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 best choice when planning and moving obstacles 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 of planning and performing obstacle motion before actually doing it if this leads to faster goal reaching. Besides, a pose generator based on reinforcement learning is proposed to decide where the robot can move the obstacle. 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

The problem of navigating among static and dynamic obstacles in robotics has many efficient solutions. However, most of the navigation algorithms are designed to avoid 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 a closed door. 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 to integrate machine learning and traditional planning to get the best out of these two approaches [2].

The real interest of moving an obstacle in NAMO is often overlooked. Many algorithms plan the shortest path 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 alternative to choose the fastest.

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 time consuming and can present large variation in execution time. Additionally, the target 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 positions will impact more on the navigation of humans or other robots and should be avoided [4]. We use Reinforcement Learning (RL) to have a fast decision on where the obstacle should be moved by taking all these constraints into account.

Following this choice, optimizing the total time requires to compare the planning and execution time for both detour and 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 predicts the total time required to plan and execute obstacle displacement based on the robot's observation of the situation and the target 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:

1) a framework for NAMO that minimizes time consumption by integrating learning-based and classical planning methods to take advantage of their strengths.
2) a method to propose a pose to unload the movable obstacle based on deep RL with consideration of the navigation cost and social context.
3) an approach to predict the total time of removing a movable obstacle before actually performing it.
4) an extensive evaluation of our approach in a simple simulation environment and a demonstration of its scalability on a more realistic Gazebo [5] simulation.

In the remaining parts of the paper, we first introduce the 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 review path planning in dynamic environments, before focusing on NAMO and specifically stock region search.

A. Path planning in dynamic environments

Classical path planning algorithms such as A* or Rapidly-Exploring Random Trees (RRT) , usually take all the obstacles as non-interactive objects and plan paths to avoid collisions. However they are not directly adapted to dynamic

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

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

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

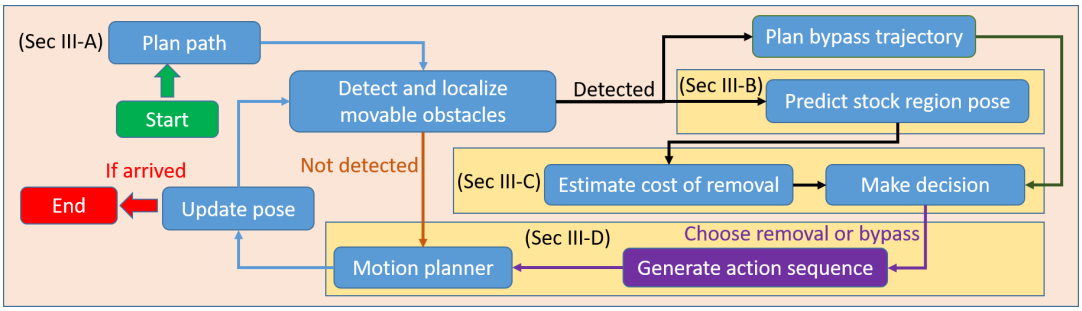

Fig. 1. Overview of our method. It includes three modules (yellow regions): stock region position prediction, cost estimation and interaction planning. It starts from path planning and iterates until the robot arrives at the destination.

environments and methods such as the dynamic window approach (DWA) [6] avoid the collisions with mobile obstacles while following the global path. In social environments with humans, the methods can be improved to take social constraints into account. For example, in [7], a RL-based DWA avoids pedestrians by passing behind them.

B. NAMO algorithms

NAMO includes movable obstacles beside the static and dynamic obstacles. The algorithms require two additional modules to detect and interact with these obstacles.

There are three solutions to identify the movable obstacles. The simplest one is to take the information as prior knowledge, as in [8]. The second is to use sensors, such as cameras in [9], to detect objects and recognize the movable ones. The third solution is based on detecting the moving-affordance [10], by interacting with the obstacle to check if it can move.

There are various possible interactions such as pick-and-place, push or pull. For example, in [11], the robot pushes boxes away until a feasible path can be planned to the goal. However, pushing is not safe especially in partially observable environment, because there may be another obstacle behind. In contrast, the pull and pick actions are safer as the working space is visible, as in [10] where a small bottle is picked to free the path. Nevertheless, the pick and place are limited to small and graspable objects.

Most NAMO algorithms give priority to remove the obstacle over planning a detour. For example, in [12], the robot follows its path and removes all the encountered obstacles. In the same way, in [11], when meeting an obstacle, the robot interacts with it to check the moving affordance and a detour is planned only when the obstacle is not movable.

