On Deep Learning for Nonprehensile Manipulation

by

Steven Caro

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2026

© Steven Caro 2026

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Steven Caro was the sole author for Chapters 1, 2, 4, and 5 which were written under the supervision of Prof. Stephen L. Smith. Chapter 3 was written for publication in the proceedings of the Conference on Robots and Vision (CRV). Chapter 4 was written for publication in the IEEE Robotics and Automation Letters (RA-L).

Research presented in Chapter 3:

Chapter 3 is based on the following collaborative work:

Ninghan Zhong, Steven Caro, Megnath Ramesh, Rishi Bhatnagar, Avraiem Iskandar, and Stephen L Smith. Bench-Push: Benchmarking pushing-based navigation and manipulation tasks for mobile robots. *arXiv preprint arXiv:2512.11736*, 2025. [60]

Steven Caro and Ninghan Zhong were the primary authors. Steven Caro, Ninghan Zhong, Megnath Ramesh, and Avraiem Iskandar collaboratively developed the Maze, Ship-Ice, Box-Delivery and Area-Clearing environments in Sec. 3.1.1, designed the evaluation metrics in Sec. 3.1.3, trained reference baselines in Sec. 3.2.1, and conducted the physical experiments in Sec. 3.2.2. Rishi Bhatnagar developed the Mujoco variants of the environments in Sec. 3.1.1.

Abstract

Nonprehensile manipulation, i.e. interaction without grasping, is a fundamental capability for mobile robots operating in unstructured environments, yet it remains a challenging control problem due to complex contact dynamics and under-actuated physics. Progress in this domain has been further hindered by the lack of standardized evaluation frameworks, leading to fragmented research efforts. This thesis addresses these gaps through two primary contributions: a unified benchmarking suite and a novel hierarchical control architecture.

First, we introduce **Bench-Push**, a comprehensive benchmark designed specifically for pushing-based mobile robot tasks. Unlike existing benchmarks that penalize environment interaction, Bench-Push provides diverse environments – ranging from navigation-centric mazes to manipulation-centric delivery tasks – and introduces novel metrics to quantify the trade-off between task efficiency and interaction effort. We validate the framework by evaluating state-of-the-art baselines and demonstrating successful zero-shot transfer to a physical robot.

Second, we propose the **Hie**rarchical **R**einforcement Learning - **D**iffusion Policy (**HeRD**), a hybrid architecture designed to solve long-horizon manipulation tasks. We identify that while Reinforcement Learning (RL) excels at strategic decision-making, it struggles to learn precise low-level contact dynamics. Conversely, Generative Diffusion Models synthesize smooth, context-aware trajectories, but lack high-level planning capabilities. HeRD bridges this gap by decoupling the control hierarchy: a high-level RL planner, utilizing a Spatial Action Map action space, selects strategic subgoals, which are then executed by a low-level, goal-conditioned diffusion policy. Extensive experiments in the *Box-Delivery* task demonstrate that HeRD significantly outperforms both state-of-the-art learning-based methods (SAM) and classical motion planners (Greedy Heuristic, Hierarchical RRT*). HeRD achieves higher success rates and greater interaction efficiency in both simulation and real-world deployments. Furthermore, we demonstrate that HeRD is capable of robust zero-shot generalization to unseen, unstructured clutter, successfully navigating complex environments where classical planners experience catastrophic failure.

Acknowledgements

I would like to thank my parents, Dino and Gioia; brothers, Nicholas and Daniel; and girlfriend, Natalie. I consider myself extremely lucky to be surrounded by a family that unconditionally loves and supports me.

To my friends and extended family – you made sure I always had a home to come back to in Windsor.

I would especially like to thank my supervisor, Stephen L. Smith. I have grown a lot over the last two years, and I attribute much of that to Steve. His unwavering support and rigorous approach to problem-solving helped me to hold my work to a high standard and showed me what it truly means to be a researcher.

I would also like to thank my fellow lab members, especially my co-authors Ninghan Zhong, Megnath Ramesh, and Avraiem Iskandar. Thank you for making all those days in the lab both productive and enjoyable. The countless hours we spent brainstorming and debugging are memories I will fondly look back on.

Dedication

To Mom, Dad, Nicholas, Daniel, and Natalie

Table of Contents

Author's Declaration ii

Statement of Contributions iii

Abstract iv

Acknowledgements v

Dedication vi

List of Figures x

List of Tables xiii

1 Introduction 1

1.1 Benchmarking Nonprehensile Manipulation Tasks 2

1.2 Nonprehensile Box Delivery 3

1.3 Thesis Contributions 4

1.4 Thesis Outline 6

2 Literature Review and Background 7

2.1 Literature Review 7

2.1.1 Pushing-based Mobile Manipulation 8

2.1.2 Mobile Robot Benchmarks . . . 9
2.1.3 Reinforcement Learning . . . 9
2.1.4 Generative Diffusion Models . . . 11
2.2 Background . . . 13
2.2.1 Deep Reinforcement Learning . . . 13
2.2.2 Diffusion Models . . . 15

3 Bench-Push: Benchmarking Pushing-based Navigation and Manipulation Tasks for Mobile Robots 19
3.1 Bench-Push . . . 19
3.1.1 Bench-Push Environments . . . 20
3.1.2 Bench-Push Implementation . . . 23
3.1.3 Metrics . . . 25
3.2 Evaluating Baselines using Bench-Push . . . 29
3.2.1 Implemented Baselines . . . 29
3.2.2 Physical Experiment Setup . . . 30
3.2.3 *Maze* Evaluations . . . 30
3.2.4 *Box-Delivery* Evaluations . . . 32
3.3 Summary . . . 33

4 Hierarchical RL-Diffusion Policy for Efficient Nonprehensile Manipulation 34
4.1 Problem Formulation and Spatial Action Map Overview . . . 34
4.1.1 Problem Formulation . . . 34
4.1.2 Spatial Action Maps . . . 35
4.2 Approach Overview . . . 37
4.3 Implementation Details . . . 38
4.3.1 Reinforcement Learning Policy . . . 38

4.3.2 Diffusion Policy 41
4.4 Results 43
4.4.1 Real-World Experiments 47
4.5 Summary 47

5 Conclusion **48**
5.1 Limitations and Future Work 49
5.1.1 Bench-Push 49
5.1.2 HeRD 49

References **51**

List of Figures

1.1 Illustrations of pushing-based navigation and manipulation environments in both 2D and 3D simulations, each with configurable complexity levels. *Maze* (top-left): the mobile robot must reach the goal point among movable obstacles. *Ship-Ice* (lower-left): an autonomous ship must navigate through ice-covered waters. *Box-Delivery* (top-right): the robot needs to push all boxes into the green receptacle. *Area-Clearing* (lower-right): the robot needs to remove all boxes from within a clearance area outlined by the green rectangle. 3

1.2 The robot (highlighted in red) is tasked with pushing the boxes into the green receptacle. This sequence of images shows a robot using the proposed HeRD policy gathering three boxes and pushing them into the receptacle. 4

1.3 *Maze* environment in 2D simulation (top-left panel), in 3D simulation (lower-left panel), and in our physical testbed (right panel). 5

3.1 The *Maze* environment (left) and an example egocentric observation for *Maze* (right). 21

3.2 The *Box-Delivery* environment (left) and an example observation for *Box-Delivery* (right). 22

3.3 Three teleoperated *Maze* paths and their performance. While the shortest path (red) has the highest efficiency score $E_{\text{nav}} = 0.86$, excessive collisions with the movable objects degrade the effort score $I_{\text{nav}} = 0.56$. In contrast, while the longest path (blue) is collision-free $I_{\text{nav}} = 1.00$, efficiency is largely compromised $E_{\text{nav}} = 0.54$. A balanced path (yellow) potentially offers the best trade-off. 26

3.4 Three teleoperated paths for the area clearing task, where all boxes must be removed from an area (green square). Each path has a task success score $S_{\text{manip}} = 2/3$. While path 1 (red) has the highest efficiency score, its effort score is low as it moves a box longer than its shortest distance to the goal (dotted lines of length l_i^*). In contrast, path 3 (blue) achieves the best effort score by pushing as little as possible, but produces a long path and low efficiency. Path 2 (yellow) potentially offers the best trade-off. 27

4.1 Visualization of the four-channel state representation in the *LargeDivider* environment (colors shown for clarity only). 36

4.2 HeRD architecture. The high-level RL policy encodes observations into a spatial action map and selects a spatial goal, which is converted to a path using SPFA. If the path intersects boxes, it is executed by a proportional controller that first rotates then translates the robot; otherwise, a diffusion policy generates a goal-conditioned trajectory through denoising and feasibility conditioning. 37

4.3 Evaluation environments used in HeRD vary in size and layout. Small environments measure 10m x 5m with 10 boxes; Large environments are 10m x 10m with 20 boxes. Robot, box, and obstacle positions are randomized per episode, with column counts ranging from 0–2 (Small) to 0–8 (Large). To test zero-shot generalization, we introduce the Apartment environment. This mimics a residential layout with irregular, non-convex obstacles (e.g., L-shaped couches) unseen during training, evaluating how well learned behaviours transfer to unstructured, realistic settings. 40

4.4 Denoising process of a generated path. The high-level RL policy selects a spatial goal (the black dot in the figure), and the low-level diffusion policy generates a context-aware trajectory from the robot to the spatial goal. . . 41

4.5 Diffusion-based trajectory generation pipeline. At each difusion timestep i, the trajectory τ^i is denoised using a 1D U-Net conditioned on observation features. The trajectory's endpoints are fixed via goal inpainting: the first point is the robot's position, and the last is the spatial goal. If $i > 0$, the intermediate points are further denoised. Once denoising completes ($i = 0$), the trajectory undergoes a feasibility pipeline: each waypoint is projected to free space if necessary, redundant waypoints are pruned, and path segments crossing obstacles are repaired via SPFA. The final output is a smooth trajectory that respects geometric constraints. 42

4.6 The difference in behaviour of HeRD compared to the baseline. The figure shows the first few actions of each policy in the same environment. HeRD immediately seeks out boxes to push into the receptacle while the baseline focuses on gathering multiple boxes together, even pushing them backwards to do so. 45

4.7 Training curves for the high-level policy. HeRD converges earlier (learns more efficiently) than the no-diffusion variant. 46

List of Tables

3.1 Interactive pushing-based tasks included in Bench-Push. 20

3.2 Rewards, actions, and observations for each task. 20

3.3 Results for *Maze* in the physical testbed and Bench-Push simulations. ↑ indicates higher is better. 31

3.4 Results for *Box-Delivery* in the physical testbed and Bench-Push simulations. ↑ indicates higher is better. 33

4.1 HeRD vs. baseline performance. Bold text indicates the highest number of boxes delivered and the shortest travel distance (evaluated only for policies exceeding an 80% delivery threshold). 44

4.2 Effect of the different components of HeRD 44

Chapter 1

Introduction

The intersection between machine learning and robotics has been a cutting-edge area of research for many years. Breakthroughs in this field have far-reaching implications that span multiple industries, improving quality, efficiency, and safety in sectors such as manufacturing and transportation. Historically, robotic control relied on analytical methods that required precise models of the environment and robot dynamics. While effective in structured settings like factory assembly lines, these "classical" controllers often falter in unstructured environments where dynamics are hard to model and disturbances are frequent [21].

Recently, deep learning has demonstrated its potential to bridge this gap, allowing robots to learn control policies directly from sensory data. Two particular approaches dominate this field with great success: Reinforcement Learning and Generative Diffusion Models.

Reinforcement Learning (RL) has been the primary focus of research for deep learning in robotics since the field grew in popularity. By framing control as an optimization problem over a cumulative reward signal, RL agents have mastered complex strategies in games like Go and Dota 2, and more notably, have solved high-dimensional motor control tasks in robotics that were previously intractable [27, 34]. In manipulation, RL allows robots to discover dynamic strategies through trial and error. However, RL faces significant challenges: it suffers from the "curse of dimensionality" in continuous spaces and often struggles to learn precise contact dynamics from sparse rewards, leading to sample inefficiency and unsafe exploration phases [21].

Generative Diffusion Models have recently emerged as a powerful alternative. Originally designed for image synthesis, diffusion models in robotics treat trajectory generation as

a conditional denoising process [8]. Unlike RL, which optimizes a single scalar reward, diffusion models learn to mimic the distribution of expert demonstrations. This allows them to generate smooth, multimodal, and human-like behaviours that naturally handle complex constraints. However, diffusion models lack an intrinsic notion of "success" or "optimality" regarding a long-term goal; they are excellent at executing a specific motion but struggle to plan sequences of actions over long horizons [19].

One area of particular focus for these learning-based tools is robotic manipulation. There is a growing reliance on robotic solutions for tasks that involve manipulating objects that are too heavy, dangerous, or inaccessible for humans. Whether positioning vehicle parts for assembly on a factory line or clearing rubble from a disaster site, we depend on robots to perform tasks that would otherwise be impossible.

Manipulation is generally categorized into two types: Prehensile Manipulation – grasping an object, typically with a multi-fingered hand or gripper; and Nonprehensile Manipulation – any non-grasping interaction, such as pushing, sliding, or blowing [37, 53].

Nonprehensile manipulation is a fundamental capability for general-purpose robots operating in cluttered or constrained environments. Compared to prehensile systems, nonprehensile manipulators can interact with multiple objects simultaneously and are capable of moving large or ungraspable items using simple, low-cost hardware [51]. Although humans intuitively learn pushing skills at a young age, the task remains highly complex for robots. The dynamics are nonlinear and contact-rich, making them difficult for classical controllers to model.

While the importance of this field becomes more apparent, there lacks a standardized system of evaluating and comparing methods. Current evaluations often rely on ad hoc, custom-built environments that limit reproducibility and cross-comparison between algorithms. Furthermore, solving the control problem itself is non-trivial; RL policies often struggle with the dual burden of learning high-level long-horizon planning alongside low-level contact dynamics.

1.1 Benchmarking Nonprehensile Manipulation Tasks

Despite a growing body of work on nonprehensile manipulation, research efforts remain fragmented. The field lacks a unified framework for systematic evaluation and reproducibility. Existing approaches address isolated aspects of the problem, such as navigating amongst moveable objects or gathering objects into a pile, but share a common theme: controlling a robot to reliably push objects toward a desired direction or goal. Yet, these

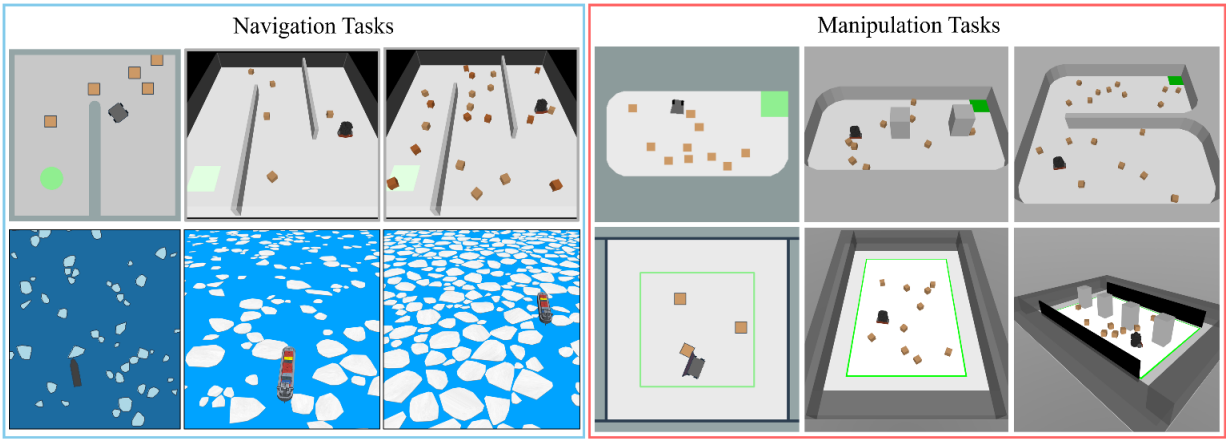


Figure 1.1: Illustrations of pushing-based navigation and manipulation environments in both 2D and 3D simulations, each with configurable complexity levels. *Maze* (top-left): the mobile robot must reach the goal point among movable obstacles. *Ship-Ice* (lower-left): an autonomous ship must navigate through ice-covered waters. *Box-Delivery* (top-right): the robot needs to push all boxes into the green receptacle. *Area-Clearing* (lower-right): the robot needs to remove all boxes from within a clearance area outlined by the green rectangle.

methods are typically tested in individually developed environments that, while similar in design, lack standardization. In addition, despite similar objectives, evaluations in existing works often rely on inconsistent metrics. While some prioritize success rates, others focus on path length, often neglecting the interaction effort: the physical cost of exertion on the environment. As a result, potentially transferable techniques [51, 50] are difficult to cross-compare.

