Navigation among movable obstacles using machine learning based total time cost optimization

Kai Zhang[1,2], Eric Lucet[1], Julien Alexandre dit Sandretto[2] and David Filliat[2]

***Abstract*— Most navigation approaches treat obstacles as static objects and choose to bypass them. However, the detour could be costly or could lead to failures in indoor environments. The recently developed navigation among movable obstacles (NAMO) methods prefer to remove all the movable obstacles blocking the way, which might be not the wisest choice when planning and performing obstacle displacement takes a long time. We propose a pipeline where the robot solves the NAMO problems by optimizing the total time to reach the goal. This is achieved by a supervised learning approach that can predict the time to plan and perform obstacle motion before actually doing it if this leads to faster goal reaching. Besides, a pose generator based on reinforcement learning is proposed to predict the stock region pose, where the robot could unload the obstacles when removing them. The method is evaluated in two kinds of simulation environments and the results demonstrate its advantages compared to the classical bypass and obstacle removal strategies.**

I. INTRODUCTION

Navigation is a long studied problem in mobile robotics, and the problem of navigating among static and dynamic obstacles has many efficient solutions. However, most of the navigation planning algorithms share the principle of avoiding all the obstacles even when they are movable like bottles or bags. This can sometimes produce long detour or simply prevent goal reaching, for example when the robot finds that a door is closed. The field of Navigation Among Movable Obstacles (NAMO) provides solutions by planning actions to remove these kinds of obstacles [1]. In particular, recent approaches seek ways to integrate machine learning and more traditional planning approaches to get the best out of these two approaches [2].

The real interest of moving an obstacle in the NAMO algorithm is often overlooked. Indeed, many algorithms plan the shortest path to the destination without consideration of movable obstacles, then follow the trajectory and remove the obstacles encountered. This raises a question whether this decision is wise in all the cases, especially when the detour is short and easy. While there are several costs that can be optimized, such as the travel distance, the energy cost, or the safety, we focus on the total time of reaching the goal which includes both planning and motion execution time. This is particularly important as the planning and execution time for obstacle displacement is usually much higher and variable than the time required to recompute and follow a detour. Therefore, to minimize total time, we must estimate the duration of each possible reaction to choose the best strategy.

A first problem in NAMO is the decision on where to put the movable object as it takes most of the planning time and has an important impact on execution time. This problem is often solved by sampling candidate positions [3] which is usually quite time consuming and can present large variation in execution time. Additionally, the definition of the goal position should free the robot path, but can also take other constraints into account, such as a social cost related to the fact that some position will reduce the allowance on the navigation of humans or other robots and should therefore be avoided [4]. We propose to use Reinforcement Learning (RL) in order to have a fast decision on where the obstacle should be moved by taking all these constraints into account.

Following this choice, the second problem, in order to optimize the total time, is to compare the planning and execution time for both the detour and the obstacle displacement. While the detour is usually very fast to plan and its execution time is easily predictable, it is more difficult for object displacement, as the planning often relies once again on sampling. We therefore propose a supervised learning approach that can predict the total time required to plan and execute obstacle displacement based on the robot's observation of the situation and the stock region position for the obstacle. This makes it possible to precisely choose between a detour and a movement of the obstacle without actually performing the motion planning before we have verified that the total cost is lower.

In summary, the main contributions of our work are listed below.

1) We propose a framework to complete a NAMO task that minimizes time consumption by integrating learning-based methods and classical planning methods to take advantage of their strengths.
2) We design a pose proposal method to generate a suitable stock region to unload the movable obstacle based on deep reinforcement learning and considering the navigation cost and social context.
3) We develop a cost estimation approach that predicts the total time consumption of removing a movable obstacle so that the robot can decide on the faster solution.
4) We demonstrate the effectiveness of our approach through evaluation on a simple simulation environment and show its scalability on a more realistic Gazebo [5] based simulation environment.

In the remaining parts of the paper, we first introduce the

Website: kai-zhang-er.github.io/namo-time-cost

[1] List, CEA, Université Paris-Saclay, 91120 Palaiseau, France kai.zhang@cea.fr

[2] U2IS, ENSTA Paris, Institut Polytechnique de Paris, 91120 Palaiseau, France

related work on the NAMO task in section II, describe our approach in section III and present the experiment details and simulation environments, as well as the results in section IV.

II. RELATED WORK

We first quickly review path planning algorithms in dynamic environments, before focusing on the NAMO task, and discussing specifically the stock region search.

A. Path planning in dynamic environments

Path planning algorithms usually take all the obstacles as non-interactive objects and plan paths to avoid collisions. The most popular and classical shortest-path planning algorithms such as A* [6], Rapidly-Exploring Random Trees (RRT) [7] or Probabilistic Road-Map planning (PRM) [8], generate global paths to bypass the static obstacles that are in the map. However they are not directly adapted to dynamic environments. Accordingly, methods such as the dynamic window approach (DWA) [9] avoid the collisions with mobile obstacles while trying to follow the global path. When it comes to social environments, where there are moving humans, the previous methods can be improved to take social constraints into account. For example, in [10], an RL-based DWA avoids moving pedestrians from their back region. [11] uses RL to estimate the danger zones of the mobile obstacles according to their types and speed, then plans a path outside these zones.

