Decentralized Non-communicating Multiagent Collision Avoidance with Deep Reinforcement Learning

Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P. How

***Abstract*— Finding feasible, collision-free paths for multiagent systems can be challenging, particularly in non-communicating scenarios where each agent's intent (e.g. goal) is unobservable to the others. In particular, finding time efficient paths often requires anticipating interaction with neighboring agents, the process of which can be computationally prohibitive. This work presents a decentralized multiagent collision avoidance algorithm based on a novel application of deep reinforcement learning, which effectively offloads the online computation (for predicting interaction patterns) to an offline learning procedure. Specifically, the proposed approach develops a value network that encodes the estimated time to the goal given an agent's joint configuration (positions and velocities) with its neighbors. Use of the value network not only admits efficient (i.e., real-time implementable) queries for finding a collision-free velocity vector, but also considers the uncertainty in the other agents' motion. Simulation results show more than 26% improvement in paths quality (i.e., time to reach the goal) when compared with optimal reciprocal collision avoidance (ORCA), a state-of-the-art collision avoidance strategy.**

I. Introduction

Collision avoidance is central to many robotics applications, such as multiagent coordination [1], autonomous navigation through human crowds [2], pedestrian motion prediction [3], and computer crowd simulation [4]. Yet, finding collision-free, time efficient paths around other agents remains challenging, because it may require predicting other agents' motion and anticipating interaction patterns, through a process that needs to be computationally tractable for real-time implementation.

If there is a reliable communication network for agents to broadcast their intents (e.g. goals, planned paths), then collision avoidance can be enforced through a centralized planner. For instance, collision avoidance requirements can be formulated as separation constraints in an optimization framework for finding a set of jointly feasible and collision-free paths [1], [5], [6]. However, centralized path planning methods can be computationally prohibitive for large teams [1]. To attain better scalability, researchers have also proposed distributed algorithms based on message-passing schemes [7], [8], which resolve local (e.g. pairwise) conflicts without needing to form a joint optimization problem between all members of the team.

This work focuses on scenarios where communication cannot be reliably established, which naturally arises when considering human-robot interactions and can also be caused by hardware constraints or failures. This limitation poses additional challenges for collision avoidance, because mobile agents would need to cooperate without necessarily having knowledge of the other agent's intents. Existing work on non-communicating collision avoidance can be broadly classified into two categories, reaction-based and trajectory-based. Reaction-based methods [9]–[11] specify one-step interaction rules for the current geometric configuration. For example, reciprocal velocity obstacle (RVO) [12] is a reaction-based method that adjusts each agent's velocity vector to ensure collision-free navigation. However, since reaction-based methods do not consider evolution of the neighboring agents' future states, they are short-sighted in time and have been found to create oscillatory and unnatural behaviors in certain situations [10], [13].

Fig. 1: Autonomous ground vehicles navigating alongside pedestrians. Collision avoidance is essential for coordinating multiagent systems and modeling interactions between mobile agents.

In contrast, trajectory-based methods explicitly account for evolution of the joint (agent and neighbors) future states by anticipating other agents' motion. A subclass of non-cooperative approaches [14], [15] propagates the other agents' dynamics forward in time, and then plans a collision-free path with respect to the other agents' predicted paths. However, in crowded environments, the set of predicted paths often marks a large portion of the space untraversable/unsafe, which leads to the *freezing robot problem* [13]. A key to resolving this issue is to account for interactions, such that each agent's motion can affect one another. Thereby, a subclass of cooperative approaches [16]–[18] has been proposed, which first infers the other agents' intents (e.g. goals), then plans a set of jointly feasible paths for all neighboring agents in the environment. Cooperative trajectory-based methods often produce paths with better quality (e.g. shorter time for all agents to reach their goal) than that of reaction-based methods [16]. However, planning paths for all other agents is computationally expensive, and such cooperative approach typically requires more information than is readily available (e.g. other agent's intended goal). Moreover, due to model

Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA, USA
{chenyuf2, miaoliu, mfe, jhow}@mit.edu

and measurement uncertainty, the other agents' actual paths might not conform to the planned/predicted paths, particularly beyond a few seconds into the future. Thus, trajectory-based methods also need to be run at a high (sensor update) rate, which exacerbates the computational problem.

The major difficulty in multiagent collision avoidance is that anticipating evolution of joint states (paths) is desirable but computationally prohibitive. This work seeks to address this issue through reinforcement learning – to offload the expensive online computation to an offline training procedure. Specifically, this work develops a computationally efficient (i.e., real-time implementable) interaction rule by learning a value function that implicitly encodes cooperative behaviors.