Benchmarks have proven essential for advancing robotics research. Successful robotics benchmarks such as the YCB dataset [6], RoboSuite [62], and the Interactive Gibson Benchmark [55] have each driven significant progress in grasping, learning-based manipulation, and interactive navigation; however, there is no comparable standard for nonprehensile mobile manipulation.

1.2 Nonprehensile Box Delivery

A typical nonprehensile task is box delivery, where a robot must push scattered boxes through an environment of obstacles to a specific receptacle. This problem exposes the limitations of using current RL methods.

In robotic settings, RL policies often operate over low-level control signals such as

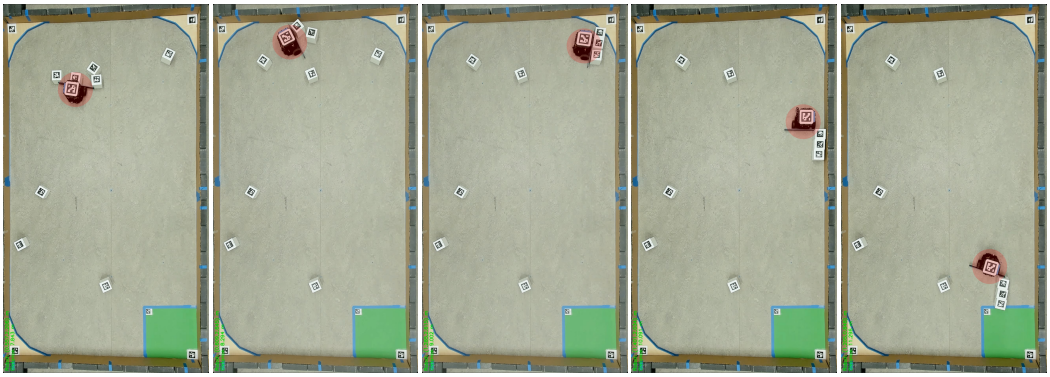


Figure 1.2: The robot (highlighted in red) is tasked with pushing the boxes into the green receptacle. This sequence of images shows a robot using the proposed HeRD policy gathering three boxes and pushing them into the receptacle.

wheel velocities or discrete movement commands (e.g., move forward, turn left) [1]. As a result, these policies must simultaneously learn task-relevant strategies and develop an understanding of the robot's dynamics. This dual burden makes learning more challenging, especially in contact-rich tasks like nonprehensile manipulation. Hierarchical RL addresses this challenge by separating high-level decision-making from low-level control. Wu et al. demonstrated that using Spatial Action Maps (SAM) [51], where an RL policy outputs spatial goals, combined with a separate controller for execution, improves performance in nonprehensile manipulation tasks.

On the other hand, diffusion policies, trained on human demonstrations, capture rich behavioural priors that reflect intuitive control strategies. In our case, these include stabilizing contact during object manipulation, anticipating future interactions, and avoiding disruptive collisions – strategies which are intuitive for a human but are difficult to learn from reinforcement alone, especially in the presence of sparse or delayed rewards. While much promise is shown here, diffusion policies are limited by the quality and diversity of the demonstrations provided.

1.3 Thesis Contributions

We explore multiple aspects of the field of nonprehensile mobile manipulation, from benchmarking and evaluating to developing efficient solutions to a well-known nonprehensile task. The primary contributions of this thesis are as follows:

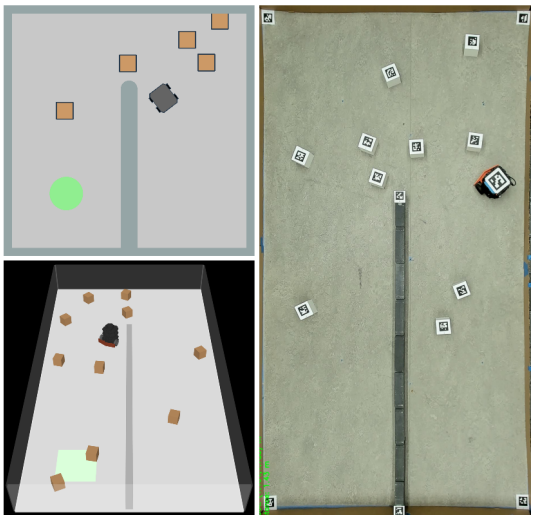


Figure 1.3: *Maze* environment in 2D simulation (top-left panel), in 3D simulation (lower-left panel), and in our physical testbed (right panel).

1. We present the first unified benchmark for pushing-based mobile manipulation, featuring diverse environments and novel metrics for interaction efficiency. We propose Bench-Push, the first comprehensive suite of tools for standardized training and evaluation of algorithms for pushing-based mobile robot tasks. We provide a range of simulated environments inspired by existing works in pushing-based mobile robot navigation and manipulation tasks. The environments are configurable by users to modify the difficulty. For each environment, we provide high-fidelity 3D simulations implemented using MuJoCo [42] to ultimately enable physical robot deployment, as shown in Fig. 1.3, and provide 2D Pymunk [5] simulations for rapid prototyping. We also propose a set of novel metrics in Bench-Push to evaluate and compare existing and future methods. The proposed metrics capture properties unique to mobile robot pushing-based tasks, such as efficiency and interaction trade-offs, and partial task completions. Additionally, we provide reference baseline implementations in Bench-Push and present their evaluation results as potential starting points for future research. Our baselines include both general reinforcement learning algorithms (SAC [14], PPO [34]) and task-specific methods such as Spatial Action Maps (SAM) [51] and ASV path planners [9, 61]. Finally, we present zero-shot sim-to-real transfer with policies trained using Bench-Push. We show that the evaluation results from the physical robot testbed generally align with those from simulation, showcasing Bench-Push as an effective workflow that scales from 2D rapid experimentation to 3D simulation and ultimately to real-world deployment.

2. We present the first hierarchical control policy that uses an RL agent as the high-level planner and a diffusion model as low-level trajectory generator. We introduce a goal-conditioned diffusion policy framework capable of producing collision-free trajectories by enforcing feasibility constraints. We then train this policy using human demonstrations to generate trajectories to navigate to spatial goals. Our proposed framework, **Hie**rarchical **R**einforcement Learning - **D**iffusion Policy (**HeRD**), combines the high-level planning of spatially-aware RL with the context-aware trajectory generation of a diffusion policy. Evaluated against both learning-based and classical motion planning baselines, HeRD achieves superior success rates and travel efficiency. Key to our approach is a new reward function that encourages the RL agent to act more efficiently. Finally, in our physical implementation, we observe a substantial increase in success rate of the HeRD policy when compared to the current state-of-the-art policy.

1.4 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 provides a comprehensive review of the related literature and establishes the theoretical foundations for this work. We examine the current state of pushing-based mobile manipulation, identifying gaps in existing benchmarks and control strategies. We then provide the mathematical background for Markov Decision Processes, Deep Reinforcement Learning (specifically Deep Q-Networks), and Generative Diffusion Models, which form the basis of our proposed methods.

Chapter 3 introduces **Bench-Push**, a unified benchmarking framework for evaluating nonprehensile mobile manipulation. This chapter details the design of diverse simulation environments and proposes a novel set of metrics to quantify interaction efficiency and effort. We also present baseline evaluations using state-of-the-art reinforcement learning algorithms and demonstrate the framework's capability for sim-to-real transfer.

Chapter 4 presents **HeRD**, a novel control architecture for efficient object manipulation. We describe the hierarchical formulation that pairs a high-level RL planner with a low-level diffusion-based trajectory generator. We provide a detailed ablation study analyzing the impact of our reward structure and hybrid control strategy, followed by experimental results demonstrating superior performance over existing baselines in both simulation and with a physical robot.

Finally, **Chapter 5** summarizes the contributions of this thesis, discusses the limitations of the proposed approaches, and outlines potential directions for future research.

Chapter 2

Literature Review and Background

This chapter provides the context and theoretical foundations necessary to understand the contributions presented in this thesis. We begin with a review of the related work in Section 2.1, examining the current state of pushing-based mobile manipulation, the limitations of existing benchmarking frameworks, and the evolution of hierarchical control strategies in robotics. Following this, Section 2.2 establishes the mathematical preliminaries required to formulate our proposed solution. We detail the fundamentals of Markov Decision Processes and Deep Reinforcement Learning, specifically focusing on value-based formulations. Finally we discuss Generative Diffusion Models, deriving their theoretical basis from Stochastic Differential Equations and explaining techniques for conditioning these models on given conditions. Together, these sections identify the gaps in the current literature and define the algorithmic tools used to address them.

2.1 Literature Review

This section reviews the state-of-the-art in nonprehensile mobile manipulation, identifying gaps in both evaluation methodologies and algorithmic control. We first examine the landscape of pushing-based tasks and the specific challenges imposed by contact dynamics. We then analyze existing benchmarks to highlight the lack of standardized metrics for interaction-rich environments. Finally, we discuss the evolution of learning-based control, specifically focusing on the trade-offs between Reinforcement Learning and Diffusion Models, which motivates the hierarchical architecture proposed in this thesis.

2.1.1 Pushing-based Mobile Manipulation

Pushing-based tasks represent a distinct subset of mobile robotics where the agent must intentionally interact with the environment to achieve a goal. Unlike traditional navigation, which treats obstacles as static constraints to be avoided, pushing-based manipulation leverages physical interaction to clear paths or rearrange objects.

Navigation vs. Manipulation

Research in this domain generally falls into two categories. **Pushing-Based Navigation** focuses on reaching a goal location by pushing obstacles aside. Recent work aims to improve adaptability in these scenarios; for instance, [15] leveraged online sensor feedback for adaptive pushing, while [58] improved maneuverability by learning non-axis-aligned pushing. Other works focus on efficiency, minimizing the total force [50] or number of pushes [33] required to traverse a crowded space. Beyond terrestrial robots, [9, 61] studied autonomous surface vehicle (ASV) navigating ice-covered waters, proposing planners that reduce ice pushing.

In contrast, **Pushing-based Manipulation** emphasizes the precise rearrangement of objects. This requires a deeper understanding of contact physics. For example, [41] studied the maintenance of stiff robot-object contacts to ensure stable pushing, and in a subsequent work [40], achieved stable pushing while the object is sliding. Others focus on the policy level to coordinate actions to push all objects toward some configurations. In [51], the Spatial Action Map (SAM) is proposed as a novel action representation suitable for multi-object pushing, where a mobile robot gathers scattered objects into a receptacle. In [56], a full-stack framework is developed for robots to push and collect trolleys in public areas autonomously.

The Control Challenge

Regardless of the specific task, nonprehensile manipulation is inherently challenging due to its underactuated nature and non-linear dynamics [37]. The robot cannot directly control the object's position, but must influence it through frictional contacts. Classical methods like Model Predictive Control (MPC) [3] attempt to model these forces explicitly, but they often struggle with the computational cost of reasoning over hybrid contact modes (sticking vs. sliding). This difficulty has driven the shift toward learning-based policies.

2.1.2 Mobile Robot Benchmarks

To measure progress in these tasks, rigorous benchmarking is essential. However, traditional mobile robot benchmarks focus primarily on collision-free navigation. BARN [30] evaluates navigation in cluttered environments, while [57] and [20] extend beyond static environments by considering both static and dynamic obstacles. Nonetheless, these benchmarks primarily assess performance in ways that discourage environmental interaction, classifying robot-environment contact as failures rather than controlled and intended interactions.

The Interactive Gibson Benchmark [55] is the closest existing framework to our work, introducing environments where robots can interact with objects. However, significant gaps remain. First, Gibson focuses *only* on interactive navigation, whereas ours supports both pushing-based navigation *and* manipulation tasks. Second, Gibson emphasizes photo-realistic rendering, which is valuable for vision research but is often heavy for rapid experimentation. In contrast, our benchmark spans both lightweight 2D simulation for pilot studies and 3D simulation for advanced evaluation. Third, Gibson includes only generic RL baselines, while we provide both transferable RL and task-specific baselines for stronger domain performances.

Evaluation metrics in existing mobile robot benchmarks [30, 57] primarily focus on navigation success rates, collision count, and path efficiency. These metrics are insufficient for pushing-based tasks where controlled interactions are required rather than avoided. Gibson [55] introduced metrics for interactive navigation that capture both efficiency and environment disturbance from the interaction. These metrics are limited to interactive navigation and are not suitable for pushing-based manipulation where modifying the environment is part of the task (i.e., box delivery). In contrast, our benchmark introduces separate evaluation metrics for pushing-based navigation and manipulation tasks, ensuring that interaction quality and efficiency are appropriately measured in both classes.

2.1.3 Reinforcement Learning

Reinforcement Learning (RL) has become the dominant paradigm for solving high-dimensional control problems in robotics. However, the design of the action spaces is critical to success.

Continuous vs. Discrete Control

Continuous control policies (e.g., PPO, SAC) typically operate directly on low-level motor commands, such as linear and angular velocities [38, 10]. While expressive, these policies

force the agent to simultaneously learn high-level planning and low-level dynamics. This dual burden often leads to sample inefficiency. Conversely, discretized motion primitives improve learning stability [59], but they introduce their own challenges: meaningful reward signals may arise only after executing long sequences of motion primitives, making exploration difficult in long-horizon tasks [51].

Spatial Action Maps

To bridge this gap, Spatial Action Maps (SAMs) [51] treat the action space as a pixel-wise grid of spatial subgoals. This decouples high-level decision-making from low-level motion control: a Deep Q-Network (DQN) selects a pixel in the image as a target, and a low-level controller executes the motion. While SAMs have shown success in pushing [51], blowing [53], and multi-agent manipulation [52], they traditionally rely on hard-coded or shortest-path low-level controllers, which fail to account for complex obstacles or object interactions during the movement.

Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) decomposes complex tasks into a hierarchy of subgoals or options [39, 47]. This reduces the effective planning horizon and improves sample efficiency, particularly in sparse-reward settings.

In continuous control, methods like HIRO [28] propose training a high-level policy to output goal states for a low-level RL controller, utilizing off-policy corrections to maintain sample efficiency. Other hybrid architectures, such as PRM-RL [13], combine learning with classical methods by using sampling-based planners to generate high-level roadmaps while relying on RL for robust low-level execution. Furthermore, skill-based methods seek to simplify long-horizon planning by extracting reusable behaviours from offline data; for instance, SPiRL [31] learns a generative prior over skills to guide high-level RL exploration.

