Training A Non-Cooperator to Identify Vulnerabilities and Improve Robustness for Robot Navigation

Quecheng Qiu[1,*], Xuyang Zhang[1,*], Shunyi Yao[1], Yu'an Chen[1], Guangda Chen[1], Bei Hua[1] and Jianmin Ji[1,†]

***Abstract*— Autonomous mobile robots have become popular in various applications coexisting with humans, which requires robots to navigate efficiently and safely in crowd environments with diverse pedestrians. Pedestrians may cooperate with the robot by avoiding it actively or ignoring the robot during their walking, while some pedestrians, denoted as non-cooperators, may try to block the robot. It is also challenging to identify potential vulnerabilities of a navigation policy, i.e., situations that the robot may cause a collision, in various crowd environments, which reduces the reliability and the safety of the robot. In this paper, we propose a deep reinforcement learning (DRL) approach to train a policy simulating the behavior of non-cooperators, which can effectively identify vulnerabilities of a navigation policy. We evaluate the approach both on the Robot Operating System (ROS) navigation stack with dynamic window approach (DWA) and a DRL-based navigation policy, which identifies useful vulnerabilities of both navigation policies for further improvements. Moreover, these non-cooperators play a game with the DRL-based navigation policy, then we can improve the robustness of such navigation policy by retraining it in the sense of asymmetric self-play. We evaluate the retrained navigation policy in various crowd environments with diverse pedestrians. The experimental results show that the approach can improve the robustness of the navigation policy. The source code for the training and the simulation platform is released online at https://github.com/DRL-Navigation.**

I. Introduction

Autonomous mobile robots are widely applied in various human coexistence environments, like cleaning robots in airports and malls and delivery robots in hotels and restaurants. These applications require the robot to navigate efficiently and safely in crowded environments with diverse pedestrians. The navigation policy of the robot is required to be robust to diverse crowd environments, while different pedestrians may behave differently w.r.t. the robot. In particular, cooperating pedestrians may cooperate with the robot, i.e., actively avoid the robot during their walking. An unaffected pedestrian may ignore the robot and keep walking unaffected. Meanwhile, a non-cooperator (NCO) may try to block the robot. It is challenging to develop a robust navigation policy in various crowd environments with diverse pedestrians to efficiently drive the robot to its target without collision.

Many navigation approaches have been proposed to address the challenge, like ROS navigation stack with dynamic window approach (DWA) [1] and various deep reinforcement learning (DRL) approaches [2]. Some of them consider pedestrians as undisturbed moving obstacles while ignoring cooperating pedestrians and NCOs in environments. Therefore, they are easy to suffer from the freezing robot problem in dense crowds. Normally, it is more efficient and safer if pedestrians cooperate with the robot. Some other approaches assume that the pedestrians cooperate with the robot following certain patterns, like social force model (SFM) [3] and Optimal Reciprocal Collision Avoidance (ORCA) [4]. However, these approaches may not be robust to crowd environments with pedestrians that behave differently, like NCOs. As far as we know, there is little work addressing NCOs in robot navigation.

On the other hand, it is also challenging to identify potential vulnerabilities of a given navigation policy. It is hard to exhaust all possible situations for real-world navigation in practice to identify cases when the robot may cause a collision. This reduces the reliability and safety of the robot in human coexistence environments.

In this paper, we propose a DRL approach to train a policy simulating the behavior of NCOs. We first introduce the network structure and the reward function that encourage the NCO to block the robot and explore situations in which the robot may cause a collision. Then we implement a distributed training framework with the asynchronous proximal policy optimization (APPO) algorithm, which runs for the training of NCOs. In particular, the DRL policy for a NCO consists of two sub-policies, i.e., the flashing policy and the moving policy. The flashing policy enables the NCO to teleport to arbitrary locations around the robot based on its surrounding environment, which can greatly increase the diversity of the interactions between the robot and the NCO and encourage the NCO to identify more vulnerabilities of the robot navigation policy. Meanwhile, the moving policy drives the NCO towards the robot with desired linear and angular velocities. In our experiments, we find out that the flashing policy is crucial for the performance of the NCO, while different moving policies have limited effect. In this paper, we focus on how to train an effective flashing policy given a simple moving policy.

These NCOs can help us to effectively identify vulnerabilities of a navigation policy for the robot. We evaluate the approach both on the ROS navigation stack with DWA and a DRL-based navigation policy, which identifies useful vulnerabilities of both navigation policies for further

[1] School of Computer Science and Technology, University of Science and Technology of China, China. {qiuqc, zxy2533, ustcysy, an11099, cgdsss}@mail.ustc.edu.cn,{jianmin, bhua}@ustc.edu.cn.

[*] These authors contributed equally to the work.

[†] The corresponding author jianmin@ustc.edu.cn.

improvements. Moreover, these NCOs play a game with the DRL-based navigation policy, then we can improve the robustness of such navigation policy by retraining it in the sense of asymmetric self-play. We evaluate the retrained navigation policy in various crowd environments with diverse pedestrians. The experimental results show that the approach can improve the robustness of the navigation policy in various environments. The source code for the training framework and the simulation platform is released online.

Our main contributions are summarized as follows:

- We propose a DRL approach to train a policy simulating the behavior of NCOs, which can effectively identify vulnerabilities of a navigation policy. We evaluate the approach both on the ROS navigation stack with DWA and a DRL-based navigation policy, which identifies useful vulnerabilities of both navigation policies for further improvements.
- We use the trained NCOs to improve the robustness of navigation policies by retraining them in the sense of asymmetric self-play. We evaluate the retrained navigation policy in various crowd environments with diverse pedestrians as well as in various real-world environments. The experimental results show that the approach can improve the robustness of the navigation policy.
- We specify the policy of a NCO as the combination of the flashing policy and the moving policy. Our experiments show that the flashing policy can greatly increase the diversity of the interactions between the robot and the NCO, which is crucial for the performance.

II. Related Work

A. Crowd-aware Robot Navigation

DRL-based navigation approaches have been applied in crowd environments with promising results [2]. They often assume that pedestrians behave following certain patterns. These patterns include the social force model (SFM) [3] that considers multiple types of force for pedestrians in the environment, and ORCA [4] makes assumptions on reciprocal behaviors of pedestrians.

