2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Robot Navigation in Crowded Environments Using Deep Reinforcement Learning

Lucia Liu[1,2], Daniel Dugas[1], Gianluca Cesari[2], Roland Siegwart[1], Renaud Dubé[2]

***Abstract*— Mobile robots operating in public environments require the ability to navigate among humans and other obstacles in a socially compliant and safe manner. This work presents a combined imitation learning and deep reinforcement learning approach for motion planning in such crowded and cluttered environments. By separately processing information related to static and dynamic objects, we enable our network to learn motion patterns that are tailored to real-world environments. Our model is also designed such that it can handle usual cases in which robots can be equipped with sensor suites that only offer limited field of view. Our model outperforms current state-of-the-art approaches, which is shown in simulated environments containing human-like agents and static obstacles. Additionally, we demonstrate the real-time performance and applicability of our model by successfully navigating a robotic platform through real-world environments.**

I. INTRODUCTION

Considering the fast-paced advancements in the field of mobile robotics, it is foreseeable that autonomous robots will shift from predominantly operating in structured industrial environments to being deployed in unstructured and dynamic spaces as well. These applications include, among others, autonomous cleaning, delivery, and inspection, where robots require the ability of safely navigating in the presence of highly dynamic obstacles, such as pedestrians. Previous efforts to explicitly model pedestrian behaviours show the difficulty associated with predicting the movements of humans [1], [2], [3]. Contrastingly, the usage of well-known obstacle avoidance algorithms that assume static obstacles [4], [5], [6] or make assumptions on the behavior of other agents [7], [8] can yield sub-optimal solutions in real-world environments.

With the rapid improvements in the fields of machine learning and deep reinforcement learning (DRL), recent works started to explore the usage of neural networks for robot navigation in dynamic environments. End-to-end solutions have been developed, allowing navigation through dynamic environments based on raw sensor input data [9], [10], [11]. While dynamic obstacle avoidance and navigating to a goal is possible using these approaches, they are limited in modeling interactions between agents and navigating in a socially acceptable manner. Additionally, it is more difficult to analyze end-to-end policies and ensuring safe operation using these algorithms. Other works do model humans explicitly to capture individual interactions but do not differentiate between pedestrians and static obstacles [12], [13], [14]. However, real-life environments often contain static obstacles of various shapes and it may be desirable to move differently when approaching static obstacles compared to standing pedestrians.

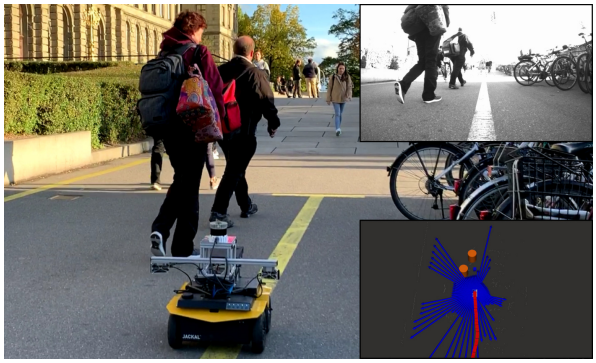


Fig. 1. Our deep reinforcement learning-based approach allows a robot to navigate autonomously through a real-world scenario including pedestrians and static obstacles. **Upper right**: Image of the stereo grayscale camera used to detect pedestrians. **Bottom right**: Angular map in blue, created using a 3D Lidar scan. Detected pedestrians are shown as orange cylinders and the robot's estimated previous path is shown in red.

In this work, we build on approaches from [13], [14] and [15] to develop a DRL network for navigation in crowded environments that successfully addresses the aforementioned limitations. We call our approach *Social Obstacle Avoidance using Deep Reinforcement Learning* (SOADRL). By passing static obstacles and pedestrians separately to the model, we enable the network to explicitly leverage the information of different obstacle types and eliminate the need to model static obstacles as pedestrians. Our algorithm allows for an arbitrary amount of pedestrians, including situations where no pedestrians are observable. For training, we use an asynchronous advantage actor-critic (A3C) framework [16] which enables us to increase the training speed through parallel usage of multiple actors. By training a policy network, no assumptions on the behaviour of other agents have to be made since it eliminates the necessity to predict the next state and its value for each possible action. Additionally, using a policy network reduces the computational costs associated with finding the best action, which is particularly important for real-time robot control. Finally, previous works often assume a 360° field of view (FOV) [12], [13], [14]. However, achieving such FOV often calls for expensive sensors and/or imposes strong constraints on the mechanical design. In this work, we show that our approach generalizes to real-life setups with a limited FOV.