The SAM framework can be viewed as a form of implicit hierarchy: the high-level agent chooses a spatial subgoal, and a low-level controller executes it. HeRD bridges these paradigms. Rather than training two RL policies concurrently or pairing a classical planner with low-level RL, we make this hierarchy explicit by pairing a goal-predicting DDQN policy with a denoising diffusion probabilistic model (DDPM) [35, 17] for path generation. DDPMs are generative diffusion models with recent applications in robotics that sample structured action sequences via iterative denoising [19] and can be conditioned on observations [8]. This mirrors broader HRL frameworks that decouple strategic planning from motor control.

2.1.4 Generative Diffusion Models

To address the limitations of hard-coded low-level controllers, we look to the emerging field of Generative Diffusion Models. Originally designed for image synthesis, Denoising Diffusion Probabilistic Models (DDPMs) [17] have recently been adapted for robotics to generate complex, multimodal trajectories.

Trajectory Generation

In the context of robotics, a trajectory is generated by iteratively denoising a sequence of random Gaussian noise, conditioned on the robot's observation [19, 8]. This allows the policy to capture human-like intuition and smooth, feasible motions that are difficult to define with cost functions. Unlike RL, which converges to a single optimal action for a given state, diffusion policies model multimodal distributions, allowing them to represent diverse valid strategies.

Conditioning and Guidance

A central challenge in diffusion-based control is guiding the generation process toward outcomes that are not only high-quality but also task-relevant and physically feasible. While unconditional diffusion models capture the underlying data distribution, they cannot inherently adapt to specific runtime goals or environment constraints. To address this, recent works have focused on three mechanisms for guidance: classifier-based guidance, goal inpainting, and environmental conditioning.

Early work in guided diffusion utilized *classifier guidance*, where an auxiliary classifier provides gradients to shift the denoising process toward high-value regions [11]. In robotics, this concept has been adapted to optimize trajectory smoothness or safety. For instance, [22] and [43] introduce hard and soft constraints directly into the denoising loop, enforcing kinematic limits or obstacle avoidance by correcting the noisy trajectory at each step to ensure it remains physically feasible.

For goal-directed tasks, *inpainting* is a dominant technique adapted from image restoration. As demonstrated in Diffuser [19], portions of the trajectory (typically the final state) can be fixed to a desired value, while the diffusion model generates the remaining sections of the path. This method can guarantee that a trajectory will terminate at the desired final state, but it does not inherently account for obstacles between the start and end unless the underlying model is conditioned on the scene.

To generate trajectories that are valid within a specific environment, the diffusion model must be conditioned on the observation space. Recent approaches leverage architectures like Cross-Attention [46] or Feature-wise Linear Modulation (FiLM) [29] to modulate the denoising network based on visual or LiDAR inputs [8, 36]. For example, Diffusion Policy [8] conditions the generation process on a history of visual observations, ensuring the robot's motion aligns with the physical affordances of the scene. This environmental conditioning is necessary for mobile manipulation, where a valid path in open space may be invalid in a cluttered corridor.

In HeRD, we leverage goal inpainting to anchor the trajectory to the subgoals selected by our high-level RL planner. Simultaneously, we condition the diffusion model with the current state using FiLM embedding. After the path is generated, we apply constraints to ensure feasibility in the environment. This strategy ensures that generated paths are both goal-directed and physically feasible.

Hierarchical Diffusion

Hierarchical variants of diffusion models have been proposed to tackle long-horizon or multi-stage tasks. HDMI [24] and Hierarchical Diffuser [7] both adopt a two-level diffusion architecture: a high-level model generates subgoals or intermediate keyframes, and a low-level diffusion model produces the full trajectory segments that reach them. HDP [26] follows a similar design in manipulation settings, predicting end-effector subgoals with a high-level diffusion model and generating transitions using a low-level trajectory diffuser.

Other approaches, such as [48, 54], combine high-level diffusion models for subgoal generation with low-level controllers trained via reinforcement learning, forming hybrid architectures that integrate generative planning with policy-based execution.

Our work, HeRD, proposes a complementary hierarchical structure. We argue that RL is superior for high-level planning due to its ability to optimize value over long horizons, while Diffusion is superior for low-level execution due to its ability to generate smooth, contact-aware motions. Therefore, we propose a hybrid hierarchy: a value-based SAM planner guiding a goal-conditioned diffusion low-level controller.

2.2 Background

2.2.1 Deep Reinforcement Learning

Markov Decision Process

We formulate the problem of nonprehensile mobile manipulation as a Markov Decision Process (MDP). Formally, an MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where: $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions available to the agent, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function, where $P(s'|s, a)$ denotes the probability of transitioning to state s' given state s and action a, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, representing the expected immediate reward received for executing action a in state s, and $\gamma \in [0, 1)$ is the discount factor that balances immediate and future rewards.

The framework relies on the Markov assumption, which implies that the future state depends only on the current state and action, independent of the history of prior events:

$$\mathbb{P}(s_{t+1}|s_t, a_t, \ldots, s_0, a_0) = P(s_{t+1}|s_t, a_t). \tag{2.1}$$

A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ defines the agent's behaviour, mapping states to a probability distribution over actions. The objective in an MDP is to find an optimal policy π^* that maximizes the expected discounted reward over an infinite horizon:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \tag{2.2}$$

To solve this optimization problem, we introduce the concept of value functions. The action-value function, $Q^{\pi}(s, a)$, represents the expected return starting from state s, taking action a, and following policy π thereafter:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right]. \tag{2.3}$$

The optimal action-value function, $Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$, satisfies the Bellman Optimality Equation, a recursive relationship that decomposes the value into the immediate reward and the discounted value of the next state:

$$Q^*(s, a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[R(s, a) + \gamma \max_{a'} Q^*(s', a') \right]. \tag{2.4}$$

If $Q^*(s,a)$ is known, the optimal policy is simply to select the action with the highest value: $\pi^*(s) = \arg\max_a Q^*(s,a)$.

If the transition dynamics P and reward function R are fully known, this equation can be solved using dynamic programming methods such as Value Iteration.

Reinforcement Learning

In the context of mobile manipulation, the transition dynamics P, which can involve complex contact physics and friction, are often unknown or computationally intractable to model analytically. Reinforcement Learning (RL) addresses this challenge by learning the optimal policy through interaction with the environment, rather than by solving the MDP analytically. With RL, both the transition and reward models are assumed to be unknown.

The goal of the RL agent is to estimate the optimal action-values $Q^*(s,a)$ from observed transitions (s,a,r,s'). Q-Learning [49] is a widely used off-policy algorithm that iteratively updates these estimates based on the temporal difference (TD) error:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right), \tag{2.5}$$

where α is the learning rate. As the number of samples approaches infinity, $Q(s,a)$ converges to $Q^*(s,a)$.

Deep Q-Networks (DQN)

For environments with high-dimensional state spaces, such as images or occupancy grids, maintaining a tabular representation of $Q(s,a)$ is intractable. Deep Q-Networks (DQN) [27] approximate the Q-function using a neural network with parameters θ, denoted as $Q_\theta(s,a)$. The network is trained by minimizing the loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(y - Q_\theta(s,a) \right)^2 \right], \tag{2.6}$$

where $\mathcal{D}$ is an experience replay buffer storing past transitions, and the target y is given by:

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s',a'). \tag{2.7}$$

Here, θ^- represents the parameters of a *target network*, a frozen copy of the online network that is updated periodically to stabilize training.

Double Deep Q-Networks (DDQN)

A known limitation of DQN is the maximization bias in the target calculation. The operator $\max_{a'} Q_{\theta^-}(s', a')$ tends to overestimate value estimates because the same network is used to both select and evaluate the action. This overestimation can propagate and lead to suboptimal policies.

Double DQN (DDQN) [45] mitigates this by decoupling the action selection from the action evaluation. The online network θ is used to select the greedy action, while the target network θ^- is used to evaluate its value. The modified target becomes:

$$y_{DDQN} = r + \gamma Q_{\theta^-}\left(s', \arg\max_{a'} Q_\theta(s', a')\right). \tag{2.8}$$

This formulation significantly reduces overestimation and is the standard backbone for value-based spatial planning methods, including the Spatial Action Maps utilised in this thesis.

2.2.2 Diffusion Models

Generative Models

The fundamental goal of generative modelling is to learn the underlying probability distribution $p_{data}(\mathbf{x})$ of a dataset $\mathbf{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, such that new samples $\mathbf{x} \sim p_{data}(\mathbf{x})$ can be synthesized. While traditional methods attempt to model this distribution directly or implicitly, Score-Based Generative Models (SGMs) take a different approach, modelling the gradient of the log-probability density, known as the score function:

$$\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}). \tag{2.9}$$

Intuitively, the score function defines a vector field pointing towards regions of high data density. By following this vector field, one can transport a random noise sample from its prior distribution to the data distribution.

Stochastic Differential Equations (SDEs)

Diffusion models generalize this concept by defining a continuous-time process that gradually destroys data structure. We consider a forward process indexed by time $t \in [0, T]$,

where $\mathbf{x}_0 \sim p_{data}$ and $\mathbf{x}_T \sim p_{prior}$. Typically, a Gaussian distribution is selected as the prior distribution: $p_{prior} = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

This forward process is modelled as the solution to a generic SDE:

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w}, \tag{2.10}$$

where $\mathbf{f}(\mathbf{x}, t)$ is the drift coefficient, representing the deterministic evolution of the state; $g(t)$ is the diffusion coefficient, a scalar function controlling the variance of the injected noise; and $\mathbf{w}$ represents a standard Wiener process. As t advances from 0 to T, the SDE transforms the complex data distribution into a tractable prior distribution.

Reverse SDE

To generate data, we must reverse this process: starting from noise $\mathbf{x}_T \sim p_{prior}$ and simulating backward in time to $\mathbf{x}_0 \sim p_{data}$. Anderson [2] proved that for any forward SDE, there exists a corresponding reverse-time SDE:

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] dt + g(t)d\bar{\mathbf{w}}, \tag{2.11}$$

where dt represents a negative time step and $\bar{\mathbf{w}}$ is a standard Wiener process flowing backward in time.

Simulating this reverse process requires knowledge of the score function $\nabla_{\mathbf{x}} \log p_t(x)$ at every time step t. Since the true density $p_t(x)$ is unknown, we train a time-dependent neural network $s_\theta(\mathbf{x}, t)$ to approximate it.

Denoising Diffusion Probabilistic Models (DDPM)

In practice, we often solve a discrete-time formulation of the SDE known as a Denoising Diffusion Probabilistic Model (DDPM) [17]. This process discretizes the continuous time interval into T integer steps (e.g., $t \in \{1, \ldots, T\}$), defined by a variance schedule $\beta_1, \ldots, \beta_T$, where $\beta_t \in (0, 1)$ controls the amount of noise added at each step.

The forward diffusion process $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ adds Gaussian noise according to:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t;\ \sqrt{1-\beta_t}\mathbf{x}_{t-1},\ \beta_t\mathbf{I}). \tag{2.12}$$

By defining $\alpha_t = 1 - \beta_t$ and the cumulative product $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, we can marginalize over the intermediate steps to sample $\mathbf{x}_t$ directly from the original data $\mathbf{x}_0$:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \text{where } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{2.13}$$

Intuitively, this shows that every noisy sample is simply a weighted combination of the original sample and pure noise, with the proportions coming from the noise scheduler at timestep t. This property allows us to efficiently generate training samples for any timestep t without simulating the entire chain.

Learning Objective: Instead of learning the score function explicitly, DDPM trains a network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ to predict the noise component $\boldsymbol{\epsilon}$ from the noisy input $\mathbf{x}_t$. It can be shown that this is mathematically equivalent to learning the score, up to a scaling constant: $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \propto -\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$. The loss function is the mean squared error between the true noise and the predicted noise:

$$\mathcal{L}(\theta) = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)||^2\right] = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)||^2\right]. \tag{2.14}$$

Sampling: Once trained, we generate new data by reversing the diffusion process. Starting from pure noise $\mathbf{x}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we iteratively remove noise using the predicted $\boldsymbol{\epsilon}_\theta$:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}, \tag{2.15}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ injects stochasticity (for $t > 1$) and σ_t is a variance term derived from β_t. This iterative denoising gradually refines the noisy signal into a coherent sample.

Conditional Generation and Guidance

Standard DDPMs model the marginal distribution $p(\mathbf{x})$. However, in robot control, we are rarely interested in generating random actions; we require actions that satisfy specific conditions $\mathbf{c}$, such as a goal location or a sensory observation. We therefore aim to model the conditional distribution $p(\mathbf{x}|\mathbf{c})$

While simply concatenating the condition $\mathbf{c}$ to the network input allows the model to learn conditional dependence, standard conditional training often fails to adhere to the condition, prioritizing diversity over fidelity [11, 18] To enforce stronger adherence to the condition, guidance mechanisms are employed during the sampling process.

Classifier Guidance: Early approaches [11] utilized a separate classifier $p_\phi(\mathbf{c}|\mathbf{x}_t)$ trained on noisy data. The gradients of this classifier, $\nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{c}|\mathbf{x}_t)$, are added to the score estimate during sampling, effectively pushing the trajectory toward regions where the condition is satisfied. However, training a robust noise-aware classifier is computationally expensive and complicates the training pipeline.

Classifier-Free Guidance (CFG): To avoid the need for an auxiliary model, Ho and Salimans [18] introduced Classifier-Free Guidance. This method modifies the training procedure to jointly train both a conditional model $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{c})$ and an unconditional model $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \emptyset)$ within the same neural network. This is achieved by randomly dropping the condition $\mathbf{c}$ by replacing it with a null token $\emptyset$ with a fixed probability during training.

During inference, the noise prediction is formed by extrapolating the difference between the conditional and unconditional estimates:

$$\tilde{\boldsymbol{\epsilon}}_t = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \emptyset) + w \cdot (\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{c}) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \emptyset)) \,, \tag{2.16}$$

where $w \geq 1$ is the guidance scale. Intuitively, this pushes the generated sample away from the unconditional distribution and strictly toward the direction of the condition $\mathbf{c}$. A higher w yields samples that align more closely with the specified goal, but may reduce the diversity of the generated trajectories.

Chapter 3

Bench-Push: Benchmarking Pushing-based Navigation and Manipulation Tasks for Mobile Robots

3.1 Bench-Push

Bench-Push consists of three main components – (i) environment, (ii) policy, and (iii) evaluation metrics. In this section, we detail the specifics of these environments, explain the implementation of custom policies in Bench-Push and introduce the metrics used to evaluate policies. First, we clarify some of the terminology used in this section.

Environment: An environment is a simulated world where the robotic agent exists, integrated with the Gymnasium [44] interface. Each environment is simulated in both 3D through MuJoCo [42] and 2D through Pymunk [5], and has multiple variations with increasing complexity (see Fig. 1.1). Environments can be run independently of a training session, and the robot can be manually operated by a user.

Task: We refer to a task as the robot's objective in a given environment. Specifically, we consider two types of tasks:

- *Navigation Task*, where the robot must reach a goal region while inevitably pushing path-blocking obstacles.

Table 3.1: Interactive pushing-based tasks included in Bench-Push.

Environments	Task Class	Variations	Baselines Evaluated
Maze	Navigation-centric	maze complexity, obstacle count	SAC [14], PPO [34], RRT [23]
Ship-Ice	Navigation-centric	ice concentrations	SAC [14], PPO [34], ASV planning [9, 61]
Box-Delivery	Manipulation-centric	w/o static obstacles, box count	SAC [14], PPO [34], SAM [51]
Area-Clearing	Manipulation-centric	w/o static obstacles, box count	SAC [14], PPO [34], SAM [51], GTSP (see project repo)

Table 3.2: Rewards, actions, and observations for each task.