B. NAMO algorithms

Different from typical navigation environments described before, environments for NAMO tasks include movable obstacles beside the static or dynamic obstacles. Therefore, the NAMO algorithms require two additional modules, movable obstacle detection and interaction.

To identify the movable obstacles, there are usually three solutions. The easiest idea is to take the information as prior knowledge, as in [12]. The second solution is to use sensors, such as cameras, to observe the environment and classify the movable obstacles. For example, [13] applies computer vision to classify movable obstacles. The third solution is based on the moving-affordance of the obstacle [14], where the robot is controlled to try to interact with an obstacle, in order to detect if it is movable or not.

There are various types of interactions to remove the obstacles, including pick-and-place, push, pull, etc. For example, in [15], when the robot meets a box, it will push the box away until a feasible path could be planned to the goal. However, the push action is not safe especially in partially observable environment, because there may be another box or even people behind the box. In contrast, the pull and pick actions are safer as the working space is visible. In [14], a small bottle blocks the path and the robot clears the path by picking it up. Nevertheless, the pick and place are limited to small and graspable objects.

Most of the current NAMO algorithms give priority to remove the obstacle over planning a detour. For example, in [16], the robot follows a predefined path and removes all the obstacles that block the path. In the same way, in [15], a path is planned based on environment map. Then, when robot meets an obstacle, it is ordered firstly to interact with the obstacle to check the moving affordance. Only when the obstacle is not movable, the robot replans a detour.

In this paper, we use simple movable obstacle detection approaches and pull object actions, and focus on more efficient removal or bypass decision strategies.

C. Stock region searching

Little research considers the movable obstacle final position (which we will call stock region) in a NAMO situation. A common idea is to move the obstacle out of the path [12] or displace it until a feasible detour could be planned to bypass it [17], [18], but these works do not consider the ultimate position of the obstacle which could result in potential danger or obstructions in the future.

Regarding the works on searching the stock region, [19] tries to find the stock region by minimizing the displacement of movable obstacle, meaning the stock region position is the closest to the obstacle but also out of the planned path and has no collision with other obstacles. To extend the static environment to a dynamic environment, [4] designs a social cost to measure the impact of moving an obstacle on the environment allowance. In particular, the stock region is expected not to be in the middle of road and narrow regions. We propose a new RL based approach to quickly find positions with the same objective.

III. METHOD

We start with the introduction of the task and overview of the method, before detailing our main contributions.

A. Task description and method overview

The task is defined by the robot goal position and the global map for static obstacles. We are in the NAMO context where the robot can interact with movable obstacles if necessary. Our objective is to minimize the task-completion time which includes both planning and execution time.

As shown in Fig. 1, the method mainly includes three modules: stock region position prediction, cost estimation and interaction planning. Firstly, task description is used to plan a shortest-path based on the global map. During path following, the robot detects unexpected obstacles within a fixed range (as explained in section IV) since the global map only contains static obstacles. When encountering unexpected obstacles, it chooses either to bypass or to remove them by estimating the cost of each option and choosing the smallest one. To predict the removal cost, the stock region is first generated by our pose generator, and the removal cost is predicted by our supervised algorithm. To predict the bypass cost, the shortest detour trajectory taking the unexpected obstacle into account is computed. If the decision is to move the obstacle, the actual planning of the interactions to remove obstacles is performed and implemented by the motion planner. Afterward, the robot resumes path following.

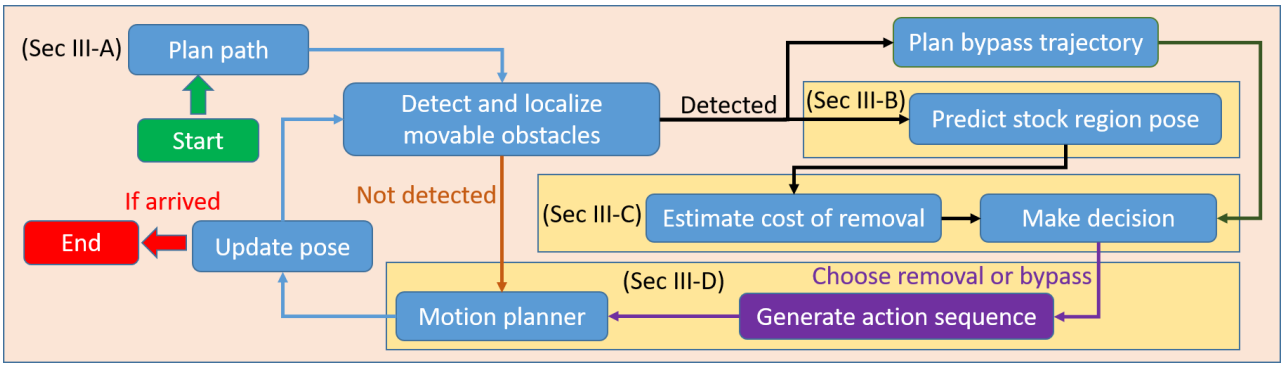

Fig. 1. Overview of our method. It includes three modules (yellow regions): stock region position prediction, cost estimation and interaction planning. It starts from path planning and iterates until the robot arrives at the destination.

