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

Anonymous Authors

***Abstract*— Mobile robots are increasingly deployed in cluttered environments with movable objects, posing challenges for traditional methods that prohibit interaction. In such settings, the mobile robot must go beyond traditional obstacle avoidance, leveraging pushing or nudging strategies to accomplish its goals. While research in pushing-based robotics is growing, evaluations rely on ad hoc setups, limiting reproducibility and cross-comparison. To address this, we present Bench-Push, the first unified benchmark for pushing-based mobile robot navigation and manipulation tasks. Bench-Push includes multiple components: 1) a comprehensive range of simulated environments that capture the fundamental challenges in pushing-based tasks, including navigating a maze with movable obstacles, autonomous ship navigation in ice-covered waters, box delivery, and area clearing, each with varying levels of complexity; 2) novel evaluation metrics to capture efficiency, interaction effort, and partial task completion; and 3) demonstrations using Bench-Push to evaluate example implementations of established baselines across environments. Bench-Push will be open-sourced as a Python library with a modular design. The code, documentation, and trained models can be found at `https://anonymous.4open.science/r/BenchPush-CDF6/README.md`.**

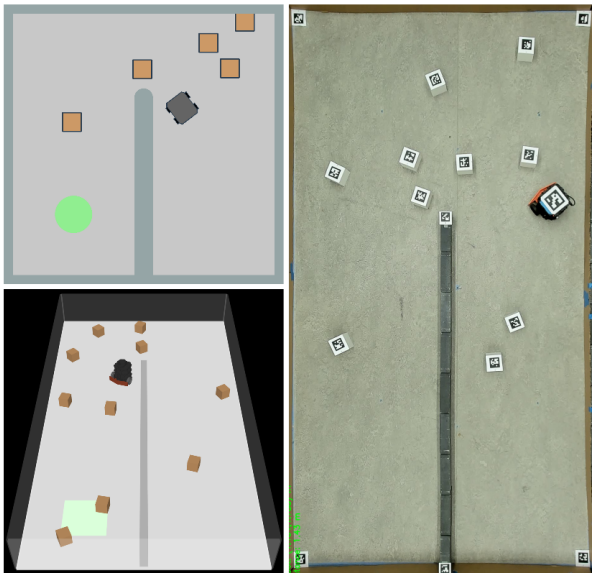

Fig. 1: *Maze* environment in 2D simulation (top-left panel), in 3D simulation (lower-left panel), and in our physical testbed (right panel).

I. Introduction

The key to robust mobile robot deployment lies in the robot's ability to operate in complex environments. Traditional mobile robot approaches typically focus on computing collision-free paths or actions. When mobile robots are deployed in cluttered and unstructured environments, such as homes, hospitals, or disaster sites, where movable objects might block feasible routes, finding collision-free paths is often impractical. Under such settings, the ability for robots to interact with movable objects and obstacles is essential for task completion. This leads to pushing-based navigation and manipulation, where the mobile robot must interact with objects through pushing or nudging actions to progress.

A parallel branch of work has traditionally studied such scenarios by assuming that mobile robots are equipped with manipulators and can use grasping actions to move obstacles out of the way [1], [2], [3]. However, manipulators are not always viable due to cost considerations, structural limitations, or task-specific constraints. Moreover, prehensile and dexterous mobile manipulation of objects remains an area of active research. This has spurred increasing interest in pushing-based mobile robot navigation and manipulation in environments with movable objects [4], [5].

Despite a growing body of work on pushing-based tasks, the field still lacks a unified framework for systematic evaluation and reproducibility. Existing approaches address different aspects of the problem but share a common theme: controlling a robot to reliably push objects toward a desired direction or goal. Yet, these 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. As a result, potentially transferable techniques [5], [6] are only evaluated in ad hoc settings.

Benchmarks have proven essential for advancing robotics research. Successful robotics benchmarks such as the YCB dataset [7], RoboSuite [8], and the Interactive Gibson Benchmark [9] have each driven significant progress in grasping, learning-based manipulation, and interactive navigation. Successful benchmarks share several guiding principles: (1) they identify the common ground across existing work; (2) they operate at an appropriate level of abstraction to ensure generalizability; and (3) they remain easy to use and configurable. Motivated by these principles and given the fragmented state of mobile robot pushing research, we present Bench-Push, the first comprehensive suite of tools for standardized training and evaluation of algorithms for pushing-based mobile robot tasks. Bench-Push will also be made available as an open-source Python library.