Environments	Action	Observation	Rewards
Maze	Angular vel.	occupancy + footprint + goal DT	collision + distance decrement + terminal
Ship-Ice	Angular vel.	occupancy + footprint + goal DT + heading encoding	collision + heading + terminal
Box-Delivery	Heading	occupancy + footprint + egocentric DT + goal DT	collision + box completion + box displacement
Area-Clearing	Heading	occupancy + footprint + egocentric DT + goal DT	collision + box completion + box displacement

- *Manipulation Task*, where the robot must push objects in the environment to some desired configurations.

Policy: A policy is a set of rules used to control the robot in an environment. It takes as input an observation of the environment and outputs a robot action. In Bench-Push, we include several baseline policies as reference implementations for users, which are listed in Table 3.1.

Metric: A metric is a score used to evaluate the effectiveness of a policy in completing a task in a given environment. In Section 3.1.3, we introduce novel metrics included in Bench-Push to evaluate and compare policies.

3.1.1 Bench-Push Environments

We now introduce the environments and tasks in Bench-Push. For each environment, we provide the rationale for choosing the environment and the specifics of the setup. We also present the actions, observations, and rewards for the robotic agents, summarized in Table 3.2. Further demonstrations can also be found in the supplemental video.

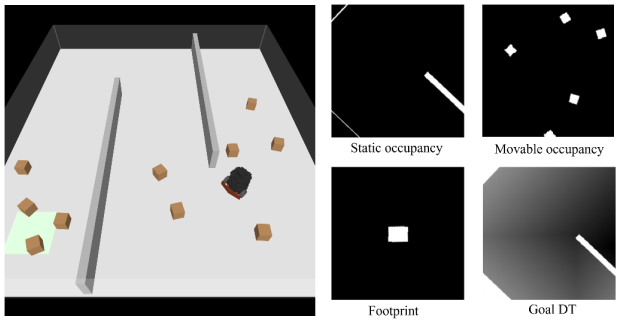


Figure 3.1: The *Maze* environment (left) and an example egocentric observation for *Maze* (right).

Navigating maze with movable obstacles *(Maze)*

Existing work in pushing-based navigation has frequently adopted maze-like environments with bounded rectangular spaces and walls [33, 50, 58]. These environments capture common structural layouts of many real-world indoor settings, such as offices and hospitals. Motivated by this, we introduce *Maze* environment, which features a static maze structure with randomly initialized obstacles (Fig. 1.1 top-left). The mobile robot agent is a TurtleBot3 Burger, and its task is to navigate from a starting position to a goal location while minimizing path length and obstacle collisions. Variations of the environment include different maze structures, obstacle sizes, and obstacle locations and densities.

The robot in this environment is configured to move with a constant forward speed and receives an angular velocity as action input from the policy. At each timestep, the robot makes a local egocentric observation consisting of four channels that correspond to (i) static obstacle occupancy, (ii) movable obstacle occupancy, (iii) robot footprint, and (iv) a distance transform (DT) originating from the goal (goal DT). Fig. 3.1 shows an example observation. The robot receives rewards for moving closer to the goal, penalties for colliding with obstacles, and a terminal reward when reaching the goal.

Autonomous ship navigation in icy waters (Ship-Ice)

To ensure broader applicability, we extend beyond structured indoor settings represented by *Maze* to field robotics scenarios. A notable example in recent research is Autonomous Surface Vehicle (ASV) navigation through ice-covered waters [9, 61]. This domain presents challenges distinct from *Maze*, such as irregular obstacles, dense clustering, and fluid dynamics. For this reason, we introduce the *Ship-Ice* environment, which simulates an autonomous ship navigating in a channel of ice-covered waters (Fig. 1.1 bottom-left). The

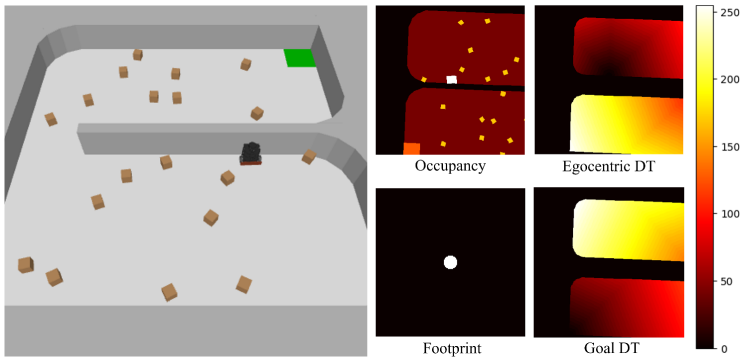


Figure 3.2: The *Box-Delivery* environment (left) and an example observation for *Box-Delivery* (right).

ship's task is to reach a horizontal goal line ahead of the ship while minimizing collisions with broken ice floes in the channel [9]. The ice floes are convex polygons with ice concentrations varying between 0% and 50%.

Ship-Ice models ship navigation at real-world scale. We include pre-generated ice fields provided by marine study experts from the National Research Council Canada (NRC), while also allowing users to generate ice fields on the fly. Because MuJoCo does not natively support fluid dynamics, we approximate them on the xy plane using combined linear and quadratic drag forces acting on all bodies. Relevant parameters such as ship size, mass, drag coefficients, and ice sizes are based on [9].

Similar to the *Maze* environment, the ship moves with a constant forward speed and receives an angular velocity as action input. The ship makes observations similar to *Maze* but with an additional channel that encodes the heading of the ship as a single-pixel-width line. This channel facilitates fine-grained movement around clusters of ice floes. The ship's reward function combines penalties for collisions with ice floes, a heading reward to encourage approaching the goal line, and a terminal reward for reaching the goal.

Delivering boxes to a receptacle (Box-Delivery)

Existing work in pushing-based manipulation has primarily focused on either contact-level control of pushing actions [41, 40], or task-level policies for moving all objects toward target configurations [51, 56], with some positioned in between [4]. The common ground across all these methods centers on requiring the robot to push objects in a desired direction or trajectory toward a goal. Inspired by this observation, we introduce *Box-Delivery* here and

Area-Clearing in the following section (Sec. 3.1.1). The *Box-Delivery* environment consists of a set of movable boxes and a designated *receptacle* (Fig. 1.1 top-right). Similar to *Maze*, the robotic agent is a TurtleBot3 Burger. The robot is tasked with delivering all boxes to the receptacle using a non-prehensile manipulator (e.g., front bumper) to push the boxes. The boxes and robot starting location are randomly generated within an environment. Further variations are possible by including static obstacles (e.g., columns) and changing the number and size of movable objects.

At each step, the robot receives a heading as its action input, in which case the robot travels for a fixed distance along that heading. The robot's observation is egocentric and includes (i) occupancy of the environment with static and movable objects encoded in different values, (ii) robot footprint, (iii) DT originating from the robot (egocentric DT), and (iv) DT originating from the receptacle (goal DT). Fig. 3.2 illustrates the environment and corresponding robot observation. The robot needs to push movable objects (brown boxes) toward the receptacle (light green square). The robot receives rewards for moving boxes toward the receptacle or delivering a box, and penalties when colliding with static obstacles or moving boxes away from the receptacle.

Clearing boxes from an area (Area-Clearing)

The *Area-Clearing* environment consists of a set of movable boxes and a *clearance area* (Fig. 1.1 bottom-right). The robotic agent is a TurtleBot3 Burger. The task of the robot is to remove the boxes from within this clearance area, with no constraints on the final positions of the cleared boxes. Variations of this environment can be created by changing the number of objects to be removed, the number of static obstacles, and workspace constraints (e.g., walls blocking the clearance area close to the boundary).

The robot actions and observations in *Area-Clearing* are the same as *Box-Delivery*, except that the goal distance transform is now computed with respect to the open boundary of the clearance area. The reward function is also similar to *Box-Delivery*, where the robot receives a reward when moving any object inside the clearance area closer to the boundary, and when pushing a box out of the boundary.

3.1.2 Bench-Push Implementation

Bench-Push is designed with a modular architecture to support flexible customization and ease of use. The benchmark has been designed with a one-line install to allow researchers

to get up and running quickly. We highlight a few core aspects of Bench-Push's implementation, designed to support diverse research needs and to make it convenient for users to integrate custom policies of their choice for the pushing-based tasks described above.

Additional Features and Configurability

In addition to the action spaces presented in Table 3.2, Bench-Push also allows direct wheel velocity commands for the TurtleBot3 Burger in *Maze*, *Box-Delivery*, and *Area-Clearing*, allowing for lower-level, more fine-grained control. Further, the robot can be equipped with three interchangeable front bumpers (inward-curved collector, straight-edge pusher, outward-curved navigation bumper). These bumpers potentially serve different tasks and can induce different interaction effects. Finally, in real-world indoor environments, robots often encounter both objects that rest flat on the ground (e.g., couches, tables, bags) and wheeled objects that move more easily (e.g., office chairs). To conceptually capture this challenge, the *Maze*, *Box-Delivery*, and *Area-Clearing* environments can have both unwheeled and wheeled boxes that exhibit distinct interaction dynamics, as shown in Fig. 1.1 top-left with dark and light brown boxes.

Simplified 2D Simulations

All environments detailed in Sec. 3.1.1 are supported with a simplified simulation through Pymunk 2D [5]. The 2D versions provide lightweight simulations where all bodies and interactions are modeled through convex planar polygons, making them ideal for rapid pilot studies. The 2D version of *Ship-Ice* uses a model ship scale similar to the setup in [9] with no fluid dynamics. Detailed comparisons between the 2D and 3D environments can be found in the project repository.

Gymnasium Integration

We integrate all environments within the Gymnasium [44] interface to standardize policy development and evaluation. The implementation supports a wide range of environment parameters through either script-level configuration or command-line arguments.

Extensible Policy Class

Bench-Push provides a standardized, extensible policy template to simplify the incorporation of new algorithms. This template follows a plug-and-play philosophy: a user only

needs to implement the required APIs within a policy class. Bench-Push then handles inference, evaluation, and logging processes.

By way of illustration, Bench-Push comes with reference implementations for well-established baselines, as described in Sec. 3.2. These examples demonstrate both how to integrate popular Reinforcement Learning algorithms from Stable Baselines 3 [32] and how to incorporate specialized or state-of-the-art approaches. Additional documentation and usage examples, including configuration files and training scripts, are provided in the project repository at https://github.com/IvanIZ/BenchPush.

3.1.3 Metrics

We now provide metrics to evaluate policies for the above tasks. These metrics are inspired by [55] and aim to capture the task efficiency and interaction efforts of the robot. Consider a robot of mass m_0 that traverses a path of length l_0 as determined by the policy. Let $O = \{o_1, \ldots, o_K\}$ be the set of K movable objects in the environment, which may have moved due to the robot's motion. Let the mass and path length for each object $i \in \{1, \ldots, K\}$ be m_i and l_i, respectively. Since navigation and manipulation are intrinsically different tasks, Bench-Push provides a distinct set of metrics for each class.

Navigation Metrics

We provide two metrics to evaluate navigation policies: task efficiency score E_{nav} and interactive effort score I_{nav}. The robot's efficiency in completing a navigation task depends on the length of its path, l_0. Let $\mathbb{1}_{\text{success}}$ be an indicator function denoting task success, which takes on a value of 1 if the robot has successfully reached the goal, and 0 otherwise. Further, let l_0^* be the shortest path distance from the robot's start position to the goal that avoids only static obstacles in the environment. The efficiency score $E_{\text{nav}} \in [0, 1]$ is then computed as

$$E_{\text{nav}} = \mathbb{1}_{\text{success}} \frac{l_0^*}{l_0}. \tag{3.1}$$

From Eq. 3.1, we see that $E_{\text{nav}} = 1$ if the robot reaches the goal with the shortest possible path length.

The interaction effort score I_{nav} quantifies the work done by the robot to push objects relative to the total work done. It is defined as the ratio of the work needed to move only the robot along its path to the total work expended moving both the robot and any movable objects it interacts with. Assuming a constant coefficient of kinetic friction μ

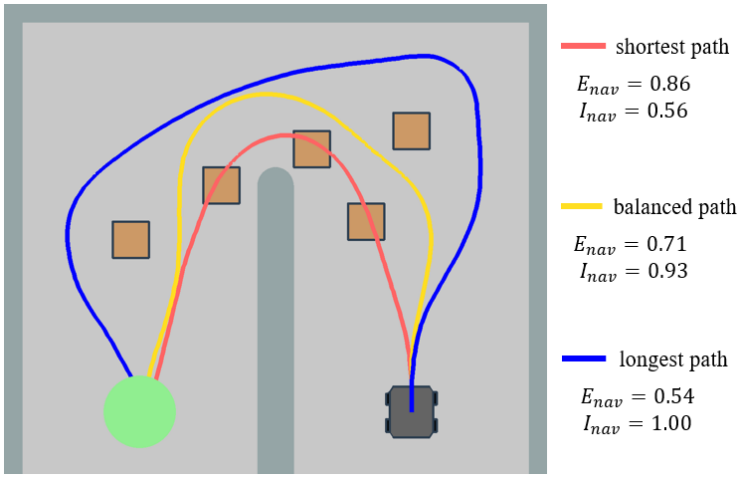


Figure 3.3: Three teleoperated *Maze* paths and their performance. While the shortest path (red) has the highest efficiency score $E_{\text{nav}} = 0.86$, excessive collisions with the movable objects degrade the effort score $I_{\text{nav}} = 0.56$. In contrast, while the longest path (blue) is collision-free $I_{\text{nav}} = 1.00$, efficiency is largely compromised $E_{\text{nav}} = 0.54$. A balanced path (yellow) potentially offers the best trade-off.

across the environment, the work done to move object i a distance l_i is $\mu m_i g l_i$. The score $I_{\text{nav}} \in [0, 1]$ is then the ratio of the robot's work to the total work:

$$I_{\text{nav}} = \frac{\mu m_0 g l_0}{\sum_{i=0}^{K} \mu m_i g l_i} = \frac{m_0 l_0}{\sum_{i=0}^{K} m_i l_i}. \tag{3.2}$$

Here, $I_{\text{nav}} = 1$ when the robot reaches the goal without moving any objects, as the interaction terms $m_i l_i$ for $i \in \{1, \ldots, K\}$ in the denominator are all zero. I_{nav} penalizes extensive interactions by the robot, such as pushing objects heavier than the robot or moving them over longer distances (increases denominator, lowers score). If the robot is significantly heavier than the objects, the work required to move the objects becomes negligible compared to the work to move the robot, which is reflected by I_{nav} as it approaches 1.

To evaluate the performance of a navigation policy, one must use both metrics to capture the trade-off between minimizing travel distance (E_{nav}) and avoiding unnecessary interactions (I_{nav}). Fig. 3.3 illustrates how E_{nav} and I_{nav} capture this unique trade-off in navigation with pushing. An ideal policy will look to maximize both scores.

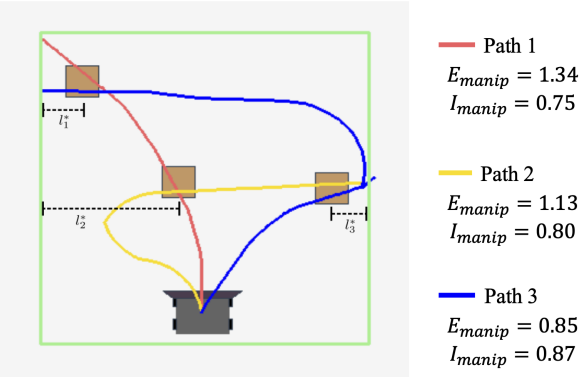


Figure 3.4: Three teleoperated paths for the area clearing task, where all boxes must be removed from an area (green square). Each path has a task success score $S_{\text{manip}} = 2/3$. While path 1 (red) has the highest efficiency score, its effort score is low as it moves a box longer than its shortest distance to the goal (dotted lines of length l_i^*). In contrast, path 3 (blue) achieves the best effort score by pushing as little as possible, but produces a long path and low efficiency. Path 2 (yellow) potentially offers the best trade-off.