[1]Autonomous Systems Lab, ETH Zurich, Zurich, Switzerland. {luliu, daniel.dugas, rsiegwart}@ethz.ch
[2]Sevensense Robotics AG, Zurich, Switzerland. {gianluca.cesari, renaud.dube}@sevensense.ch

In summary, our main contributions are:

1) a novel DRL based approach combining pedestrian and static obstacle representations for robot navigation in crowded environments,
2) a strategy that allows training and optimal policy generation in the absence of pedestrians and with a limited FOV,
3) an extensive evaluation of the algorithm performance in diverse challenging simulation environments and a demonstration in real-world situations[1].

II. BACKGROUND

A. Related Work

In the field of robotics, collision avoidance and safe navigation are highly researched topics as safety is of utmost importance when operating robots [17]. This becomes especially important in dynamic and populated environments, where the robot might be required to deviate from precalculated paths to avoid humans and other obstacles.

Navigation in crowded environments combines the two areas of collision-free navigation and modeling human behaviour. In fact, many approaches combine these methods by first predicting future trajectories of the humans in the environment and then calculating the best possible paths to reach a goal. Chen et al. [18] use a socially conscious recurrent neural network model for trajectory prediction and sample feasible waypoints for an optimal path. In [19] a pedestrian motion pattern is learned using Gaussian processes (GP) and chance-constrained rapidly exploring random trees are used for path planning. However, these methods are computationally expensive especially in crowded environments as they require the prediction of the trajectory for each pedestrian. Additionally, uncertainty in the predictions can quickly result in *frozen robot* situations, where a guaranteed safe path cannot be found. By taking the reactive behaviour of humans into account and assuming them to actively avoid the robot, these situations can be prevented. Common approaches for dynamic robot navigation are based on such reactive methods that assume human movements to follow certain policies, such as Optimal Reciprocal Collision Avoidance (ORCA) [7] and Reciprocal Velocity Obstacles [8], where the pedestrians' movements are expected to account for half of the necessary distance for avoidance. While these models are sufficient in artificial environments where all agents execute this behaviour, they are sub-optimal in real-world environments with decision-making agents that might move differently.

Other recent work explored the usage of DRL for navigation in dynamic environments. For this, a network is trained to provide a control action which navigates a robot around obstacles towards a goal. This includes end-to-end solutions that directly process data from a 2D laser range finder using convolutional neural networks (CNN) and output an action command for the robot [9]. Pfeiffer et al. [10] use a similar approach and combine imitation learning (IL) with DRL for the training of the network, which results in a significant reduction of training time and allowed the usage of sparse rewards. However, end-to-end approaches are generally harder to introspect and require more training to model interactions between agents and to differentiate between different kinds of obstacles, which is crucial for executing a socially compliant behaviour.

To explicitly incorporate interactions between agents, Chen et al. [20] obtain information about the agents in the environment and process them separately using a value network to determine the robot's optimal control action. This work was later extended by modifying the reward function to make the agent comply to social norms [12]. To avoid the need for assumptions on the pedestrians, Everett et al. [13] train a policy network based on the previous work that does not require a one-step lookahead. They additionally make use of LSTMs to allow arbitrary number of pedestrian observations. Our method builds upon the approach of Chen et al. [14], where the authors developed a neural network architecture that explicitly models human-robot and robot-robot interaction. However, the presented approaches do not specifically model static obstacles but incorporate them as standing pedestrians. This can result in sub-optimal paths as the networks are trained to keep a socially acceptable distance to pedestrians and expect reactive behaviour. Pokle et al. [21] use lidar data and trajectories of observed humans, as well as a global plan and the robot's odometry as inputs to their planner. The authors propose to separate the action generation into first obtaining a local plan, which is an input to a velocity controller network that predicts the low-level commands. However, while their experiments in simulated environments were successful, the performance in real-world scenarios is yet to be shown.

B. Deep Reinforcement Learning using A3C

In reinforcement learning (RL) the environment is formulated as a Markov decision process described by tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, state transition probability matrix $\mathcal{P}$, reward function $\mathcal{R}$ and discount factor $\gamma \in [0,1)$. The goal is to find a policy π that maximizes the expected return for every state $s \in \mathcal{S}$ described as

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \Big| s_0 = s\right] \tag{1}$$

with $r_t \in \mathcal{R}$ being a scalar reward received when moving from state s_t at time step t to state s_{t+1} at the next time step $t+1$. In our case the states contain information about the humans in the environment, static obstacles, the goal position and the agent's current speed and size. The action space $\mathcal{A}$ contains discrete control commands to the agent, described as tuple of velocity and rotation. π returns the likelihood of being the optimal action for each possible action $a \in \mathcal{A}$ given a current state s_t. In RL the policy π is found by taking actions and letting the agent explore its learning environment with the objective of maximizing the cumulative rewards. For this, we simulate environments including pedestrians controlled

[1] https://youtu.be/Sccs1y16S3g

by different policies and static obstacles of different sizes and shapes. To obtain a positive reward from the environment, the agent is required to reach a goal location without colliding with obstacles.