The main contributions of this work are (i) a two-agent collision avoidance algorithm based on a novel application of deep reinforcement learning, (ii) a principled way for generalizing to more ($n > 2$) agents, (iii) an extended formulation to capture kinematic constraints, and (iv) simulation results that show significant improvement in solution quality compared with existing reaction-based methods.

II. Problem Formulation

A. Sequential Decision Making

A non-communicating multiagent collision avoidance problem can be formulated as a partially-observable sequential decision making problem. Let $\mathbf{s}_t$, $\mathbf{a}_t$ denote an agent's state and action at time t. The agent's state vector can be divided into two parts, that is $\mathbf{s}_t = [\mathbf{s}_t^o, \mathbf{s}_t^h]$, where $\mathbf{s}_t^o$ denotes the observable part that can be measured by all other agents, and $\mathbf{s}_t^h$ denotes the hidden part that is only known to the agent itself. In this work, let position and velocity vectors in 2D be denoted by $\mathbf{p}$ and $\mathbf{v}$, respectively; let action be the agent's velocity, $\mathbf{a} = \mathbf{v}$; let the observable states be the agent's position, velocity, and radius (size), $\mathbf{s}^o = [p_x, p_y, v_x, v_y, r] \in \mathbb{R}^5$; and let the hidden states be the agent's intended goal position, preferred speed, and heading angle, $\mathbf{s}^h = [p_{gx}, p_{gy}, v_{pref}, \theta] \in \mathbb{R}^4$.

The following presents a two-agent[1] collision avoidance problem formulation, where an agent's own state and the other agent's state are denoted by $\mathbf{s}$ and $\tilde{\mathbf{s}}$, respectively. The objective is to minimize the expected time, $\mathbb{E}[t_g]$, of an agent to reach its goal by developing a policy, $\pi : (\mathbf{s}_{0:t}, \tilde{\mathbf{s}}^o_{0:t}) \mapsto \mathbf{a}_t$, that selects an action given the observed state trajectories,

$$\underset{\pi(\mathbf{s}, \tilde{\mathbf{s}}^o)}{\operatorname{argmin}} \quad \mathbb{E}\left[t_g | \mathbf{s}_0, \tilde{\mathbf{s}}^o_0, \pi, \tilde{\pi}\right] \tag{1}$$

$$s.t. \quad ||\mathbf{p}_t - \tilde{\mathbf{p}}_t||_2 \geq r + \tilde{r} \qquad \forall t \tag{2}$$

$$\mathbf{p}_{t_g} = \mathbf{p}_g \tag{3}$$

$$\mathbf{p}_t = \mathbf{p}_{t-1} + \Delta t \cdot \pi(\mathbf{s}_{0:t}, \tilde{\mathbf{s}}^o_{0:t})$$

$$\tilde{\mathbf{p}}_t = \tilde{\mathbf{p}}_{t-1} + \Delta t \cdot \tilde{\pi}(\tilde{\mathbf{s}}_{0:t}, \mathbf{s}^o_{0:t}), \tag{4}$$

where (2) is the collision avoidance constraint, (3) is the goal constraint, (4) is the agents' kinematics, and the expectation in (1) is with respect to the other agent's policy and hidden states (intents). Note that static obstacles can be modeled as stationary agents, which will be discussed in more details in Section IV-D.

Although it is difficult to solve for the optimal solution of (1)-(4), this problem formulation can be useful for understanding the limitations of the existing methods. In particular, it provides insights into the approximations/assumptions made by existing works. A common assumption is reciprocity, that is $\pi = \tilde{\pi}$, such that each agent would follow the same policy [11], [16]. Thereby, the main difficulty is in handling the uncertainty in the other agent's hidden intents (e.g. goals).

Reaction-based methods [10], [11] often specify a Markovian policy, $\pi(\mathbf{s}_{0:t}, \tilde{\mathbf{s}}^o_{0:t}) = \pi(\mathbf{s}_t, \tilde{\mathbf{s}}^o_t)$, that optimizes a one-step cost while satisfying collision avoidance constraints. For instance, in velocity obstacle approaches [11], an agent chooses a collision-free velocity that is closest to its preferred velocity (i.e., directed toward its goal). Given this one-step nature, reaction-based methods do not anticipate the other agent's hidden intent, but rather rely on a fast update rate to react quickly to the other agent's motion. Although computationally efficient given these simplifications, reaction-based methods are myopic in time, which can sometimes lead to generating unnatural trajectories [17] (e.g., Fig. 4a).

