to reach the specified threshold. Rows correspond to different combinations of rollout horizon and initial push direction range: **Row 1** uses Horizon 4 with wide orientation sampling $[-45°, 45°]$; **Row 2** uses Horizon 4 with narrow orientation sampling $[-5°, 5°]$; **Row 3** uses Horizon 3 with wide sampling; and **Row 4** uses Horizon 3 with narrow sampling. The horizontal axis in each plot categorizes tasks based on translation distance $\{10\,\text{cm–}20\,\text{cm},\ 20\,\text{cm–}30\,\text{cm}\}$ and rotation difference $\{0°–30°,\ 30°–60°,\ 60°–90°\}$ between initial and target poses.

gregate results in Table 4.1. Figure 4.7 compares the robot travel distance between our primitive "goal cost" method and the model-free baseline. Since the model-free baseline

continuously pushes from the same spot while leveraging its knowledge of the dynamics, it achieves greater efficiency in terms of travel distance. Our method achieves 100% task completion rate for both of the thresholds. However, the model-free method performs poorly in addition to the sharp decline between. While it could be argued that the model-free baseline might perform better with extended training time and further hyperparameter optimization, this introduces a significant computational burden. Model-free methods often demand extensive training data and time due to their trial-and-error nature, which can become impractical in real-world applications where data collection is expensive or time-consuming. In contrast, our model-based approach, by leveraging predictive dynamics and goal-directed planning, requires significantly fewer training iterations and exhibits stable behavior across a wide range of initial conditions and target poses. Furthermore, we observed that our method has the potential to achieve extremely high accuracy—errors as low as 1 mm in translation and 1° in orientation—when the stopping thresholds are relaxed and operation is allowed to continue until convergence. This level of precision, however, comes at the cost of increased number of steps, which complicates systematic evaluation under fixed time or step constraints. Nevertheless, this flexibility emphasizes the strength of model-based approaches in adapting to diverse performance criteria and operational constraints.

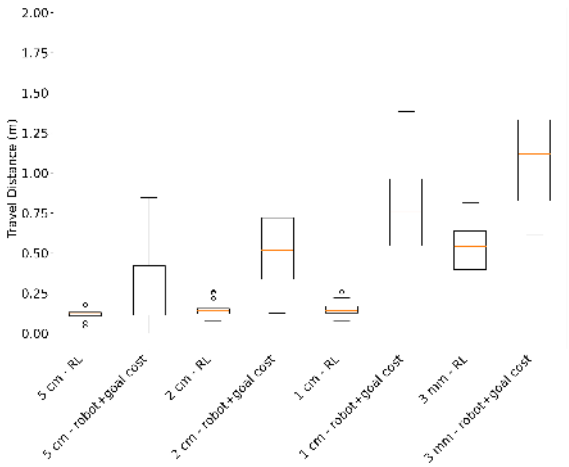


Figure 4.7: The Robot Travel Comparison Between Our Primitive Method Against The Model-free Baseline [Ferrandis et al., 2023].

Table 4.1: Comparison of Our Method and Model-Free Baseline [Ferrandis et al., 2023].

Threshold	**Our Method**	**Model-Free Baseline**
3 mm, 2 degrees	1.000	0.303
7.5 mm, 5 degrees	1.000	0.886

Lastly, we log the object pose predictions during our experiments. The average error between the model prediction and the object pose after taking the selected action was 1 ± 0.84 mm for distance and 0.0075 ± 0.0082 radians for orientation, similar to the "big set" model evaluation results for our primitive method.

Trajectory Following: We conduct experiments using both circular and "L"-shaped trajectories composed of dense waypoints. Our controller uses Eq. 3.8 and performs single-step roll-outs to decide the next action. Waypoints are switched once the object reaches sufficiently close proximity to the current target. We tested circular trajectories with radii of 10 cm, 20 cm, and 30 cm, and "L"-shaped paths with side lengths of 10 cm, 20 cm, and 30 cm. For each shape and size, we evaluated two cases: one with no orientation objective and one with a small orientation cost of 0.01.

Table 4.3 summarizes the trajectory following performance for circular shapes. The average position error in each control step varied between 3 mm and 5.9 mm, depending on the radius and whether orientation control was used. The average push size ranged between 1.05 cm and 1.57 cm. These results confirm that our controller maintains accurate path-following performance across different path scales, while orientation objectives slightly increase the positional error.

Table 4.2 presents the results for L-shaped trajectories. The controller successfully followed the L-shaped paths with high positional accuracy. Average position error remained low, between 2.1 mm and 5.9 mm, with higher errors associated with nonzero orientation cost. The push size was consistent with the controller's sampling settings, ranging from 1 cm to 2 cm.