Our work uses a policy-based model-free RL method, which directly parameterizes the policy $\pi(a|s,\theta)$ and updates the parameters θ by performing gradient ascent on the expected return. The framework used to perform policy optimization is called asynchronous advantage actor-critic (A3C) [16] which maintains both, a policy function $\pi(a_t|s_t;\theta)$ ('Actor') with parameters θ and a value function $V(s_t;\theta_v)$ ('Critic') with parameters θ_v and updates these after every n steps. During training, A3C has the ability to run separate learning environments in parallel, all updating a joint learner network. This provides the possibility to leverage multiple cores, which greatly accelerates the learning process.

III. APPROACH

A. Neural network model

The proposed network architectures for SOADRL are depicted in Fig. 2. They consist of three different channels that separately process (i.) information about the goal location and the robot, (ii.) pedestrians and (iii.) static obstacles. Our model supports two techniques to represent static obstacles: angular maps and occupancy grids. The outputs of these channels are concatenated and fed into two fully connected (FC) layers. The policy function and the value function are obtained by passing the output of these layers into two separate final fully connected layers. The final layer for the value function outputs a single value, which is an estimate for the future discounted reward of the state. The final policy layer is trained to output the probability of being the optimal action for each possible action in the discrete action space, which is normalized using a softmax activation function.

1) Goal and Robot channel: Our first input contains information about the goal and robot state. The robot state is parametrized as follows:

$$\mathbf{s} = \left[d_g, v_{pref}, \alpha, r, v'_x, v'_y\right], \tag{2}$$

where $d_g = ||\mathbf{p}_r - \mathbf{p}_g||_2$ is the distance between the robot's center point and the goal in meters. v_{pref} is the preferred forward velocity of the robot and provides an upper limit to the robot's speed. r is the robot's radius and $[v'_x, v'_y]$ is the robot's current velocity in the robot centered frame. The origin of the robot centered frame is fixed to the robot's location and the x-axis points to the direction of the goal. α is the difference of the current heading to the angle to the goal bearing.

2) Pedestrian channel: To process the pedestrian features, we integrated the interaction module and the pooling module introduced by Chen et al. [14]. This network is designed to model the human-human and human-robot interactions and calculates an attention score for each pedestrian in the robot's environment. The interaction module captures these interactions through maps that contain all agents in the neighborhood for each human. The pooling module computes an attention score for each pedestrian and combines these to a value that represents the crowd.

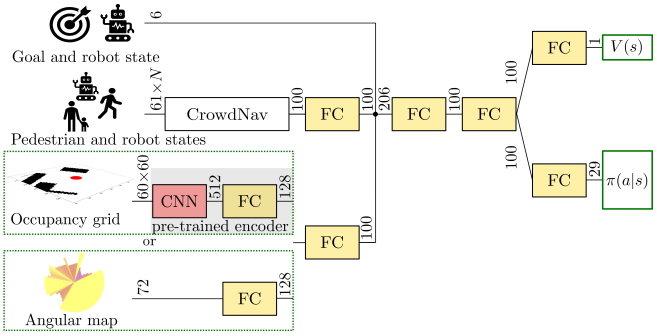


Fig. 2. The proposed neural networks for modeling the value function and policy function for motion planning in potentially crowded environments. The networks consist of three input channels whose outputs are concatenated. Pedestrian information is processed following [14]. To use an occupancy grid as static obstacle input, the upper network for the third channel is chosen. The occupancy grid is first processed by a pre-trained encoder, following [15]. When using an angular map as static obstacle input, the second network is applied where the angular map features are processed using two FC layers. The network is split at the final FC layer to obtain a policy $\pi(a|s)$ and a value $V(s)$.

3) Static obstacle channel: To encode static obstacles in the environment we investigated two different representations, angular maps and occupancy grids.