B. Stock region prediction

When the robot meets a movable obstacle, a suitable stock region should be proposed to put the obstacle so that it won't block the path anymore. Instead of defining a fixed stock region for all movable obstacles, a stock region proposal method generates a pose to store the movable obstacles with the consideration of moving distance and social cost.

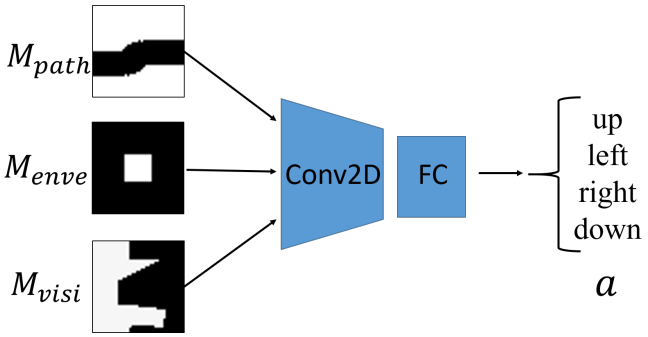

Fig. 2. Policy for stock region pose prediction. Three binary maps: local path map M_{path}, obstacle envelope map M_{enve} and local visibility map M_{visi}, are stacked and inputted to a network to get actions, indicating movement at four directions. Conv2D means 2D convolution layers while FC means fully connected layers.

The proposal of stock region pose is based on RL to train a policy that will propose optimal obstacle displacements to a safe region.

As illustrated in Fig. 2, the input of the policy consists of a set of three local maps:

- The path map is generated from the shortest-path trajectory with a dilation radius equal to the robot radius. The dilated path is marked as zeros on a ones background. The local path map, M_{path}, is cropped from the path map around the stock region position.
- The obstacle envelope map M_{enve} encodes the shape of the obstacle in ones on zeros background, which also indicates the minimum size of the stock region.
- The visibility map is generated from the Lidar data observed at the robot pose. The zeros region corresponds to obstacles or invisible region that is unsafe to set the stock region, while the ones area is clear space. Then, the local visibility map, M_{visi}, is obtained by cropping the visibility map around the stock region.

These three maps are stacked in a tensor ($80 \times 80 \times 3$ pixels) and inputted into a policy network to propose an action a indicating the movement of the stock region. The network consists of two parts, a set of two convolution layers (Conv2D) followed by two fully connected layers (FC) (see appendix for details).

There are four actions: $a \in \{up, left, right, down\}$. The stock region moves in the 2D map with a fixed distance (2 pixels in our experiments) in the direction of a. Then, M_{visi} and M_{path} translate along the direction of a. The initial pose of stock region is the same as the obstacle pose.

The iteration stops when the final pose of the stock region satisfies the following constraints:

1) The stock region should be in the clear region, which means its envelop has no overlap with the obstacle or invisible region shown in the visibility map. In other words, $M_{enve} \wedge M_{visi} = M_{enve}$
2) The stock region shouldn't block the navigation path. In other words, the white region in envelop map has no intersection with the black region in path map. In other words, $M_{enve} \wedge M_{path} = M_{enve}$

The reward at step t for this RL is defined as:

$$r_t = \begin{cases} -5 & \text{if } M_{enve} \wedge M_{visi} \wedge M_{path} \neq M_{enve} \\ 0 & \text{if success} \\ p_{soc}(x,y) + p_{step} & \text{otherwise} \end{cases} \tag{1}$$

where p_{soc} is the social penalty [4] of the current position (x,y) and p_{step} is a fixed step penalty ($p_{step} = -1$ in our experiments). The objective of applying social cost is to avoid putting the obstacle in narrow passage and in the middle of corridors. It has two advantages, one is that it reduces the risks of collision when the robot continues navigating to the destination while the other one is that the new pose of obstacle results in less risks when there are people or other dynamic obstacles in the environment. The social cost map is generated from the obstacle map using:

$$p_{soc}(x,y) = -f_{conv}(d_{allow}(x,y)) \tag{2}$$

$$d_{allow} = 2 \times d_{obs}(x,y) \tag{3}$$

where f_{conv} is a conversion function defined in [4] to map space allowance d_{allow} to social cost. The $d_{obs}(x,y)$ is the distance from (x,y) to the closest obstacle. Because the social cost is between 0 and 1, it's negative value is taken as p_{soc}.

The policy network is trained on a set of randomly generated situations (see details in section IV-A.2) with Proximal Policy Iteration (PPO) [20] using the default setting given in the paper (see details in appendix V). In the prediction step, i.e. during navigation of the robot, the stock region is proposed by applying the learned policy a maximum of N times starting from the observed situation and stopping when all the constraints are met. The termination pose is the proposed stock region that will be used for estimating the obstacle removal cost. If it fails to find a valid stock region, the removal cost is set to infinite.

C. Cost estimation and decision making

This module aims to estimate the cost of bypassing or removing the obstacle, and select the appropriate action. The cost is represented by the time consumption. To calculate the cost of bypassing the obstacle, the movable obstacle is temporarily added to the obstacle map, then the shortest-path planner generates an alternative path to skirt the obstacle. If there is no alternative, the cost is set at infinity. If there is, we use the average robot speed to compute the time cost from the path length.