Manipulation Metrics

The metrics for manipulation tasks, unlike the navigation metrics, must incorporate the minimum path length and effort required by the robot to manipulate objects towards completing its task. We also look to compare robot paths that may achieve *partial completion* of tasks, e.g., push two out of four boxes into the receptacle. To capture this effectively, we propose three metrics: (i) task success score S_{manip}, (ii) efficiency score E_{manip}, and (iii) interaction effort score I_{manip}.

To simplify our explanation, we consider that each of the K movable objects must be manipulated to a desired set of configurations (delivered or cleared). As such, the robot must complete a series of smaller manipulation *sub-tasks* corresponding to each box to complete the overall task. Let $\mathbb{1}^i_{\text{success}}$ be an indicator function that denotes whether the sub-task corresponding to the i^{th} movable object has been completed. Specifically, $\mathbb{1}^i_{\text{success}} = 1$ if object $o_i \in O$ has been successfully pushed into the receptacle for *Box-Delivery* (or removed from the clearance area for *Area-Clearing*), and $\mathbb{1}^i_{\text{success}} = 0$ otherwise. Let $K' \leq K$ be the number of sub-tasks that have been completed, i.e., $K' = \sum_{i=1}^{K} \mathbb{1}^i_{\text{success}}$. The task success score S_{manip} is given as

$$S_{\text{manip}} = \frac{K'}{K}. \tag{3.3}$$

Given that K' sub-tasks were completed, let $O' \subseteq O$ be the corresponding set of completed objects, where $|O'| = K'$. Let $L^*(O')$ be (a lower bound on) the minimum path length required to complete the successful sub-tasks. For example, if the robot completes 2 out of 4 boxes, $L^*(O')$ is the minimum path length the robot must take to move the two boxes to the receptacle for *Box-Delivery* (or removed from the clearance area for *Area-Clearing*). The efficiency score for the manipulation task E_{manip} is given by

$$E_{\text{manip}} = \frac{L^*(O')}{l_0}. \tag{3.4}$$

Intuitively, E_{manip} quantifies the robot's executed path length in relation to the length of an idealized shortest path to complete the K' sub-tasks. In Bench-Push, we compute $L^*(O')$ using a minimum spanning tree (MST) of a graph G that we construct as follows. The vertex set includes K' vertices for the initial positions of the completed movable objects in O', another K' vertices for the closest point to the goal (receptacle or clearance area boundary) from each object, and a vertex for the robot. Edges are added between all pairs of objects, from the robot to each object, and from each object to its nearest goal, with the edge weights being the shortest distance between the two points. The MST of this graph is not a strict lower bound on the total path length, and as such, it is not guaranteed that $E_{\text{manip}} \in [0, 1]$. However, we find that this is a practical bound, as doubling the edges of the MST provides a path that can complete the sub-tasks (ignoring object-to-object interactions). Note that a true lower bound can be obtained by computing the MST of a similar graph with a single goal vertex, i.e., where the robot stacks all boxes and goes to the goal once. In practice, however, we found this bound to be overly optimistic, typically yielding E_{manip} values closer to 0 and providing little insight into relative policy performance.

Using both S_{manip} and E_{manip}, we determine how many sub-tasks the robot has completed, and the robot's efficiency in completing them. The interaction effort score I_{manip} for manipulation quantifies the robot's effort to interact with pushable objects. Unlike in navigation, pushing objects is a necessary part of the task. So, we define I_{manip} as the ratio of the minimum necessary work to complete the K' sub-tasks to the total work done. We define the minimum necessary work as the work done to move the robot along its path ($\mu g m_0 l_0$) plus the work required to push each successfully delivered (or cleared) object along its shortest path to the goal ($\mu g m_i l_i^*$). The total work done is the same as the denominator of I_{nav}, i.e., the work done to move the robot and push all objects. The resulting score $I_{\text{manip}} \in [0, 1]$ is given as

$$I_{\text{manip}} = \frac{m_0 l_0 + \sum_{i=1}^{K} \mathbb{1}^i_{\text{success}} m_i l_i^*}{\sum_{i=0}^{K} m_i l_i}. \tag{3.5}$$

Similar to the navigation metrics, one must use all manipulation metrics together to evaluate a policy's ability to complete a task (S_{manip}), minimize path length (E_{manip}), and avoid unnecessary interactions (I_{manip}). Fig. 3.4 illustrates these metrics evaluating three trajectories for the area clearing task. All trajectories achieve a task success score of 2/3, but with different trade-offs in path efficiency and interaction effort.

3.2 Evaluating Baselines using Bench-Push

We now describe the baseline algorithms available through Bench-Push. These baselines are intended to confirm the extensibility of Bench-Push and provide reference points for future studies. Further, we present physical robot experiments on a subset of tasks from Sec. 3.1.1. Specifically, we focus on one navigation task (*Maze*) and one manipulation task (*Box-Delivery*), where we evaluate sim-to-real transfer with policies trained using Bench-Push. Due to space constraints, we include evaluation results for the remaining tasks using Bench-Push simulations in the project repository.

3.2.1 Implemented Baselines

To support both general mobile robot pushing research and task-specific studies, Bench-Push includes baselines that fall into two categories: (i) reinforcement learning (RL) baselines applicable to all tasks in Section 3.1.1 and (ii) task-specific baselines. Table 3.1 summarizes the baselines included for each task. The reinforcement learning (RL) baselines include Soft Actor-Critic (SAC) [14] and Proximal Policy Optimization (PPO) [34]. These baselines were implemented using Stable Baselines 3 [32] and trained across all tasks. To handle image observations, both baselines employ a customized ResNet18 [16] backbone with the last linear layer removed as the high-dimensional observation encoder, followed by a 3-layer MLP to output actions or value estimates.

In addition to the RL baselines, Bench-Push includes specialized policies for some of the tasks, of which we highlight two policies in this paper. For *Maze*, we implemented a rapidly-exploring random tree (RRT) planner based on [23], which drives the robot through the free space among pushable obstacles, albeit without considering the nature of the interactions. Secondly, for *Box-Delivery* and *Area-Clearing*, we adopt the Spatial Action Map (SAM) policy [51], designed for multi-object pushing. The SAM policy learns to output a series of waypoints in the environment that enable the robot to complete a manipulation-centric task. To implement SAM, we configured the *Box-Delivery* and *Area-Clearing* environments

to receive waypoints in the environment as action inputs. We additionally include task-specific baselines such as lattice-planning [9] and predictive-planning [61] policies for *Ship-Ice*, and a Generalized Traveling Salesman Problem (GTSP) based policy for *Area-Clearing* (details in project repository). The evaluations of these additional baselines in simulations and model weights are included in the project repository as references for future studies.

3.2.2 Physical Experiment Setup

We conducted physical robot experiments for the *Maze* and *Box-Delivery* as representative scenarios for navigation and manipulation tasks, respectively. The experiment testbed is shown in Fig. 1.3 (right), which uses a TurtleBot3 Burger. The vision system is an overhead camera that tracks the robot and detects objects at 10 Hz. The robot is required to interact with the 3D-printed boxes under different tasks.

The policies deployed on the robot are trained in Bench-Push 3D environments with similar configurations. For both *Maze* and *Box-Delivery*, we test each configuration with 20 episodes in simulation and 3 episodes in the testbed. Through the physical experiments, we seek to evaluate the effectiveness of transferring Bench-Push-trained policies to real hardware. Specifically, we examine whether task success rates transfer from simulation to reality, and behavioral trends on efficiency and interaction effort remain consistent.

3.2.3 *Maze* Evaluations

We evaluated the performance of SAC, PPO, and RRT policies in a U-shaped maze under 3-obstacle, 6-obstacle, and 10-obstacle settings, both in 3D simulation and in the physical testbed. The robot starts at one end of the maze and must reach the goal region located on the other end.

Table 3.3 summarizes the evaluation results. Observe that results from the physical testbed overall mirror those from Bench-Push simulations. In particular, interaction scores for all baselines exhibit downward trends with added obstacles, suggesting that environments with more clutter are increasingly difficult for the policies to navigate the robot. Furthermore, evaluations from both platforms indicate that PPO generally outperforms other baselines by having higher efficiency scores with comparable interaction scores, while SAC struggles the most in 10-obstacle settings.

These observations suggest that policies trained in Bench-Push transfer well to real robots, with consistent behaviors observed across simulation and physical deployment.

Table 3.3: Results for *Maze* in the physical testbed and Bench-Push simulations. ↑ indicates higher is better.

# Obs	Platform	Alg.	$E_{\text{nav}} \uparrow$	$I_{\text{nav}} \uparrow$
3-Obs	Sim	PPO	**0.809 ± 0.186**	0.985 ± 0.016
		SAC	0.730 ± 0.307	0.985 ± 0.020
		RRT	0.779 ± 0.056	**0.987 ± 0.010**
	Testbed	PPO	**0.757 ± 0.011**	**0.988 ± 0.008**
		SAC	0.708 ± 0.009	0.984 ± 0.005
		RRT	0.683 ± 0.001	0.985 ± 0.005
6-Obs	Sim	PPO	**0.853 ± 0.006**	**0.981 ± 0.018**
		SAC	0.765 ± 0.256	0.977 ± 0.018
		RRT	0.784 ± 0.051	0.979 ± 0.015
	Testbed	PPO	**0.754 ± 0.021**	0.955 ± 0.024
		SAC	0.724 ± 0.023	**0.987 ± 0.012**
		RRT	0.655 ± 0.005	0.977 ± 0.007
10-Obs	Sim	PPO	0.725 ± 0.305	0.960 ± 0.022
		SAC	0.708 ± 0.298	0.957 ± 0.023
		RRT	**0.758 ± 0.072**	**0.967 ± 0.018**
	Testbed	PPO	**0.770 ± 0.052**	**0.966 ± 0.012**
		SAC	0.618 ± 0.020	0.913 ± 0.037
		RRT	0.681 ± 0.022	0.930 ± 0.001

Further, we posit that PPO's slight edge in *Maze* mainly comes from more stable on-policy optimization under a fixed training budget. PPO updates on fresh rollouts with a clipped objective, which yields steadier improvements and more reliable exploration in long-horizon navigation.

We do note, however, that scores in the physical testbed are slightly lower than in simulation, likely reflecting sim-to-real gaps such as localization noise and control imperfections.

3.2.4 *Box-Delivery* Evaluations

We evaluated the performance of SAC, PPO, and SAM in *Box-Delivery* settings, where the robot must push boxes to the receptacle with no static obstacles impeding its path. A 2D example setup is shown in the left-most *Box-Delivery* snapshot in Fig. 1.1. We performed experiments under 3-obstacle and 5-obstacle settings in both 3D simulation and in the testbed. The positions of the robot and boxes are randomly placed at the start of each trial. For the testbed experiments, a time limit of 5 minutes is enforced to ensure prompt completion of the trial.

The evaluation results are summarized in Table 3.4. We observe that SAM drastically outperforms PPO and SAC in simulation and testbed experiments, as it achieves high values for all metrics. Between the RL baselines, SAC outperforms PPO by achieving better task success and efficiency scores, with comparable interaction effort. Similar to *Maze*, the metric values decrease with an increase in the number of boxes, indicating an increase in task difficulty.

We attribute the large performance gap between SAM and the RL baselines to the action representation. In the Box-Delivery setup, PPO and SAC output a short-horizon low-dimensional heading at each step, which implicitly bundles multiple hard decisions such as which box to target, where to make contact, and how to sequence pushes over a long horizon. Consequently, the policy must discover long sequences of correct choices before receiving meaningful reward signals, making learning unstable and inefficient. In contrast, SAM uses a spatial action map over the global pixel space to select waypoints in the environment. SAM can reason over the full scene to choose which object interaction to prioritize, while a lower-level controller handles realizing that waypoint. This decoupling reduces the burden on the learned policy to simultaneously reason about dynamics, contact, and long-horizon task structure. As a result, SAM achieves consistently higher task success and efficiency than PPO and SAC in both simulation and the physical testbed. Such

Table 3.4: Results for *Box-Delivery* in the physical testbed and Bench-Push simulations. ↑ indicates higher is better.

# Obs	Platform	Alg.	$S_{\text{manip}} \uparrow$	$E_{\text{manip}} \uparrow$	$I_{\text{manip}} \uparrow$
3-Obs	Sim	PPO	0.08 ± 0.18	0.004 ± 0.008	**0.997 ± 0.002**
		SAC	0.13 ± 0.22	0.008 ± 0.014	0.985 ± 0.005
		SAM	**1.00 ± 0.00**	**0.282 ± 0.122**	0.987 ± 0.007
	Testbed	PPO	0.00 ± 0.00	0.000 ± 0.000	**0.992 ± 0.003**
		SAC	0.33 ± 0.27	0.189 ± 0.148	0.969 ± 0.009
		SAM	**0.78 ± 0.16**	**0.341 ± 0.111**	0.981 ± 0.015
5-Obs	Sim	PPO	0.20 ± 0.22	0.012 ± 0.013	**0.994 ± 0.003**
		SAC	0.15 ± 0.17	0.018 ± 0.019	0.978 ± 0.007
		SAM	**0.98 ± 0.06**	**0.243 ± 0.096**	0.982 ± 0.007
	Testbed	PPO	0.00 ± 0.00	0.000 ± 0.000	0.987 ± 0.003
		SAC	0.13 ± 0.09	0.093 ± 0.073	**0.990 ± 0.001**
		SAM	**0.67 ± 0.09**	**0.417 ± 0.089**	0.972 ± 0.003

an observation suggests the need for meaningful task decomposition for navigation and manipulation in cluttered environments.

Finally, the task success scores in the physical testbed are lower in most cases compared to the simulation, mainly due to sim-to-real gaps stemming from small controller errors that result in imperfect manipulation of the objects (e.g., boxes slipping from the robot's bumper). Despite this score reduction, the relative performances of the policies in the testbed are consistent with the simulations, indicating good policy transfer from Bench-Push simulations to real robots.

3.3 Summary

We introduce Bench-Push, the first comprehensive benchmark for pushing-based navigation and manipulation tasks in mobile robots. Bench-Push offers a suite of customizable tasks spanning both navigation and manipulation, implemented in simulation with varying levels of complexity. We also provide novel evaluation metrics that capture efficiency and interaction effort. Bench-Push comes with implementations of established RL algorithms and task-specific baselines, demonstrating the benchmark's extensibility.

Chapter 4

Hierarchical RL-Diffusion Policy for Efficient Nonprehensile Manipulation

4.1 Problem Formulation and Spatial Action Map Overview

In this section, we first formulate the task as a Markov decision process and specify the objectives we aim to optimize. We then provide an overview of spatial action maps, including details about the state and action spaces, and the reward function used in the original paper.

4.1.1 Problem Formulation

We consider an environment $\mathcal{W} \subset \mathbb{R}^2$ containing n fixed obstacles $\mathcal{O}_1, \ldots, \mathcal{O}_n \subset \mathcal{W}$, a receptacle $\mathcal{B} \subset \mathcal{W}$, m movable boxes, and a mobile robot. At time $t \geq 0$, each box $j \in \{1, \ldots, m\}$ is described by a time-varying set $b_j(t) \subset \mathcal{W}$, defining the 2D space occupied by the box. Likewise, the robot is described by $\mathcal{R}(t) \subset \mathcal{W}$. The robot navigates within the free space $\mathcal{W}_{\text{free}} = \mathcal{W} \setminus (\cup_i \mathcal{O}_i)$. The task of the robot is to successfully push all boxes into the receptacle by a given time $T_{\max}$: That is, to reach a state at some time $t' \in [0, T_{\max}]$ where $b_j(t') \subset \mathcal{B}$ for all $j \in \{1, \ldots, m\}$.