Angular map: The angular map encodes the distance of the next obstacle within angular ranges, as shown in Fig. 3, which is inspired by the angular pedestrian grid in [15]. The output of this map is an one dimensional vector $\mathbf{l}$ with the elements l_k being described as:

$$l_k = \min\left(l_{\max}, \left\{\rho_{i,k} | i \in \{1,\ldots,N_{obs}\}\right\}\right) \tag{3}$$

l_{max} is the maximum range of the angular map, N_{obs} is the total number of obstacles in the FOV of the agent that ranges from $\alpha_{\min}$ to $\alpha_{\max}$ with $\alpha_{\min}, \alpha_{\max} \in [-\pi, \pi]$ and $k \in \{1,\ldots,K\}$. K defines the number of uniform angular cells with the opening angles:

$$\psi_k = (k-1)\frac{\alpha_{\max} - \alpha_{\min}}{K} \tag{4}$$

$\rho_{i,k}$ is the distance of the closest point of obstacle i to the agent within the angular range of slice k and is infinity if the obstacle has no component that lies within this range.

To simplify obstacle avoidance, we incorporate the robot's dimensions into the angular map. When using point clouds for the map generation, this requires the inflation of each point by the dimensions of the agent. This allows the agent to move in any direction of free space depicted by the map without collision.

The advantage of using an angular map is that it can be easily obtained in real-time from the robot's sensors, e.g. through laser range finders. Additionally, angular maps reflect the importance of close obstacles as they influence more cells. This provides a more detailed representation of close obstacles compared to distant obstacles which are of less importance to the robot. We choose a discretization of 5°, resulting in 72 slices with a maximum range $l_{\max}$ of 6 m for

(a) Angular map with full FOV.

(b) Angular map with limited FOV.

(c) Top-down view of the environment with robot.

Fig. 3. Representation of static obstacles with an angular map. The number of slices for both, full FOV and limited FOV is 72. The length and color of each slice depict the distance of the closest obstacle to the robot within the slice's angular range - the darker the color, the closer the obstacle. The maximum distance of both angular maps is 6 m.

each cell. For the angular map with a limited FOV we choose $\alpha_{\min} = -0.23464\,\text{rad}\ (-13.4°)$ and $\alpha_{\max} = 0.20015\,\text{rad}\ (11.5°)$, which is the FOV of the Sevensense stereo cameras used on our hardware platform. It is asymmetrical due to the blockmatching procedure for obtaining the obstacle point clouds from the two images.

Occupancy grid: Our second static obstacle representation is an occupancy grid of a local extract of the robot's environment. It is centered around the robot and rotated around its heading. Occupied cells are marked with the value 0, free cells with the value 1. The grid is processed using a CNN and two FC layers. The CNN is pre-trained separately from the rest of the network using a convolutional auto-encoder (AE), following the implementation in [15]. The CNN weights then stay fixed for the RL step, which results in a significant reduction in training time.
The advantage of the occupancy grid is that it encodes all obstacles within a certain range, allowing the agent to take more optimal actions from a long-term perspective. However, this method requires prior knowledge of the environment to create an accurate occupancy map. We choose the size of the local grid map to $6\times6\,\text{m}$ around the agent with a resolution of 0.1 m per cell.

4) Adaptive model selection: An advantage of the integrated pedestrian model architecture from [14] is its ability to allow various numbers of humans. However, the original network does assume at least one pedestrian feature vector to generate an action. While it is less of a problem in the original setup where static obstacles are modeled as pedestrians, it is important that our network can determine an action to avoid static obstacles when no pedestrians are in the robot's FOV. To allow action generation in these situations, we have created a *no-humans* network, identical to the main network in Fig. 2 only without the pedestrian channel. In the final network graph we maintain both, a *no-humans* model and a main model in parallel. During training and execution, the correct model for inference and updating needs to be chosen at every step.

B. Reinforcement learning

1) Action space: Our action space is discretized and designed for a non-holonomic robot. Each action consists of a combination of a forward velocity and a rotational action. The action space of the forward velocity contains four unitless factors k_v with values in $[0,1]$. The forward velocity is then obtained by scaling the robot's preferred speed v_{pref} with this factor in a post-processing step: $v = k_v \cdot v_{pref}$.
For the rotational action space we use $\Delta\phi$ as change in heading angle per time step which is equally spaced with seven values from $-\frac{\pi}{20}$ to $\frac{\pi}{20}$.

2) Reward: The objective of the reward function is to enable the network to learn which actions are preferable within any given environment. For this, we formulate a sparse reward that provides the necessary feedback as follows:

$$r(s_t) = \begin{cases} 1 & \text{if goal reached} \\ -0.25 & \text{else if collision} \\ r_{disc,ped} + r_{disc,obst} + r_{rot} & \text{otherwise} \end{cases} \tag{5}$$