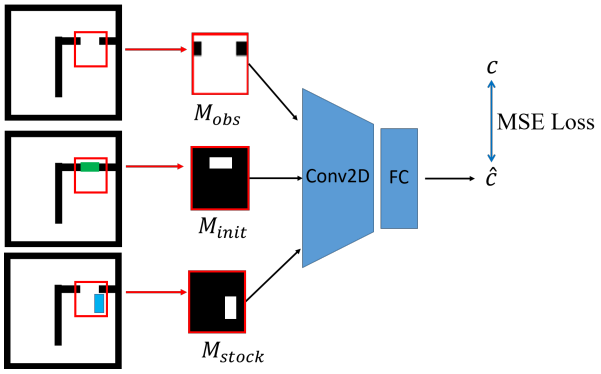

Fig. 3. Neural network for removal cost estimation. The local obstacle map M_{obs} is cropped from the global map. The stock region pose map M_{stock} is generated based on the output of stock region pose estimation method in section III-B. The initial pose map M_{init} is also an input. The predicted cost $\hat{c}$ is optimized using the true cost value c through MSE Loss. Conv2D means 2D convolution layers while FC means fully-connected layers.

For the removal cost, we propose to use a simple convolution neural network (CNN) consisting of three layers of 2D convolutions and three FC layers, to predict the cost based on the environment map (Fig. 3, see details in appendix). Centered at the robot, the local obstacle map M_{obs} is cropped from the global map. The initial pose of obstacle and predicted stock region pose are converted to binary masks by setting the obstacle as white and the background as black. These three binary masks are fed into the CNN network to output the predicted cost value $\hat{c}$. The mean square error (MSE) loss with the real value c is used to train the network.

The training dataset consists of a set of quadruples (robot position, obstacle position, stock region position and time consumption). With a predefined obstacle map, we randomly generate robot positions around the movable obstacle. Then, we apply the stock region prediction method to find the stock region position. The robot motion is planned and executed to move the obstacle to the stock region (see section III-D). The total planning and execution time is recorded. We repeat the above steps several times to prepare the training dataset.

In the testing step, when the robot encounters a movable obstacle, this trained network is used to predict the time for obstacle displacement. This time is added to the time of the remaining trajectory to reach the goal.

Finally, the bypassing cost is compared with the predicted obstacle removal cost and the smaller one is chosen as the strategy to apply.

D. Interaction planning

The interaction planning module includes the action sequence generation and motion planning. When the robot decides to remove the obstacle, the generator would generate a sequence of abstract actions along with the configurations. The motion planner will convert the abstract actions into control parameters, like linear and angular speeds. These low-level control parameters are communicated to the hardware to implement the actions.

The action sequence generator is based on PDDLStream [21]. Firstly, we define some abstract actions, also named symbolic operators $A_i, i = 1,...,m$ (m=3 in our case), as the basis of PDDL planning. Each symbolic operator A_i consists of the name, precondition and effect of the action. For example, the *Nav* operator from position $P1$ to $P2$ can be expressed as:

Nav(P1,P2)
Pre: (atPos(P1)) (not(atPos(P2)))
Eff: (atPos(P2)) (not(atPos(P1)))

Similarly, we also define the operators *Load* and *Unload* signifying the attachment and detachment of obstacle with the robot. Then, we describe the initial state S_0 and goal state S_g of the environment. Here, we limit the planning region to a local area instead of the global environment for the purpose of efficiency. Afterward, the environment description along with the action operators are transferred to the planner of PDDLStream and a solution containing a sequence of actions with parameters is produced:

$$PDDLStream(S_0, S_g) = \{A_1, ..., A_j\} \tag{4}$$

With the parameters and action operators, a motion controller generates the control commands. For the *Nav* operation, we use A* [6] to plan the path, then the robot follows the trajectory through a PID controller. For the *Load* and *Unload* operations, the inverse-kinematic controller calculates the configuration of each joint of the arm and a PID is applied to control the action.

IV. EXPERIMENTAL RESULTS

We present experiments implemented on two simulation environments: one is based on Pybullet [22] while the other is based on Gazebo [5], which is more realistic. Beside the comparison of the whole NAMO results, the performance of our stock region proposal method and cost estimator are evaluated individually.

A. Pybullet simulation

Pybullet is a simplified but fast simulation environment, we use it to demonstrate the robustness of our method on various configurations and compare it to other methods.

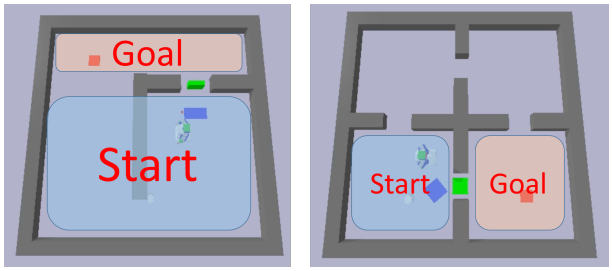

Fig. 4. Our two Pybullet simulation environments. The robot starts randomly from the start region (light blue) and has to arrive at the destination (red square) randomly chosen from the goal region (orange). The blue rectangle is the virtual stock region.