Recently, Yao et al. [5] propose a map-based DRL approach for crowd-aware robot navigation, where the information of obstacles and the information of pedestrians are specified in the sensor map and the pedestrian map, respectively. This approach performs well in crowd environments with static obstacles and diverse pedestrians that are driven by ORCA and SFM. In this paper, we propose a DRL approach to train NCOs w.r.t. the robot navigation. We apply these NCOs to identify vulnerabilities of the ROS navigation stack with DWA and the DRL navigation policy trained by [5].

B. Adversarial training

Sukhbaatar et al. [6] employ the asymmetric self-play framework for the training, which allows agents to automatically generate curriculums and understand the environments better. Zheng et al. [7] apply a DRL approach with adversarial training to identify possible errors in the game environment. In navigation domain, Xu et al. [8] propose a Self-supervised Environment Synthesis (SES) framework that uses generative adversarial network for environment synthesis. Their SES framework aims to identify potentially challenging navigation scenarios that need to be added to the training distribution. In this paper, we consider a game between the robot and NCOs and retrain the DRL-based robot navigation policy in the sense of asymmetric self-play. We show that the robustness of these DRL policies can be improved by retraining them in environments with NCOs.

III. Approach

In this section, we first propose a DRL approach to train a non-cooperator (NCO), i.e., the policy for the pedestrian that tries to block the robot and identify situations in which the robot may cause a collision. Then we show how to retrain a DRL-based robot navigation policy using the trained NCOs in adversarial training. Fig. 1(a) shows the training process.

A. Non-Cooperators

Here we specify details on how to train the DRL policy for a NCO. In our work, the DRL policy is composed of two sub-policies, i.e., the flashing policy and the moving policy, which output two different types of actions, i.e., the flash action and the move action, as shown in Fig. 1(b). Intuitively, the NCO first decides where to place itself around the robot, i.e., the flashing policy, then decides how to drive towards the robot, i.e., the moving policy. In our experiments, we find out that the performance of the NCO is primarily determined by the flashing policy. As a result, we first specify a moving policy for the NCO, then we train the flashing policy based on the given moving policy.

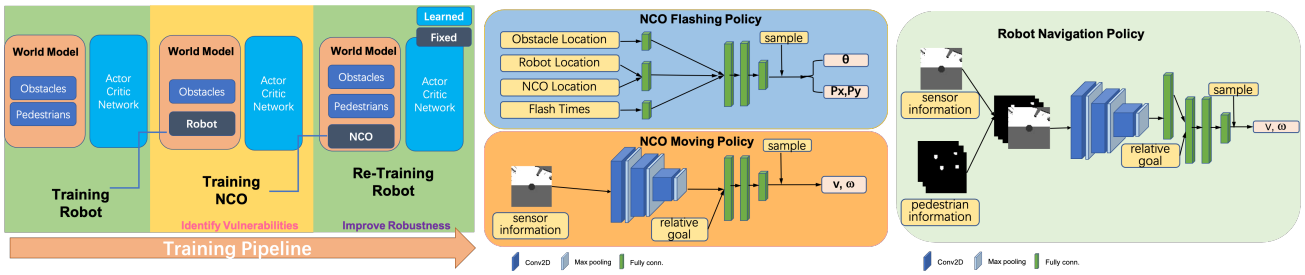
1) Flash and Move Actions: Intuitively, the NCO first uses the flash action to place itself around the robot, then uses the move action to drive towards the robot.

The flash action indicates a position (p_x, p_y) that the NCO should be placed, which intends to set the effective initial states for the NCO and the robot efficiently. In specific, it enables the NCO to teleport itself to the location away from the robot in the range of $1.5m$. We can train a DRL-based flashing policy for the flash action, which places the NCO around the robot properly while considering existing obstacles around the robot. The move action indicates the desired linear and angular velocities (v, ω) for the NCO, which drives the NCO to block the robot. We can specify a moving policy to output move actions after the NCO has been placed by the flash action.

Notice that, for both flash and move actions, the corresponding positions of the NCO are allowed to overlap with other obstacles. This is a reasonable assumption in practice, as for obstacles like low stools, pedestrians can pass through these obstacles easily while the robot is blocked.

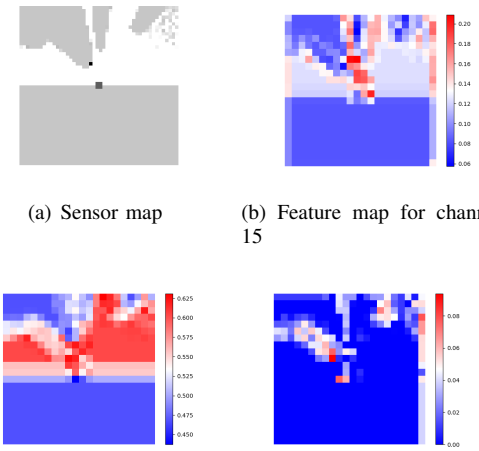
2) The Moving Policy: The moving policy drives the NCO by move actions towards the robot.

As shown in Fig. 1(b), the input of the moving policy consists of two parts, i.e., the sensor information and the robot's pose (position and orientation) w.r.t. the NCO. The sensor information specifies the surrounding environment of



(a) Robot training pipeline (b) NCO policy (c) Robot navigation policy

Fig. 1. (a) Overview of the robot training process. We first train a robot navigation policy, then train a policy simulating the behavior of NCOs to identify vulnerabilities of the robot policy. Finally, we re-train the robot policy with the help of these trained NCOs. (b) The policy of a NCO consists of two sub-policies, i.e., the flashing policy and the moving policy. (c) The robot navigation policy.



(a) Sensor map (b) Feature map for channel 15

(c) Feature map for channel 26 (d) Feature map for channel 30

Fig. 2. A representative input of sensor map and its corresponding feature map. In the NCO moving policy network, the convolutional layer that receives the sensor map has 64 channels. We make the feature maps for these channels and illustrate three of them.

the NCO which benefits to better block the robot. In our experiments, the sensor information is constructed from a $1 \times 48 \times 48$ egocentric costmap[1] that is generated from a 2D laser sensor with 180° field of view. Since robot pose may not be specified in NCO's sensor info when robot is outside NCO's view or behind an obstacle, we need to specify relative pose of robot as additional input for moving policy.