The reward function awards the agent when reaching the goal and penalizes it when colliding with pedestrians or static obstacles. Additionally, we want to avoid proximity to pedestrians as this violates their comfort zone. $r_{disc,ped} = \min(0,\ 0.5(d_{\min,ped} - d_{disc})\Delta t)$, which penalizes the agent when its distance to the closest pedestrian $d_{\min,ped}$ is smaller than a distance d_{disc}, which we choose to be 0.2 m. $r_{disc,obst} = -0.05 \cdot \Delta t$ penalizes the agent when it is closer to a static obstacle than 0.2 m and is 0 otherwise. Finally, we want to encourage a smooth trajectory to make the robot's movements more predictable for other agents. For this, we add a small negative reward for angular movements: $r_{rot} = -|\Delta\phi| \cdot 0.005$.
Following [13] and [14], our discount factor for the discounted cumulative reward is dependent on the preferred speed and time step Δt.

$$R_t = r_t + \sum_{k=t+1}^{T} \gamma^{(k-t-1)\cdot\Delta t\cdot v_{pref}} r_k \tag{6}$$

3) Environment: For the training of SOADRL we use an environment that simulates the movements of humans in crowds. We insert human-like agents into the environment controlled by different policies, such as ORCA [7], the trained policy of our robot at that iteration, and random forward movements. Both in training and testing we randomize the ability of other agents to perceive our robot to account for possible non-cooperative behaviour. Human and robot sizes and preferred speeds can be randomized by uniformly sampling from $r \in [0.3,0.5]$m and $v_{pref} \in [0.6,1.2]$m/s. All agents in the environment are given a goal in a randomized distance from their starting position. Once humans reach their destination in the environment they stay at their goal location and become static pedestrians. Furthermore, static obstacles shaped as walls or squares are added into the environment with randomized sizes and locations.

4) Training process: Following [9], [10], [13], we use supervised imitation learning (IL) to reduce the training time of our model. During the IL step we collect state-action pairs and their corresponding rewards from ORCA agents moving through our environment towards a goal. While the goal can

be reached occasionally for easy setups after the IL step, we observed that the agent cannot yet generalize the avoidance of obstacles to more complex situations. During RL training, the action chosen by the agent is sampled from the policy distribution which is the output of our policy network to encourage exploration. For testing and execution the action with the highest probability is chosen.

Joint training of *no-humans* model and main model: To allow our model to select an optimal policy for environments without other agents, we train a *no-humans* model in addition to our main model. This *no-humans* model is first trained separately on environments with only static obstacles. Once the networks are trained separately, they are both included in the final graph and trained in a joint environment. This is done by maintaining separate batches - one for transitions that include humans in the agent's FOV and one for transitions that do not. When updating the network, both models are updated using their respective batch. The target value R_t for each transition is the discounted cumulative reward over the following transitions within the horizon, regardless of the number of humans in the FOV.

Training stages: We train our model using curriculum learning where we increase the complexity of our environment in multiple stages. The first stage of training uses an environment with a randomized goal distance $d_g \in [4,10]$m, 5 ORCA agents and a randomized number of square and wall shaped obstacles $n_s, n_w \in [0,4]$. We first run 10 iterations of IL where each iteration collects experiences from 500 episodes and executes 50 epochs for training with a batch size of 100. The success rate after this step is ~ 0.18. Afterwards, we run RL using A3C with 12 workers which converges after 6h with $1.7 \cdot 10^6$ transitions in total and a success rate of 0.96 when using an angular map. For the model using the occupancy grid with a pre-trained CNN, it takes 7h and $1.1 \cdot 10^6$ transitions in total to converge to a success rate of 0.94.

Afterwards, we increase the amount of static obstacles and add humans executing different policies. The total amount of agents is sampled from [4, 12]. The goal distance is increased to $d_g \in [6,14]$m and the speed and size of humans and robot are randomized for each training episode. This stage converges after 3h ($0.5 \cdot 10^6$ transitions) at a success rate of 0.88 for the angular map model and after 4.5h ($0.9 \cdot 10^6$ transitions) at a success rate of 0.80 for the occupancy grid model.

Training parameters: The learning rate for IL is chosen as $1 \cdot 10^{-4}$ and for RL as $2 \cdot 10^{-5}$. An update is performed after every $T = 5$ transitions, discount $\gamma = 0.9$, entropy coefficient $\beta = 1 \cdot 10^{-3}$ and Adam [22] is used for optimization. The time step for each transition is chosen as $\Delta t = 0.2$ s. An episode is terminated after 35 s without reaching the goal or collision.

IV. EXPERIMENTS

A. Computational details

The model is implemented in Python using Tensorflow and trained on a computer with a GeForce RTX 2080 Ti GPU and an Intel Core i7-8700 CPU with 6 2-threaded cores. Testing was done on an Intel Core i7-4900MQ CPU.