Importantly, in the scenarios where a non-zero orientation cost was used, the objective was to preserve the object's initial orientation—i.e., the cube was expected to maintain a 0° rotation throughout the trajectory. This was particularly challenging around the corner segments of the L-shape, where lateral pushes could inadvertently induce undesired ori-

entation changes. The results indicate that our controller handles these challenges effectively, though the orientation cost leads to slightly increased positional error as the system balances between translational and rotational objectives. Figure 4.8 illustrates representative simulation cases of trajectory following without orientation cost, highlighting both L-shaped and circular trajectories at different push sizes and position errors.

We also evaluated our improved framework on trajectory-following tasks in simulation. The results are reported in Tables 4.4 and 4.5. The experiments sampled the push direction θ within a range of -45° to 45°. We did not observe significant differences in the L-shaped trajectories, but the performance decreased for circular trajectories. We hypothesize that this decline may be due to the higher magnitude of pushes in our improved method compared to the primitive approach. Additionally, our improved method does not account for pushes from the edges of corners, which could be beneficial for trajectory-following scenarios.

Table 4.2: Trajectory following performance on L-shaped paths with different lengths and orientation costs with our primitive method.

Shape	Length (m)	Avg. Push Size (m)	Avg. Pos. Error (m)	Orientation Cost
L	0.2	0.01	0.0027	0
L	0.2	0.01	0.0040	0.01
L	0.2	0.02	0.0029	0
L	0.2	0.02	0.0059	0.01
L	0.1	0.01	0.0021	0
L	0.1	0.01	0.0039	0.01
L	0.1	0.02	0.0026	0
L	0.1	0.02	0.0059	0.01

Table 4.3: Trajectory following performance on circular shapes of various radii, with and without orientation cost with our primitive method.

Shape	Radius (m)	Avg. Push Size (m)	Avg. Pos. Error (m)	Orientation Cost
Circle	0.1	0.0105	0.0035	0
Circle	0.1	0.0105	0.0046	0.01
Circle	0.1	0.0157	0.0040	0
Circle	0.1	0.0157	0.0059	0.01
Circle	0.2	0.0105	0.0032	0
Circle	0.2	0.0105	0.0049	0.01
Circle	0.2	0.0157	0.0039	0
Circle	0.2	0.0157	0.0056	0.01
Circle	0.3	0.0105	0.0034	0
Circle	0.3	0.0105	0.0049	0.01
Circle	0.3	0.0157	0.0039	0
Circle	0.3	0.0157	0.0067	0.01

Table 4.4: Trajectory following performance on L-shaped paths with different lengths and orientation costs with our improved version.

Shape	Length (m)	Avg. Push Size (m)	Avg. Pos. Error (m)	Orientation Cost
L	0.2	0.04	0.0042	0
L	0.2	0.04	0.008	0.01
L	0.1	0.01	0.003	0
L	0.1	0.01	0.0039	0.01
L	0.1	0.02	0.0032	0
L	0.1	0.02	0.0043	0.01

Table 4.5: Trajectory following performance on circular shapes of various radii, with and without orientation cost with our improved version.

Shape	Radius (m)	Avg. Push Size (m)	Avg. Pos. Error (m)	Orientation Cost
Circle	0.1	0.0105	0.0037	0
Circle	0.1	0.0105	0.0041	0.01
Circle	0.1	0.0157	0.0061	0
Circle	0.1	0.0157	0.0078	0.01
Circle	0.1	0.02	0.007	0
Circle	0.1	0.02	0.0079	0.01

Obstacle Avoidance: As mentioned before, this task is precise posing with an additional obstacle avoidance term, as defined in Eq. 3.9. We use the objective function defined in Eq. 3.6 for posing so that the motion planning cost does not implicitly help obstacle avoidance with less side switching. We use rectangular-prism objects of height 7cm and square base of 3cm and 5cm to simulate obstacles in the environment and evaluate our method under various obstacle configurations from single to multiple obstacles. We ensure each configuration has a collision-free path and repeat them 5 times. Certain configurations require large orientation changes to traverse narrow passages. For instance, Our method was not successful where the object was required to turn above 60 degrees to get through the passage.

Configurations with single objects or two objects with relatively wide separation are typically easy to handle for the method. As the separation between obstacles decreases, the success rate drops. Additionally, the success rate decreases with the number of obstacles, as expected. Our approach struggles in scenarios requiring the object to rotate significantly—more than 60°—to pass through narrow gaps between two closely placed obstacles. These larger rotation demands highlight the need for more advanced planning capabilities, as the current system's limited planning horizon and fixed action sequences restrict its ability to adapt to complex, intricate maneuvers.