1) Environment description: Two simulation environments, shown in Fig. 4, are constructed for evaluation. In each experiment, the starting point and goal are chosen randomly from the start region and goal region. The movable obstacle is a small light box that can be moved by the robot. We use a PR2 robot equipped with a Lidar sensor, which can identify and localize the movable obstacle with a provided object id. In the experiment, since we assume the robot can only partially observe the environment, it is impossible to find whether there are other obstacles behind the movable obstacle. Therefore, for safety, the robot can only pull the movable obstacle and put it in the visible free region.

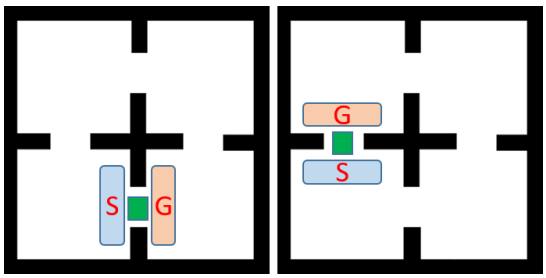

Fig. 5. Training cases of stock region proposal network. There are a total of four cases, two of them are shown in the figure while the other two are similar but with exchanged start and goal region

2) Stock region proposal results: Following the method described in section III-B, the proposal network is firstly trained under different cases to learn the pose generation strategy. For example, in the second environment demonstrated in Fig. 4, we design four scenarios shown in Fig. 5. At each RL training episode, the instance is created by planning the robot to move from a random start point and to a random goal in a randomly chosen environment.

We evaluate our trained policy on 300 test cases with random start and goal positions and compare it with a brute-force search method. This search method is implemented by searching all the points from the visible region and selecting the one closest to the initial position of the movable obstacle in order to give a performance upper bound. In contrast, we also compare to a sampling method which intends to balance the search time and performance by picking several points before searching. We also compare these methods with the social cost considered, which look for the point with minimum weighted sum of navigation cost and social cost. The results are presented in Table I and Fig. 6. Three criteria are reported: average moving distance (AMD), average social cost (ASC) at proposed position and average prediction time (APT). When analyzing the impact of social cost on our RL method, we find that with the consideration of social cost, the network proposes stock region closer to the wall which reduces the risk of collision and has less effect on environment allowance. Besides, comparing to search method, the stock region proposed by our RL method is closer to the initial position, which means less navigation cost. What's more, our RL approach requires a much smaller computation time even compared with sampling method.

TABLE I

QUANTITATIVE COMPARISON OF STOCK REGION PROPOSAL METHODS. AMD: AVERAGE MOVING DISTANCE, ASC: AVERAGE SOCIAL COST AT PROPOSED POSITION, APT: AVERAGE PREDICTION TIME.

	AMD	ASC	APT
RL method (without scost)	0.95	0.43	0.02
Search method (without scost)	0.32	0.62	0.53
RL method (with scost)	1.32	0.28	0.02
Search method (with scost)	1.52	0.20	0.54
Sampling method (with scost)	1.47	0.23	0.47

3) Cost prediction results: We collect 100 samples as the dataset to train and evaluate our cost estimator following the methods explained in section III-C. 80 samples are used for training while the remaining $N = 20$ are employed to evaluate the performance. The average absolute difference D_{abs} between the estimated cost values $\hat{c}$ and true values c is computed for evaluation.

$$D_{abs} = \frac{\sum_i^N |c_i - \hat{c}_i|}{N} \tag{5}$$

In our experiment, $D_{abs} = 0.6s$, which is relative small with respect to the average removal time of $5.03s$. It is better than a simple predictor that would use the average time of the training dataset as a prediction ($D_{avg} = 0.8s$). This shows that our cost estimator is able to efficiently take the situation into account to predict the removal cost.

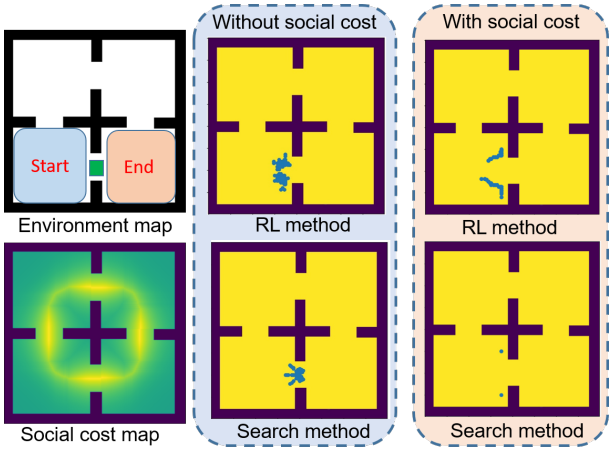

Fig. 6. The footprints of proposed stock region. The social cost map is generated by the method in [4] where the bright region means higher cost while the darker means smaller cost. The blue points in last two columns are the position of proposed stock region.