Our key contributions from Bench-Push are as follows. First, we provide a range of simulated environments inspired by existing works in pushing-based mobile robot navigation and manipulation tasks. The environments are configurable

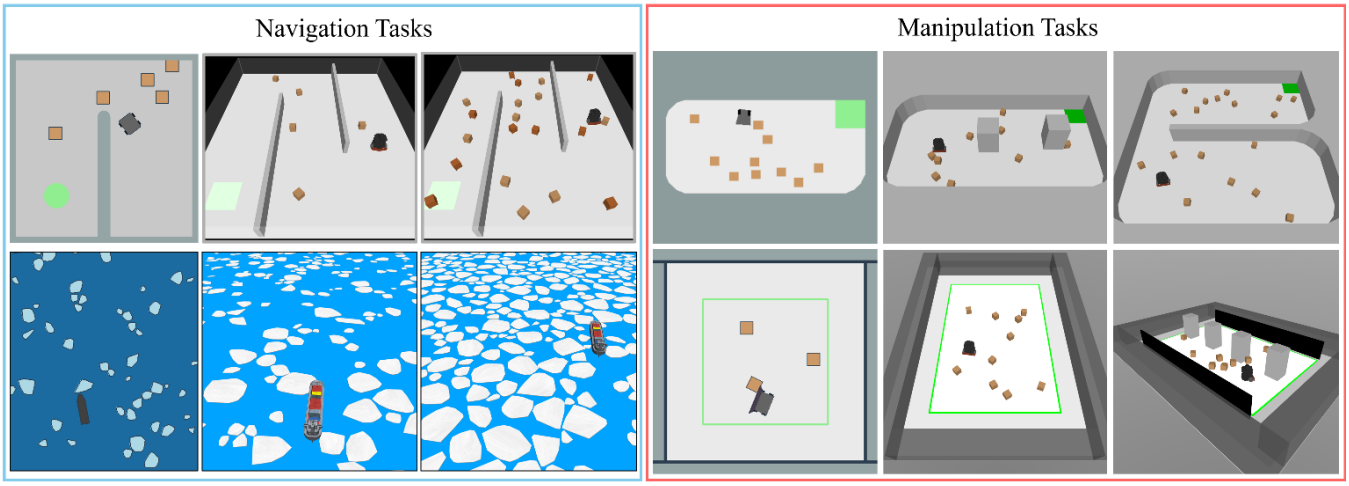

Fig. 2: Illustrations of pushing-based navigation and manipulation environments in both 2D and 3D simulations, each with configurable complexity levels. *Maze* (top-left): the mobile robot must reach the goal point among movable obstacles. *Ship-Ice* (lower-left): an autonomous ship must navigate through 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.

by users to modify the difficulty. For each environment, we provide high-fidelity 3D simulations implemented using MuJoCo [10] to ultimately enable physical robot deployment, as shown in Fig. 1. We additionally provide 2D Pymunk [11] simulations for rapid prototyping.

Second, we 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.

Third, 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 [12], PPO [13]) and task-specific methods such as Spatial Action Maps (SAM) [5] and ASV path planners [14], [15].

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.

II. Related Work

In this section, we review related work in pushing-based mobile robot tasks and the most closely related benchmarks.

A. Pushing-based Navigation and Manipulation Tasks

Pushing-based navigation tasks focus on navigating the robot to a goal location while pushing path-blocking obstacles aside. Recent work primarily focuses either on improving the robot's navigation adaptability in environments with pushable objects or on reducing total pushing effort. For instance, [16] leveraged online sensor feedback for adaptive pushing action selection, while [4] improved maneuverability by learning non-axis-aligned pushing. To reduce pushing effort during navigation, [17] studied minimizing the number of pushes required for a mobile robot to move obstacles, while [6] aims to reduce accumulated pushing force for navigation. Beyond terrestrial robots, [14], [15] studied autonomous surface vehicle (ASV) navigating ice-covered waters, proposing planners that reduce ice pushing.

Pushing-based manipulation tasks emphasize manipulating the movable objects in the environment to reach desired configurations. Existing work on the control level focuses on stable pushing by modeling robot-object contact. For example, [18] studied maintaining a stiff robot-object contact for stable pushing, while a subsequent work [19] achieves stable pushing while the object is sliding. Other works focus on the policy level to coordinate actions to push all objects toward some configurations. In [5], 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 [20], a full-stack framework is developed for robots to autonomously push and collect trolleys in public areas.