Objective. Our objective is to compute a robot policy π that optimizes a two-tiered objective, capturing both task success and the efficiency of robot behaviour. The *Primary*

Objective is to maximize the expected number of boxes placed in the receptacle by time $T_{\max}$. Formally, the primary objective corresponds to maximizing the success indicator: $\sum_{j=1}^{m} \mathbf{1}[b_j(T_{\max}) \subset \mathcal{B}]$.

The *Secondary Objective* is to minimize the time to complete the task. Formally, we seek to minimize:

$$\min \left\{ t \in [0, T_{\max}] \mid b_j(t) \subset \mathcal{B} \; \forall \; j \in \{1, \ldots, m\} \right\},$$

where the completion time is $T_{\max}$ if not all boxes are in the receptacle by time $T_{\max}$.

This tiered formulation encourages the agent to reliably complete the task while favouring efficient policies.

MDP Model. We model the problem as a Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}, R, \gamma)$, where $\mathcal{S}$ is the state space consisting of the robot position and environment configuration and $\mathcal{A}$ is the action space. At each time step, the robot observes a state $s \in \mathcal{S}$ and chooses an action $a \in \mathcal{A}$ according to a policy $\pi : \mathcal{S} \to \mathcal{A}$. The environment transitions to a new state $s' \sim \mathbb{P}(s'|s, a)$ and emits a scalar reward $r = R(s, a)$, where the scalar reward acts as a proxy for the multi-objective optimization stated above. The constant $\gamma \in (0, 1)$ is the discount factor.

4.1.2 Spatial Action Maps

In this section, we review the spatial action map (SAM) policy as proposed in [51]. To motivate the use of SAMs, we first formulate the task as a reinforcement learning problem. To solve the above MDP, we adopt a value-based RL approach using a Double Deep Q-Network (DDQN) [45]. The goal is to learn an action-value function $Q_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, parameterized by θ, that estimates the expected cumulative discounted reward when taking action a in state s and following an optimal policy thereafter.

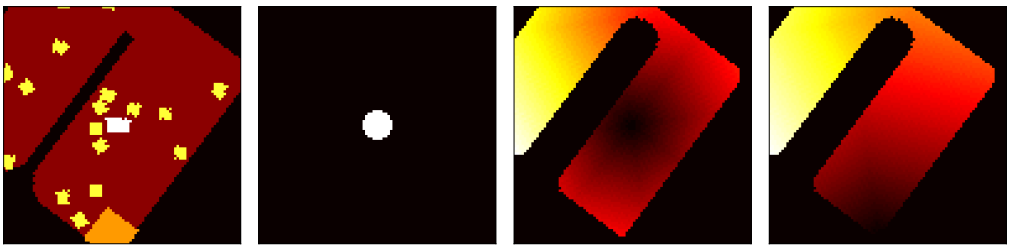
During training, transitions (s, a, r, s') are uniformly sampled from an experience replay buffer [25], and the online network parameters θ are updated to minimize the following temporal difference loss:

$$\mathcal{L}(\theta) = \left| r + \gamma Q_{\theta'} \left(s', \arg\max_{a'} Q_\theta(s', a') \right) - Q_\theta(s, a) \right|,$$

where $Q_{\theta'}$ is the target network with frozen parameters θ' periodically updated from θ.

The resulting policy is derived greedily from the learned Q-function:

$$\pi_\theta(s) = \arg\max_a Q_\theta(s, a).$$



(a) Overhead Map (b) Robot Footprint (c) Shortest path from robot (d) Shortest path to receptacle

Figure 4.1: Visualization of the four-channel state representation in the *LargeDivider* environment (colors shown for clarity only).

SAMs define a dense, pixel-aligned action space where each action corresponds to a goal coordinate in the robot's environment. The Q-network $Q_\theta(s, a)$ outputs a heatmap over all spatial locations, representing the value of navigating to each coordinate. The policy $\pi_\theta(s)$ selects the pixel with the highest Q-value, which becomes the high-level goal for the robot. A path is then generated to this goal point using the Shortest Path Faster Algorithm (SPFA) [12], and a low-level controller is used to traverse the path. The rest of this section details the state space, action space, and reward function, which are central to this formulation.

State Space. Each state $s \in \mathcal{S}$ is encoded as a 4-channel image aligned to the robot's local coordinate frame, with spatial resolution (H, W). The channels include: (1) A semantic segmentation map encoding object classes (e.g., obstacles, floor, boxes, robot, receptacle); (2) A binary mask representing the robot's footprint and location; (3) A shortest-path distance map from the robot's current position to each pixel; and (4) A shortest-path distance map from each pixel to the receptacle. This yields a structured state space $\mathcal{S} = [0, 1]^{H \times W \times 4}$. An example visualization is shown in Fig. 4.1.

Action Space. Each action $a \in \mathcal{A}$ represents a pixel location (u, v) in the spatial map: $\mathcal{A} = \{1, \ldots, W\} \times \{1, \ldots, H\}$. The selected coordinate becomes the high-level goal, and the robot uses a low-level controller to navigate to it.

Reward Function. The total reward at a given step k is composed of three components:

$$r(k) = r_{\text{goal}}(k) + r_{\text{progress}}(k) + r_{\text{penalty}}(k), \tag{4.1}$$

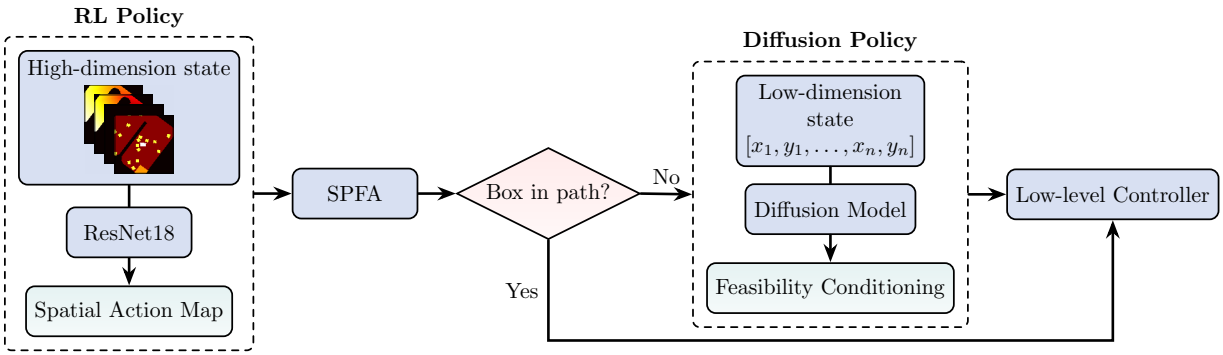


Figure 4.2: HeRD architecture. The high-level RL policy encodes observations into a spatial action map and selects a spatial goal, which is converted to a path using SPFA. If the path intersects boxes, it is executed by a proportional controller that first rotates then translates the robot; otherwise, a diffusion policy generates a goal-conditioned trajectory through denoising and feasibility conditioning.

where:

$$r_{\text{goal}}(k) \quad = +1 \text{ for each box placed in receptacle,} \tag{4.2}$$

$$r_{\text{progress}}(k) = \alpha \sum_j \Delta d_{b_j}(k), \tag{4.3}$$

$$r_{\text{penalty}}(k) = -0.25 \text{ for a collision or nonmovement.} \tag{4.4}$$

Here, $\Delta d_{b_j}(k)$ denotes the signed change in shortest-path distance from box b_j to the receptacle at step k, and α is a scaling coefficient that ensures the progress reward is on a comparable scale to the other reward terms. The value of α is chosen empirically based on the environment size, such that progress rewards do not dominate the reward signal and overwhelm goal completion or penalty terms.

4.2 Approach Overview

We design a hierarchical control framework, **HeRD**, that separates high-level semantic decision-making from low-level path planning. The high-level policy selects spatial goals using reinforcement learning with a SAM action representation [51], while a low-level controller determines how to move the robot toward that goal.

Our key contribution lies in adapting the trajectory generation strategy depending on whether the path to the spatial goal requires interacting with movable objects (i.e., boxes):

1. If the SPFA-computed path to the spatial goal intersects a boxes, we retain the original path to exploit pushing behaviour.

2. Otherwise, we generate a new trajectory using a learned diffusion policy π_d, trained on human demonstrations, which captures human-like strategies for positioning, navigation, and setup in scenarios where no immediate box interaction is required.

Our decision to use the diffusion policy only when no boxes are intersected stems from an asymmetry in the learning signal provided by the reward function. There is strong, immediate reward feedback when the robot pushes a box (r_{progress}), but when just navigating, the only available reward signal is the occasional penalty for colliding with an obstacle or not moving, and a distance-based penalty (See Eqn. 4.6 in Sect. 4.3.1). We hypothesize that the strong feedback when pushing enables the policy to specialize in this task. In contrast, there are complex and subtle concepts involved in efficiently navigating the robot that are difficult to encode into a reward function. Strategically repositioning or avoiding boxes en route to a spatial goal are tasks that are easy for a human to carry out, but very hard to encode into a reward function. By deferring to the diffusion policy in these situations, we leverage the intuition of the human demonstrator to generate effective trajectories for the robot.

The overall architecture is visualized in Fig. 4.2. Further details on the policy and training are provided in Section 4.3.

4.3 Implementation Details

4.3.1 Reinforcement Learning Policy

We formulate the high-level RL policy as a DDQN with a SAM action representation, following the architecture described in Section 4.1.2, with two notable modifications to the reward function. Qualitatively, we observe that the current policy often engages in inefficient behaviour: rather than directly pushing boxes to the receptacle, the robot tends to gather them into a corner before attempting group pushes along walls. It frequently performs backward movements or pushes boxes away from the receptacle in hopes of later combining them. This strategy leads to excessive travel and coordination failures, particularly in larger or more constrained environments. To address this we (1) modify the r_{progress} reward term and (2) add a new r_{motion} term.

Progress-based reward. In the original SAM formulation, the robot received a reward based on the total signed progress of all boxes toward the receptacle (Eqn. 4.3), which heavily incentivized the agent to push multiple boxes at once. While this inherently is not a negative quality, it often leads to unnecessarily long and inefficient paths in order to gather boxes to push them together.

To align with our objective of minimizing time to task completion, we redefine the reward to consider only the signed progress of the single most-advancing box per step:

$$r_{\text{progress}}(k) = \alpha \cdot \Delta d_{b_j^*}(k), \quad \text{where } j^* = \arg\max_j |\Delta d_{b_j}(k)|, \tag{4.5}$$

and $\Delta d_{b_j}(k)$ represents the distance box j travels towards the receptacle at step k. This preserves the directional component of progress while avoiding reward inflation from simultaneous multi-box pushes. As a result, the agent tends to act more efficiently, choosing to push nearby boxes individually rather than engaging in complex, inefficient maneuvers to gather and push multiple boxes at once. Nevertheless, the agent can still exploit favourable multi-box pushes if they arise naturally.

We use $\alpha = 0.2$, consistent with [51], to normalize progress rewards and prevent them from dominating other terms.

Distance-based penalty. To further encourage efficient trajectories, we introduce a penalty based on the robot's displacement at each step. The penalty is scaled to be smaller than the reward for a correct box push, ensuring that the agent is not heavily penalized for non-pushing actions, giving it the

$$r_{\text{motion}}(k) = -\frac{\alpha}{\beta} \cdot \Delta d_{\mathcal{R}}(k). \tag{4.6}$$

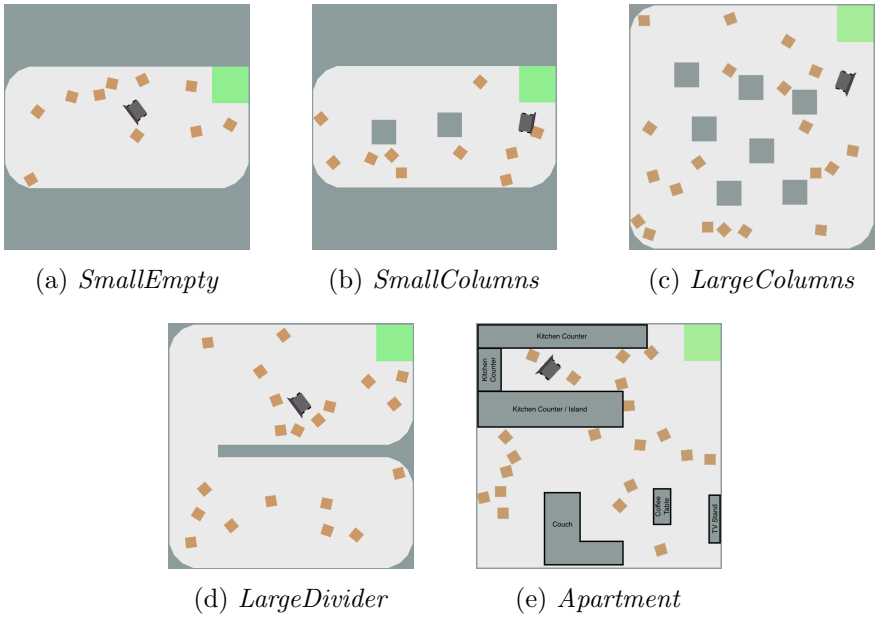
Here, $\Delta d_{\mathcal{R}}(k)$ denotes the distance the robot travels at step k. We use $\beta = 8$, but found that values in $[2, 16]$ work with similar effectiveness.

Training Details. We adopt the hyperparameters used in [51], summarized as follows: *Replay Buffer* of 10,000 transitions; *Optimizer* is SGD with learning rate 0.01, momentum 0.9, and weight decay 0.0001; *Loss Function* is Smooth L1 loss with gradients clipped at 10; *Batch Size* is 32; *Discount Factor* is $\gamma = 0.99^{0.25 \cdot d_{\mathcal{R}(t)}}$, to reflect distance travelled per step (see [51] for more details); *Exploration* via ϵ-greedy with ϵ linearly annealed from 1.0 to 0.01 over 6,000 steps; *Training Duration* is 60,000 steps; *Random Warm-up* is 1,000 random exploration steps before training begins.

We train the RL policy after the diffusion policy has been trained and frozen. During the RL training phase, the agent executes actions using the full hierarchical framework.

This ensures the RL agent learns a value function that accounts for the specific dynamics and limitations of the diffusion-based low-level controller.

While [51] trains four specialized policies, one per environment type, due to our more efficient reward design we are able to train one generalized policy with minimal performance loss due to the generalization. To do this, we randomly select between the *LargeColumns* or *LargeDivider* environments every episode (see Fig. 4.3 for the environment types), which forces the policy to adapt to various obstacle configurations. We define an episode as terminated when either all of the boxes are pushed into the receptacle or if the robot has not pushed any boxes into the receptacle for 100 steps. We assume access to ground-truth state observations in simulation to eliminate partial observability and better isolate the impact of our reward and architecture modifications. Training takes about 8 hours on an NVIDIA L40S GPU.



(a) *SmallEmpty* (b) *SmallColumns* (c) *LargeColumns*

(d) *LargeDivider* (e) *Apartment*

Figure 4.3: Evaluation environments used in HeRD vary in size and layout. Small environments measure 10m x 5m with 10 boxes; Large environments are 10m x 10m with 20 boxes. Robot, box, and obstacle positions are randomized per episode, with column counts ranging from 0–2 (Small) to 0–8 (Large). To test zero-shot generalization, we introduce the Apartment environment. This mimics a residential layout with irregular, non-convex obstacles (e.g., L-shaped couches) unseen during training, evaluating how well learned behaviours transfer to unstructured, realistic settings.