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

C. Stock region searching

Few works consider the final position of the moved obstacle (called stock region). A common idea is to move it out of the path [8] or until a feasible detour is found [13], [14], but they do not consider the impact of final position of the obstacle on the future danger. However, [15] searches the stock region by minimizing the displacement of the movable obstacle out of the planned path, and [4], working in dynamic environments, designs a social cost to measure the impact of the stock region: it is expected not to be in the middle of corridors 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 navigation goal and the global map for static obstacles. We are in the NAMO context where the robot can interact with movable obstacles if necessary, but we assume a static environment without dynamic obstacles. 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. During path following, the robot detects unexpected (i.e., movable) obstacles within a fixed range (as explained in section IV). Upon such detection, it chooses either to bypass or to remove them by estimating the cost of each option and pick 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 executed. Afterward, the robot resumes path following.

B. Stock region prediction

A suitable stock region should be proposed to put the obstacle to free the path. Instead of defining a fixed stock region for all movable obstacles, a stock region proposal method generates a pose to store each movable obstacle by considering moving distance and social cost.

The stock region proposal 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:

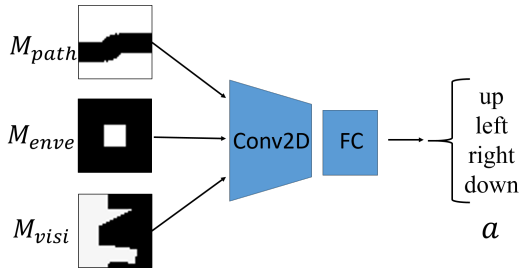

Fig. 2. Policy for stock region prediction. See text for details.

- 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.

The local map size is set to 80×80 pixels, equivalent to 2×2 meters, which is determined by the sensor range at which movable obstacles can be reliably detected. Larger visibility map could be used in more complex environments but would require more training and computation time. These three maps are stacked in a tensor 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 RL is defined as:

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

where p_{soc} is the social penalty [4] of the current position (x,y) and p_{step} is a fixed step penalty (-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: it reduces the risks of collision when the robot continues navigating to the destination, and it 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:

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

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

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) [16] using the default setting given in the paper (see details in appendix). 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. For bypassing the obstacle, the movable obstacle is temporarily added to the obstacle map, and the shortest-path planner generates an alternative path. If there is no alternative, the cost is set at infinity. If there is, the average robot speed is used to compute the cost from the path length.

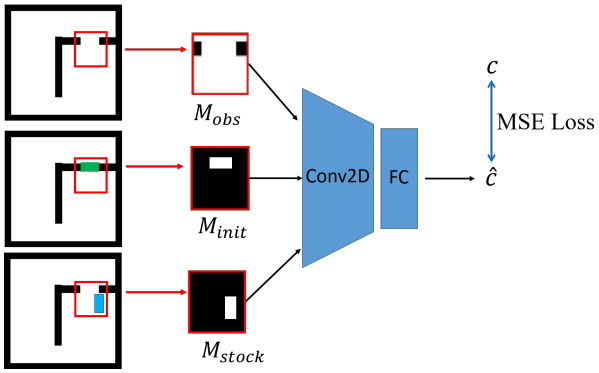

Fig. 3. Neural network for removal cost estimation. See text for details.

For the removal cost, we use a simple neural network consisting of three Conv2D 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 one and the background as zero. These three binary masks are fed into the network to give the predicted cost value $\hat{c}$.

The training dataset consists of a set of quadruples (robot position, obstacle position, stock region position and time consumption c) randomly generated from a predefined obstacle map. We randomly generate robot positions around the movable obstacle and 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 c is recorded.

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 action sequence generation and motion planning. For obstacle removal, the generator produces a sequence of abstract actions along with the configurations. Then, the motion planner converts the abstract actions into control parameters, like linear and angular speeds which are then executed.

The action sequence generator uses PDDLStream [17]. We define abstract actions $A_i, i = 1,...,m$ ($m = 3$ in our case), each consisting of the name, precondition and effect. For example, the *Nav* action from position $P1$ to $P2$ is:

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

Similarly, we define the actions *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 to produce a solution as a sequence of actions with parameters:

$$\{a_1,....,a_j\} = PDDLStream(S_0, S_g) \tag{4}$$

A motion controller generates the control commands for each actions with its parameters. For the *Nav* operation, we use A* 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 with two simulation environments: one using Pybullet [18] and one using Gazebo [5], which is more realistic. Beside the comparison of the whole NAMO results, we evaluate the performance of our stock region proposal and cost estimator.