Due to the local nature of the MPC framework, the proposed obstacle avoidance method is not complete. For example, certain cul-de-sac configurations lead to inevitable failures because the controller lacks global re-planning capability.

4.2 Real World

Real World Setup: We conduct a series of real-world experiments to evaluate the effectiveness of our planar pushing approach. The robotic setup consists of a Franka Emika Panda robot, which holds a 15 cm stick equipped with a elastic spherical ball attached to its tip. A Kinect 360 RGBD camera is used to perform markerless tracking of object poses in real time. We use an old sensor without markers to have a challenging noisy environment to provide a sharp contrast to the simulation environment. The experiments were performed on a 90 cm by 60 cm table. The tracking standard deviation was around 1 mm and 1 degrees but some far away parts of the table resulted in systematic errors due to camera angles which we avoid. This setup provides a realistic and challenging evaluation environment and can be seen in Fig. 1.1. To ensure quasi-static conditions during real-world trials, the robot's velocity is limited to a maximum of 10 cm/s. Unlike in simulation, where both velocity and step duration are explicitly defined, the real-world execution uses a simple position controller to push the object according to the selected action. To maintain accurate pose tracking, the robot takes a brief step backward after each action, allowing the perception system to reacquire the object pose before selecting and executing the next action. The speed of our real-world experiments is limited by the backward movement required at each step; however, we have demonstrated that our system operates efficiently and fast enough in simulation.

Figure 4.12: The objects used in the real-world experiments. The box at the bottom, object 1, has dimensions of 9 $\times$ 11 cm, the box at the top, object 2, has dimensions of 10 $\times$ 12 cm, and the box on the right, object 3, has dimensions of 13 $\times$ 11 cm.

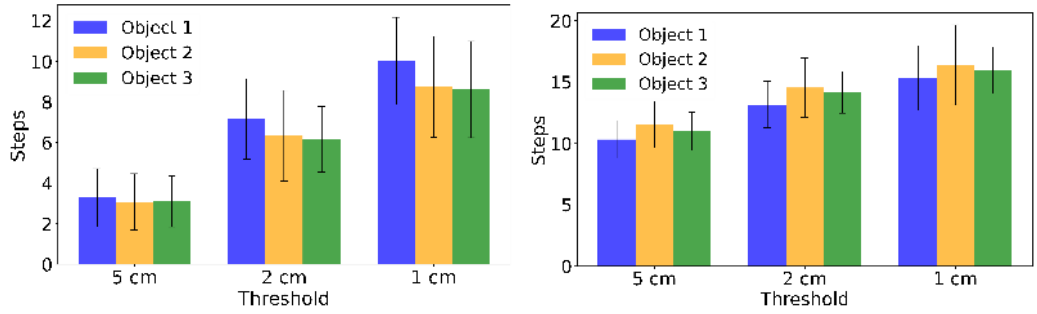
Precise Manipulation:

We perform the precise posing task on three objects, shown in Fig. 4.12. We divide the experiments into two categories: short distances (position: 9-15 cm, orientation: 0-60°) and long distances (position: 15-20 cm, orientation: 60-90°), with all objects starting at the same initial pose. We perform a total of 300 push episodes. Additionally, we conduct twenty experiments in which both the initial and target poses are randomly chosen.

Fig. 4.13 illustrates the number of steps required to reach the target poses. The results mirror that of the simulation case. In addition, we can see that the required number of steps increase with object size. Table 4.6 shows the success rate of all the pushes for 1cm position and two a posteriori orientation thresholds. Note that the objective included orientation but the stopping condition did not, which explains less than 100% success rates. The numbers are fairly close for all cases and they are all relatively high. This shows the real-world performance of our approach. We lastly run 10 short distance object 1 cases with a (5 mm, 2°) threshold with average 10.8 steps and 100% success rate.

We also measured the one-step prediction error of the push dynamics model. The average distance error is 3.8 mm for the same starting pose cases and 4.8 mm for the random starting pose cases, while the average orientation error is 2.5° for aforementioned

cases.



(a) (b)

Figure 4.13: The performance of our method in real-world experiments. The left subfigure correspond to short distance (9-15 cm, 0°–60°) configurations and the right subfigure to long distance (15-20 cm, 60°–90°) configurations.

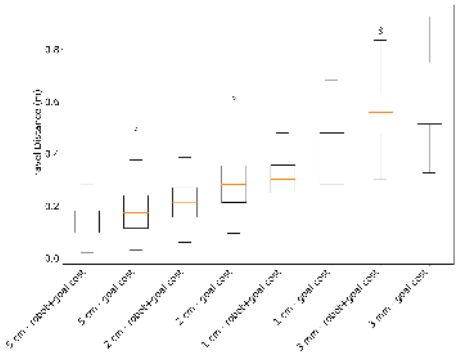
Table 4.6: Success rates for reaching the target pose in manipulation tasks for less than the mentioned thresholds.