The emergence of these diverse approaches with shared objectives underscores the need for a unified benchmark.

B. Mobile Robot Benchmarks

Traditional mobile robot benchmarks focus primarily on collision-free navigation. BARN [21] evaluates navigation in cluttered environments, while [22] and [23] 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 [9] introduces interactable environments and is perhaps the closest to our work. Our benchmark differs from Gibson [9] in three ways. 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 [21], [22] 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 [9] 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.

III. 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 [24] interface. Each environment is simulated in both 3D through MuJoCo [10] and 2D through Pymunk [11], and has multiple variations with increasing complexity (see Fig. 2). 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.
- *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 I.

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

A. 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 II. Further demonstrations can also be found in the supplemental video.

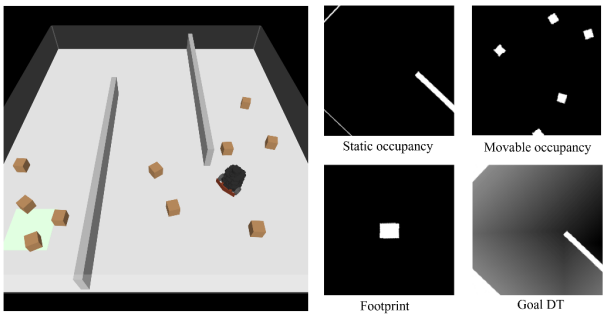

Fig. 3: The *Maze* environment (left) and an example egocentric observation for *Maze* (right).

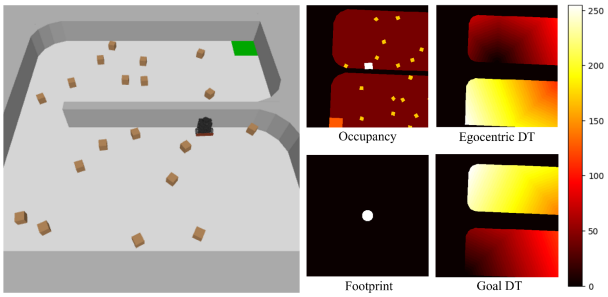

Fig. 4: The *Box-Delivery* environment (left) and an example observation for *Box-Delivery* (right).

1) Navigating maze with movable obstacles (Maze): Existing work in pushing-based navigation has frequently adopted maze-like environments with bounded rectangular spaces and walls [17], [6], [4]. 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. 2 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 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.

2) 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 [14], [15]. 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. 2 bottom-left). The

TABLE I: Interactive pushing-based tasks included in Bench-Push.

Environments	Task Class	Variations	Baselines Evaluated
Maze	Navigation-centric	maze complexity, obstacle count	SAC [12], PPO [13], RRT [25]
Ship-Ice	Navigation-centric	ice concentrations	SAC [12], PPO [13], ASV planning [14], [15]
Box-Delivery	Manipulation-centric	w/o static obstacles, box count	SAC [12], PPO [13], SAM [5]
Area-Clearing	Manipulation-centric	w/o static obstacles, box count	SAC [12], PPO [13], SAM [5], GTSP (see project repo)

TABLE II: 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

ship's task is to reach a horizontal goal line ahead of the ship while minimizing collisions with broken ice floes in the channel [14]. 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 [anonymous for review], 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 [14].

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.

3) Delivering boxes to a receptacle (Box-Delivery): Existing work in pushing-based manipulation has primarily focused on either contact-level control of pushing actions [18], [19], or task-level policies for moving all objects toward target configurations [5], [20], with some positioned in between [26]. 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. III-A.4). The *Box-Delivery* environment consists of a set of movable boxes and a designated *receptacle* (Fig. 2 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. 4 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.

4) Clearing boxes from an area (Area-Clearing): The *Area-Clearing* environment consists of a set of movable boxes and a *clearance area* (Fig. 2 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.

B. 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.

1) Additional Features and Configurability: In addition to the action spaces presented in Table II, 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. 2 top-left with dark and light brown boxes.

2) Simplified 2D Simulations: All environments detailed in Sec. III-A are supported with a simplified simulation through Pymunk 2D [11]. 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 [14] with no fluid dynamics. Detailed comparisons between the 2D and 3D environments can be found in the project repository.

3) Gymnasium Integration: We integrate all environments within the Gymnasium [24] 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.