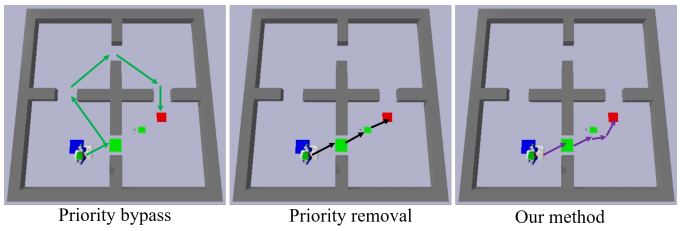

Fig. 7. Demonstration of different NAMO strategy

4) NAMO results: We compare our method using all the presented modules with two other types of NAMO strategies: priority bypass and priority removal. Priority bypass strategy stands for most of the classical navigation methods, where the robot bypasses obstacles during navigation to avoid collision. On the contrary, the priority removal strategy, which is widely applied in recent NAMO algorithms [15], [14], gives priority to remove obstacle when the robot meets one. Fig. 7 shows examples of navigation trajectories when the robot adopts these different NAMO strategies with two obstacles. In the third figure, our method compares the cost of bypass and removal each time and chooses to remove the first obstacle and bypass the second obstacle for the purpose of minimizing the time cost.

The quantitative results on the time consumption of each method can be found in Table II for six different navigation tasks, as shown in Fig. 8. In environment (*a-e*), our method outperforms the priority bypass method by taking less time to accomplish the task, especially in environment *b* and *d* since the detour is quite long and time consuming. Comparing with priority remove, both methods have similar performance because the removal of obstacle tends to be less costly than the detour. In addition, from the results in environment *e*, when the detour becomes shorter, our method demonstrates its advantage comparing to priority removal, through changing the decision according to the environment. In environment *f*, we evaluate the multiple obstacle case. The priority bypass strategy chooses to bypass all the obstacles while priority removal moves the two movable obstacles. Differently, ours tends to remove the first obstacle then bypass the second one, which accomplishes the task with less time cost. Regarding to the standard deviation, our method achieves the tasks with more stable performance on the time cost.

TABLE II

QUANTITATIVE COMPARISON RESULTS ON AVERAGE TIME COST AND STANDARD DEVIATION OF 100 TEST EXAMPLES IN PYBULLET ENVIRONMENT

Env.	Priority bypass (s)	Priority removal (s)	Our method (s)
a	17.8 ± 8.69	16.0 ± 6.83	15.3 ± 6.44
b	28.2 ± 9.68	18.1 ± 3.03	18.7 ± 2.52
c	18.7 ± 3.61	16.5 ± 2.95	16.2 ± 2.69
d	22.3 ± 1.42	15.0 ± 2.53	14.8 ± 1.86
e	15.3 ± 2.28	16.2 ± 3.18	15.3 ± 2.31
f	23.4 ± 1.21	21.0 ± 2.35	18.0 ± 1.83

B. Gazebo simulation

The Gazebo simulation aims to validate the scalability of the proposed method in a more realistic environment. For example, it adds the sensor noise and the action uncertainty, which exist in real world but are not considered in the Pybullet environment.

1) Environment description: In the Gazebo environment, shown in Fig. 9, we use a Clearpath Jackal robot accompanied with an arm at its bottom. The movable obstacle can be detected and localized through a camera on top of the robot using an ARTag placed on the obstacle. Besides, a Lidar is used to detect the static obstacles. The robot can load/unload the obstacle by extending and retracting its arm.

2) Stock region proposal: We re-use the pretrained model in Pybullet environment to predict the stock region pose in the Gazebo environment. As shown in Fig. 2 and described in section III-B, the input of stock region proposal method are binary images which are independent of the simulators and allowed to transfer the model between the Pybullet and Gazebo environments.

3) Cost prediction: To collect the training dataset for cost estimation, we randomly sample the robot start point in the environment and select a collision free goal position around the start point at a distance of 2 meters. Then the A* algorithm plans the shortest path and we pick the middle point of the path as the obstacle position. When the robot starts, it estimates the obstacle's pose and plans the interaction sequence, which includes *nav, load* and *unload*. We count the time until the robot unloads the obstacle.

Due to the uncertainty and noise during the movement of the robot, we have to preprocess the collected dataset to remove the outliers, meaning remove failed obstacle displacements. We obtain 118 samples and 98 of them are used for training while 20 are used for testing. We reach an average difference $D_{abs} = 0.4s$ that is 2.1% of the average removal time ($19.1s$). In contrast, the average time predictor reaches a much lower performance of $D_{abs} = 2.5s$.

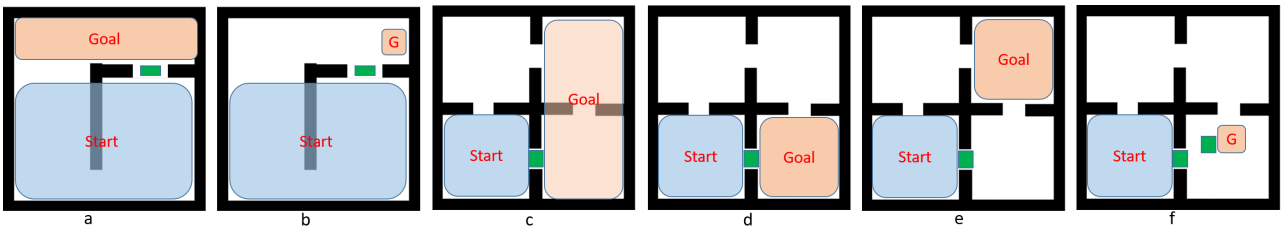

Fig. 8. Evaluation environments referred in table II for NAMO tasks. The green box is the movable obstacle while black areas are static obstacles.

