arXiv:2303.02407v1 [cs.RO] 4 Mar 2023

Local Navigation Among Movable Obstacles with Deep Reinforcement Learning

Linghong Yao, Valerio Modugno, Yuanchang Liu, Danail Stoyanov, and Dimitrios Kanoulas

***Abstract*— Autonomous robots would benefit a lot by gaining the ability to manipulate their environment to solve path planning tasks, known as the Navigation Among Movable Obstacle (NAMO) problem. In this paper, we present a deep reinforcement learning approach for solving NAMO locally, near narrow passages. We train parallel agents in physics simulation using an Advantage Actor-Critic based algorithm with a multi-modal neural network. We present an online policy that is able to push obstacles in a non-axial-aligned fashion, react to unexpected obstacle dynamics in real-time, and solve the local NAMO problem. Experimental validation in simulation shows that the presented approach generalises to unseen NAMO problems in unknown environments. We further demonstrate the implementation of the policy on a real quadrupedal robot, showing that the policy can deal with real-world sensor noises and uncertainties in unseen NAMO tasks.**

I. INTRODUCTION

Studied extensively by Mike Stilman [1], the Navigation Among Movable Obstacle (NAMO) problem tackles planning problems where obstacles can be manipulated via pushing, pulling, or lifting to aid a robot's navigation in cluttered environments. Just like how humans intuitively nudge and move furniture out of their way when walking through a tightly packed room, robots can also become much more useful if they can manipulate their surroundings to help them reach their target location. Practical use cases for NAMO solutions include robots that perform routine maintenance of factories where idle containers and boxes may obstruct doorways; personal service robots in households where rooms are packed with furniture and items; and industrial inspection robots in underground caves and tunnels where rocks and debris may block the walkway. In such environments, the ability to reliably manipulate obstacles will significantly increase the efficiency of autonomous robots.

However, even simplified versions of the NAMO problem have been proven to be NP-hard [2], [3]. Past literature has tackled the NAMO problem using algorithms based on iterative and recursive methods [4], [5], [6], but usually with simplifications to the problem setting, such as prior knowledge of the environment, offline planning, and axial-aligned manipulation. Only a few studies have tackled the online NAMO problem and considered the implications of sensor errors and unexpected object dynamics [7], [8]. Furthermore, methods utilizing recursive and iterative search-based techniques usually result in a computational complexity that is exponential to the number of local obstacles [5], and the search time in practice can sometimes be too long when many obstacles are present.

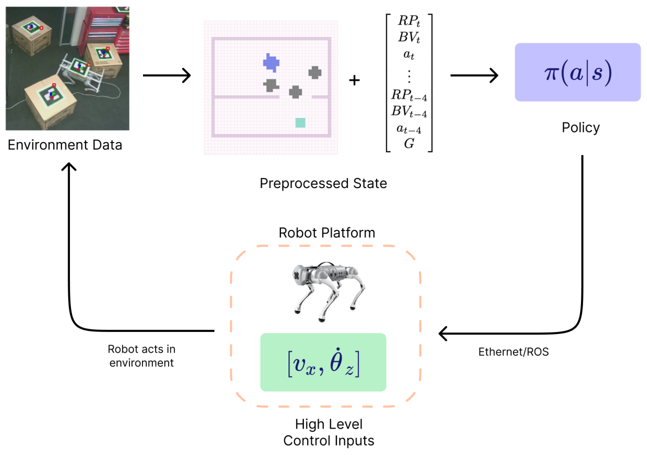

Environment Data
Preprocessed State
$\begin{bmatrix} RP_t \\ BV_t \\ a_t \\ \vdots \\ RP_{t-4} \\ BV_{t-4} \\ a_{t-4} \\ G \end{bmatrix}$
+
$\pi(a|s)$
Policy
Robot Platform
$[v_x, \dot{\theta}_z]$
High Level Control Inputs
Robot acts in environment
Ethernet/ROS

Fig. 1. The environment observation is obtained via visual sensors and preprocessed into the desired state format. The input is fed into a trained policy network that computes a high-level action output for the robot to follow in the environment, in order to solve the local NAMO task.

In this paper, we aim to address some of these shortcomings using a deep reinforcement learning approach (Fig. 1). By utilizing a neural network for policy-based RL algorithms, we develop an online agent that can solve unseen NAMO tasks without the constraint of axis-aligned pushing, and is able to handle uncertainties in sensor inputs and obstacle dynamics. Specifically, we focus on solving keyhole problems[1] near one or two narrow passages. We also constrain the problem to use only pushing actions, as this is the most universal manipulation method amongst mobile robots – pulling and lifting obstacles might need extra robotic equipment such as arms. Our proposed method is suitable to be implemented within workflows where global path planning algorithms such as A* [9] would be unable to resolve local problems that require NAMO solutions. Therefore, we formulate the environment in a small region near narrow passages, with the objective of the robot reaching from one disjointed space to the other via obstacle manipulation. Such

[1]The authors are with the Department of Computer Science and Department of Mechanical Engineering, University College London, Gower Street, WC1E 6BT, London, UK. {leon.yao.18, v.modugno, yuanchang.liu, danail.stoyanov, d.kanoulas}@ucl.ac.uk

This work was supported by the UKRI Future Leaders Fellowship [MR/V025333/1] (RoboHike) and the Engineering and Physical Sciences Research Council (EPSRC) [EP/P012841/1]. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

[1]Defined in [1], keyhole is the problem of moving one or more objects to connect two disjointed spaces to solve local NAMO tasks.

framing of the problem removes the necessity for the RL agent to learn how to perform path planning with obstacle avoidance. The goal of the policy is only to get through the local narrow passage using obstacle manipulation, such that reliable path tracking can resume.