The network structure for the moving policy is shown in Fig. 1(b). The network first produces feature maps from the sensor information using three convolutional layers. Then, these refined features combined with the vector of relative position of the robot are passed to two fully-connected layers with 512 units. The $\langle$input channel, output channel, kernel size, stride length$\rangle$ of the three 2-D convolutional layers are $\langle 3,64,3,1\rangle$, $\langle 64,128,3,1\rangle$, $\langle 128,256,3,1\rangle$, respectively. At last, the network outputs the move action from the Gaussian distribution. The feature map (the output of the convolution kernel) of a trained moving policy for an NCO is shown in Fig. 2. We can see that, in Fig. 2(c), this channel focuses more on empty spaces, which response to potential collision areas with the target robot, while the channel in Fig. 2(d) focuses more on obstacles detected by the laser, including the target robot. In Fig. 2(b), the channel is more attentive to the path that the NCO may take in the future, and this path tends to be closer to static obstacles, which indicates that the NCO learns to use obstacles to block the robot.

The output of the moving policy is the move action (v, ω), where $v \in [0, 0.6]$ and $\omega \in [-0.9, 0.9]$ are two continuous values that respectively denote the desired linear and angular velocities of the NCO. Note that, $v \geq 0$, i.e., moving backward is not permitted to simulate normal behaviors of walking pedestrians. The moving policy is pre-trained using Proximal Policy Optimization (PPO) [9] in a customized 2D simulator and will be frozen while training the flashing policy. In this paper, we focus on how to train an effective flashing policy given a specified moving policy.

3) The Flashing Policy: The NCO uses the flashing policy to place itself around the robot.

As shown in Fig. 1(b), the input of the flashing policy consists of three parts, i.e., the position and circumcircle radius of obstacles, the position and orientation of both the robot and the NCO, and the number of times that the NCO has executed the flash action, with the dimension of 10×3, 2×3, 2×1, respectively.

The output of the flashing policy consists of two parts, i.e., the position (p_x, p_y) for the flash action and the real number θ indicating whether or not to execute the flash action. In specific, we set $\theta > 0$, when the NCO executes the flash action and would be placed at (p_x, p_y). When $\theta \leq 0$, i.e., the NCO decides not to execute the flash action and decides to switch to the moving policy.

The flashing policy network first uses three fully-connected layers to encode the three parts of the input. Then these refined features are combined together and passed to two fully-connected layers. At last, the policy network outputs the real number θ and the flash action, i.e., the position (p_x, p_y), from the Gaussian distribution.

Now we specify the reward function r for the flashing policy, which encourages the NCO to construct a situation

[1]http://wiki.ros.org/costmap_2d

that the robot may cause a collision in minimum steps.

$$r = r_{hit} + r_{step} + r_{flash},$$

$$r_{hit} = \begin{cases} \alpha_{hit} & \text{if the NCO collides with the front of robot,} \\ 0 & \text{otherwise,} \end{cases}$$

$$r_{flash} = \begin{cases} \alpha_{f1} & \text{if the NCO flashes } \beta_{f1} \text{ times continuously,} \\ \alpha_{f2} & \text{if the NCO doesn't flash within } \beta_{f2} \text{ steps,} \\ 0 & \text{otherwise,} \end{cases}$$

where $\alpha_{hit} > 0$ specifies the reward when there is a collision between the NCO and the front of the robot. In this case, the robot were blocked by the NCO and caused a collision. $\alpha_{f1} < 0$ denotes the penalty when the NCO uses its flash ability too often, $\alpha_{f2} < 0$ denotes the penalty when the NCO does not use its flash ability for a long time, and $r_{step} < 0$ denotes a small penalty to encourage short paths.

In our experiments, we set $\alpha_{hit} = 600$, $\alpha_{f1} = -400$, $\alpha_{f2} = -400$, $\beta_{f1} = 2$, $\beta_{f2} = 15$ and $r_{step} = -10$. Notice that, the NCO earns rewards only when there is a collision with the front of robot, which avoids the cases in which the NCO hits the robot from the back.

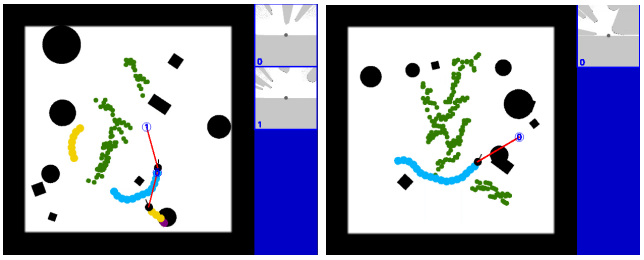
Given a specified moving policy, we train the flashing policy in a distributed training framework with the asynchronous PPO (APPO) algorithm, using simulation environments that are constructed by the same customized simulation platform for the moving policy. Similar to scenarios in Fig. 3, a training environment contains the NCO, a robot driven by its navigation policy, and a number of static obstacles. In particular, at each time step, the NCO first applies the flashing policy and receives θ and (p_x, p_y). If $\theta > 0$, the NCO will be placed at (p_x, p_y) and then apply the moving policy and perform the resulting move action. If $\theta < 0$, the NCO will directly apply the moving policy and perform the resulting move action.

In our experiments, we show that the performance of the NCO is primarily determined by the performance of the flashing policy and the trained flashing policy allows the NCO to cooperate with existing obstacles to block the robot.

B. Retraining Robot Navigation

Now we show how to retrain a DRL-based robot navigation policy using the trained NCOs. We first review the DRL approach for crowd-aware robot navigation proposed in [5] and train such a navigation policy in our simulation environments. Then we specify how to retrain this DRL policy using the trained NCOs in adversarial training.