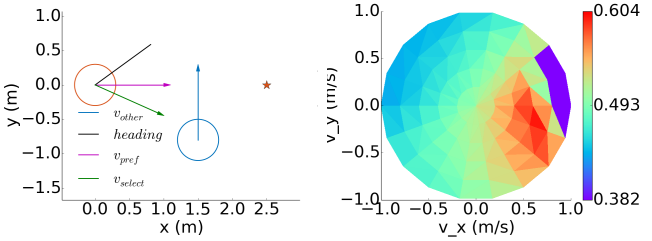
Trajectory-based methods [16]–[18] solve (1)-(4) in two steps. First, the other agent's hidden state is inferred from its observed trajectory, $\hat{\tilde{\mathbf{s}}}^h_t = f(\tilde{\mathbf{s}}^o_{0:t})$, where $f(\cdot)$ is a inference function. Second, a centralized path planning algorithm, $\pi(\mathbf{s}_{0:t}, \tilde{\mathbf{s}}^o_{0:t}) = \pi_{central}(\mathbf{s}_t, \tilde{\mathbf{s}}^o_t, \hat{\tilde{\mathbf{s}}}^h_t)$, is employed to find jointly feasible paths. By planning/anticipating complete paths, trajectory-based methods are no longer myopic. However, both the inference and the planning steps are computationally expensive, and need to be carried out online at each new observation (sensor update $\tilde{\mathbf{s}}^o_t$).

Our approach uses a reinforcement learning framework to solve (1)-(4) by pre-computing a value function $V(\mathbf{s}, \tilde{\mathbf{s}}^o)$ that estimates the expected time to the goal. As a result, the proposed method offloads computation from the online planning step (as in trajectory-based methods) to an offline learning procedure. The learned value function enables the use of a computationally efficient one-step lookahead operation, which will be defined in (11) and explained in Section III. Repeating this one-step lookahead operation at each sensor update leads to generating better paths, as shown later in Fig. 4d.

B. Reinforcement Learning

Reinforcement learning (RL) [19] is a class of machine learning methods for solving sequential decision making problems with unknown state-transition dynamics. Typically, a sequential decision making problem can be formulated as a Markov decision process (MDP), which is defined by a tuple $M = \langle S, A, P, R, \gamma \rangle$, where S is the state space, A is the action space, P is the state-transition model, R is the reward function, and γ is a discount factor. By detailing each of these elements and relating to (1)-(4), the following provides a RL formulation of the two-agent collision avoidance problem.

[1] This formulation can be generalized to multiagent ($n > 2$) scenarios by replacing the other agent's state $\tilde{\mathbf{s}}^o_t$ with all other agents' states $\tilde{\mathbf{S}}^o_t = [\mathbf{s}^o_{1,t}, \ldots, \mathbf{s}^o_{n,t}]$, and expand (2) to include all pairwise collision constraints.



(a) Input joint state (b) Value function

Fig. 2: RL policy. (a) shows a joint state of the system (geometric configuration) in the red agent's reference frame, with its goal aligned with the x-axis and marked as a star. (b) shows the red agent's value function at taking each possible action (velocity vector). Given the presence of the blue agent, ORCA [11] would choose an action close to the current heading angle (black vector), whereas the RL policy chooses to cut behind (green vector) the blue agent, leading to generating better paths similar to that in Fig. 4.

State space: The system's state is constructed by concatenating the two agents' individual states, $\mathbf{s}^{jn} = [\mathbf{s},\ \tilde{\mathbf{s}}^o] \in \mathbb{R}^{14}$.

Action space: The action space is the set of permissible velocity vectors. Here, it is assumed that an agent can travel in any direction at any time, that is $\mathbf{a}(\mathbf{s}) = \mathbf{v}$ for $||\mathbf{v}||_2 < v_{pref}$. It is also straightforward to impose kinematic constraints, which will be explored in Section III-D.