We utilize a policy based on Advantage Actor-Critic algorithm [10], and we use state-of-the-art physics engine NVIDIA Isaac Gym [11] to simulate and train parallel agents in obstructed narrow passages. We present results for both policies trained to solve unseen obstacle positions in known environments, and unseen obstacle positions in unknown environments. We further demonstrate real-world robot experiments conducted on a Unitree Go1 quadrupedal robot to show that the policy is able to handle sensor noises and real-world obstacle dynamics in practice. The rest of this paper is organized as follows. In Sec. II, we discuss the literature on the NAMO problem, while in Sec. III we state the problem formulation in the reinforcement setting, and how we implement training in simulation. In Sec. IV we present our results both in simulation and with real robot experiments. Lastly, Sec. V concludes the paper and points to future directions for solving NAMO with RL.

II. RELATED WORK

The complexity of motion planning with movable obstacles was first analyzed by [2], where simplified variations of NAMO have been proven to be NP-hard. Demaine et al. [3] further showed that problems involving square blocks constrained to planar translations are still NP-hard. Stilman et al. [1] first formulated the navigation among movable obstacle problem in 2005, and proposed a graph-based planner that decomposes the global problem into a set of subclass keyhole problems called LP_1, where disjointed free spaces can be connected by moving a single obstacle. The search space of NAMO was greatly reduced via this decomposition and was used to generate a complete solution. This work was extended in [4], which proposed a planner that can solve LP_k subproblems, where k obstacles need to be moved independently to connect two disjointed free spaces. Nieuwenhuisen et al. [6] proposed a method based on Rapidly-exploring Random Trees (RRT), where the node expansion process is probabilistic and is adaptively guided by a heuristic. A probabilistically complete algorithm was proposed by Berg et al. [12], in which the computation of robot's motions is decoupled with the obstacle movements and the tree-based method finds the sequence of axis-aligned obstacle movements that connects disjointed spaces between the robot and the goal position. In recent works, Moghaddam et al. [5] proposed a recursive algorithm capable of solving nonlinear and non-monotone problems with axis-aligned object manipulations, and a quadruped robot applied pushing actions to free space in [13] via search-based methods.

In the aforementioned approaches, planning is performed offline with complete prior knowledge of the environment and known movability of obstacles. The computational complexity of these methods grows exponentially to the number of obstacles, which is inefficient in practice as the number

TABLE I
COMPARISON TO ONLINE SOTA METHODS

Method	Uncertainties	Manipulation	Non-Axis-aligned	Real-world
[14]	None	Push	No	No
[15]	Pos. Mov.	Grasp	No	**Yes**
[8]	**Pos. Mov. Kin.**	Motion Primitives	**Yes**	**Yes**
Ours	**Pos. Mov. Kin.**	Push	**Yes**	**Yes**

of obstacles in the scene increases. Wu et al. [14] examined realistic situations where the robot must navigate through an unknown environment with unknown object movability. The robot is constrained to only pushing actions, and the authors proposed an efficient online method where re-planning is only performed when new information changes the optimality of the current plan. Kakiuchi et al. [7] devised an action strategy and performed real-world testing on humanoid robots for unknown environments and unknown object movability. In Levihn et al. [15], the authors introduced a novel method based on Hierarchical Reinforcement Learning and Monte Carlo Tree search and performs planning online with uncertain sensor information. The proposed method is shown to have approximately linear computational complexity with respect to the number of obstacles. Non-axial manipulation of obstacles is examined in [8] by utilizing a physics-based reinforcement learning framework to adapt to unexpected obstacle behaviors such as rotation. Further extensions to NAMO, such as socially aware obstacle placement, have also been examined in [16], [17], [18].

In this paper, we propose a Deep Reinforcement Learning (DRL) approach. There have been numerous recent studies in which agents trained with DRL have been shown to generalize to a wide range of tasks in simulation, including 2D arcade games [19], multi-agent collaborative and competitive tasks [20], and navigation tasks [21]. These studies have shown promising results in agents performing highly difficult tasks with very little to no prior knowledge about the tasks. Xia et al. [22] trained vision-based DRL agents in home settings with interactive objects. The authors presented agents which can efficiently optimize trade-off between path-efficiency and object interaction but did not focus specifically on NAMO tasks where object manipulation is necessary for agents to reach goal position. To the best of our knowledge, we are the first to utilize deep reinforcement learning to solve the NAMO problem. Table I compares our method to other state-of-the-art online NAMO methods. We highlight that our approach removes the constraint of axis-aligned manipulation, and imposed positional (Pos.), movement (Mov.), and kinematics (Kin.) uncertainties in the environment. We further note that compared to [8], our method runs in constant time complexity and therefore completes a typical similar task about five times faster. Furthermore, our method is capable of solving harder problems such as non-linear problems, which we will demonstrate in Sec. III.

The main contributions of this work are:

- We propose a reinforcement learning policy that can solve local NAMO problems with non-axial-aligned pushing.