As shown in Fig. 1, the network architecture for the robot's navigation policy is similar to the architecture for the moving policy of the NCO. The input consists of the sensor map, the pedestrian map, and the relative goal. The sensor map is a $1 \times 48 \times 48$ egocentric costmap generated by 2D laser scans. The dimension of the pedestrian map is $3 \times 48 \times 48$ where the three channels of pedestrian maps indicate the location and speed of pedestrians around the robot. The relative goal is a 3-dimensional vector representing the relative position and orientation of the robot. Yao et al. [5] show that the distinguish between obstacles and pedestrians by the sensor information and the pedestrian information can greatly improve the navigation performance of the robot in crowd environments.



(a) Training scenario (b) Testing scenario

Fig. 3. Two examples of the training scenario and the testing scenario. The small black spot denotes the robot and blue spots denote the robot's trajectory. Yellow spots denote the trajectories of the NCOs. Purple spots denote the positions where the NCOs have just flashed. Green spots denote the trajectories of other pedestrians, and other block spots or boxes denote various static obstacles.

The output of the robot's policy is the same as the moving policy for NCOs, where the ranges of the linear velocities and the angular velocities are the same as the ones for NCOs. We follow the reward function and the setting of corresponding parameters specified in [5] for the training of the robot navigation policy in our simulation environments. Fig. 3 illustrates the training scenario and the testing scenario for the retrained robot. In specific, the training scenario contains a NCO, a robot driven by the navigation policy, ten randomly placed static obstacles, and five pedestrians that are driven by ORCA with the max walking speed of 0.5 m/s. The testing scenario contains the robot and a number of randomly placed static obstacles and pedestrians driven by ORCA. Note that, we still assume that the NCOs and pedestrians can pass through static obstacles, which increases the navigating challenges for the robot.

Based on the trained NCOs, we can retrain the robot navigation policy in the sense of asymmetric self-play to improve its performance. In the above process, the navigation policy is trained in environments with pedestrians only driven by ORCA. The policy would not be robust to environments with diverse pedestrians that may behave differently. With the help of NCOs, the robustness of retrained navigation policy can be improved.

Notice that, the original robot navigation policy is trained in simulation environments with ten randomly placed obstacles and five pedestrians, where these pedestrians are driven by ORCA. In the following, we denote these training environments as the 'mild' environments. With the help of NCOs, we can enhance these mild environments by replacing a pedestrian with the NCO and result in new environments which are used in the training scenario as shown in Fig. 3(a). In our experiments, we find out that replacing one pedestrian with the NCO will be a great help in retraining the robot's navigation policy. However, replacing more pedestrians with NCOs would make the scenario too challenging for the retraining of the navigation policy, as shown in Fig. 6.We de-

note these constructed new environments as the 'aggressive' environments. Note that, we can obtain multiple versions of trained NCOs following the training process in the above subsection. Then we can use all of these versions in aggressive environments to improve the robustness of the robot navigation policy. To overcome the catastrophic forgetting problem, we can collect experiences for the training from four aggressive environments and four mild environments in parallel in our distributed training framework. Later, the robot navigation policy can be retrained based on all of these experiences.

C. Training Framework and Simulation Environments

We implemented a distributed training framework with APPO, following the architecture of Rapid [10]. In our experiments, we set one backpropagation node, one forward propagation node, and eight environment sampling nodes running in parallel. The environment sampling nodes are implemented in our customized 2D simulator, which generates the sensor information from a simulated 2D laser sensor and the pedestrian information from a simulated depth camera on the robot.

IV. Experiments

In this section, we first train the NCOs and use them to identify vulnerabilities of the two baseline robot's navigation policies based on different algorithms, i.e., DWA and PPO. Then we retrain the PPO based policy with the adversarial model and make comparisons with the above baseline methods in simulation environments. Finally, we deploy the refined PPO based policy in a real robot. The experimental results show that the NCOs can improve the performance and robustness of the retrained navigation policy.

In the following, with a slight abuse of notion, we use 'DWA' to denote the ROS navigation stack with DWA. We use 'APPO' to denote the navigation policy following the DRL approach in [5] as specified in Sec. III-B, 'Pure-PPO' as the non-asynchronous training counterpart. We use 'APPO-Normal-NCO' and 'Pure-PPO-Normal-NCO' to denote the retrained policy with the help of one NCO following the approach in Sec. III-B, respectively. We use 'APPO-two-Normal-NCOs' and 'APPO-three-Normal-NCOs' to denote the retrained policy with the help of two and three NCOs. Moreover, we implement 'naive' NCOs that are only driven by the moving policy without the flashing policy, i.e., the NCOs just rush towards the robot while ignoring other obstacles. We use 'APPO-Naive-NCO' to denote the retrained policy using the naive NCOs. The experimental results show that the flashing policy for the NCOs is crucial for the performance of the retrained navigation policy. All policies are trained on a computer with an i9-9900k CPU and an NVIDIA 3090 GPU. Detailed training parameters are specified at https://github.com/DRL-Navigation.

A. Identifying Vulnerabilities in Baseline Policies

We consider two baseline navigation policies based on DWA and PPO, respectively.

1) DWA: As shown in Fig. 4, we can train various NCOs for navigation policies based on DWA with different values of parameters in the following equation:

$$G(v,\omega) = \sigma(\alpha \cdot \text{heading}(v,\omega) + \beta \cdot \text{dist}(v,\omega) + \gamma \cdot \text{velocity}(v,\omega)),$$

where v and ω denote the linear and angular velocities of the robot, operations heading, dist, and velocity denote the robot's orientation towards its target, the distance to the closest obstacle, and the robot's forward velocity, respectively. σ, β, and γ are corresponding parameters. Notice that, the clearance coefficient β adjusts the influence of the distance from the robot to the closes obstacle, which would greatly affect the behavior of the robot w.r.t. the NCO. Without loss of generality, we set $\sigma = 1$, $\alpha = 1$, and $\gamma = 0.01$ to identify vulnerabilities of DWA with different values of β, denoted as 'DWA(β)'. Fig. 4 illustrates some vulnerabilities for 'DWA(β)' by using either the naive NCO or the NCO and scenarios without NCOs. In scenarios without NCOs, both 'DWA(0.25)' and 'DWA(5)' reach the goal successfully. In scenarios with naive NCOs, as illustrated in Fig. 4(b) and 4(g), 'DWA(0.25)' passes through two obstacles and reaches the destination with the naive NCO pursuing behind. 'DWA(5)' chooses to bypass the obstacle for more free space and the naive NCO prefers to move close to the robot rather than 'bother' it. In scenarios with NCOs, as illustrated in Fig. 4(c) and 4(h), 'DWA(0.25)' passes through two obstacles but causes the collision while there is a sudden appearance of the NCO. 'DWA' fails to avoid the collision in this situation. 'DWA(5)' continuously rotates at some places or gets stuck in a local optimum w.r.t. the NCO and nearby obstacles. Thereby, 'DWA' is not efficient in these cases.