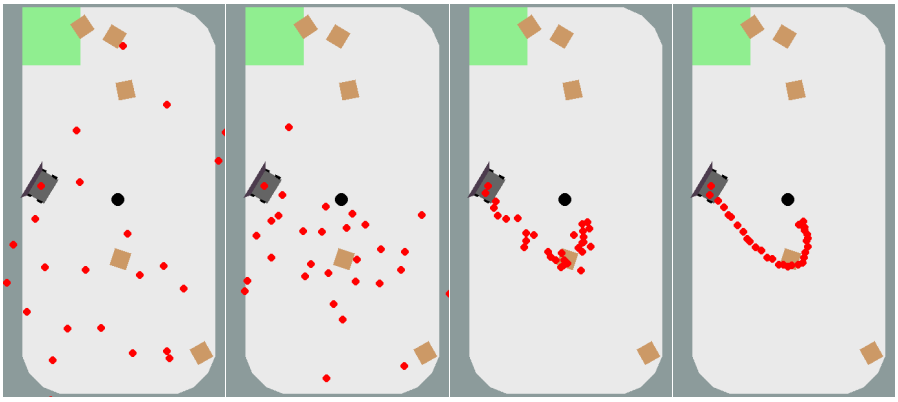


Figure 4.4: Denoising process of a generated path. The high-level RL policy selects a spatial goal (the black dot in the figure), and the low-level diffusion policy generates a context-aware trajectory from the robot to the spatial goal.

4.3.2 Diffusion Policy

Our diffusion-based controller generates trajectories from demonstration data to guide the robot toward a spatial goal in non-pushing scenarios. It is queried during execution when the SAM-selected goal is reachable without box interaction. Fig. 4.4 shows one such diffused path.

Model Architecture and Conditioning. Following [8], we implement a conditional diffusion model using a 1D U-Net for denoising, conditioned via FiLM layers [29] on low-dimensional observation features. Unlike the receding-horizon approach used in [8], we generate the trajectory in a single forward pass. This is necessary to support goal conditioning via inpainting [19], where at every denoising step we fix the first point of the trajectory to the robot's position and the last point to the goal. This guides the waypoints into a coherent trajectory. Receding-horizon rollouts are incompatible here, as they lack a consistent final waypoint.

The denoising process is defined iteratively as

$$\tau_t^{i-1} = a\left(\tau_t^i - \lambda\,\epsilon_\theta(\mathbf{O}_t, \tau_t^i, i) + \mathcal{N}(0, \sigma^2 I)\right), \tag{4.7}$$

where τ^i is the noisy trajectory at timestep i, $\mathbf{O}_t$ is the observation vector, ϵ_θ is the noise prediction network, $a(\cdot)$ enforces the goal conditioning, and $\lambda \in (0, 1)$ is the discount factor in the diffusion process. The model is trained to minimize the standard denoising loss: $\mathcal{L} = \text{MSE}\left(\epsilon^i, \epsilon_\theta(\mathbf{O}_t, \tau^0 + \epsilon^i, i)\right)$, and generates 32-point trajectories using DDIM sampling at inference time.

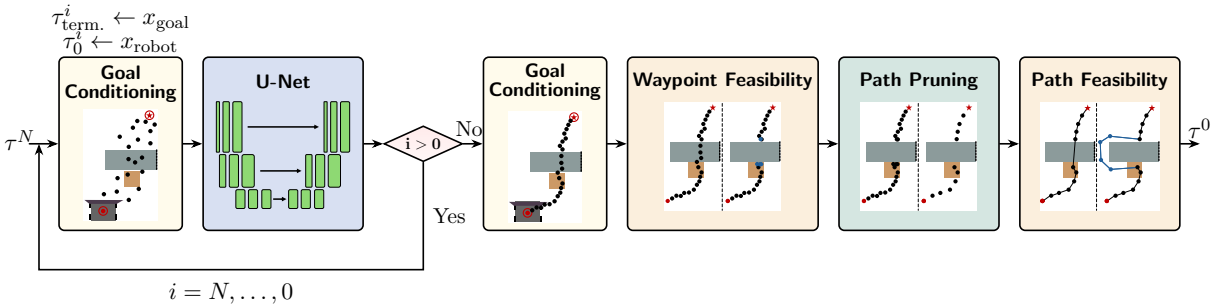


Figure 4.5: Diffusion-based trajectory generation pipeline. At each difusion timestep i, the trajectory τ^i is denoised using a 1D U-Net conditioned on observation features. The trajectory's endpoints are fixed via goal inpainting: the first point is the robot's position, and the last is the spatial goal. If $i > 0$, the intermediate points are further denoised. Once denoising completes ($i = 0$), the trajectory undergoes a feasibility pipeline: each waypoint is projected to free space if necessary, redundant waypoints are pruned, and path segments crossing obstacles are repaired via SPFA. The final output is a smooth trajectory that respects geometric constraints.

Demonstration Data and Augmentation. To collect expert demonstrations for training the diffusion policy, we deployed a pretrained high-level policy in a large, obstacle-free environment (*LargeEmpty*) to generate spatial goals. For each episode, a human demonstrator teleoperated the robot to navigate toward the SAM-selected goal while strategically considering the most effective path to the goal. These considerations included grouping boxes against walls, avoiding interactions with boxes that would be easier to push later, and pushing multiple boxes at once.

Each demonstration episode begins at the robot's initial position and ends at the SAM-specified goal location. Trajectories are recorded by logging the robot's (x, y) position at fixed intervals of 0.3 meters of travel, producing a sequence of sparse, evenly spaced waypoints. To improve smoothness and standardize input dimensions for the diffusion policy, we then apply linear interpolation between these waypoints to generate a dense 32-point trajectory per episode.

Approximately 1,000 episodes were collected. To augment the dataset, we generate additional demonstrations by selecting any non-interpolated waypoint from an existing trajectory as the new starting position, and padding the end of the trajectory with the spatial goal to maintain the fixed length. This yields ∼13,500 valid demonstrations for training.

Each demonstration is paired with a state observation composed of: (1) the 4-corner vertices of the robot and receptacle, (2) the (x, y) positions of the four closest boxes to

the robot, and (3) the spatial goal. If fewer than four boxes remain in the environment, placeholder boxes centered within the receptacle are used to preserve a fixed input size. All state inputs are normalized following the protocol from [8]. We made the design decision to include only the four closest boxes based on our empirical hypothesis that human demonstrators struggled to consider more than four boxes when teleoperating the robot. In addition, we found that the orientation of the boxes was not essential to include in the state, allowing us to further reduce the state space by only including the box positions.

Training Details. The diffusion model was trained for 3,500 epochs using a batch size of 256, with a validation split of 2%. Optimization was performed using AdamW ($\beta_1 = 0.95$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, weight decay $= 10^{-6}$, learning rate $= 10^{-4}$). Validation was performed every 10 epochs. Although full training took approximately 1 hour on a single NVIDIA L40S GPU, we selected the checkpoint with the lowest validation loss, which occurred at epoch 350. This corresponds to roughly 6 minutes of training time in practice. We trained with 100 denoising steps but use 15-step DDIM sampling at inference to improve efficiency.

Trajectory Postprocessing. To ensure the diffusion path is feasible, we use the following postprocessing procedure:

1. *Goal Conditioning:* All sampled trajectories should start at the robot and end at the spatial goal. We use conditioning-by-inpainting to replace the first and last sampled points in the trajectory with the robot's position and spatial goal.
2. *Waypoint Feasibility:* Each waypoint is checked for collision; if it lies within an obstacle, it is projected to the nearest valid point in $\mathcal{W}_{free}$.
3. *Pruning:* Waypoints are sparsified via a distance threshold. This results in a more efficient trajectory without compromising fidelity.
4. *Trajectory Feasibility:* Even if all waypoints lie in free space, sequential points may straddle obstacles. Line segments crossing obstacles are repaired using SPFA.

The complete denoising process is shown in Fig. 4.5.

4.4 Results

We evaluate the performance of HeRD on the *Box-Delivery* task from Bench-Push [60]. We test across four environments of increasing difficulty (Fig. 4.3). For fair comparison, all

Table 4.1: HeRD vs. baseline performance. Bold text indicates the highest number of boxes delivered and the shortest travel distance (evaluated only for policies exceeding an 80% delivery threshold).

	Ours		SAM		Hierarchical RRT*		Greedy Heuristic Policy	
Environment	Boxes	Distance	Boxes	Distance	Boxes	Distance	Boxes	Distance
SmallEmpty	**10.00**	115 ± 23	9.80 ± 0.40	177 ± 54	9.75 ± 0.54	208 ± 80	9.65 ± 1.53	$\mathbf{82 \pm 16}$
SmallColumns	**10.00**	114 ± 31	9.50 ± 0.74	315 ± 176	9.95 ± 0.22	175 ± 50	9.80 ± 0.87	$\mathbf{81 \pm 19}$
LargeColumns	$\mathbf{19.80 \pm 0.51}$	$\mathbf{339 \pm 67}$	19.30 ± 0.95	655 ± 269	16.25 ± 3.99	550 ± 172	15.75 ± 7.45	197 ± 84
LargeDivider	$\mathbf{19.60 \pm 0.92}$	$\mathbf{345 \pm 69}$	18.05 ± 2.64	573 ± 131	17.55 ± 2.50	604 ± 180	8.00 ± 6.15	153 ± 107
Apartment	$\mathbf{17.60 \pm 3.32}$	$\mathbf{934 \pm 308}$	11.30 ± 4.53	565 ± 327	5.55 ± 4.91	501 ± 335	5.55 ± 3.76	171 ± 68

Table 4.2: Effect of the different components of HeRD

	Ours		No diffusion policy		Only diffusion policy		Naive trajectory with feasibility conditioning		Cumulative progress reward		No step distance penalty	
Environment	Boxes	Distance	Boxes	Distance	Boxes	Distance	Boxes	Distance	Boxes	Distance	Boxes	Distance
SmallEmpty	**10.00**	$\mathbf{115 \pm 23}$	9.95 ± 0.22	130 ± 38	9.55 ± 0.50	246 ± 79	9.95 ± 0.22	225 ± 86	**10.00**	150 ± 39	**10.00**	119 ± 25
SmallColumns	**10.00**	114 ± 31	**10.00**	131 ± 25	9.65 ± 0.48	200 ± 90	9.55 ± 0.92	216 ± 98	9.80 ± 0.87	138 ± 42	**10.00**	$\mathbf{104 \pm 26}$
LargeColumns	$\mathbf{19.80 \pm 0.51}$	$\mathbf{339 \pm 67}$	19.55 ± 0.86	403 ± 136	18.90 ± 1.22	540 ± 186	15.80 ± 3.43	606 ± 218	18.75 ± 2.81	435 ± 106	19.15 ± 3.26	360 ± 77
LargeDivider	$\mathbf{19.60 \pm 0.92}$	345 ± 69	19.90 ± 0.44	547 ± 138	19.15 ± 0.85	639 ± 141	17.75 ± 2.38	702 ± 197	18.90 ± 1.41	408 ± 95	19.40 ± 1.24	$\mathbf{318 \pm 58}$

models are evaluated on the same random seed, with 20 trials per environment. We report two primary metrics: the number of boxes successfully delivered, and the total distance travelled by the robot within the episode.

Comparison to baselines. To rigorously evaluate our approach, we compare HeRD against three baselines: (1) **SAM** [51], trained separately per environment; (2) **Greedy Heuristic Policy**, an A* transit planner with clearance penalties; and (3) **Hierarchical RRT***, a sampling planner using a semantic edge cost to encourage advantageous box pushes. In contrast to SAM, we train a single generalized policy for HeRD and for the Hierarchical RRT* baseline. We described the SAM implementation in Section 4.1.2. Please visit our project repository for details on the latter two baselines.

Evaluation results in Table 4.1 show HeRD consistently achieves higher success rates and shorter paths than SAM. While the **Greedy Heuristic** achieves low travel distance in simple environments by rigidly avoiding box interactions, this behaviour causes catastrophic failure in complex settings (dropping to 8.00 boxes in *LargeDivider*). Similarly, while **Hierarchical RRT*** navigates safely, it often samples geometrically valid but strategically suboptimal paths, falling short of HeRD's success rate (17.55 boxes vs. 19.60 in *LargeDivider*).

These results highlight the difficulty of manually encoding foresight for nonprehensile



(a) HeRD (b) Baseline

Figure 4.6: The difference in behaviour of HeRD compared to the baseline. The figure shows the first few actions of each policy in the same environment. HeRD immediately seeks out boxes to push into the receptacle while the baseline focuses on gathering multiple boxes together, even pushing them backwards to do so.

manipulation. By replacing the heuristic low-level planner with a learned diffusion policy, HeRD captures intuitive, context-aware navigation priors, resulting in superior robustness and efficiency across all tested environments.

Zero-Shot Generalization in Complex Environment. To test robustness to complex and unseen obstacles, we evaluated the policies in the novel *Apartment* environment without any additional training. Since the SAM baseline requires environment-specific training, we evaluated all previously trained SAM models and report the performance of the best one (the model trained on *LargeColumns*).

The generalized HeRD policy significantly outperformed the best SAM baseline (17.60 boxes vs. 11.30 boxes). The Greedy Heuristic and Hierarchical RRT* often became trapped in the non-convex obstacles (both dropping to just 5.55 boxes delivered), while HeRD maintained a high success rate. This demonstrates that the diffusion policy learns robust, generalizable obstacle-avoidance priors.

Effect of separating pushing and non-pushing actions. We hypothesize that the high-level policy specializes in pushing actions but struggles to plan efficient navigation actions, which is why we utilize diffusion for these scenarios. To validate this design, we compare HeRD to two ablations: one trained *without* a diffusion policy, and one that uses the diffusion policy *exclusively* for all trajectories. See Table 4.2.

The "No diffusion trajectories" variant outperforms the "Only diffusion trajectories" variant in both success rate and efficiency. This supports our claim that the high-level policy excels at pushing, while highlighting the limitations of relying entirely on diffusion-generated paths, which lack the fine-grained precision required for effective pushing, especially in cluttered or multi-box scenarios. A much larger demonstration dataset would be

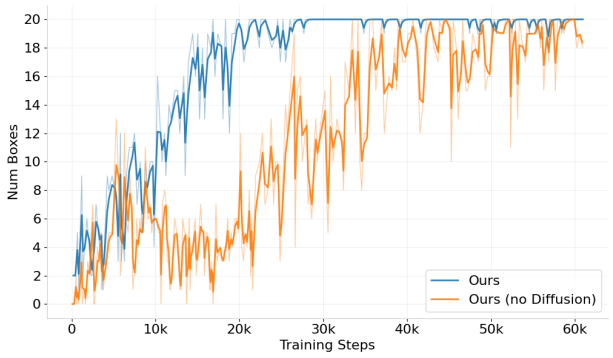


Figure 4.7: Training curves for the high-level policy. HeRD converges earlier (learns more efficiently) than the no-diffusion variant.

required to overcome this.

HeRD outperforms both ablations, demonstrating that selectively deploying diffusion trajectories complements the high-level policy's pushing proficiency. We observe improved sample efficiency as well: Fig. 4.7 shows that HeRD converges roughly 16,000 samples earlier than the no-diffusion variant. This suggests that delegating low-reward decision-making to the diffusion model helps smooth the learning signal and accelerates training.

Effect of feasibility conditioning. To investigate reliance on the post-processing feasibility pipeline, we introduced a 'Naive trajectory with feasibility conditioning' ablation. In this variant, the diffusion model is replaced by a simple straight-line path generator, which is then passed through the same feasibility pipeline used in HeRD.

Table 4.2 shows that the post-processing alone is insufficient. The naive method's travel distances nearly double across all environments, and its success rate drops sharply in challenging environments compared to HeRD. This suggests that the diffusion policy is learning efficient, context-aware navigation priors that post-processing alone cannot replicate.