A. Pybullet simulation

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

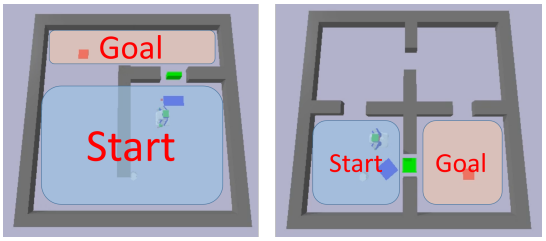

Fig. 4. Our two Pybullet environments. Robot starts randomly from the start region and its goal (red square) is randomly chosen from the goal region. The green rectangle is the movable object and the blue rectangle is the stock region for a specific episode.

1) Environment description: Two simulation environments are used (Fig. 4). In each experiment, the starting point and goal are chosen randomly from the start region and goal regions. The movable obstacle is a small 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. Since it is impossible to detect whether there are other obstacles behind the movable obstacle, we limit actions to pulling the obstacle and putting it in the visible free region.

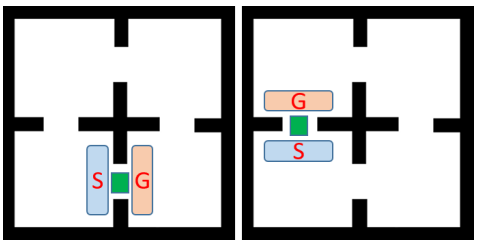

Fig. 5. Two of the four training cases for stock region proposal network. The other two are similar but with exchanged start and goal regions.

2) Stock region proposal results: Following the method described in section III-B, the proposal network is trained under different cases to learn the pose generation policy. We design four scenarios shown in Fig. 5. At each episode, an instance is created by planning from a random start point to a random goal in a randomly chosen environment. The training takes around 5 hours for 100000 steps.

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. 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 seeks a 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 future collision. 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 the sampling method. In these scenarios all the above methods could find a stock region for each test case. We could imagine failure at producing a stock regions in more complex environments which would lead to our approach choosing to bypass obstacles.

TABLE I

QUANTITATIVE COMPARISON OF STOCK REGION PROPOSAL METHODS. SEE TEXT FOR LEGENDS.

Method	AMD	ASC	APT
RL (w/o scost)	0.95	0.43	0.02
Search (w/o scost)	0.32	0.62	0.53
RL (w scost)	1.32	0.28	0.02
Search mthod (w scost)	1.52	0.20	0.54
Sampling (w scost)	1.47	0.23	0.47

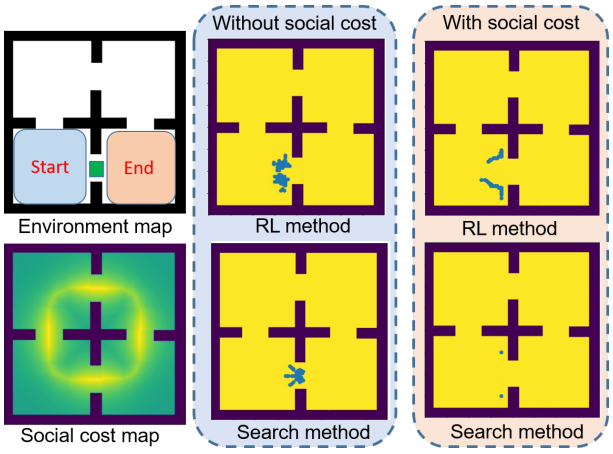

Fig. 6. Left: map and social cost generated by the method in [4] (brighter is higher). Right: Footprints of proposed stock region (blue points).

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

$$D_{abs} = \sum_{i}^{N} |c_i - \hat{c}_i| / N \tag{5}$$

We measured $D_{abs} = 0.6s$, which is relatively small compared to the average time of $5.03s$. It is better than a simple predictor using the average time of the training dataset as a prediction ($D_{avg} = 0.8s$). This shows that our cost estimator can take the situation into account to predict the cost.