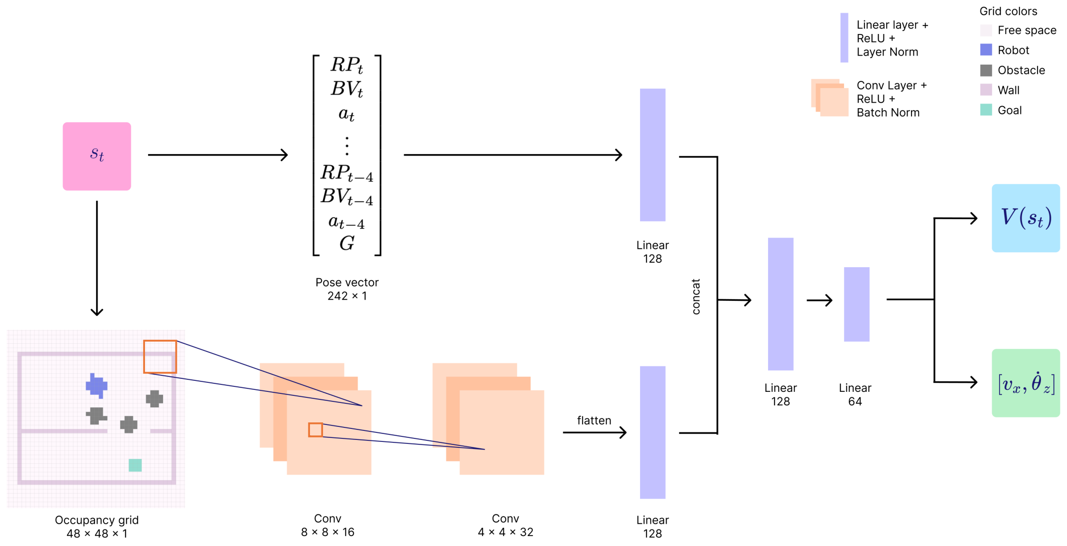


Fig. 2. The presented policy uses a multi-modal neural network to handle both image and vector inputs. The state s_t, shown on the left side of the diagram, consists of a vector and a grid. The vector is 242 in length, containing information about the agent's pose, box vertices, previous action, and goal position. The grid consists of 48 cells semantically labeled. The vector is first passed through a linear block, whilst the grid image is passed forward via two convolution blocks and a linear block. The outputs of the two streams are concatenated, then passed through two more linear blocks. Finally, the network diverges via two separate sets of weights to output the value estimate and the action output shown on the right end of the diagram.

- We demonstrate that the proposed policy is able to generalize to unseen obstacle positions in known environments, and has the potential to generalize to unseen obstacle positions in unknown environments.
- We develop a framework for generating local NAMO problems to train agents in parallel simulated environments, which can be used to benchmark future studies.
- Lastly, we show that reliable sim-to-real transfer of the proposed method is possible, and the policy can handle sensor noises and uncertain dynamics.

III. METHODS

A. RL Background

We consider a typical episodic reinforcement learning setting where the agent interacts with the environment over discrete time steps. At every time step, the agent receives state s_t and samples an action a_t from the policy distribution $\pi(a_t|s_t;\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the parameter of some function approximation. The agent then receives the next state s_{t+1} and a scalar reward r_t. In policy-based settings, the objective is to find the parameter $\boldsymbol{\theta}$ which maximizes the expected cumulative return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ with discount factor $\gamma \in (0, 1]$.

In Advantage Actor-Critic methods, the algorithm computes both a policy $\pi(a_t|s_t;\boldsymbol{\theta})$ and a value function $V(s_t;\boldsymbol{w})$. The value function estimates the expected return $\mathbb{E}_\pi[R_t|s_t = s]$ by following the current policy π from s_t. The value function V, also known as the critic, updates via the parameter $\boldsymbol{w}$ whilst the policy π, also known as the actor, updates via the parameter $\boldsymbol{\theta}$.

We implemented a deep neural network for function approximation and updated the parameters with stochastic gradient descent. The parameters are updated using

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t;\boldsymbol{\theta}) A(s_t, a_t; \boldsymbol{w}) \tag{1}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_w \nabla_w V(s_t;\boldsymbol{w}) A(s_t, a_t; \boldsymbol{w}) \tag{2}$$

where α and α_w represent the learning rate of the policy and value function, respectively. Note that $A(s_t, a_t; \boldsymbol{w}) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\boldsymbol{w}) - V(s_t;\boldsymbol{w})$ computes the n-step advantage for the state-action pair (s_t, a_t), where k determines the number of steps to look ahead.

In practice, the actor and critic parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ share a large set of weights, as shown in Fig. 2 to help stabilize the learning. In addition, we add an entropy term $\nabla_\theta H(\pi(s_t;\boldsymbol{\theta}))$ to Eq. (1) and Eq. (2) as recommended by [10] to help regularize learning. We also use a clipped surrogate objective from [23]:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

where ϵ is a hyper-parameter to be set. Updating policies in such ways further improves the stability of training.

B. Problem Formulation

In this section, we describe the formulation of the reinforcement learning problem. We tackle local NAMO keyholes where the objective is to connect two disjointed, adjacent free spaces separated by a set of obstacles. We define an agent which interacts with the environment over a set of time steps with continuous action space in forward velocity v_x and angular velocity $\dot{\theta}_z$. The agent receives a reward r_t from the environment at each time step, and the objective is to maximise the cumulative return $\sum_{k=0}^{\infty} \gamma^k r_{t+k}$.

The environment consists of a small local region that includes a narrow passage and several obstacles near the passage. We set the goal behind the passage. The episode ends if the agent either reaches the goal position or if the maximum episode length is exceeded.