The experimental results show that the trained NCOs can help us to identify vulnerabilities in 'DWA' and help us to analyze the effect of different values of its parameters. In specific, we find out that, 'DWA' with the large value of β is easy to get stuck by surrounding obstacles and pedestrians. Meanwhile, 'DWA' with a small value of β is not sensitive to incoming pedestrians, which would cause a collision with pedestrians who have not noticed the robot.

By comparing the performances of naive NCOs and NCOs, we can find out that the strategy of the naive NCO tends to chase behind the robot, which is not effective in identifying potential vulnerabilities of the robot navigation policy,

2) PPO: We use the trained NCO to evaluate the performance of 'APPO', as shown in Fig. 5.

In the scenario with multiple pedestrians and static obstacles as shown in Fig. 5(a), 'APPO' tends to pass through the narrow area. When there are multiple passed by pedestrians in the narrow area and one NCO moving towards the robot, 'APPO' would have trouble avoiding the collision and hit other pedestrians. In the scenario with only pedestrians and no static obstacles or NCOs as shown in Fig. 5(d), 'APPO' tends to take a big turn to avoid other pedestrians while sacrificing efficiency.

The experimental results show that the baseline DRL policy, i.e., 'APPO', has two vulnerabilities. Firstly, the

trained policy tends to stay away from pedestrians as far as possible in spite of sacrificing efficiency, since it's difficult to balance safety and efficiency in the training phase. Secondly, due to the lack of immediate reaction ability, the policy is not robust enough to ensure safety in highly constrained spaces like narrow corridors with multiple pedestrians.

B. Improving Navigation Policy

The above subsection show that the trained NCOs can find out situations that the robot's navigation policy would cause a collision in short steps. Therefore, we can use these trained NCOs to improve a DRL-based navigation policy by retraining it in environments with them.

1) Robustness of PPO-NCO: Note that, 'APPO-Normal-NCO' is retrained based on 'APPO' with NCOs. We can see that 'APPO-Normal-NCO' is more robust than 'APPO' in environments with NCOs.

Our experiments[2] show that these NCOs allow the robot to learn multiple strategies to interact with diverse pedestrians. As illustrated in Fig. 5(c), 'APPO-Normal-NCO' allows the robot to get rid of an incoming NCO by moving around static obstacles. Note that, the max speed of the robot is slightly slower than the NCO, which is a reasonable assumption in practice. Then the robot cannot easily escape from the chasing by moving to free space. However, with the help of other obstacles, the robot can escape from the chasing much easier. An interesting observation is that this strategy is widely applied to the chasing between two persons in real life.

As illustrated in Fig. 5(f), 'APPO-Normal-NCO' also allows the robot to move across a bunch of walking pedestrians without making a big turn and taking a long journey to avoid pedestrians. It can move along a short path and avoid collision by adjusting its speed. This shows that 'APPO-Normal-NCO' is robust to environments with pedestrians that may or may not cooperate with the robot.

Fig. 5(b) shows that 'APPO-Naive-NCO' also learns to move around static obstacles to get rid of an incoming NCO. However, as shown in Fig. 5(e), its strategy is very sensitive to the positions of the NCO and other pedestrians, then the robot would take more steps to avoid collisions. The following quantitative experimental results show that, without the flashing policy, 'APPO-Naive-NCO' can hardly adapt to complex scenes.

2) Performance of PPO-NCO: Here we show that NCOs are also helpful for improving the performance of the DRL navigation policy in environments without NCOs.

We compare the performance of 'DWA', 'PPO', and two retrained policies, 'APPO-Naive-NCO' and 'APPO-Normal-NCO', in various environments. And we use four metrics to evaluate the performance of navigation policies:

- *Reach rate* (*SU*): The ratio of the tests that the robot reaches the target without collision in a limited time.
- *Pedestrian collision rate* (*PC*): The percentage $(\%)$ ratio of the tests that a collision occurs between the robot and a pedestrian during the navigation.
- *Obstacle collision rate* (*OC*): The percentage $(\%)$ratio of the tests that a collision occurs between the robot and a static obstacle.
- *Average velocity speed* (*AV*): The average velocity speed of the moving robot.
- *NCO collision rate* (*KC*): The percentage $(\%)$ ratio of the tests that a collision occurs between the robot and the NCO.
- *Average navigation score (NS)*: A comprehensive assessment of a task completion speed by factoring in both navigation time and task completion [11]. The score function is as follows, where T_s means the completion time, $T_{\max}$ means the maximum time allowed by one navigation task. We set $T_{\max}$ = 30 s.

$$NS = \begin{cases} 1 - \frac{2T_s}{T_{\max}}, & \text{if success} \\ -1, & \text{otherwise} \end{cases}$$

.