Reward function: A reward function is specified to award the agent for reaching its goal (3), and penalize the agent for getting too close or colliding with the other agent (2),

$$R(\mathbf{s}^{jn}, \mathbf{a}) = \begin{cases} -0.25 & \text{if} \quad d_{min} < 0 \\ -0.1 - d_{min}/2 & \text{else if} \quad d_{min} < 0.2 \\ 1 & \text{else if} \quad \mathbf{p} = \mathbf{p}_g \\ 0 & \text{o.w.} \end{cases}, \quad (5)$$

where d_{min} is the minimum separation distance between the two agents within a duration of Δt, assuming the agent travels at velocity $\mathbf{v} = \mathbf{a}$, and the other agent continues to travel at its observed velocity $\tilde{\mathbf{v}}$. Note that the separation d_{min} can be calculated analytically through simple geometry.

State transition model: A probabilistic state transition model, $P(\mathbf{s}^{jn}_{t+1}, \mathbf{s}^{jn}_t | \mathbf{a}_t)$, is determined by the agents' kinematics as defined in (4). Since the other agent's choice of action also depends on its policy and hidden intents (e.g. goal), the system's state transition model is unknown. As in existing work [11], this work also assumes reciprocity $\pi = \tilde{\pi}$, which leads to the interesting observation that the state transition model depends on the agent's learned policy.

Value function: The objective is to find the optimal value function

$$V^*(\mathbf{s}^{jn}_0) = \sum_{t=0}^{T} \gamma^{t \cdot v_{pref}}\, R(\mathbf{s}^{jn}_t, \pi^*(\mathbf{s}^{jn}_t)), \quad (6)$$

where $\gamma \in [0, 1)$ is a discount factor. Recall v_{pref} is an agent's preferred speed and is typically time invariant. It is used here as a normalization factor for numerical reasons, because otherwise the value function of a slow moving agent could be very small. The optimal policy can be retrieved from the value function, that is

$$\pi^*(\mathbf{s}^{jn}_0) = \underset{\mathbf{a}}{\operatorname{argmax}}\, R(\mathbf{s}_0, \mathbf{a}) + \gamma^{\Delta t \cdot v_{pref}} \int_{\mathbf{s}^{jn}_1} P(\mathbf{s}^{jn}_0, \mathbf{s}^{jn}_1 | \mathbf{a}) V^*(\mathbf{s}^{jn}_1) d\mathbf{s}^{jn}_1. \quad (7)$$

This work chooses to optimize $V(\mathbf{s}^{jn})$ rather than the more common choice $Q(\mathbf{s}^{jn}, \mathbf{a})$, because unlike previous works that focus on discrete, finite action spaces [20], [21], the action space here is continuous and the set of permissible velocity vectors depends on the agent's state (preferred speed).

III. Approach

The following presents an algorithm for solving the two-agent RL problem formulated in Section II-B, and then generalizes its solution (policy) to multiagent collision avoidance. While applications of RL are typically limited to discrete, low-dimensional domains, recent advances in Deep RL [20]–[22] have demonstrated human-level performance in complex, high-dimensional spaces. Since the joined state vector $\mathbf{s}^{jn}$ is in a continuous 14 dimensional space, and because a large amount of training data can be easily generated in a simulator, this work employs a fully connected deep neural network with ReLU nonlinearities to parametrize the value function, as shown in Fig. 3a. This value network is denoted by $V(\cdot; \mathbf{w})$, where $\mathbf{w}$ is the set of weights in the neural network.

A. Parametrization

From a geometric perspective, there is some redundancy in the parameterization of the system's joint state $\mathbf{s}^{jn}$, because the optimal policy should be invariant to any coordinate transformation (rotation and translation). To remove this ambiguity, an agent-centric frame is defined, with the origin at the agent's position, and the x-axis pointing toward the agent's goal, that is,

$$\begin{aligned} \mathbf{s}' &= \texttt{rotate}\left(\mathbf{s}^{jn}\right) \\ &= [d_g,\ v_{pref},\ v'_x,\ v'_y,\ r,\ \theta',\ \tilde{v}'_x,\ \tilde{v}'_y,\ \tilde{p}'_x,\ \tilde{p}'_y,\ \tilde{r}, \\ &\quad r + \tilde{r},\ \cos(\theta'),\ \sin(\theta'),\ d_a], \end{aligned} \quad (8)$$

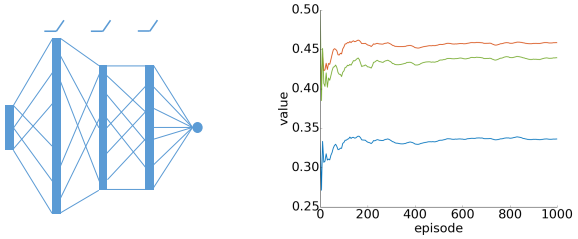
where $d_g = ||\mathbf{p}_g - \mathbf{p}||_2$ is the agent's distance to goal, and $d_a = ||\mathbf{p} - \tilde{\mathbf{p}}||_2$ is the distance to the other agent. An illustration of this parametrization is shown in Fig. 2a. Note that this agent-centric parametrization is only used when querying the neural network.