We assume the following upstream inputs are present when constructing the state of the agent: a semantically labeled,

TABLE II

REWARD GIVEN AT EACH TIME STEP

reward	description	weight
goal	1 if reach goal, 0 otherwise	10
progress	[-1, 1] $\propto$ velocity towards goal	1
dist	[0, 1] $\propto$ distance to goal	0.1
wall collision	-1 if collision with wall	0.2
box collision	-1 if collision with box	0.1
vel effort	$[-1, 0] \propto v_{target}$ target velocity	0.05
rot effort	$[-1, 0] \propto \dot{\theta}_{target}$	0.1
vel offset	$[-1, 0] \propto \lvert v_{actual} - v_{target} \rvert$	0.2
rot offset	$[-1, 0] \propto \lvert \dot{\theta}_{actual} - \dot{\theta}_{target} \rvert$	0.1
time	-1	1

coarse occupancy grid, which can be obtained via LIDAR or camera sensors and semantic segmentation methods [24]; bounding boxes detection on obstacles, which can be obtained by object detection algorithms; and internal pose data about the agent's current state, which can be obtained via robot's internal sensors. We assume a degree of sensor noise from all the inputs. From these upstream inputs, we then construct the agent state which includes the goal position G, obstacle vertices BV_t, robot internal state (pos, vel, rot, ang vel) RP_t, previous action a_t, and a semantically labeled occupancy grid. We include a history of concatenated past states for RP_t, BV_t, and a_t to provide the agent with some temporal information, in a similar fashion as [19]. We choose to include the past 5 frames, as from our experiments this was proved to provide a good trade-off between performance and computational cost. The final state is formed by the occupancy grid and a vector containing data G, $RP_{t-4:t}$, $BV_{t-4:t}$, and $a_{t-4:t}$, as shown in Fig. 2.

In order to keep the policy practical and generalizable, we adopt a high-level control strategy to control the robots. We utilize a unicycle model with two degrees of freedom v_x and $\dot{\theta}_z$, i.e., linear and angular velocity. This policy can easily be applied to different robot models, as we demonstrate in our robot experiments in Sec. IV.

The objective of the agent is to maximize the cumulative return. The reward given at each time step is described in Table II. We give a large reward upon completion of the task, and no reward if the task is not completed. At each time step, a small amount of positive reward is given if the agent moves towards the goal (*progress*), or is close to the goal (*dist*). A small amount of negative reward is also given proportionally to the effort of the action taken (*vel effort*, *rot effort*), any collision with walls (*wall collision*), and any contacts from pushing obstacles (*box collision*). We also penalize the agent for inducing large offsets between target and actual actions, which typically results from collisions with other objects or abrupt changes to actions (*vel offset*, *rot offset*).

C. Algorithm Implementation

We train our agent in simulation using NVIDIA Isaac Gym, which uses PhysX as the back-end physics engine[11]. A large benefit of using Isaac Gym is that we can train parallel agents on a single GPU. Such parallel training greatly helps policy-based algorithms to produce stable learning with less training time [10].

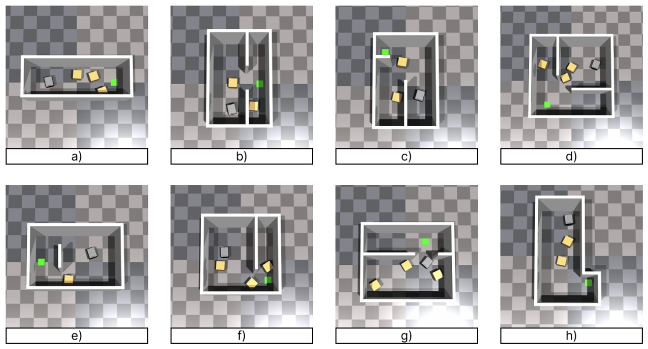


Fig. 3. Local NAMO scenes are generated with a set of fixed maps and random obstacle positions. The task of the agent (grey) is to manipulate the boxes (yellow) to reach the goal position (green). Maps (a) and (h) are used for training, representing a variety of different local NAMO environments including corridor (a,b), mid-doorway (c,d), side doorway (e,f,i), and diagonal doorway (g,h).

The advantage actor-critic method is implemented with a multi-modal neural network is designed to fuse two components of the state space, as illustrated in Fig. 2. The state includes a vector and grid component that are normalized to $[-1, 1]$. The vector input is passed through a linear block consisting of one layer of perceptron with 128 units, ReLU activation, and a normalization layer. The grid input is passed through two convolutional blocks each consisting of a convolutional filter, ReLU activation, and batch normalization. The first block contains 16 filters of 8×8, whilst the second block contains 32 filters of 4×4. The output is flattened and passed through a linear block with 128 units. The two streams are concatenated and passes through two linear blocks with 128 units and 64 units. The output is then finally connected to two separate linear layers outputting the value estimate $V(s_t)$ and the action $[v_x, \dot{\theta}_z]$, respectively.

D. Scene Generation and Curriculum Training

In Fig. 3, we show eight different map configurations that were developed in Isaac Gym. We designed these maps to cover a wide range of local NAMO settings: narrow corridor (map a and b), doorway with space to the sides (map c and d), doorway against a wall (map e, f and i), and diagonal doorways (map g and h).

Each agent is spawned in a pre-defined room with a dimension around 6×6 m^2. This is a reasonable setting for mobile robot sizes of around 0.5 to 1 meter in length to move through narrow passages that are 1 to 2 meters wide. An obstruction of obstacles near such passages would make it difficult for the robot to pass through. The robot and the goal position are randomly spawned within their own predefined area, but with a small probability (e.g., 5%) the robot can also spawn anywhere inside the map, such that the agent has a finite chance of visiting every position.