Effect of reward structure. We perform two ablations to evaluate how the reward function components contribute to performance. The first reverts our maximum-based progress reward (Eqn. 4.5) back to the cumulative formulation (Eqn. 4.3). The results under the "Cumulative progress reward" column in Table 4.2 show a noticeable decline in efficiency and task success. This cumulative formulation encourages the robot to gather multiple boxes before pushing to maximize per-step reward, often leading to complex setup actions like backward movements and detours. In constrained environments like *LargeDivider*, this results in longer paths and reduced robustness. By contrast, our maximum-based formu-

lation reduces this bias and encourages steady, directed progress toward the goal leading to more robust and efficient behaviour.

The second removes the step distance penalty (Eqn. 4.6), which yields metrics competitve with HeRD, but degrades success rates in complex environments. This penalty helps discourage inefficient motion while retaining exploration.

4.4.1 Real-World Experiments

We evaluate HeRD and the SAM baseline on a physical TurtleBot3 in a lab recreation of the *SmallEmpty* environment (Fig. 1.2). Using an overhead camera and visual markers, we obtain accurate pose estimations of the robot, boxes, receptacle, and environment boundaries. We utilized this controlled perception setup to isolate the performance of the control policy from perception noise, ensuring that any failures or inefficiencies observed were due to the policy itself rather than state estimation errors. Furthermore, this setup provided the high-precision ground truth data required to calculate the distance metrics reported below.

Each trial is run for $T_{\max} = 25$ minutes. We use the number of boxes pushed into the receptacle in the trial as a proxy for our two-tiered objective of maximizing completed boxes and minimizing distance. We tested HeRD and the baseline for 5 trials each. Using HeRD, the robot was pushed 8.0 out of 10 boxes on average into the receptacle in the given time limit, and only 3.2 boxes using the baseline policy.

A notable sim-to-real gap was the wall friction. The robot struggled to overcome this force when many boxes were on the wall at once. This severely degraded the baseline policy, which relies heavily on gathering many boxes in a corner and pushing all of them against the wall towards the receptacle. In contrast, the HeRD policy acts more efficiently, pushing one or two boxes in at a time if they are in accessible positions rather than pushing them backwards into a far corner.

4.5 Summary

We introduce HeRD, a hierarchical architecture pairing high-level spatial goal selection with low-level diffusion trajectory generation. This decoupling enables efficient pushing in clutter, consistently outperforming the baseline [51] in success and efficiency, particularly in constrained settings. By leveraging human priors for smooth navigation within a hierarchical framework, HeRD achieves robust generalization across diverse environments.

Chapter 5

Conclusion

This thesis addressed the challenges of nonprehensile mobile manipulation, a domain where robots must interact with objects to navigate or rearrange their environment. The complexity of contact dynamics, coupled with a lack of standardized evaluation tools, has historically fragmented research efforts in this field. We approached this problem from two complementary directions: first, by establishing a rigorous benchmarking framework to standardize evaluation; and second, by developing a novel hierarchical control architecture that leverages the strengths of both reinforcement learning and generative diffusion models.

In Chapter 3, we introduced **Bench-Push**, the first unified benchmark designed specifically for pushing-based mobile robot tasks. Recognizing that traditional navigation metrics penalize the interaction required for manipulation, we proposed a novel set of metrics, Task Efficiency (E_{manip}) and Interaction Effort (I_{manip}), that quantify the trade-off between path length and physical exertion. We provided a suite of high-fidelity simulation environments spanning both navigation-centric (Maze, Ship-Ice) and manipulation-centric (Box-Delivery, Area-Clearing) tasks. Through extensive baseline evaluations, we demonstrated that Bench-Push enables fair cross-comparison of algorithms and supports zero-shot transfer to physical hardware, bridging the gap between theoretical algorithm design and real-world deployment.

In Chapter 4, we utilized this benchmarking framework to develop **HeRD** (**He**rarchical **R**einforcement Learning - **D**iffusion Policy), a hybrid architecture for efficient mobile manipulation. We identified that while an RL agent with a spatial action map action space provides effective high-level planning, they suffer from inefficient low-level execution when relying on standard trajectory generation methods. HeRD addresses this by decoupling

the decision-making: a high-level RL agent selects spatial subgoals, while a low-level goal-conditioned diffusion policy generates smooth, context-aware trajectories. By introducing a max-based progress reward and separating pushing from non-pushing navigation, HeRD achieved a significant reduction in travel distance and higher success rates compared to both learning-based and classical motion planning baselines in cluttered environments. Furthermore, HeRD demonstrated robust zero-shot generalization to unseen, complex environments, successfully navigating unstructured non-convex obstacles that caused catastrophic failures in the classical baseline planners.

5.1 Limitations and Future Work

It is our hope that the work described in this thesis contributes positively to the community. Below, we outline some limitations of our methods and potential avenues for future work to extend these contributions.

5.1.1 Bench-Push

A primary limitation of Bench-Push is its reliance on bird's-eye view environment observations and a priori knowledge of the environment. In our current implementation, the robot receives ground-truth state observations with the assumptions of access to an unobstructed overhead camera and perfect environment segmentation. These are simplifying assumptions that allow researchers to focus only on the challenges of nonprehensile manipulation, without the burden of partial state observability.

Future work will include different onboard sensor options to attach to the robot, such as RGB-D cameras or LiDAR sensors. This reflects the constraints typical in a real-world environment, where overhead cameras are typically not available. Researchers will have the opportunity to use reconstruction methods (e.g. SLAM) to construct a map of the environment. In addition, while we have made the assumption of a priori environmental knowledge for the currently implemented baselines, researchers are encouraged to explore different state representations, which is a readily available option of each environment.

5.1.2 HeRD

As HeRD was developed and validated within the Bench-Push framework, it inherits the limitation of requiring a global, unobstructed overhead perspective. In many real-world

scenarios, this setup is rarely feasible. Future work will focus on transitioning from global state observations to ego-centric perception. By integrating a front-facing RGB-D camera, the observations could be constructed from local, online-generated maps.

The current diffusion policy utilizes a low-dimension state space that explicitly encodes the coordinates of only the four closest boxes to the robot. This heuristic was based on the hypothesis that human demonstrators prioritize closer boxes when making manipulation decisions. However, this limited "attention window" may prevent the model from capturing higher-order latent strategies. For example, the actions of the demonstrators could have been influenced by the state of boxes closer to the receptacle, with the goal of setting up a multi-box push later.

Extending the state representation to include a higher-dimensional vector or transition to a direct image-to-action (visual) policy would allow the model to reason over the entire scene geometry. Such an expansion would enable HeRD to leverage more nuanced and strategically robust manipulation behaviours, albeit at the cost of extra demonstrations.

References

[1] Elie Aljalbout, Felix Frank, Maximilian Karl, and Patrick van der Smagt. On the Role of the Action Space in Robot Manipulation Learning and Sim-to-Real Transfer. *IEEE Robotics and Automation Letters*, 9(6):5895–5902, June 2024.

[2] Brian DO Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

[3] Maria Bauza, Francois R. Hogan, and Alberto Rodriguez. A Data-Efficient Approach to Precise and Controlled Pushing. In *Proceedings of The 2nd Conference on Robot Learning*, pages 336–345. PMLR, October 2018.

[4] Filippo Bertoncelli and Lorenzo Sabattini. Streamlining object pushing: Behavior tree-based coordination of control and planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5203–5210, 2024.

[5] Victor Blomqvist. Pymunk: A easy-to-use pythonic rigid body 2d physics library. (version 6.7.0), 2024.

[6] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *IEEE International Conference on Advanced Robotics*, pages 510–517, 2015.

[7] Chang Chen, Fei Deng, Kenji Kawaguchi, Caglar Gulcehre, and Sungjin Ahn. Simple hierarchical planning with diffusion. In *The Twelfth International Conference on Learning Representations*. The International Conference on Learning Representations (ICLR), 2024.

[8] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via

action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

[9] Rodrigue de Schaetzen, Alexander Botros, Ninghan Zhong, Kevin Murrant, Robert Gash, and Stephen L Smith. Auto-icenav: A local navigation strategy for autonomous surface ships in broken ice fields. *arXiv preprint arXiv:2411.17155*, 2024.

[10] Juan Del Aguila Ferrandis, João Moura, and Sethu Vijayakumar. Nonprehensile Planar Manipulation through Reinforcement Learning with Multimodal Categorical Exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5606–5613, October 2023.

[11] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[12] Duan Fanding. SPFA fast algorithm for the shortest path. *Journal of Southwest Jiaotong University*, 29(2):6, 1994.

[13] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *IEEE ICRA*, pages 5113–5120, 2018.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[15] Botao He, Guofei Chen, Wenshan Wang, Ji Zhang, Cornelia Fermuller, and Yiannis Aloimonos. Interactive-far: Interactive, fast and adaptable routing for navigation among movable obstacles in complex unknown environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5402–5409, 2024.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[17] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

[18] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[19] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[20] Linh Kästner, Teham Bhuiyan, Tuan Anh Le, Elias Treis, Johannes Cox, Boris Meinardus, Jacek Kmiecik, Reyk Carstens, Duc Pichel, Bassel Fatloun, et al. Arena-bench: A benchmarking suite for obstacle avoidance approaches in highly dynamic environments. *IEEE Robotics and Automation Letters*, 7(4):9477–9484, 2022.

[21] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[22] Kota Kondo, Andrea Tagliabue, Xiaoyi Cai, Claudius Tewari, Olivia Garcia, Marcos Espitia-Alvarez, and Jonathan P How. Cgd: Constraint-guided diffusion policies for uav trajectory planning. *arXiv preprint arXiv:2405.01758*, 2024.

[23] Steven M. LaValle and James J. Kuffner. Randomized Kinodynamic Planning. *The International Journal of Robotics Research*, 20(5):378–400, May 2001.

[24] Wenhao Li, Xiangfeng Wang, Bo Jin, and Hongyuan Zha. Hierarchical Diffusion for Offline Decision Making. In *Proceedings of the 40th International Conference on Machine Learning*, pages 20035–20064. PMLR, July 2023.

[25] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992.

[26] Xiao Ma, Sumit Patidar, Iain Haughton, and Stephen James. Hierarchical Diffusion Policy for Kinematics-Aware Multi-Task Robotic Manipulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18081–18090, Seattle, WA, USA, June 2024. IEEE.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[28] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *NeurIPS*, 31, 2018.

[29] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[30] Daniel Perille, Abigail Truong, Xuesu Xiao, and Peter Stone. Benchmarking metric ground navigation. In *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 116–121. IEEE, 2020.

[31] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Proc. CoRL*, pages 188–204, 2021.

[32] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[33] Zhongqiang Ren, Bunyod Suvonov, Guofei Chen, Botao He, Yijie Liao, Cornelia Fermuller, and Ji Zhang. Search-based path planning in interactive environments among movable obstacles. *arXiv preprint arXiv:2410.18333*, 2024.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[35] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2256–2265. PMLR, June 2015.

[36] Ajay Sridhar, Dhruv Shah, Catherine Glossop, and Sergey Levine. Nomad: Goal masked diffusion policies for navigation and exploration. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–70. IEEE, 2024.

[37] Jochen Stüber, Claudio Zito, and Rustam Stolkin. Let's Push Things Forward: A Survey on Robot Pushing. *Frontiers in Robotics and AI*, 7:8, February 2020.

[38] Xilong Sun, Jiqing Li, Anna Vladimirovna Kovalenko, Wei Feng, and Yongsheng Ou. Integrating Reinforcement Learning and Learning From Demonstrations to Learn Nonprehensile Manipulation. *IEEE Transactions on Automation Science and Engineering*, 20(3):1735–1744, July 2023.

[39] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, August 1999.

[40] Yujie Tang, Martijn Wisse, and Wei Pan. Unwieldy object delivery with nonholonomic mobile base: A free pushing approach. *IEEE Robotics and Automation Letters*, 2024.

[41] Yujie Tang, Hai Zhu, Susan Potters, Martijn Wisse, and Wei Pan. Unwieldy object delivery with nonholonomic mobile base: A stable pushing approach. *IEEE Robotics and Automation Letters*, 8(11):7727–7734, 2023.

[42] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[43] Yuki Tomita, Kohei Matsumoto, Yuki Hyodo, and Ryo Kurazume. Colson: Controllable learning-based social navigation via diffusion-based reinforcement learning. *arXiv preprint arXiv:2503.13934*, 2025.

[44] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[45] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[47] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal Networks for Hierarchical Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3540–3549. PMLR, July 2017.

[48] Vivienne Huiling Wang, Tinghuai Wang, and Joni Pajarinen. Hierarchical reinforcement learning with uncertainty-guided diffusional subgoals. *arXiv preprint arXiv:2505.21750*, 2025.

[49] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[50] Joris J Weeda, Saray Bakker, Gang Chen, and Javier Alonso-Mora. Pushing through clutter with movability awareness of blocking obstacles. *arXiv preprint arXiv:2502.20106*, 2025.

[51] Jimmy Wu, Xingyuan Sun, Andy Zeng, Shuran Song, Johnny Lee, Szymon Rusinkiewicz, and Thomas Funkhouser. Spatial action maps for mobile manipulation. *arXiv preprint arXiv:2004.09141*, 2020.

[52] Jimmy Wu, Xingyuan Sun, Andy Zeng, Shuran Song, Szymon Rusinkiewicz, and Thomas Funkhouser. Spatial intention maps for multi-agent mobile manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8749–8756. IEEE, 2021.

[53] Jimmy Wu, Xingyuan Sun, Andy Zeng, Shuran Song, Szymon Rusinkiewicz, and Thomas Funkhouser. Learning pneumatic non-prehensile manipulation with a mobile blower. *IEEE Robotics and Automation Letters*, 7(3):8471–8478, 2022.

[54] Zixuan Wu, Sean Ye, Manisha Natarajan, and Matthew C Gombolay. Diffusion-reinforcement learning hierarchical motion planning in adversarial multi-agent games. *arXiv e-prints*, pages arXiv–2403, 2024.

[55] Fei Xia, William B Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese. Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5(2):713–720, 2020.

[56] Peijia Xie, Bingyi Xia, Anjun Hu, Ziqi Zhao, Lingxiao Meng, Zhirui Sun, Xuheng Gao, Jiankun Wang, and Max Q-H Meng. Autonomous multiple-trolley collection system with nonholonomic robots: Design, control, and implementation. *Journal of Field Robotics*, 42(1):20–36, 2025.

[57] Zifan Xu, Bo Liu, Xuesu Xiao, Anirudh Nair, and Peter Stone. Benchmarking reinforcement learning techniques for autonomous navigation. In *IEEE International Conference on Robotics and Automation*, pages 9224–9230, 2023.

[58] Linghong Yao, Valerio Modugno, Andromachi Maria Delfaki, Yuanchang Liu, Danail Stoyanov, and Dimitrios Kanoulas. Local path planning among pushable objects based on reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3062–3068, 2024.

[59] Weihao Yuan, Johannes A. Stork, Danica Kragic, Michael Y. Wang, and Kaiyu Hang. Rearrangement with Nonprehensile Manipulation Using Deep Reinforcement Learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 270–277, May 2018.

[60] Ninghan Zhong, Steven Caro, Megnath Ramesh, Rishi Bhatnagar, Avraiem Iskandar, and Stephen L Smith. Bench-Push: Benchmarking pushing-based navigation and manipulation tasks for mobile robots. *arXiv preprint arXiv:2512.11736*, 2025.

[61] Ninghan Zhong, Alessandro Potenza, and Stephen L Smith. Autonomous navigation in ice-covered waters with learned predictions on ship-ice interactions. In *IEEE International Conference on Robotics and Automation*, pages 10157–10163, 2025.

[62] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.