Episode Type	5 degrees	10 degrees
Long Distant Episodes	0.875	0.98
Short Distant Episodes	0.86	0.97
Random Episodes	0.9	0.95

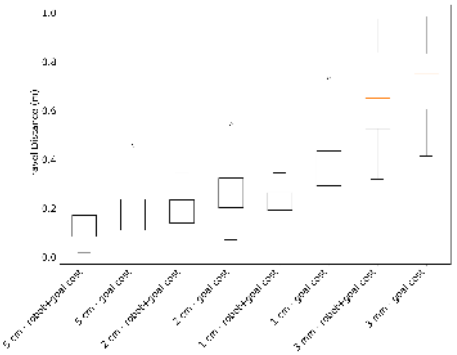
Trajectory Following: The trajectory following evaluations are conducted with two distinct shapes for the object's path: a circular trajectory and an L-shaped trajectory. In each experiment, the robotic pusher was tasked with guiding the object along these predefined shapes while minimizing the distance between successive waypoints to less than half of the set distances. The average displacement error ranged from 5 to 7 mm, with the average push length between 1.67 cm and 2 cm for L-shaped trajectories. For circular trajectories, the average displacement error ranged from 6 to 7 mm, with the average push length again between 1.67 cm and 2 cm. Fig. 4.10b shows an example L-shaped trajectory following result.

Obstacle Avoidance: We evaluated the obstacle avoidance task through five experiments with a single obstacle and five experiments with two obstacles. In these tests, our method successfully reached the target pose while avoiding obstacles, maintaining a safe distance of at least 1 cm without any collisions. Fig. 4.11 illustrates the robot pushing the object to the target pose while avoiding a single obstacle. Fig. 4.10a shows the object trajectory for

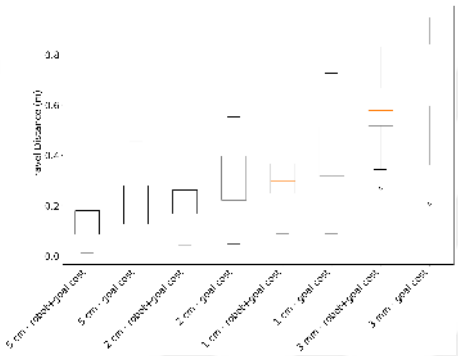
a more difficult 2 obstacle case. However, similar to the simulation case and as expected, our method struggles to find a feasible solutions for more complex configurations. The reasons are the same as the simulation case with the added issues of perception noise.



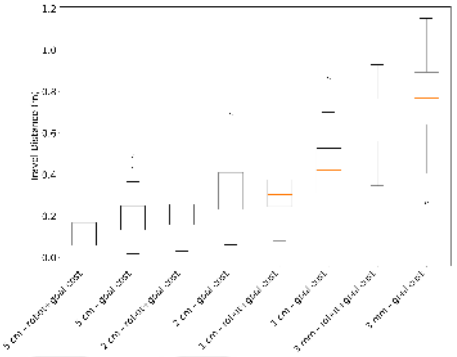
(a) -45° to 45°, long dist, horizon 4.



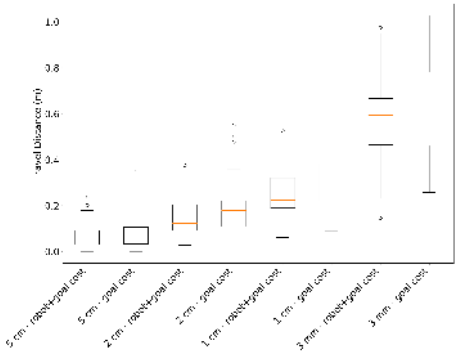
(b) -45° to 45°, long dist, horizon 3



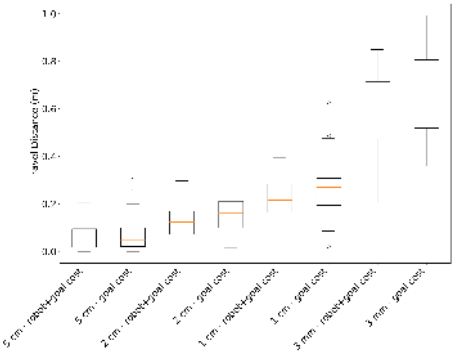
(c) -5° to 5°, long dist, horizon 4.