We choose boxes of size around $60cm^3$, and spawn up to 5 boxes in each room. We choose 5 boxes because it appropriately fills the room with enough variety, adding more would overfill the room and make the box position generation too difficult. Note that if obstacle positions are generated at

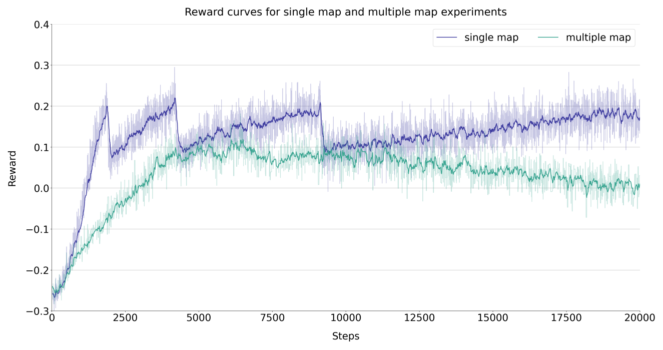


Fig. 4. The reward over policy update steps for both experiments conducted on a single map and on multiple maps are shown in purple and green, respectively. The effect of curriculum learning can be observed from sharp drops of the purple curve as the task becomes harder, whilst this effect is smoothed out in the multiple map setting as not all maps reach curriculum at the same time. The single map setting converges to a stable equilibrium, but the policy trained in multiple map setting often does not reach such equilibrium, and early stopping is performed.

random, most of the problems would be too easy, i.e. not requiring any interaction between the agent and the obstacles. Therefore, obstacle positions are picked randomly with a higher probability to spawn in difficult positions. Concretely, each obstacle i has probability λp_i to spawn in any random location in the room, with random rotation. With probability $\lambda(1-p_i)$ it spawns in pre-defined challenging positions (e.g. that blockades the narrow passage, or positioned near the narrow passage) plus some random perturbations. Finally, with probability $1-\lambda$ the obstacle does not spawn in the room, in which case their vertices are represented with zeros in the input. We typically use $p_i \in [0.2, 0.6]$ for each box, and we gradually increase the value of λ in discrete increments starting from 0.2. By randomizing the positions of obstacles, we force the agent to find a solution for new obstacle positions every time, and thereby learn to reason about the relationship between different obstacles, such as which order to push the obstacles.

The parameter λ allows us to control how many boxes spawn and thereby the difficulty of the NAMO problem. This effectively creates a curriculum training scheme where we can update λ such that the agent receives more difficult problems as it learns to solve easier ones. We set a threshold of completion rate at the current difficulty (e.g., 90%). Starting with $\lambda = 0.2$, we train the agent until its performance reaches 90% completion rate, and we increase λ by 0.2. This process repeats 4 times until λ reaches 1, upon which around 4 to 5 boxes will spawn at the start of the scene every time. The effect of this curriculum learning scheme over a training run can be seen in Fig. 4.

E. Domain Randomization

Heavy domain randomization is applied to both the state and the action space in the form of Gaussian white noise. We anticipate real-world problems to be highly noisy in both sensor inputs and object dynamics, and a good policy should be able to robustly handle such noises. By adding noises to input state space, we simulate sensor noises that the robot is likely to encounter in the real world, such as temporal fluctuations in estimated obstacle poses and robot pose. Note that we add noises to the vector state and the grid state independently, as the real-world sources of these sensor noises are usually not correlated. We add similar Gaussian noise to the action outputs of the model, in order to simulate unexpected robot dynamics. Overall, applying domain randomization in our training decreases the sim-to-real gap, and ensures that the policy is stable in the presence of noisy inputs and unexpected object dynamics, as we will demonstrate in the next section.

TABLE III
PERFORMANCE OF POLICY WHEN TRAINED ON SINGLE MAP CONFIGURATION WITH VARYING λ

λ	boxes	completion rate	time taken	boxes pushed
0	0	99.9	6.80	0
0.2	0.7	98.5	7.13	0.34
0.4	1.3	98.3	7.68	0.64
0.6	2.1	97.4	8.09	1.00
0.8	2.8	94.5	8.14	1.37
1	3.7	91.0	8.69	1.92

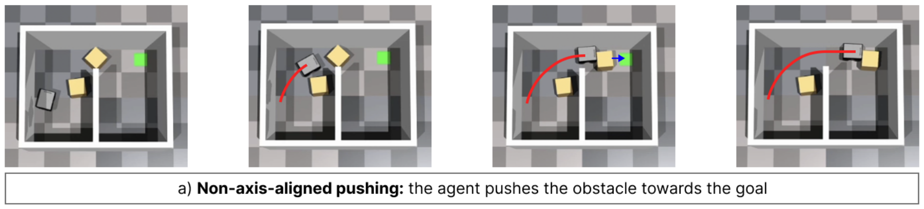
IV. EXPERIMENTS

A. Simulation

In this section, we demonstrate simulated performance of the trained policy to solve unknown problems both quantitatively and qualitatively. We train the policy with two settings, single room configuration, and multiple room configurations. In the single room setting we train the agent with random box positions in the same room (shown in Fig. 5, whilst in the multiple rooms setting the agent is trained with eight different room layouts. Our results show that in the single room configuration, we can train a policy to generalise to unseen obstacle positions in a known environment with optimized behaviours and a low failure rate. In the multiple room setting, we show that a policy can also generalise to unseen obstacle positions in unseen environments at the cost of less optimized behaviour.

The following training settings are used for both experiments. The scene generation and randomization protocols are described in Sec. III, where we randomize the positions of the agent, goal, and boxes, train with a curriculum, and apply domain randomization to the input and output space. We use ADAM as the optimizer [25], with l2 regularization to help with stability [26]. We also add gradient clipping to avoid exploding gradients, which typically occurs when off-policy learning, function approximation, and value bootstrapping are all present (deadly triad) [27]. An adaptive learning rate is utilized based on the KL threshold. We use a horizon length of 50, and policy update frequency of 20 physics steps (333ms). The maximum episode length is set to 45 seconds, or 2700 physics steps. We train $4,000$ environments in parallel with mini-batch of $2,000$ samples every update. Training is conducted using an NVIDIA RTX 3080 GPU for $20,000$ policy update steps.