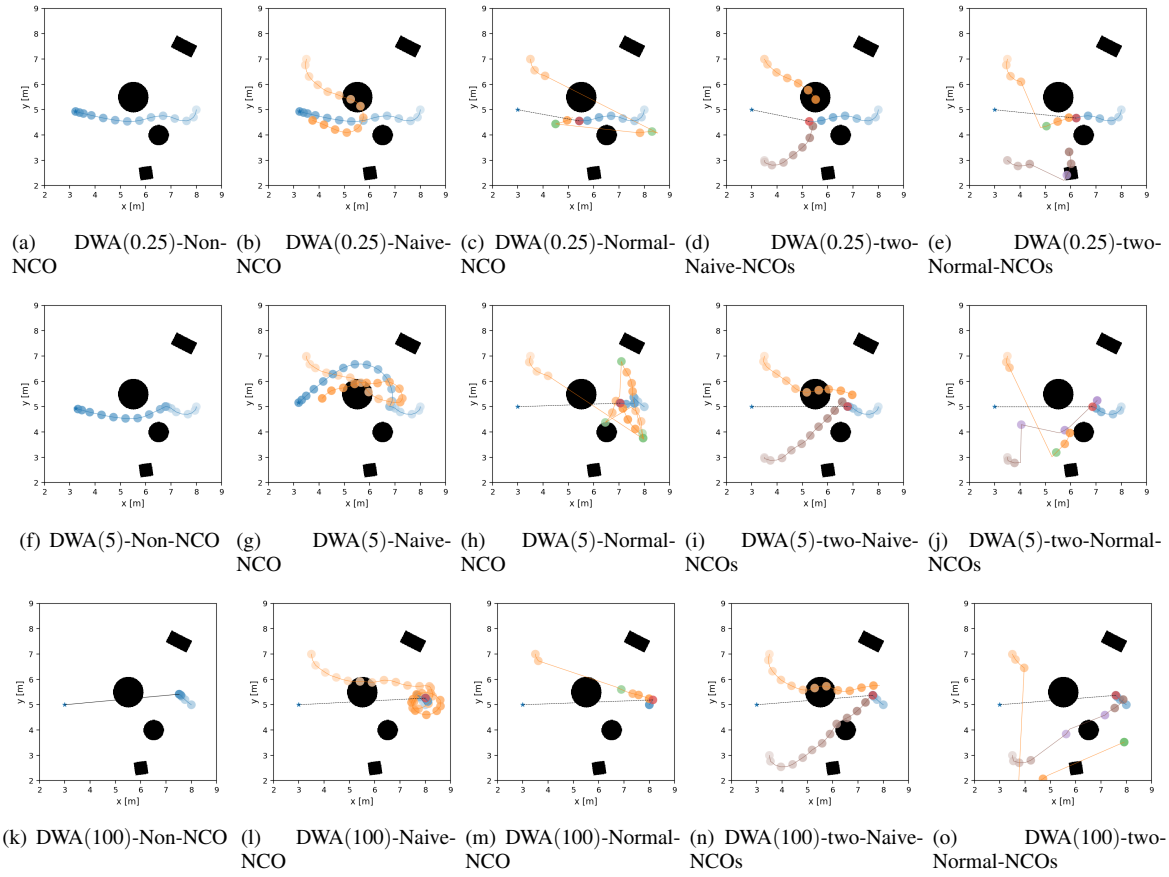
Fig. 6 shows the learning curve of 'APPO', 'Pure-PPO' and their corresponding retrained models. The reach rate curve in the training scenario in Fig. 6(a) is calculated based on the average value of the latest 100 episodes at the time step. Notice that, our method dose not have significant advantage compared with others, since the robot would be blocked by the NCO in the training process resulting lower performance. Meanwhile, there are no NCOs in the training scenario of 'Pure-PPO' and 'APPO'. The performance of the retrained model, i.e., 'Pure-PPO-Normal-NCO' and 'APPO-Normal-NCO' are enhanced compared to their baselines, which demonstrates the effectiveness of our method.

As shown in Fig. 6(b), we pick up several training models of each method at different time steps and test the generalization in 5000 random scenarios with the number of obstacles in the range of $10\sim20$ and the number of pedestrians in $5\sim10$, which greatly increased the difficulty compared to the baseline 'mild' environments. The results show that 'APPO-Normal-NCO' and 'APPO-Naive-NCO' have better performance in more constrained environments.

Table I summarizes the performance of four policies in four scenarios with static obstacles and pedestrians driven by OCRA. In specific, $\pi(N,M,K)$ denotes the scenario with N static obstacles, M pedestrians and K NCOs. We randomly generate 1000 testing environments for each scenario and record the average performance of each policy.

In constrained environments with pedestrians and obstacles, like $\pi(20,5,0)$ and $\pi(10,5,1)$, the collision rate for 'APPO-Normal-NCO' decreases greatly. The reason is that in the training phase, the NCO can pass through the obstacles and suddenly appear in front of the robot, which forces the robot to learn the ability to move away from the obstacles in environments. Fig. 5(a) illustrates such a situation. Moreover, 'APPO' does not reach the same performance as 'DWA' in the scenario with dense obstacles, like $\pi(20,0,0)$. This shows that the baseline 'APPO' policy has poor generalization in environments with dense static obstacles. Meanwhile, 'APPO-Normal-NCO' performs better in these environments, as the NCOs increase the diversity of the training experiences

[2]More illustrations can be found in the demonstration video.

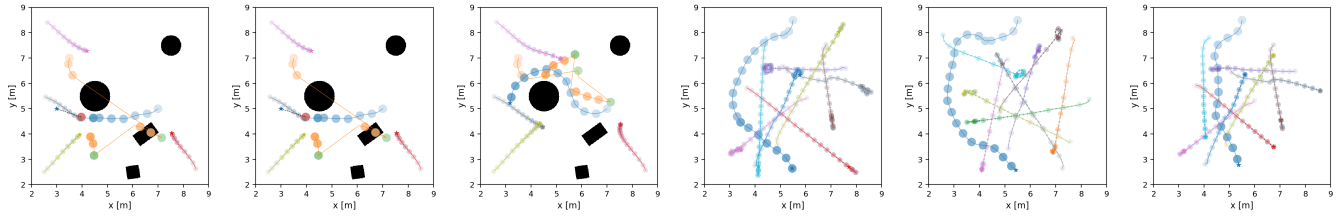


(a) DWA(0.25)-Non-NCO (b) DWA(0.25)-Naive-NCO (c) DWA(0.25)-Normal-NCO (d) DWA(0.25)-two-Naive-NCOs (e) DWA(0.25)-two-Normal-NCOs

(f) DWA(5)-Non-NCO (g) DWA(5)-Naive-NCO (h) DWA(5)-Normal-NCO (i) DWA(5)-two-Naive-NCOs (j) DWA(5)-two-Normal-NCOs

(k) DWA(100)-Non-NCO (l) DWA(100)-Naive-NCO (m) DWA(100)-Normal-NCO (n) DWA(100)-two-Naive-NCOs (o) DWA(100)-two-Normal-NCOs

Fig. 4. Trajectories of the robot driven by DWA with different values of clearance coefficient β cooperating with various NCOs. DWA(0.25) with no NCO, Naive NCO, Normal-NCO are applied in (a), (b), and (c). DWA(5) with no NCO, Naive NCO, Normal-NCO are applied in (d), (e), and (c). Gradient orange spots denote trajectories of the NCOs. Gradient blue spots denote trajectories of the robot, while red spots denote the places that the robot hits the NCO and blue stars denote the robot's targets. Orange spots denote trajectories of one NCO with green spots donote flashing action, and brown represents another NCO, whose flashing action is purple. Black spots and boxes denote various obstacles.