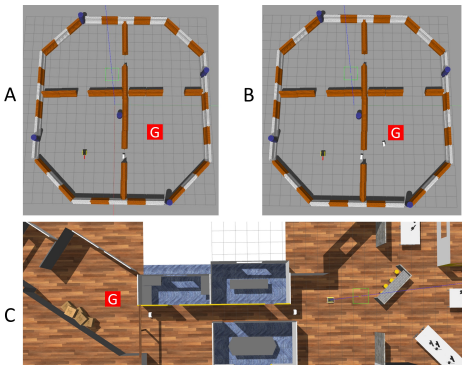

Fig. 9. Test environments in Gazebo simulator. All the objects are static except the small white boxes in the passage that are movable. The goal region is marked in red. There is one movable obstacle in environment A while two movable obstacles in environments B and C.

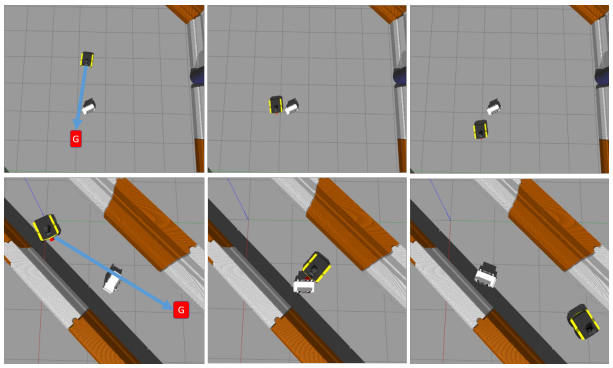

Fig. 10. Qualitative results of different interaction strategies. The blue arrow shows the initially planned shortest path to the goal. The first row shows an example of bypass strategy while the second row shows an example of removal strategy, that are automatically selected by our approach based on environment context. The first column is the start state, the second one shows the interaction while the last one shows the termination state.

4) NAMO results: We test the Jackal robot in two cases, as shown in Fig. 10. From the first row, we can see that in a large free environment, even when an obstacle that blocks the shortest path is movable, the robot chooses to bypass it to save time. On the contrary, in a narrow passage, shown in the second row of Fig.10, a movable obstacle blocks the path but the robot doesn't choose to bypass it because the detour is too costly.

In addition to the qualitative results, we measure the average time cost of different strategies for the tasks shown in Fig. 9 (Table III). In environment A, due to the long detour, our method chooses to remove the obstacle then navigates to the goal, which is the same as the priority removal. In contrast, the priority bypass at this case is very costly and it takes 20 seconds longer than ours. In environment B and C, our method chooses to remove the first obstacle and bypass the second one. Differently, the priority removal removes the two movable obstacles, which takes more time to reach the goal. The priority bypass fails at finding a detour for the first obstacle in environment C so this strategy cannot deal with such case. In addition, the standard deviation of time cost in the three environments show that our method achieves better stability with a reduced variance in most cases. In summary, our method outperforms the other methods that follow predefined strategies by proposing more adaptive decisions.

TABLE III

QUANTITATIVE COMPARISON ON AVERAGE TIME COST AND STANDARD DEVIATION OF 10 TEST EXAMPLES IN GAZEBO ENVIRONMENT

Env.	Priority bypass (s)	Priority removal (s)	Our method (s)
A	76.6 ± 3.14	53.0 ± 3.41	54.0 ± 2.32
B	76.5 ± 4.22	68.9 ± 2.31	60.6 ± 2.43
C	-	105.1 ± 8.58	95.8 ± 6.14

V. CONCLUSION AND FUTURE WORK

This paper presents a framework to solve the NAMO problem while minimizing goal reaching time. The proposed method integrates learning-based methods to make decision on interaction strategies when a movable obstacle blocks the path. Besides, a stock region proposal method generates the storage position of a movable obstacle with consideration of social cost and navigation cost. Two types of simulation environments are presented to demonstrate its effectiveness and scalability.

Our future work will focus on solving the uncertainty in the NAMO tasks, such as the observation noise and action uncertainties. In our Gazebo simulation, the observation uncertainty leads to localization error, which occasionally results in collision with obstacles and failure when the robot loads the obstacle. In addition, the control algorithm produces perturbation when robot rotates and the carried obstacle tends to fall off the arm. To tackle these problems, we plan to integrate a replanning module which would control the robot to pick the fallen obstacle. Finally, the complete method will be employed on a real mobile robot equipped with an arm.

APPENDIX

A. System setup

All the experiments are implemented on Python with the deep learning library PyTorch [23]. The computation is accelerated by the NVIDIA GTX1070 GPU with 8G memory. The CPU is Intel i5-9600F with 16G memory. All the quantitative results are collected on the same computer.