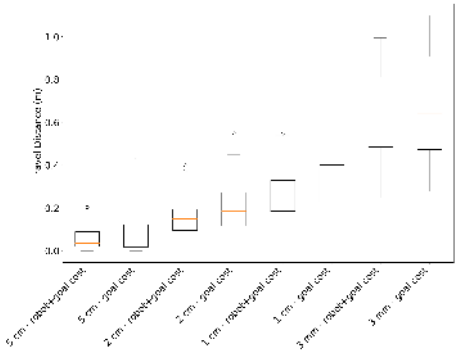
(d) -5° to 5°, long dist, horizon 3



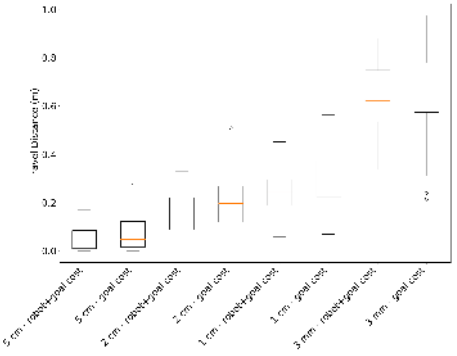
(e) -45° to 45°, short dist, horizon 4



(f) -45° to 45°, short dist, horizon 3

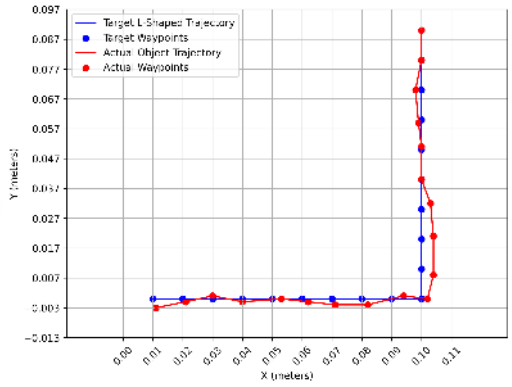


(g) -5° to 5°, short dist, horizon 4.

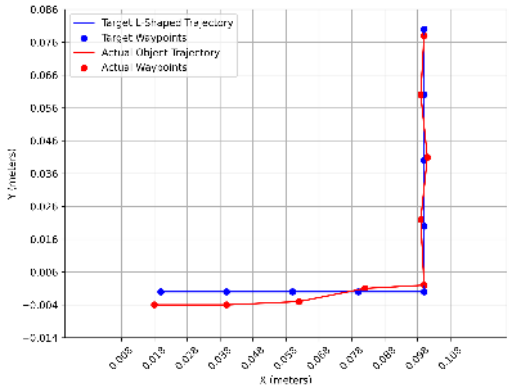


(h) -5° to 5°, short dist, horizon 3

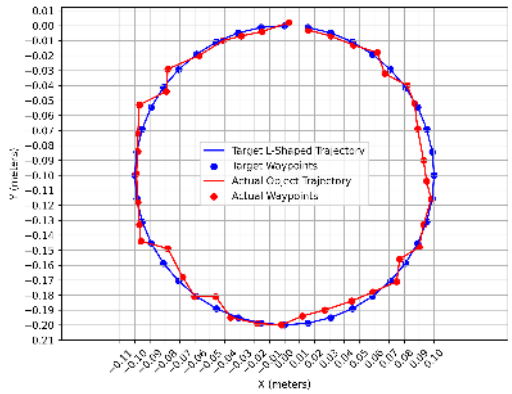
Figure 4.6: Robot travel distance for the improved version in simulation. Comparison of ”goal cost” (Eq. 3.6) and ”robot cost + goal cost” (Eq. 3.7) across different pushing scenarios. Initial push directions are sampled from either a wide range ($[-45°, 45°]$) in **(a)**, **(b)**, **(e)**, **(f)** or a narrow range ($[-5°, 5°]$) in **(c)**, **(d)**, **(g)**, **(h)**. Each pair contrasts long vs. short pushing distances and roll-out horizons: **(a)** wide, long, horizon=4; **(b)** wide, long, horizon=3; **(c)** narrow, long, horizon=4; **(d)** narrow, long, horizon=3; **(e)** wide, short, horizon=4; **(f)** wide, short, horizon=3; **(g)** narrow, short, horizon=4; **(h)** narrow, short, horizon=3.



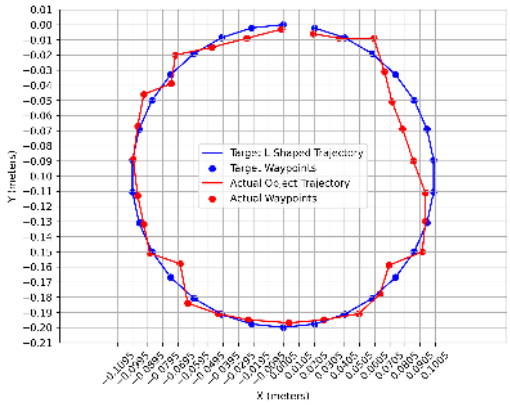
(a) L shape, Average distance : 1 cm, error : 2.1 mm.