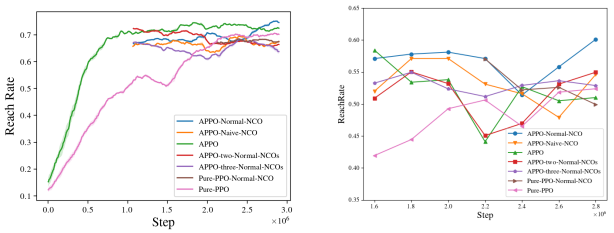
(a) APPO (b) APPO-Naive-NCO (c) APPO-NCO (d) APPO (e) APPO-Naive-NCO (f) APPO-NCO

Fig. 5. Trajectories of 'APPO', 'APPO-Normal-NCO', and 'APPO-Naive-NCO' in two scenarios. (a), (b), and (c) are environments of the scenario with four static obstacles, four pedestrians, and one NCO. (d), (e), (f) are environments of the scenario with eight pedestrians. Similar to Fig. 4, gradient blue spots and orange spots denote trajectories of the robot and the NCO, respectively. Spots with other colors denote trajectories of other pedestrians, where stars denote their target positions.

TABLE I

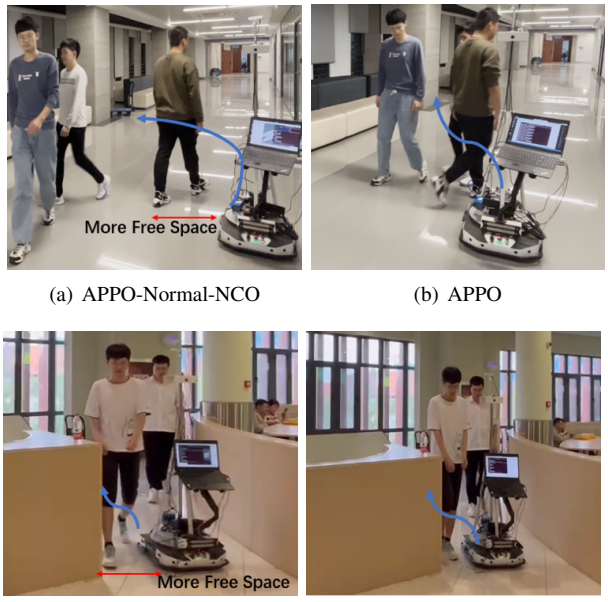
PERFORMANCE OF DIFFERENT NAVIGATION METHODS

	$\pi(20,0,0)$					$\pi(0,10,0)$					$\pi(20,5,0)$					$\pi(10,5,1)$					
Methods	SU	OC	PC	AV	NS	SU	OC	PC	AV	NS	SU	OC	PC	AV	NS	SU	OC	PC	AV	NS	KC
PPO-NCO	**81.2**	16.8	0	**0.51**	**0.09**	70.3	0	29.5	**0.52**	**-0.14**	**60.1**	26.9	**12.3**	**0.49**	**-0.24**	**57.9**	12.6	11.7	**0.55**	**-0.37**	17.4
PPO-Naive-NCO	75.5	12.4	0	0.34	0.03	**72.8**	0	**27.1**	0.49	-0.19	54.6	30.0	14.5	0.43	-0.34	51.6	17.2	**14.9**	0.52	-0.43	16.2
PPO	70.9	26.3	0	0.47	-0.08	67.1	0	32.8	0.49	-0.16	51.0	31.1	17.8	0.47	-0.36	51.5	14.6	14.2	0.54	-0.42	19.6
DWA	78.0	**0**	0	0.24	0.01	38.9	0	61.1	0.40	-0.50	47.0	**0**	41.3	0.24	-0.44	27.3	**0**	37.1	0.41	-0.65	35.6



(a) (b)

Fig. 6. Learning curve of 'APPO', 'Pure-PPO' and their corresponding retrained models. (a) is the reach rate curve in the training scenario. (b) is the reach rate curve tested in 5000 random scenarios at several specific time steps.



(a) APPO-Normal-NCO (b) APPO

(c) APPO-Normal-NCO (d) APPO

Fig. 7. Different behaviors of 'APPO-Normal-NCO' and 'APPO' in a cross circle scenario with 3 pedestrians (a, b), and a narrow corridor scenario when facing 4 incoming pedestrians (c, d).

for the navigation policy and improve its robustness in unseen environments.

In terms of the reach rate and the average navigation score, the 'APPO-Normal-NCO' achieves better performance than 'APPO-Naive-NCO' and maintains a higher velocity speed in most scenarios, which indicates that the flashing policy can greatly increase the diversity of the training experiences and encourage the robot to learn a more robust policy to navigate safely and efficiently in constrained environments.

In environments of the $\pi(0,10,0)$ scenario, which only contains ten pedestrians, 'APPO-Naive-NCO' achieves a higher success rate than 'APPO-Normal-NCO'. The reason is that these pedestrians behave similarly to the naive NCO. Then 'APPO-Naive-NCO' performs better in environments without static obstacles, while not good in environments with static obstacles.