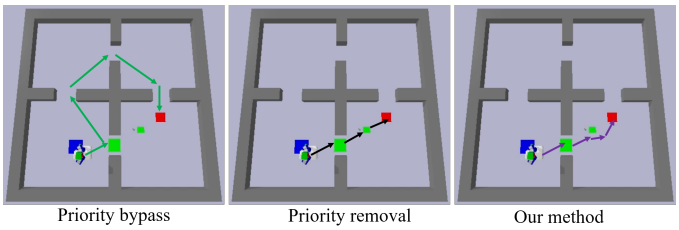

Fig. 7. Demonstration of different NAMO strategy

TABLE II

AVERAGE TIMINGS ON 100 TEST EXAMPLES.

Env.	Priority bypass (s)		Priority removal (s)		Our method (s)	
	Total	Planning	Total	Planning	Total	Planning
a	17.8 ± 8.7	0.9 ± 0.03	16.0 ± 6.8	0.2 ± 0.01	15.3 ± 6.4	1.3 ± 0.03
b	28.2 ± 9.7	0.8 ± 0.01	18.1 ± 3.0	0.2 ± 0.01	18.7 ± 2.5	1.1 ± 0.05
c	18.7 ± 3.7	0.7 ± 0.04	16.5 ± 2.9	0.2 ± 0.01	16.2 ± 2.7	1.0 ± 0.09
d	22.3 ± 1.4	1.0 ± 0.02	15.0 ± 2.5	0.2 ± 0.01	14.8 ± 1.9	1.3 ± 0.02
e	15.3 ± 2.3	0.6 ± 0.03	16.2 ± 3.2	0.2 ± 0.01	15.3 ± 2.3	0.8 ± 0.03
f	23.4 ± 1.2	1.1 ± 0.01	21.0 ± 2.3	0.5 ± 0.04	18.0 ± 1.8	1.4 ± 0.08

4) NAMO results: We compare our method with two other NAMO strategies: priority bypass and priority removal. Priority bypass strategy stands for most of the classical navigation methods. On the contrary, the priority removal strategy is widely applied in recent NAMO algorithms [11], [10]. Fig. 7 shows examples 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 one for the purpose of minimizing the time cost.

The average total navigation time and planning time of each method is in Table II for six different tasks (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 environment *e*, when the detour becomes shorter, our method demonstrates its advantage by adapting the decision according to the task. In environment *f*, we evaluate the multiple obstacle case where our method tends to remove the first obstacle and bypass the second one, which is faster. We also observe the stability of our method which achieves the tasks with a smaller standard deviation.

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.

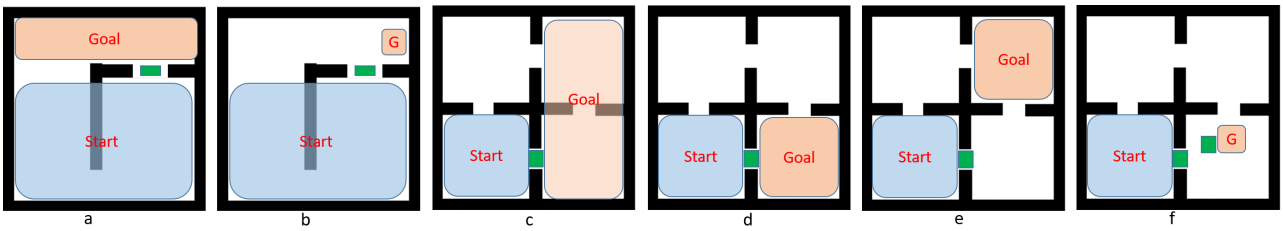

Fig. 8. Evaluation environments referred in table II for NAMO tasks. The green box is the movable obstacle while black areas are static obstacles.

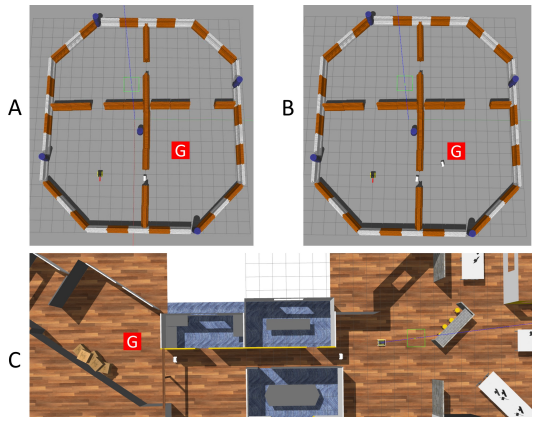

Fig. 9. Test environments in Gazebo. All the objects are static except the small white boxes in the passage that are movable. The goal is in red. There is one movable obstacle in env. A and two in env. B and C.