(b) L shape, Average distance : 1 cm, error : 2.6 mm.

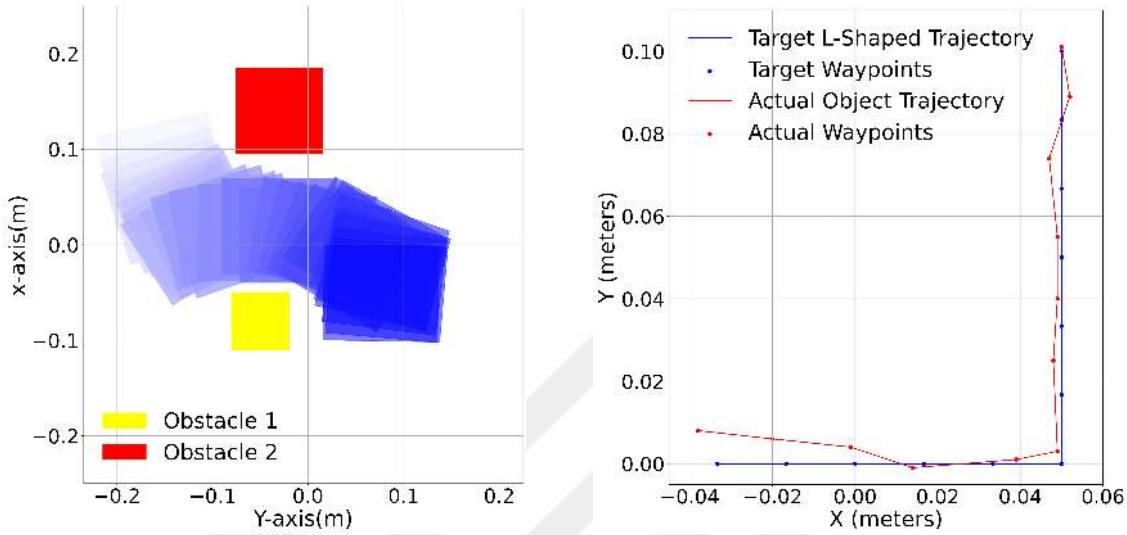


(c) O shape, Average distance : 1.57 cm, error : 4 mm.



(d) O shape, Average distance : 2.1 cm, error : 6.6 mm

Figure 4.8: Representative simulation results of trajectory following without orientation cost. Top row: L-shaped trajectories with push sizes of 1 cm and corresponding average position errors of 2.1 mm and 2.6 mm. Bottom row: circular (O-shaped) trajectories with push sizes of 1.57 cm and 2.1 cm, and corresponding average position errors of 4 mm and 6.6 mm. These cases demonstrate the controller's high accuracy across different path geometries and step sizes.



(a) Obstacle Avoidance (b) Trajectory Following

Figure 4.10: Obstacle avoidance (a) and trajectory following (b) cases in the real world. (a) illustrates a two-obstacle scenario where the object rotates to pass through the narrow passage and then rotates again to reach the target pose.(b) depicts an ”L”-shaped trajectory following in a real-world scenario, with a one-step average prediction error of 5.6 mm.

(a) (b) (c) (d) (e) (f)

Figure 4.11: Frames from real-world object avoidance case.

Chapter 5

DISCUSSION

In this work, we proposed a framework that decouples the dynamics model from the controller for planar pushing. A detailed comparison with other methods is not straightforward, since many of them incorporate additional objects and perception-based state representations, making their models inherently aware of object shape and size. In contrast, our model achieved high accuracy: less than 1 and 0.5° error on a basic dataset, and less than 2 and 2° error on a more diverse dataset. Considering the variety in pushing lengths and directions, as well as the consistent performance observed in the precision positioning task, these results demonstrate that our model successfully learns the dynamics of cubic object pushes and generalizes to new environments and object sizes with our representation. Moreover, while other model-based approaches without perception cannot exhibit side-switching behavior, our framework enables it through a carefully designed state representation and sampling of orientation and side during control. While prior works such as [Bauza and Rodriguez, 2017] and [Wang et al., 2024] employ non-parametric methods for learning forward dynamics models, our approach leverages a recurrent neural network to incorporate history encoding and to enable faster prediction on our large-scale dataset. The proposed model demonstrated improved performance compared to the method presented in [Cong et al., 2020]. Specifically, we incorporated additional MLP layers at the final stages of the GRU-based architecture, enabling the model to better capture the non-linear dynamics inherent in pushing interactions. Furthermore, by including the relative pose in our state representation in addition to the push magnitude, we enabled side-switching behavior during execution. Side-switching allows the controller to adaptively change the pushing side and select optimal contact points to guide the object more efficiently toward the target pose. A critical design aspect that enabled this capability was the computation of relative pose and push magnitudes with respect to the object's current orientation, ensuring consistency and generalization across varying scenarios.