TABLE II

PERFORMANCE ON THE PEDESTRIAN TRAJECTORY DATASETS

	ETH		HOTEL		Zara01	
methods	SU	NS	SU	NS	SU	NS
PPO-Normal-NCO	**86.3**	-0.44	**85.0**	**-0.28**	**83.8**	**-0.39**
PPO-Naive-NCO	51.3	-0.45	57.5	-0.72	72.5	-0.39
PPO	77.5	-0.26	66.3	-0.42	53.8	-0.40
DWA(0.25)	66.3	**-0.17**	47.5	-0.40	25.0	-0.63

TABLE III

PERFORMANCE ON THE REAL ROBOT

	Cross Scenario				Narrow Corridor			
Methods	SU	OC	PC	NS	SU	OC	PC	NS
PPO-Normal-NCO	**100**	0	**0**	**-0.09**	**100**	**0**	**0**	**0.15**
PPO	80	0	20	-0.19	80	**0**	20	-0.15
DWA(0.25)	20	0	100	-0.61	40	20	40	-1.00

C. Real World Experiments

1) Pedestrian Trajectory Datasets: We conduct evaluation experiments on three real-world pedestrian trajectory datasets with different outdoor social scenarios and pedestrian movement patterns. These datasets include the ETH dataset [12], the Hotel dataset [12], the ZARA01 dataset [13]. We conduct different human-robot interaction experiments such as following the pedestrian flow, moving against the pedestrian flow, and crossing the pedestrian flow for each dataset, and perform 500 trials for each model on every dataset. Table II lists the comparative results, which show that our proposed model reaches much higher performance than the baseline PPO, further demonstrating the effectiveness of our method. Interestingly, we found that our method performed better in several datasets than in our randomly generated simulation experiments. This is because the pedestrian movement pattern in these outdoor datasets is to follow the pedestrian flow, with fewer crossing pedestrians, and no obstacles present in the dataset, which provides more free space for the robot. Note that, these datasets are pre-recorded, with no real-time interaction between human and robot. Then, we conduct our method in a real robot that pedestrians can interact with them in real time.

2) Real Robot: As shown in Fig. 7, we use Agilex Tracer as a real-world differential robot with 0.7m long and 0.5m wide, and we use a ThinkPad P15v(i7-11800H CPU and Nvidia-T600 4GB GPU) for the computation device. The robot is set with a RoboSense 16 lines 3D Lidar, while we convert its outputs as the results of a 2D laser scanner in a fixed height. The robot is also set with a RealSense D455 depth camera for real-time pedestrian detection and tracking [14] based on Yolo-v3-tiny [15]. We conduct quantitative comparison experiments of 'APPO-Normal-NCO', 'APPO' and DWA(0.25) in two different scenes to test robot's ability to navigate in narrow scenes:

- *Cross Circle*: The circle cross scene where 3 pedestrians and the robot move across in a 5m range circle.
- *Narrow Corridor*: A narrow corridor with 0.9m wide in the campus cantoon where the running robot needs to face 4 coming pedestrians.

The average results are shown in Table III with each scene

conducted 5 times. The results show that 'APPO-Normal-NCO' behaves more stable than 'PPO' in these scenarios. Through adversarial training, 'APPO-Normal-NCO' improve its ability to comprehend pedestrian movement intention and navigate safely and efficiently in pedestrian constrained environments. A main improvement of our method over 'PPO' is that it would keep an appropriate distance from pedestrians to make free space for itself to guarantee safety, as shown in Fig. 7. Additionally, the frequency of the robot having to stop was lower for 'APPO-Normal-NCO', suggesting that our method was more efficient in navigating around pedestrians while maintaining a safe distance.

V. Conclusions

In this paper, we identify a special kind of pedestrians as non-cooperators (NCOs), that try to block the robot and find out collisions caused by the robot. We propose a DRL approach to train a policy simulating the behavior of these NCOs. We specify the policy of NCOs as the combination of two sub-policies, i.e., the flashing policy and the moving policy, where the flashing policy places the NCOs around the robot and the moving policy drives the NCOs towards the robot. We show that the trained NCOs can not only identify vulnerabilities of a given navigation policy, but also improve its performance by retraining the policy in environments with these NCOs. We evaluate our approach on the ROS navigation stack with DWA and a DRL-based navigation policy. The experimental results show that the trained NCOs can identify vulnerabilities of both policies effectively. The experiments also show that the flashing policy of the NCOs can greatly increase the diversity of the interactions between the robot and the NCO, which is crucial for the performance. We evaluate the retrained navigation policy in various crowd environments with diverse pedestrians. The experimental results show that the approach can improve the robustness of the navigation policy in various environments. The source code for the training framework and the simulation platform are released online.

References

[1] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[2] R. Mirsky, X. Xiao, J. Hart, and P. Stone, "Prevention and resolution of conflicts in social navigation–a survey," *arXiv preprint arXiv:2106.12113*, 2021.

[3] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[4] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011, pp. 3–19.

[5] S. Yao, G. Chen, Q. Qiu, J. Ma, X. Chen, and J. Ji, "Crowd-aware robot navigation for pedestrians with multiple collision avoidance strategies via map-based deep reinforcement learning," in *Proceedings of IROS 2021*. IEEE, 2021, pp. 8144–8150.

[6] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *arXiv preprint arXiv:1703.05407*, 2017.

[7] Y. Zheng, X. Xie, T. Su, L. Ma, J. Hao, Z. Meng, Y. Liu, R. Shen, Y. Chen, and C. Fan, "Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning," in *in Proceedings of 34th ASE-2019*. IEEE, 2019, pp. 772–784.

[8] Z. Xu, A. Nair, X. Xiao, and P. Stone, "Learning real-world autonomous navigation by self-supervised environment synthesis," *arXiv preprint arXiv:2210.04852*, 2022.

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

[10] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dêbiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[11] W. Zhang, N. Liu, and Y. Zhang, "Learn to navigate maplessly with varied lidar configurations: A support point-based approach," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1918–1925, 2021.

[12] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," in *2009 IEEE 12th ICCV*. IEEE, 2009, pp. 261–268.

[13] A. Lerner, Y. Chrysanthou, and D. Lischinski, "Crowds by example," in *Computer graphics forum*, vol. 26, no. 3. Wiley Online Library, 2007, pp. 655–664.

[14] T. Linder, S. Breuers, B. Leibe, and K. O. Arras, "On multi-modal people tracking from mobile platforms in very crowded and dynamic environments," in *Proceedings of ICRA-2016*. IEEE, 2016, pp. 5512–5519.

[15] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.