In our first experiment with a single map setting, we train the policy on the same map (see Fig. 5) but randomize the initial agent, goal, and obstacle positions. We then evaluate the network with $1,000$ scenes in the same map configuration


a) **Non-axis-aligned pushing:** the agent pushes the obstacle towards the goal

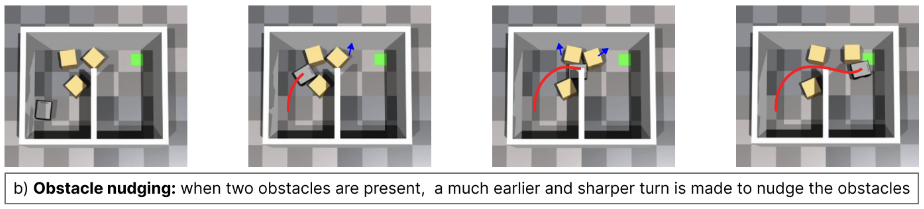

b) **Obstacle nudging:** when two obstacles are present, a much earlier and sharper turn is made to nudge the obstacles

Fig. 5. Qualitative demonstration of the policy in simulation, trained on single map setting. The agent is able to adapt to different obstacle positioning and find an efficient path through the walkway in both cases.

TABLE IV

COMPARISON BETWEEN POLICY PERFORMANCE ON TRAINING MAPS AND TEST MAP, WITH $\lambda = 0.8$

Experiment	completion rate	time taken (s)	boxes pushed
Training (8 maps)	79.8	10.99	1.12
Testing (map i)	54.3	13.05	1.51

but with unseen box positions. We repeat the evaluation six times by varying the parameter λ to increase the number of boxes in the scene, thereby increasing the difficulty of the problems.

Fig. 5 illustrates some qualitative results of the experiment. In the top row, we see that the agent is able to push obstacles along a trajectory in a non-axis-aligned fashion along its trajectory. In the second row, we highlight that the agent takes a much more narrow turn at the doorway because the most efficient way to get through is by nudging both boxes aside. This is because pushing boxes into the door will cause them to block the goal (shown in green). We highlight that the demonstrated behaviours cannot be achieved with the restriction of axis-aligned pushing, which many past works require.

The quantitative results of this experiment are shown in Table III. The rows represent increased difficulty from top to bottom as λ increases. Note that the number of boxes in the scene is not directly proportional to λ due to the random spawning of boxes, as sometimes there are no more vacancies left for new boxes to spawn. We observe that as the problem becomes harder with more boxes, the completion rate decreases often due to the agent being stuck in an unrecoverable position such as a blocked pathway or blocked goal position. In the last column of the table we list the number of mean boxes pushed when the task is completed. We note that this value is usually only half of the mean number of boxes in the scene, demonstrating that the agent only pushes boxes that are necessary for clearing the path. Overall, the observed agent's behaviour is stable and robust in the single map setting. It is able to generalise to unseen obstacle positioning to find the correct solution in unseen initial configurations 91% of the time, even in the hardest problems.

In our second experiment, we train one policy on multiple maps simultaneously and examine whether or not the trained policy is able to generalise to unseen environments as well as unseen obstacle positions. We train the agent on maps a to h, shown in Fig. 3, and test the agent on $1,000$ environments with an unseen map (same map shown used in single map setting). Our results show that the stability of learning is difficult to maintain in this setting. Despite having utilized several regularisation techniques, stable convergence of the policy is still not guaranteed on every run. Fig 4 shows that the policy is unable to reach the same performance as the single map case. Therefore, we perform early stopping to pick a policy snapshot that performs best during training.

In the multiple map setting, we notice that the agent learns strategies that generalise across different problems, namely the ability to sequentially push obstacles, and the ability to react in real-time to unexpected dynamics and input noises. Fig. 6-a shows an example of non-linear sequential pushing, where the agent must push the left box out of the way before going pushing the right box through the doorway. We highlight that the agent is able to optimize this behaviour by making only small detours along its trajectory, which would not usually be possible to achieve with axis-aligned pushing constraints. Fig 6-b highlights the ability of the agent to recover from an undesirable position. Due to limited resolution in the input image, it's often hard for the agent to precisely localize the doorway and it sometimes runs into the wall near the doorway, as shown in frame 2. However, when such expected dynamics occur in the environment, the agent adopts a strategy of backing out and moving forward again to get through the passage, shown in frames 3 and 4. Such behaviour illustrates that a less optimal policy is reached compared to the single map setting, however, the agent still displays interesting ways to recover itself from a bad state.

The quantitative results are shown in Table IV. We test the performance of the agent in known environments (same eight maps) but unknown obstacle positions, and unknown environment and unknown obstacle positions. The agent achieves around 80% completion rate across all eight maps in the hardest cases and achieves 54% completion rate in an unseen map. Whilst the completion rate on the unseen environment is relatively lower, we believe it still demonstrates

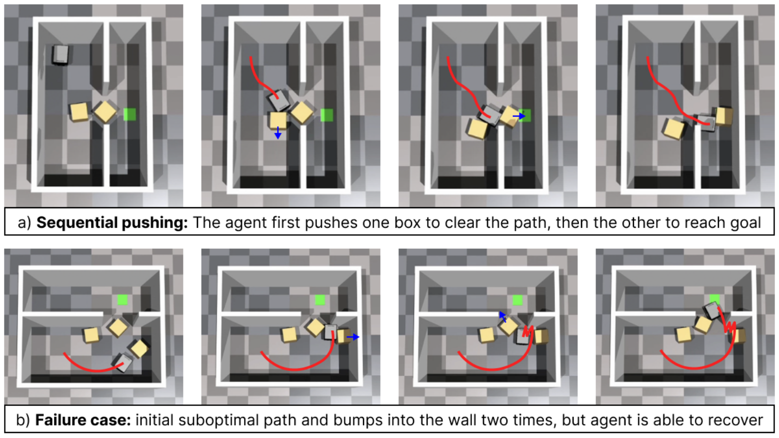

a) **Sequential pushing:** The agent first pushes one box to clear the path, then the other to reach goal
b) **Failure case:** initial suboptimal path and bumps into the wall two times, but agent is able to recover