Our extensive experiments demonstrate that the proposed method can achieve high accuracy of sub-centimeter and reaching within 2 cm of the goal pose in under 10 steps, even for long-horizon tasks spanning distances of 20–30 cm. This represents a significant improvement over prior non-prehensile manipulation methods, which typically aimed only to push the object from its initial pose to the target pose without explicitly handling precise intermediate positioning. While our primitive method relied on MPPI in its raw form, we improved the framework by extending the push length, incorporating balanced sampling across sides, and introducing greedy action selection. These enhancements effectively reduced the number of steps required to reach the goal and lowered the overall motion planning effort of the robot. Visual and quantitative comparisons between the primitive and improved versions clearly show the benefits: the improved framework achieves faster convergence while balancing position and orientation corrections, particularly in longer-distance tasks where greater positional freedom allows for better orientation adjustments.

Overall, our results highlight significant improvements over previous model-based approaches that do not incorporate perception and, at the same time, demonstrate advantages over model-free methods. Unlike model-free strategies, which often require task-specific training, our learned model can be adapted to multiple tasks simply by modifying the cost function or control objective, without additional retraining. This flexibility underscores the broader potential of combining model-based learning with sampling-based control for robust and versatile non-prehensile manipulation.

Chapter 6

CONCLUSION AND FUTURE WORK

In this thesis, we proposed a novel GRU-based dynamics model for planar pushing, trained in simulation with extensive domain randomization to improve generalization to real-world conditions. Coupled with a sampling-based Model Predictive Path Integral (MPPI) controller, our approach effectively exploits the learned dynamics to generate adaptive, task-oriented actions. By leveraging a state–action representation that encodes both relative pose and push directions, the framework enables versatile capabilities such as object-side switching, variable-length pushes, and integration of diverse objectives. These features collectively allow the system to achieve not only precise positioning but also more complex behaviors such as trajectory following and obstacle avoidance.Through systematic evaluation, we demonstrated the efficacy of our method in both simulated environments and real-world experiments with a Franka Panda robot equipped with markerless tracking. The results confirmed high success rates in precise positioning tasks under demanding thresholds of position and orientation, while also showing robust performance in dynamic tasks like obstacle avoidance and path following. Building upon this foundation, we also introduced an improved version of our framework designed to address long-horizon tasks more efficiently. Specifically, we trained a model capable of handling longer pushes and incorporated a balanced sampling strategy across object sides. Combined with a greedy action selection mechanism, this enhancement allowed the controller to prioritize high-quality actions, significantly reducing the number of steps required to achieve longer-horizon objectives. This improved framework demonstrated faster convergence toward task goals while maintaining robustness.

Despite these encouraging results, several avenues remain for improvement. Our current controller, although enhanced with balanced sampling, still relies on sampling-based rollouts, which can be computationally demanding. Future work can explore more sophisticated, goal-aware sampling strategies, for example by learning a sampling policy or

developing an inverse dynamics model to guide exploration. Additionally, learning a dedicated sampling function could streamline the selection of optimal actions within rollouts, thereby reducing planning time while improving efficiency. These enhancements would allow the framework to scale to more complex environments, enabling it to handle more challenging obstacle-avoidance and long-horizon planning tasks. Another promising direction is to integrate our method with a global planner, allowing the system to seamlessly combine local pushing strategies with long-range planning objectives. Finally, to broaden the applicability of our approach, future efforts should focus on integrating perceptual representations derived from vision-based inputs, which would allow the model to generalize across a wider variety of object geometries.

BIBLIOGRAPHY

[Arruda et al., 2017] Arruda, E., Mathew, M. J., Kopicki, M., Mistry, M., Azad, M., and Wyatt, J. L. (2017). Uncertainty averse pushing with model predictive path integral control. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 497–502. IEEE.

[Bauza et al., 2019] Bauza, M., Alet, F., Lin, Y.-C., Lozano-Pérez, T., Kaelbling, L. P., Isola, P., and Rodriguez, A. (2019). Omnipush: accurate, diverse, real-world dataset of pushing dynamics with rgb-d video. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4265–4272. IEEE.

[Bauza and Rodriguez, 2017] Bauza, M. and Rodriguez, A. (2017). A probabilistic data-driven model for planar pushing. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3008–3015. IEEE.