B. Simulation results

For evaluation we compare the usage of an angular map (SOADRL-AM) and an occupancy grid (SOADRL-OG). Additionally, we trained models for full FOV and limited FOV operation. We compare our models to ORCA [7], SARL with Local Map [14] and GA3C-CADRL [13]. For the evaluation of GA3C-CADRL, we used the weights provided by the authors. For SARL we trained their model with the parameters suggested by the authors for the full FOV operation. For limited FOV operation we fed a dummy pedestrian with $p_x = p_y = v_x = v_y = r = 0$ into the network in cases where no pedestrian exists in the agent's FOV.

All tests were conducted in two different environments with v_{pref} and r randomized for all agents and a goal distance uniformly sampled from [8, 18]m. Additionally, we randomized the humans' ability to perceive our robot. The first environment contains only pedestrians and no static obstacles with 5 to 15 humans. The second environment contains between 2 and 7 humans and 0 to 20 static obstacles. Tests ran the same 500 episodes of randomized environments for all compared policies. ORCA was used as human policy.

For evaluation we compared the metrics: Success rate, collision rate, time to goal, discomfort frequency. Collision cases may include situations where humans collide with the agent as not all humans are able to perceive our agent. The time to goal is calculated as average time for all success cases for each policy. The discomfort frequency describes the proportion of time steps where the robot is closer to a static obstacle or a pedestrian than our defined discomfort distance of 0.2 m.

1) Environments without static obstacles: With more pedestrians in the environment, our models perform better than SARL in reaching the goal, as they were trained on a more diverse environment (see Table Ia). CADRL shows a similar success rate but has a higher collision rate. As evaluated in [14], CADRL pays less attention to the heading of the pedestrians and their interactions. ORCA assumes other agents to perceive the robot and actively avoid it. We included humans that do not have this ability, which causes ORCA agents to occasionally get stuck behind other agents. Collision rate and discomfort frequency are 0 for ORCA by design.

2) Environments with static obstacles: Static obstacles are fed into the SARL and CADRL networks as multiple standing pedestrians that add up to the size of the obstacle. The collision rate of SARL and CADRL is higher than our models' and it could be observed that ORCA agents often get stuck behind static obstacles when they are blocking the agent's path, which results in a low success rate (see Table Ib). The ability of our model to differentiate between standing pedestrians and static obstacles allows the identification of the best action when facing different obstacles. While standing pedestrians might start to move, a greater distance should be ensured for moving past pedestrians than

static obstacles. On the other hand, paths blocked by standing pedestrians might be subject to change while paths blocked by static obstacles are permanently infeasible. A sample trajectory is shown in Fig. 4.

TABLE I

COMPARISON OF THE DIFFERENT POLICIES WITH FULL FOV. "SUCCESS" DESCRIBES THE PERCENTAGE OF CASES WHERE THE AGENT REACHES THE GOAL, "COLLISION" IS THE PERCENTAGE OF CASES WHERE THE AGENT COLLIDES. "TIME" DESCRIBES THE AVERAGE TIME TO REACH THE GOAL. "DISC." STANDS FOR DISCOMFORT FREQUENCY.

Policy	Success	Collision	Time [s]	Disc.
ORCA	0.74	**0.00**	20.19	**0.00**
CADRL	0.82	0.17	18.49	0.03
SARL	0.77	0.14	18.79	0.05
SOADRL-AM	0.84	0.09	17.99	0.10
SOADRL-OG	**0.87**	0.13	**17.92**	0.03

(a) Environment without static obstacles.

Policy	Success	Collision	Time [s]	Disc.
ORCA	0.42	**0.01**	19.15	0.12
CADRL	0.65	0.26	**18.93**	0.06
SARL	0.35	0.29	19.84	0.20
SOADRL-AM	**0.80**	0.11	18.94	0.08
SOADRL-OG	0.78	0.17	19.60	**0.03**

(b) Environment with static obstacles.