B. Generating Paths Using a Value Network

Given a value network V, an RL agent can generate a path to its goal by repeatedly maximizing an one-step lookahead value (7), as outlined in Algorithm 1. This corresponds to choosing the action that on average, leads to the joint state with the highest value. However, the integral in (7) is difficult to evaluate, because the other agent's next state $\tilde{\mathbf{s}}^o_{t+1}$ has an unknown distribution (depends on its unobservable intent). We approximate this integral by assuming that the other agent would be traveling at a filtered velocity for a short duration

Algorithm 1: CADRL (Coll. Avoidance with Deep RL)

1 **Input:** value network $\mathbf{V}(\cdot;\mathbf{w})$
2 **Output:** trajectory $\mathbf{s}_{0:t_f}$
3 **while** *not reached goal* **do**
4 update t, receive new measurements $\mathbf{s}_t$, $\tilde{\mathbf{s}}_t^o$
5 $\hat{\tilde{\mathbf{v}}}_t \leftarrow$ filter($\tilde{\mathbf{v}}_{0:t}$)
6 $\hat{\tilde{s}}_{t+1}^o \leftarrow$ propagate($\tilde{\mathbf{s}}_t^o, \Delta t \cdot \hat{\tilde{\mathbf{v}}}_t$)
7 $A \leftarrow$ sampleActions()
8 $\mathbf{a}_t \leftarrow \text{argmax}_{\mathbf{a}_t \in A} R(\mathbf{s}_t^{jn}, \mathbf{a}_t) + \bar{\gamma} V(\hat{\mathbf{s}}_{t+1}, \hat{\tilde{\mathbf{s}}}_{t+1}^o)$
where $\bar{\gamma} \leftarrow \gamma^{\Delta t \cdot v_{pref}}$, $\hat{\mathbf{s}}_{t+1} \leftarrow$ propagate($\mathbf{s}_t, \Delta t \cdot \mathbf{a}_t$)
9 **return** $\mathbf{s}_{0:t_f}$



(a) Value network (b) Convergence

Fig. 3: Convergence of a Deep RL policy. (a) shows a neural network used to parameterize the value function. (b) shows the values of three distinct test cases converge as a function of RL training episodes. For example, the blue line corresponds to the test case shown in the bottom row of Fig. 4.

Δt (line 5-6)[2]. The use of a filtered velocity addresses a subtle oscillation problem as discussed in [12]. This propagation step amounts to predicting the other agent's motion with a simple linear model, which has been shown to produce good accuracy over small time scales [23]. It is important to point out that this approximation is *not* assuming a linear motion model for $t > \Delta t$; uncertainty in the other agent's future motion is captured in the projected next state's value, $V(\hat{\mathbf{s}}_{t+1}, \hat{\tilde{\mathbf{s}}}_{t+1}^o)$. Furthermore, the best action is chosen from a set of permissible[3] velocity vectors (line 8). An example of this one-step lookahead operation is visualized in Fig. 2a, in which the red agent chooses the green velocity vector to cut behind the blue agent, because this action maximizes the value of the projected state shown in Fig. 2b.

C. Training a Value Network

The training procedure, outlined in Algorithm 2, consists of two major steps. First, the value network is initialized by supervised training on a set of trajectories generated by a baseline policy (line 3).

Specifically, each training trajectory is processed to generate a set of state-value pairs, $\{(\mathbf{s}^{jn}, y)_k\}_{k=1}^N$, where $y = \gamma^{t_g \cdot v_{pref}}$ and t_g is the time to reach goal. The value network is trained by back-propagation to minimize a quadratic regression error, $\text{argmin}_{\mathbf{w}} \sum_{k=1}^N \left(y_k - V(\mathbf{s}_k^{jn}; \mathbf{w})\right)^2$. This work uses optimal reciprocal collision avoidance (ORCA) [11] to generate a training set of 500 trajectories, which contains approximately 20,000 state-value pairs.