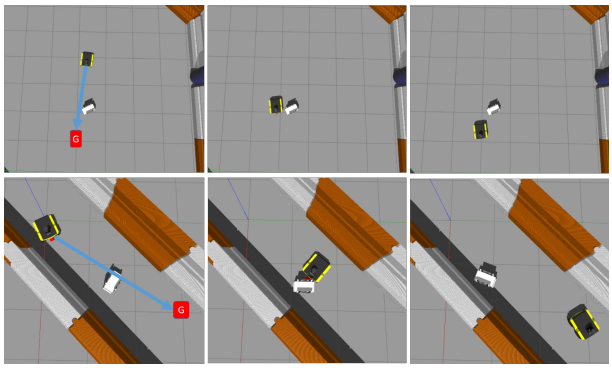

Fig. 10. Qualitative results of different interaction strategies. The blue arrow shows the initially planned shortest path to the goal. The first row shows an example of bypass strategy while the second row shows an example of removal strategy, that are automatically selected by our approach based on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

1) Environment description: We design 3 environments (Fig. 9) and use a Clearpath Jackal robot with an arm at its bottom that can load/unload the obstacle. The movable obstacles are detected and localized through a camera on the robot using an ARTag placed on the obstacle. Besides, a Lidar is used to detect static obstacles and localize the robot.

2) Stock region proposal: We re-use the model trained in the Pybullet environment for stock region prediction as the input of the network are binary images (Fig. 2) which are strongly independent of the simulator.

3) Cost prediction: To collect the training dataset for cost estimation, we randomly sample starting points in the environment and a collision free goal around the start point at a distance of 2 meters. We plan the shortest path using A* and put a movable obstacle on the middle point of the path. 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 obstacle is unloaded.

Due to the uncertainty and noise during the movement of the robot, we preprocess the collected dataset by removing failed obstacle displacements. We obtain 118 samples; 98 of them are used for training and 20 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 $D_{abs} = 2.5s$.

4) NAMO results: We show two examples in Fig. 10. The first row shows that, in a large free environment, the algorithm chooses to bypass the obstacle to save time. In the second row, in a narrow passage, the algorithm chooses to move the obstacle as the detour is too costly.

TABLE III

QUANTITATIVE COMPARISON ON AVERAGE TIME COST ON 10 TEST EXAMPLES IN GAZEBO.

Env.	Priority bypass (s)	Priority removal (s)	Our method (s)
A	76.6 ± 3.1	53.0 ± 3.4	54.0 ± 2.3
B	76.5 ± 4.2	68.9 ± 2.3	60.6 ± 2.4
C	-	105.1 ± 8.6	95.8 ± 6.1

In addition to the qualitative results, we measure the average time cost of different strategies (Table III) for the tasks shown in Fig. 9. 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 shows 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.

V. CONCLUSION AND FUTURE WORK

We present 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 extending the approach to deal with dynamic obstacles such as pedestrians as well as the corresponding collision avoidance modules. Besides, we will work 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 the robot rotates and the carried obstacle can 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 deployed on a real mobile robot and evaluated in a more complex environment.

ACKNOWLEDGMENT

This work was part of the OTPaaS project with French government funding from the "Cloud Acceleration Strategy".

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] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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 *IEEE ICRA*, 2021, pp. 4583–4590.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

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

APPENDIX

A. System setup

All the experiments are implemented in Python with the PyTorch deep learning library. We use an NVIDIA GTX1070 GPU with 8G memory and Intel i5-9600F CPU with 16G memory for all the quantitative results.

B. Parameters of stock region prediction

Name	Value
input dim	$80 \times 80 \times 3$
Conv2d(1)	kernel=5, stride=2, out_chan=16, ReLU, maxpool2D
Conv2d(2)	kernel=3, stride=2, out_chan=32, ReLU, maxpool2D
FC(1)	in_chan=128, out_chan=64, ReLU
FC(2)	in_chan=64, out_chan=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, pad=1, out_chan=16, Batchnorm2d, ReLU, maxpool2D
Conv2d(2)	Kernel=3, stride=2, pad=1, out_chan=32, Batchnorm2d, ReLU, maxpool2D
Conv2d(3)	Kernel=3, stride=1, pad=1, out_chan=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