4) 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. IV. These examples demonstrate both how to integrate popular Reinforcement Learning algorithms from Stable Baselines 3 [27] 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://anonymous.4open.science/r/BenchPush-CDF6/README.md.

C. Metrics

We now provide metrics to evaluate policies for the above tasks. These metrics are inspired by [9] 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.

1) 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

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

From Eq. 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 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:

$$I_{\text{nav}} = \frac{\mu m_0 g l_0}{\sum_{i=0}^{K} \mu m_i g l_i} = \frac{m_0 l_0}{\sum_{i=0}^{K} m_i l_i}. \tag{2}$$

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. 5 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.

2) 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*

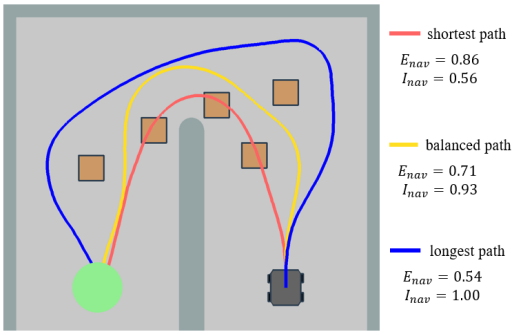

Fig. 5: Three teleoperated *Maze* paths and their performance. While the shortest path (red) has the highest efficiency score $E_{\text{nav}} = 0.86$, excessive collisions with the movable objects degrade the effort score $I_{\text{nav}} = 0.56$. In contrast, while the longest path (blue) is collision-free $I_{\text{nav}} = 1.00$, efficiency is largely compromised $E_{\text{nav}} = 0.54$. A balanced path (yellow) potentially offers the best trade-off.

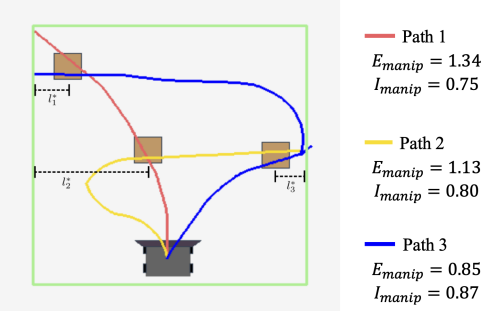

Fig. 6: Three teleoperated paths for the area clearing task, where all boxes must be removed from an area (green square). Each path has a task success score $S_{\text{manip}} = 2/3$. While path 1 (red) has the highest efficiency score, its effort score is low as it moves a box longer than its shortest 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.

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

$$S_{\text{manip}} = \frac{K'}{K}. \tag{3}$$

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

$$E_{\text{manip}} = \frac{L^*(O')}{l_0}. \tag{4}$$

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). 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

$$I_{\text{manip}} = \frac{m_0 l_0 + \sum_{i=1}^{K} \mathbb{1}^i_{\text{success}} m_i l_i^*}{\sum_{i=0}^{K} m_i l_i}. \tag{5}$$

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. 6 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.

IV. 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. III-A. 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.

A. 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 III-A and (ii) task-specific baselines. Table I summarizes the baselines included for each task. The reinforcement learning (RL) baselines include Soft Actor-Critic (SAC) [12] and Proximal Policy Optimization (PPO) [13]. These baselines were implemented using Stable Baselines 3 [27] and trained across all tasks. To handle image observations, both baselines employ a customized

ResNet18 [28] 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 [25], 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 [5], 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 [14] and predictive-planning [15] 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.

B. 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 (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.

C. 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 III 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.

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

# Obs	Platform	Alg.	E_{nav} ↑	I_{nav} ↑
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

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

D. 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. 2. 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 IV. 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.

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.

V. Conclusion, Limitations and Future Work

We introduce Bench-Push, the first comprehensive benchmark for pushing-based navigation and manipulation tasks

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

# Obs	Platform	Alg.	S_{manip} ↑	E_{manip} ↑	I_{manip} ↑
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

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.

A current limitation of Bench-Push is its reliance on bird's-eye environment observations and access to the knowledge of which objects are pushable. Future extensions will incorporate onboard sensing and diverse observation modalities to extend to more realistic perception pipelines.

We recognize that no single benchmark can meet the needs of all researchers in mobile robot pushing. Nonetheless, given the importance of this area and the absence of standardized tools, we believe Bench-Push addresses a timely need. By providing shared environments, metrics, and baselines, we hope it will serve as a foundation for advancing research on pushing-based navigation and manipulation and inspire further progress in this emerging field.