[2]This work calculates the average velocity of the past 0.5 seconds and sets Δt to 1.0 second.

[3]This work uses 25 pre-computed actions (e.g. directed toward an agent's goal or current heading) and 10 randomly sampled actions.

Algorithm 2: Deep V-learning

1 **Input:** trajectory training set D
2 **Output:** value network $V(\cdot;\mathbf{w})$
3 $V(\cdot;\mathbf{w}) \leftarrow$ train_nn(D) //step 1: initialization
4 duplicate value net $V' \leftarrow V$ //step 2: RL
5 initialize experience set $E \leftarrow D$
6 **for** *episode*=$1, \ldots, N_{eps}$ **do**
7 **for** *m times* **do**
8 $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
9 $\mathbf{s}_{0:t_f} \leftarrow$ CADRL(V), $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL(V)
10 $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues(V', $\mathbf{s}_{0:t_f}$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f}$)
11 $E \leftarrow$ assimilate$\left(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\right)$
12 $e \leftarrow$ randSubset(E)
13 $\mathbf{w} \leftarrow$ backprop(e)
14 **for** *every C episodes* **do**
15 Evaluate(V), $V' \leftarrow V$
16 **return** V

We make a few remarks about this initialization step. First, the training trajectories do not have to be optimal. For instance, two of the training trajectories generated by ORCA [11] are shown in Fig. 4a. The red agent was pushed away by the blue agent and followed a large arc before reaching its goal. Second, the initialization training step is not simply emulating the ORCA policy. Rather, it learns a time to goal estimate (value function), which can then be used to generate new trajectories following Algorithm 1. Evidently, this learned value function sometimes generates better (i.e. shorter time to goal) trajectories than ORCA, as shown in Fig. 4b. Third, this learned value function is likely to be suboptimal. For instance, the minimum separation d_{min} between the two agents should be around 0.2m by (5), but d_{min} is greater than 0.4m (too much slack) in Fig. 4b.

The second training step refines the policy through reinforcement learning. Specifically, a few random test cases are generated in each episode (line 8), and two agents are simulated to navigate around each other using an ϵ-greedy policy, which selects a random action with probability ϵ and follows the value network greedily otherwise (line 9). The simulated trajectories are then processed to generate a set of state-value pairs (line 10). For convergence reasons, as explained in [20], rather than being used to update the value network immediately, the newly generated state-value pairs are assimilated (replacing older entries) into a large experience set E (line 11). Then, a set of training points is randomly sampled from the experience set, which contains state-value pairs from many different simulated trajectories (line 12). The value network is finally updated by stochastic gradient descent (back-propagation) on the sampled subset.

To monitor convergence of the training process, the value network is tested regularly on a few pre-defined evaluation test cases (line 15), two of which are shown in Fig. 4. Note that the generated paths become tighter as a function of the number of training episodes (i.e. d_{min} reduces from 0.4m to 0.2m). A plot of the test cases' values $V(\mathbf{s}_0^{jn})$ shows that the value network has converged in approximately 800 episodes (Fig. 3b). It is important to point out that training/learning is performed on *randomly* generated test cases (line 8), but not on the evaluation test cases.

In addition to the standard Q-learning update [19], an important modification is introduced when calculating the state-value pairs (line 10). In particular, cooperation is encouraged by adding a penalty term based on a comparison of the two agents' extra time to reach the goal, $t_e = t_g - d_g/v_{pref}$. If $t_e < e_l$ and $\tilde{t}_e > e_u$[4], which corresponds to a scenario where the agent reached its goal quickly but the other agent took a long time, an penalty of 0.1 would be subtracted from the training value. Albeit simple, this modification is crucial for discouraging aggressive behaviors such as exhibited by the blue agent in Fig. 4a. Without this modification, an agent would frequently travel straight toward its goal, expecting the other agent to yield.