B. Parameters of stock region prediction

Name	Value
input dim	$80 \times 80 \times 3$
Conv2d(1)	Kernel=5, stride=2, out_channel=16, ReLU(), maxpool2D
Conv2d(2)	Kernel=3, stride=2, out_channel=32,ReLU(), maxpool2D
FC(1)	In_channel=128, Out_channel=64,ReLU()
FC(2)	In_channel=64, Out_channel=4
Total training steps	700,000
Episode length	$N = 30$
Actor learning rate	0.0003
Critic learning rate	0.001

C. Parameters of cost estimation

Name	Value
input dim	$40 \times 40 \times 3$
Conv2d(1)	Kernel=3, stride=2, padding=1, out_channel=16, Batchnorm2d, Relu, maxpool2D
Conv2d(2)	Kernel=3, stride=2, padding=1, out_channel=32, Batchnorm2d, Relu, maxpool2D
Conv2d(3)	Kernel=3, stride=1, padding=1, out_channel=32, Batchnorm2d, Relu, maxpool2D
FC(1)	In_channel=128, out_channel=128, Relu
FC(2)	In_channel=128, out_channel=64, Relu
FC(3)	In_channel=64, out_channel=128, Relu
Learning rate	0.001
Epoch	500

ACKNOWLEDGMENT

This work was carried out in the scope of OTPaaS project. This project has received funding from the French government as part of the "Cloud Acceleration Strategy" call for manifestation of interest.

REFERENCES

[1] H.-n. Wu, M. Levihn, and M. Stilman, "Navigation among movable obstacles in unknown environments," in *2010 IEEE/RSJ IROS*. IEEE, 2010, pp. 1433–1438.

[2] K. Zhang, E. Lucet, J. A. D. Sandretto, S. Kchir, and D. Filliat, "Task and motion planning methods: applications and limitations," in *19th ICINCO 2022*. SCITEPRESS-Science and Technology Publications, 2022, pp. 476–483.

[3] B. Kim, L. Shimanuki, L. P. Kaelbling, and T. Lozano-Pérez, "Representation, learning, and planning algorithms for geometric task and motion planning," *The International Journal of Robotics Research*, vol. 41, no. 2, pp. 210–231, 2022.

[4] B. Renault, J. Saraydaryan, and O. Simonin, "Modeling a social placement cost to extend navigation among movable obstacles (namo) algorithms," in *2020 IEEE/RSJ IROS*. IEEE, 2020, pp. 11 345–11 351.

[5] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ IROS*, vol. 3. IEEE, pp. 2149–2154.

[6] 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.

[7] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[8] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[9] E. J. Molinos, A. Llamazares, and M. Ocaña, "Dynamic window based approaches for avoiding obstacles in moving," *Robotics and Autonomous Systems*, vol. 118, pp. 112–130, 2019.

[10] U. Patel, N. K. S. Kumar, A. J. Sathyamoorthy, and D. Manocha, "Dwa-rl: Dynamically feasible deep reinforcement learning policy for robot navigation among mobile obstacles," in *2021 IEEE ICRA*. IEEE, 2021, pp. 6057–6063.

[11] S. S. Samsani and M. S. Muhammad, "Socially compliant robot navigation in crowded environment by human behavior resemblance using deep reinforcement learning," *IEEE RA-L*, vol. 6, no. 3, pp. 5223–5230, 2021.

[12] K.-H. Zeng, L. Weihs, A. Farhadi, and R. Mottaghi, "Pushing it out of the way: Interactive visual navigation," in *Proceedings of the IEEE/CVF CVPR*, 2021, pp. 9868–9877.

[13] Z. Meng, H. Sun, K. B. Teo, and M. H. Ang, "Active path clearing navigation through environment reconfiguration in presence of movable obstacles," in *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2018, pp. 156–163.

[14] M. Wang, R. Luo, A. Ö. Önol, and T. Padir, "Affordance-based mobile robot navigation among movable obstacles," in *2020 IEEE/RSJ IROS*. IEEE, 2020, pp. 2734–2740.

[15] K. Ellis, H. Zhang, D. Stoyanov, and D. Kanoulas, "Navigation among movable obstacles with object localization using photorealistic simulation," in *2022 IEEE/RSJ IROS*. IEEE, 2022, pp. 1711–1716.

[16] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese, "Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation," in *2021 IEEE ICRA*. IEEE, 2021, pp. 4583–4590.

[17] J. Scholz, N. Jindal, M. Levihn, C. L. Isbell, and H. I. Christensen, "Navigation among movable obstacles with learned dynamic constraints," in *2016 IEEE/RSJ IROS*. IEEE, 2016, pp. 3706–3713.

[18] B. Kim and L. Shimanuki, "Learning value functions with relational state representations for guiding task-and-motion planning," in *CoRL*. PMLR, 2020, pp. 955–968.

[19] A. Thomas and F. Mastrogiovanni, "Minimum displacement motion planning for movable obstacles," in *Intelligent Autonomous Systems 17: Proceedings of the 17th International Conference IAS-17*. Springer, 2023, pp. 155–166.

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

[21] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *ICAPS*, vol. 30, 2020, pp. 440–448.

[22] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[23] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.