REFERENCES

[1] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *International Journal of Humanoid Robotics*, vol. 2, no. 04, pp. 479–503, 2005.

[2] A. Petrovskaya and A. Y. Ng, "Probabilistic mobile manipulation in dynamic environments, with application to opening doors." in *IJCAI*, 2007, pp. 2178–2184.

[3] M. Stilman, K. Nishiwaki, S. Kagami, and J. J. Kuffner, "Planning and executing navigation among movable obstacles," *Advanced Robotics*, vol. 21, no. 14, pp. 1617–1634, 2007.

[4] L. Yao, V. Modugno, A. M. Delfaki, Y. Liu, D. Stoyanov, and D. Kanoulas, "Local path planning among pushable objects based on reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2024, pp. 3062–3068.

[5] J. Wu, X. Sun, A. Zeng, S. Song, J. Lee, S. Rusinkiewicz, and T. Funkhouser, "Spatial action maps for mobile manipulation," *arXiv preprint arXiv:2004.09141*, 2020.

[6] J. J. Weeda, S. Bakker, G. Chen, and J. Alonso-Mora, "Pushing through clutter with movability awareness of blocking obstacles," *arXiv preprint arXiv:2502.20106*, 2025.

[7] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in *IEEE International Conference on Advanced Robotics*, 2015, pp. 510–517.

[8] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," *arXiv preprint arXiv:2009.12293*, 2020.

[9] F. Xia, W. B. Shen, C. Li, P. Kasimbeg, M. E. Tchapmi, A. Toshev, R. Martín-Martín, and S. Savarese, "Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 713–720, 2020.

[10] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[11] V. Blomqvist, "Pymunk: A easy-to-use pythonic rigid body 2d physics library. (version 6.7.0)," 2024. [Online]. Available: https://pymunk.org

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[14] R. de Schaetzen, A. Botros, N. Zhong, K. Murrant, R. Gash, and S. L. Smith, "Auto-icenav: A local navigation strategy for autonomous surface ships in broken ice fields," *arXiv preprint arXiv:2411.17155*, 2024.

[15] N. Zhong, A. Potenza, and S. L. Smith, "Autonomous navigation in ice-covered waters with learned predictions on ship-ice interactions," in *IEEE International Conference on Robotics and Automation*, 2025, pp. 10 157–10 163.

[16] B. He, G. Chen, W. Wang, J. Zhang, C. Fermuller, and Y. 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*, 2024, pp. 5402–5409.

[17] Z. Ren, B. Suvonov, G. Chen, B. He, Y. Liao, C. Fermuller, and J. Zhang, "Search-based path planning in interactive environments among movable obstacles," *arXiv preprint arXiv:2410.18333*, 2024.

[18] Y. Tang, H. Zhu, S. Potters, M. Wisse, and W. Pan, "Unwieldy object delivery with nonholonomic mobile base: A stable pushing approach," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7727–7734, 2023.

[19] Y. Tang, M. Wisse, and W. Pan, "Unwieldy object delivery with nonholonomic mobile base: A free pushing approach," *IEEE Robotics and Automation Letters*, 2024.

[20] P. Xie, B. Xia, A. Hu, Z. Zhao, L. Meng, Z. Sun, X. Gao, J. Wang, and M. Q.-H. Meng, "Autonomous multiple-trolley collection system with nonholonomic robots: Design, control, and implementation," *Journal of Field Robotics*, vol. 42, no. 1, pp. 20–36, 2025.

[21] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2020, pp. 116–121.

[22] Z. Xu, B. Liu, X. Xiao, A. Nair, and P. Stone, "Benchmarking reinforcement learning techniques for autonomous navigation," in *IEEE International Conference on Robotics and Automation*, 2023, pp. 9224–9230.

[23] L. Kästner, T. Bhuiyan, T. A. Le, E. Treis, J. Cox, B. Meinardus, J. Kmiecik, R. Carstens, D. Pichel, B. Fatloun *et al.*, "Arena-bench: A benchmarking suite for obstacle avoidance approaches in highly dynamic environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9477–9484, 2022.

[24] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[25] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[26] F. Bertoncelli and L. Sabattini, "Streamlining object pushing: Behavior tree-based coordination of control and planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2024, pp. 5203–5210.

[27] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.