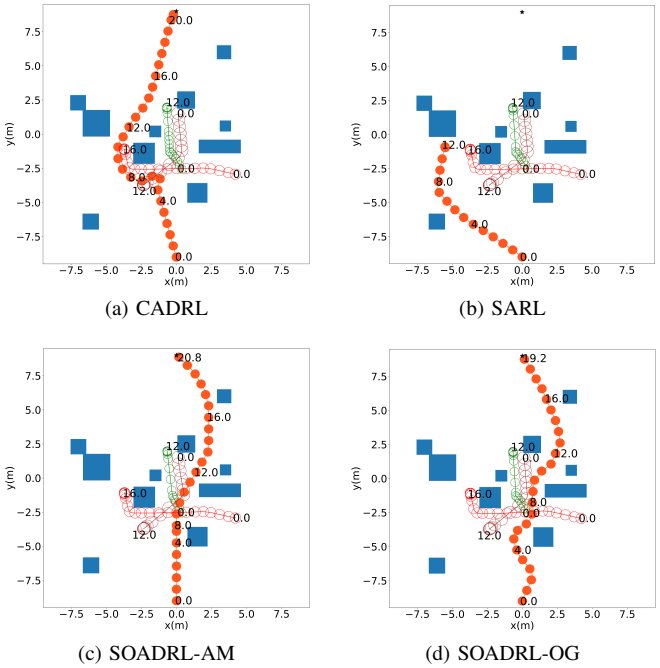
3) Limited FOV: To evaluate the policies with limited FOV, we distinguish between collisions with humans inside the robot's FOV and collisions with humans outside of the robot's FOV. For cases with no static obstacles, SARL and our model perform similarly well (see Table IIa). It could be observed that our learned policy and SARL avoid moving into directions outside of the FOV, resulting in slowing down more often to wait for pedestrians to pass. CADRL often causes collisions by agents that are outside of its FOV.

When adding static obstacles, the success rate of our model becomes twice as high compared to SARL and CADRL (see Table IIb). In many of the scenarios where our model succeeds and SARL and CADRL fail, our learned policy is seen to explore the space more than others, which results in finding paths outside of its FOV (see Fig. 5).

4) Runtime evaluation: Our model uses a policy network which is able to return the best action after one pass through the network which takes ~0.5ms on a CPU. This makes our algorithm significantly faster compared to SARL, which only trains a value network and therefore needs to infer the value of every possible state s_{t+1} for every action in the action space at every step. With 80 possible discrete actions in the original SARL setup, we were able to measure our algorithm to be ~80x faster.

C. Real-world experiments

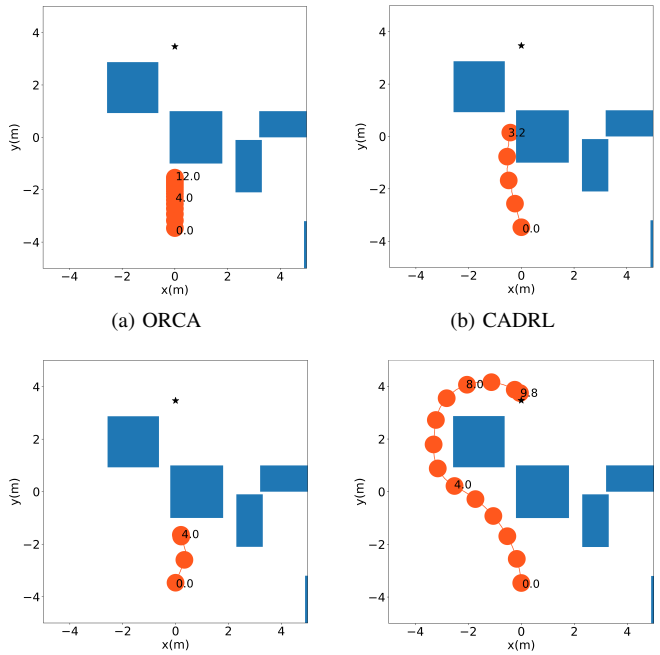
The SOADRL-AM network was integrated into our navigation framework on a mobile robot platform to test its real-world performance. It is equipped with an Ouster 3D Lidar and a Sevensense grayscale stereo camera in the front. The 3D Lidar is used to generate the angular map representation. The stereo camera with a FOV of $\alpha \in [-0.23464, 0.20015]$rad



(a) CADRL (b) SARL

(c) SOADRL-AM (d) SOADRL-OG

Fig. 4. Example trajectories for the compared policies with full FOV with static obstacles and three ORCA agents. The robot trajectory is shown in orange. The SARL agent stops behind a static obstacle instead of moving around it. While the CADRL agent decides to take a detour, our SOADRL-AM agent waits for the pedestrians to pass the narrow path before continuing. The SOADRL-OG agent is also able to identify a direct path.



(a) ORCA (b) CADRL

(c) SARL (d) SOADRL-AM

Fig. 5. Example trajectories for the compared policies with limited FOV and with static obstacles. The ORCA agent stops behind a static obstacle as well as the SARL agent. The CADRL agent collides while trying to pass between the two obstacles. Only our SOADRL agent identifies a feasible path despite the limited FOV and two large obstacles blocking the path.

TABLE II

COMPARISON OF THE DIFFERENT POLICIES WITH LIMITED FOV. "COLL. SELF" DESCRIBES THE RATE OF COLLISIONS WITH AGENTS WITHIN THE FOV OF THE ROBOT. "COLL. OTHER" IS THE RATE OF COLLISIONS CAUSED BY AGENTS OUTSIDE OF THE FOV.