[Bledt et al., 2018] Bledt, G., Powell, M. J., Katz, B., Di Carlo, J., Wensing, P. M., and Kim, S. (2018). Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2245–2252.

[Cong et al., 2020] Cong, L., Grner, M., Ruppel, P., Liang, H., Hendrich, N., and Zhang, J. (2020). Self-adapting recurrent models for object pushing from learning in simulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5304–5310. IEEE.

[Cong et al., 2022] Cong, L., Liang, H., Ruppel, P., Shi, Y., Görner, M., Hendrich, N., and Zhang, J. (2022). Reinforcement learning with vision-proprioception model for robot planar pushing. *Frontiers in Neurorobotics*, 16.

[Dengler et al., 2024] Dengler, N., Ferrandis, J. D. A., Moura, J., Vijayakumar, S., and Bennewitz, M. (2024). Learning goal-directed object pushing in cluttered scenes with location-based attention. *arXiv preprint arXiv:2403.17667*.

[Ferrandis et al., 2023] Ferrandis, J. D. A., Moura, J., and Vijayakumar, S. (2023). Non-prehensile planar manipulation through reinforcement learning with multimodal categorical exploration. In *2023 International Conference on Intelligent Robots and Systems (IROS)*, pages 5606–5613. IEEE.

[Gao et al., 2020] Gao, Z., Elibol, A., and Chong, N. Y. (2020). Non-prehensile manipulation learning through self-supervision. In *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, pages 93–99.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Goyal et al., 1989] Goyal, S., Ruina, A., and Papadopoulos, J. (1989). Limit surface and moment function descriptions of planar sliding. In *1989 IEEE International Conference on Robotics and Automation*, pages 794–795. IEEE Computer Society.

[Herzog et al., 2023] Herzog, A., Rao, K., Hausman, K., Lu, Y., Wohlhart, P., Yan, M., Lin, J., Arenas, M. G., Xiao, T., Kappler, D., et al. (2023). Deep rl at scale: Sorting waste in office buildings with a fleet of mobile manipulators. *arXiv preprint arXiv:2305.03270*.

[Jia and Erdmann, 1999] Jia, Y.-B. and Erdmann, M. (1999). Pose and motion from contact. *The International Journal of Robotics Research*, 18(5):466–487.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kloss et al., 2020] Kloss, A., Bauza, M., Wu, J., Tenenbaum, J. B., Rodriguez, A., and Bohg, J. (2020). Accurate vision-based manipulation through contact reasoning.

In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6738–6744. IEEE.

[Lee et al., 2015] Lee, G., Lozano-Pérez, T., and Kaelbling, L. P. (2015). Hierarchical planning for multi-contact non-prehensile manipulation. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 264–271. IEEE.

[Li et al., 2018] Li, J., Lee, W. S., and Hsu, D. (2018). Push-net: Deep planar pushing for objects with unknown physical properties. *Robotics: Science and Systems XIV*.

[Lynch, 1992] Lynch, K. (1992). The mechanics of fine manipulation by pushing. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pages 2269–2276 vol.3.

[Mason, 1986] Mason, M. T. (1986). Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71.

[Peng et al., 2018] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE.

[Singh et al., 2024] Singh, S., Vaishnav, R., Gautam, S., and Banerjee, S. (2024). Agricultural robotics: A comprehensive review of applications, challenges and future prospects. In *2024 2nd international conference on artificial intelligence and machine learning applications theme: healthcare and internet of things (AIMLA)*, pages 1–8. IEEE.

[Stüber et al., 2020] Stüber, J., Zito, C., and Stolkin, R. (2020). Let's push things forward: A survey on robot pushing. *Frontiers in Robotics and AI*, 7:8.

[Stüber et al., 2020] Stüber, J., Zito, C., and Stolkin, R. (2020). Let's push things forward: A survey on robot pushing. *Frontiers in Robotics and AI*, 7.

[Tobin et al., 2017] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE.

[Wang et al., 2024] Wang, G., Ren, K., and Hang, K. (2024). Uno push: Unified nonprehensile object pushing via non-parametric estimation and model predictive control. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9893–9900.

[Wang et al., 2023] Wang, S., Sun, L., Zha, F., Guo, W., and Wang, P. (2023). Learning adaptive reaching and pushing skills using contact information. *Frontiers in Neurorobotics*, 17:1271607.

[Williams et al., 2016] Williams, G., Drews, P., Goldfain, B., Rehg, J. M., and Theodorou, E. A. (2016). Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440.

[Yu et al., 2016] Yu, K.-T., Bauza, M., Fazeli, N., and Rodriguez, A. (2016). 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)*, pages 30–37. IEEE.