D. Incorporating Kinematic Constraints

Kinematics constraints need to be considered for operating physical robots. Yet, in many existing works, such constraints can be difficult to encode and might lead to a substantial increase in computational complexity [5], [12]. In contrast, it is straightforward to incorporate kinematic constraints in the RL framework. We impose rotational constraints,

$$\mathbf{a}(\mathbf{s}) = [v_s,\ \phi] \quad \text{for} \quad v_s < v_{pref},\ |\phi - \theta| < \pi/6 \tag{9}$$

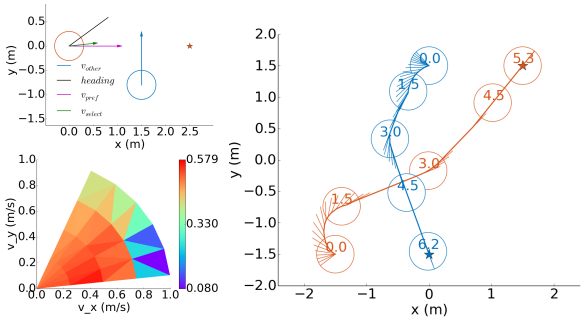
$$|\theta_{t+1} - \theta_t| < \Delta t \cdot v_{pref}, \tag{10}$$

where (9) limits the direction that an agent can travel, and (10) specifies a maximum turning rate that corresponds to a minimum turning radius of 1.0m. Figure 5a illustrates the application of these rotational constraints to the same test case in Fig. 2a. Here, the red agent chooses to slow down given the set of more constrained actions. Notice the agent is allowed to spin on the spot, which leads to an interesting optimal control problem when an agent's current heading angle is not perfectly aligned with its goal. In this case, an agent can either travel at its full speed while turning toward its goal, or first spin on the spot before traveling in a straight line. Figure 5b shows that CADRL has learned a policy that balances between these two options to minimize the time to goal. With the thin lines showing its heading angle, the red agent chooses to initially turn on the spot, and then start moving before its heading angle is perfectly aligned with its goal.

E. Multiagent Collision Avoidance

The two-agent value network can also be used for multiagent collision avoidance. Let $\tilde{\mathbf{s}}_i^o$ denote the observable part

[4]This work uses e_l = 1.0 and e_u = 2.0.



(a) Constrained action (b) Sample trajectories

Fig. 5: Rotational kinematic constraint. Top left shows the same test case as in Fig. 2a, but here the red agent chooses to slow down due to a rotational kinematic constraint (9). Bottom left shows the set of permissible velocity vectors for the red agent. Right shows a pair of sample trajectories generated by CADRL with rotational constraints, where the thin lines show the agents' heading angles. To minimize the time to goal, the red agent initially turns on the spot, and then starts moving (while continuing to turn) before its heading angle is aligned with its goal.

of the ith neighbor's state, and $\mathbf{s}_i^{jn} = [\mathbf{s},\ \tilde{\mathbf{s}}_i^o]$ denote the joint state with the ith neighbor. CADRL (Algorithm 1) can be extended to $n > 2$ agents by propagating every neighbor's state one step forward (line 5-6), and then selecting the action that has the highest value with respect to any neighbor's projected state; that is, replace line 8 with

$$\underset{\mathbf{a}_t \in A}{\text{argmax}} \ \min_i \ R(\mathbf{s}_{i,t}^{jn},\ \mathbf{a}_t) + \gamma^{\Delta t \cdot v_{pref}} V(\hat{\mathbf{s}}_{t+1},\ \hat{\tilde{\mathbf{s}}}_{i,t+1}^o). \tag{11}$$