Fig. 6. Qualitative demonstration of the policy in simulation, on multiple map setting. In a) we demonstrate that the agent is able to push obstacles sequentially whilst making minimal detours to its trajectory and in b) we demonstrate that the agent is able to recover from unexpected dynamics.

the ability of the agent to generalize to unseen environments. We hypothesise that the performance could be increased further with higher randomization in the training data (e.g., more map configurations), and by repeat experiences on harder NAMO problems with techniques such as prioritised experience replay [28].

B. *Failure Cases*

In the single map setting, most failure cases often involve the agent pushing the box into an unrecoverable position, such that the passage is completely blocked. On the other hand, in the multiple map setting, we see more failure cases. This often includes agents, taking unnecessary turns, and sometimes getting stuck in a corner of the wall as seen in Fig. 6. We believe this is caused by a combination of insufficient feature extraction from a neural network trained on highly correlated data, and a lack of resolution in the input image. The former can be improved by increasing the number of training maps and using experience replay to decorrelate the data, and the latter can be improved by increasing the input image dimension.

C. *Robot Experiments*

In this section, we present our results with a real quadruped robot. We use a neural network trained in the single map setting, as it produces more robust and stable results, and shows that the presented policy can be implemented in real-world where uncertain robot dynamics and sensor inputs are present.

The experiments are performed on a quadrupedal Unitree Go1 robot. Although in simulation we used a wheeled robot, the high-level control outputs of the network allow us to easily transfer the learned policy to a different robot. The only modifications we added to the robot are the aluminum struts on top to ensure that obstacle pushing is possible. We use cardboard boxes as obstacles, with size of around $50 \times 50 \times 50 cm^2$, which is slightly smaller than those used in training. Note that due to friction between the boxes and the carpet floor, it's very difficult for the robot to push more than one box at once. The state space is constructed from external sensors with two bird's eye view cameras to track obstacles and the robot using ArUco Markers [29]. From the camera readings, we derive the robot pose, the obstacle vertices, and compute a 2D grid to construct the state space. We note that the above can easily be derived by robot internal sensors such as camera, IMU, and LIDAR. The implementation of obstacle detection and object mapping is outside the scope of this paper, and we only show that the robot is able to handle sensor noises and unexpected obstacle behaviours.

Our testing shows that the robot can consistently navigate through the narrow passage in the presence of obstacles. In Fig. 7 we show snapshots from a video recording of the robot performing a NAMO task. We present an example NAMO problem to the robot with three obstacles, where at least two individual boxes need to be pushed to clear the path. The robot is tasked to leave the room, and reach the target location shown in the green dot. The robot immediately chooses a trajectory that avoids collision with obstacle 3 (at $t = 4$). It then starts to move box 2 via rotational pushing just enough for an opening to occur (at $t = 10$). The robot then proceeds to push box 1 out of the doorway until it reaches the goal position (at $t = 23$). In our experiments, we observe that it's very typical for the robot to perform pushing with only small nudges to rotate the obstacle, which is often the most efficient way to manipulate obstacles to create a small opening. We also note that almost all of the pushing by the robot follow non-linear trajectories, and the robot usually guides obstacles along a curved trajectory. This is also the more efficient way for the robot to simultaneously manipulate obstacles whilst adjusting its heading towards the goal.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we presented a deep learning method to solve the Navigation Among Movable Obstacles (NAMO) problem locally. Our approach allows the robot to perform non-axial pushing to solve LP_k NAMO problems with constant computational complexity. We further demonstrated that the method can be implemented on a real robot with sensor noises. Our future works will be focused on stabilising