Policy	Success	Coll. self	Coll. others
ORCA	0.44	0.01	0.54
CADRL	0.27	0.02	0.72
SARL	0.42	0.03	**0.14**
SOADRL-AM	**0.55**	**0.00**	0.42

(a) Environment without static obstacles.

Policy	Success	Coll. self	Coll. others
ORCA	0.32	**0.01**	0.18
CADRL	0.18	0.48	0.34
SARL	0.11	0.13	**0.04**
SOADRL-AM	**0.60**	0.06	0.26

(b) Environment with static obstacles.

is used for dynamic obstacle detection using block matching and MobileNetSSD for classification [23]. All high-level computations, such as network inference and pedestrian detection are performed on an Intel NUC, with an Intel Core i7-8559U CPU with 4 cores and 2 threads per core, which is mounted on the robot directly. The motion commands are updated every 0.2 s. No additional fine-tuning of the SOADRL-AM network was required for the real-world application.

The associated video demo shows the behaviour of our robot in different setups. These tests show that the robot is able to navigate through environments with both static and dynamic obstacles without collision. Even when navigating through very crowded environments and with pedestrians trying to deliberately block the robot's path, the robot is able to avoid collisions and find a path to its goal.

V. CONCLUSIONS

In this work we have developed SOADRL: a novel DRL architecture for robot navigation in crowded environments that contain pedestrians and static obstacles. We built on top of the socially aware pedestrian avoidance network from [14] by adding a static obstacle representation, for which the usage of an angular map is compared to the usage of an occupancy grid. Additionally, we trained a policy network for action generation, which resulted in a runtime efficiency that outperforms the original network by almost two orders of magnitude. In environments with dynamic and static obstacles, our model often achieves higher success rates than other state-of-the-art DRL navigation methods while minimizing the number of collisions. We have shown that even for robots with very limited FOV, our model is able to navigate safely in crowded environments and finds a path to a goal location. Finally, we have deployed our algorithm on a robotic platform and were able to successfully navigate through real-world environments among pedestrians.

ACKNOWLEDGMENT

This work was partially supported by the EU H2020 project CROWDBOT under grant nr. 779942

REFERENCES

[1] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, no. 5, p. 4282–4286, May 1995. [Online]. Available: http://dx.doi.org/10.1103/PhysRevE.51.4282

[2] 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 International Conference on Computer Vision*, Sep. 2009, pp. 261–268.

[3] A. Vemula, K. Muelling, and J. Oh, "Modeling cooperative navigation in dense human crowds," 2017.

[4] 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, March 1997.

[5] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, Sep. 1989.

[6] F. A. Cosío and M. P. Castañeda, "Autonomous robot navigation using adaptive potential fields," *Mathematical and Computer Modelling*, vol. 40, no. 9, pp. 1141 – 1156, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0895717704003097

[7] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–19.

[8] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," 05 2008, pp. 1928–1935.

[9] T. Fan, X. Cheng, J. Pan, D. Manocha, and R. Yang, "Crowdmove: Autonomous mapless navigation in crowded scenarios," 2018.

[10] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for map-less navigation by leveraging prior demonstrations," 2018.

[11] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," 2018.

[12] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," 2017.

[13] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," 2018.

[14] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," 2018.

[15] M. Pfeiffer, G. Paolo, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments," 2017.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.

[17] S. Haddadin, A. Albu-Schäffer, and G. Hirzinger, "Requirements for safe robots: Measurements, analysis and new insights," *The International Journal of Robotics Research*, vol. 28, no. 11-12, pp. 1507–1527, 2009. [Online]. Available: https://doi.org/10.1177/0278364909343970

[18] Z. Chen, C. Song, Y. Yang, B. Zhao, Y. Hu, S. Liu, and J. Zhang, "Robot navigation based on human trajectory prediction and multiple travel modes," *Applied Sciences*, vol. 8, p. 2205, 11 2018.

[19] B. Luders, G. Aoude, J. Joseph, N. Roy, and J. How, "Probabilistically safe avoidance of dynamic obstacles with uncertain motion patterns," 07 2011.

[20] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," 2016.

[21] A. Pokle, R. Martín-Martín, P. Goebel, V. Chow, H. M. Ewald, J. Yang, Z. Wang, A. Sadeghian, D. Sadigh, S. Savarese, and M. Vázquez, "Deep local trajectory replanning and control for robot navigation," 2019.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[23] T. Eppenberger, G. Cesari, M. T. Dymczyk, R. Siegwart, and R. Dubé, "Leveraging stereo-camera data for real-time dynamic obstacle detection and tracking," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.