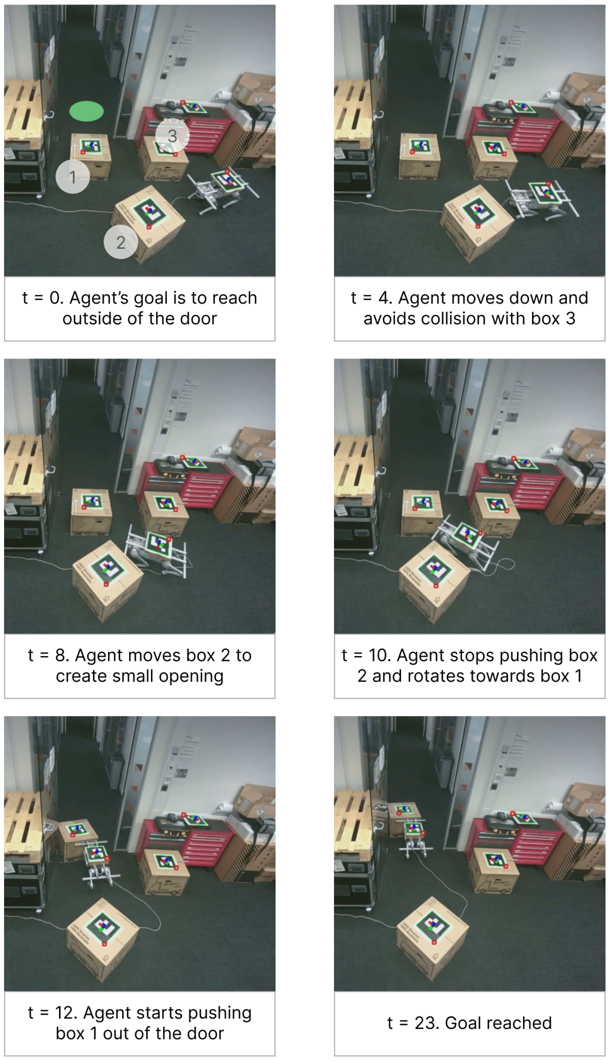

1
2
3
t = 0. Agent's goal is to reach outside of the door
t = 4. Agent moves down and avoids collision with box 3
t = 8. Agent moves box 2 to create small opening
t = 10. Agent stops pushing box 2 and rotates towards box 1
t = 12. Agent starts pushing box 1 out of the door
t = 23. Goal reached

Fig. 7. Robot experiment conducted with a quadrupedal robot, the goal position is shown as a green dot to be just outside of the room. We show that the robot performs non-axis-aligned obstacle manipulation by first nudging box 2 then subsequently pushing box 1 along its trajectory to create an opening.

learning for agents trained in multiple maps, in order to create a generalised agent that can solve NAMO problems in any map configurations. We will also investigate how agents can deal with obstacles with unknown movability and unseen shapes by incorporating these complexities in the training.

References

[1] M. Stilman and J. J. Kuffner, "Navigation Among Movable Obstacles: Real-Time Reasoning in Complex Environments," *World Scientific IJHR*, vol. 2, no. 04, pp. 479–503, 2005.

[2] G. Wilfong, "Motion Planning in the Presence of Movable Obstacles," in *4th Annual Symp. on Computational Geometry*, 1988, pp. 279–288.

[3] E. D. Demaine, M. L. Demaine, and J. O'Rourke, "Pushpush and push-1 are np-hard in 2d," *12th Canadian Conference on Computational Geometry*, pp. 211–219, 2000.

[4] M. Stilman and J. Kuffner, "Planning Among Movable Obstacles with Artificial Constraints," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1295–1307, 2008.

[5] S. K. Moghaddam and E. Masehian, "Planning robot navigation among movable obstacles (namo) through a recursive approach," *Journal of Intelligent & Robotic Systems*, vol. 83, pp. 603–634, 2016.

[6] D. Nieuwenhuisen, A. Stappen, and M. H. Overmars, "An Effective Framework for Path Planning Amidst Movable Obstacles," in *Springer Algorithmic Foundation of Robotics VII*, 2008, pp. 87–102.

[7] Y. Kakiuchi, R. Ueda, K. Kobayashi, K. Okada, and M. Inaba, "Working with Movable Obstacles using On-line Environment Perception Reconstruction using Active Sensing and Color Range Sensor," in *IEEE/RSJ IROS*, 2010, pp. 1696–1701.

[8] J. Scholz, N. Jindal, M. Levihn, C. L. Isbell, and H. I. Christensen, "Navigation Among Movable Obstacles with Learned Dynamic Constraints," in *IEEE/RSJ IROS*, 2016, pp. 3706–3713.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *PMLR International Conference on Machine Learning*, 2016, pp. 1928–1937.

[11] V. Makoviychuk *et al.*, "Isaac Gym: High Performance GPU-Based Physics Simulation for Robot Learning," *arXiv preprint arXiv:2108.10470*, 2021.

[12] J. v. d. Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha, "Path Planning Among Movable Obstacles: a Probabilistically Complete Approach," in *Springer Algorithmic Foundation of Robotics VIII*, 2009, pp. 599–614.

[13] V. S. Raghavan, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Reconfigurable and Agile Legged-Wheeled Robot Navigation in Cluttered Environments With Movable Obstacles," *IEEE Access*, vol. 10, pp. 2429–2445, 2022.

[14] H. N. Wu, M. Levihn, and M. Stilman, "Navigation Among Movable Obstacles in Unknown Environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1433–1438.

[15] M. Levihn, J. Scholz, and M. Stilman, "Hierarchical Decision Theoretic Planning for Navigation Among Movable Obstacles," in *Springer Algorithmic Foundations of Robotics X*, 2013, pp. 19–35.

[16] B. Renault, J. Saraydaryan, and O. Simonin, "Towards S-NAMO: Socially-aware Navigation Among Movable Obstacles," in *Springer Robot World Cup*, 2019, pp. 241–254.

[17] K. Ellis, H. Zhang, D. Stoyanov, and D. Kanoulas, "Navigation Among Movable Obstacles with Object Localization using Photorealistic Simulation," in *IEEE/RSJ IROS*, 2022.

[18] K. Ellis *et al.*, "Navigation Among Movable Obstacles via Multi-Object Pushing Into Storage Zones," *IEEE Access*, vol. 11, pp. 3174–3183, 2023.

[19] V. Mnih *et al.*, "Human-level Control through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[20] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[21] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu *et al.*, "Learning to Navigate in Complex Environments," *arXiv preprint arXiv:1611.03673*, 2016.

[22] 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 RA-L*, vol. 5, no. 2, pp. 713–720, 2020.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[24] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping," in *IEEE ICRA*, 2020, pp. 1689–1696.

[25] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd ICLR*, 2014.

[26] J. Farebrother, M. C. Machado, and M. Bowling, "Generalization and Regularization in DQN," *arXiv preprint arXiv:1810.00123*, 2018.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[28] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[29] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic Generation and Detection of Highly Reliable Fiducial Markers Under Occlusion," *Elsevier Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.