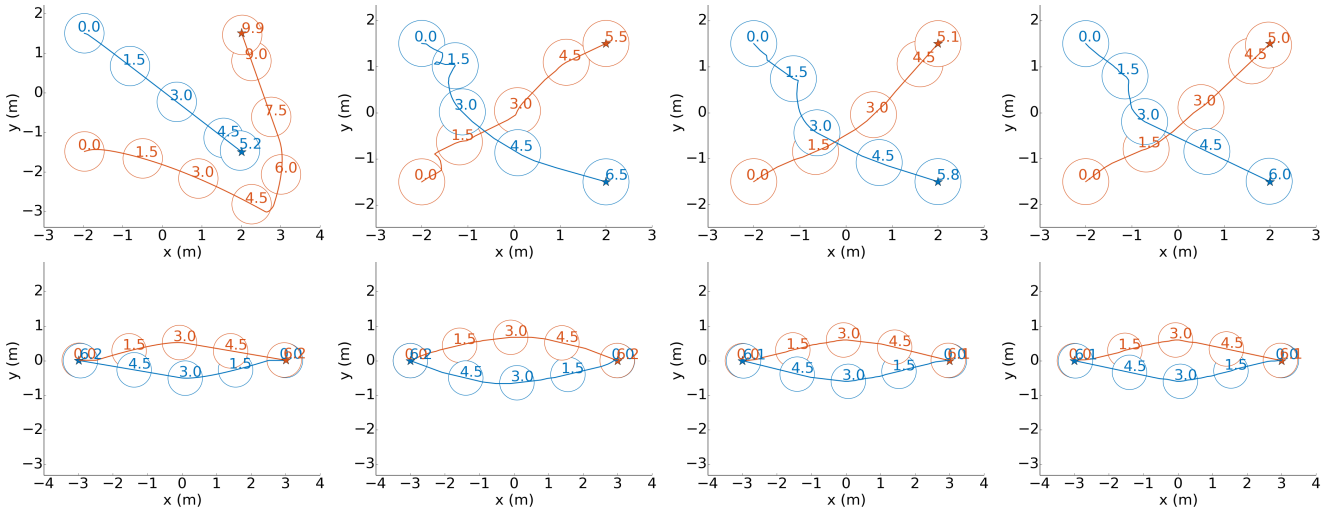
Note that the agent's projected next state $\hat{\mathbf{s}}_{t+1}$ also depends on the selected action $\mathbf{a}_t$. Although using a two-agent value network, CADRL can produce multiagent trajectories that exhibit complex interaction patterns. Figure 6a shows six agents each moving to the opposite side of a circle. The agents veer more than the two-agent case (bottom row of Fig. 4), which makes more room in the center to allow every agent to pass smoothly. Figure 6b shows three pairs of agents swapping position horizontally. The pair in the center slows down near the origin so the outer agents can pass first. Both cases demonstrate that CADRL can produce smooth, natural looking trajectories for multiagent systems, which will be explored in further detail in Section IV. However, we acknowledge that (11) is only an approximation to a true multiagent RL value function – an n-agent value network by simulating n agents navigating around each other – which will be studied for future work.

IV. Results

A. Computational Complexity

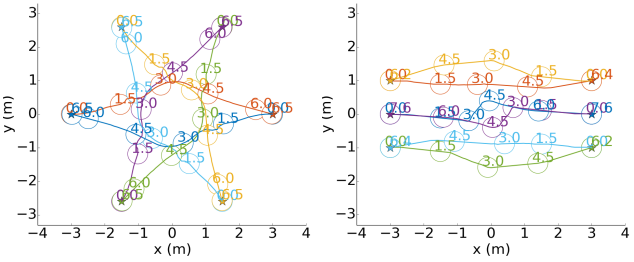
This work uses a neural network with three hidden layers of width (150, 100, 100), which is the size chosen to achieve real-time performance.[5] In particular, on a computer with an i7-5820K CPU, a Python implementation of CADRL

[5]We also experimented with other network structures. For example, a network with three hidden layers of width (300, 200, 200) produced similar results (paths) but was twice as slow.



(a) ORCA (b) Episode 0 (c) Episode 50 (d) Episode 1000

Fig. 4: Training the value network. Circles show each agent's position at the labeled time, and stars mark the goals. (a) illustrates trajectories generated by the two agents each following ORCA [11], and (b-d) illustrate trajectories generated by following the value network at various training episodes. Top (a) shows a test case which ORCA results in unnatural trajectories, where the red agent has traversed a large arc before reaching its goal. Top (b-d) show CADRL has learned to produce cooperative behaviors, as the blue agent slows down and cuts behind the red agent. Bottom (b-d) show the trajectories become more tight (better performing) during the training process, since the minimum separation d_{min} reduces from 0.4m to 0.2m, as specified in (5).



(a) (b)

Fig. 6: Multiagent trajectories produced by CADRL. Circles show each agent's position at the labeled time, and stars mark the goals. Although CADRL uses a two-agent value network for multiagent scenarios (11), more complex interaction patterns have emerged. In particular, although both test cases involve three pairs of agents swapping position, each agent follows a path much different from the two agent case shown in the bottom row of Fig. 4.

(Algorithm 1), on average, takes 5.7ms per iteration on two-agent collision avoidance problems. By inspection of (11), computational time scales linearly in the number of neighboring agents for a decentralized implementation where each agent runs CADRL individually; and scales quadratically in a centralized implementation where one computer controls all agents. For decentralized control on ten agents, each iteration of CADRL is observed to take 62ms. Moreover, CADRL is parallelizable because it consists of a large number of independent queries of the value network (11).

Furthermore, offline training (Algorithm 2) took less than three hours and is found to be quite robust. In particular, using mini-batches of size 500, the initialization step (line 3) took 9.6 minutes to complete 10,000 iterations of back-propagation. The RL step used an ϵ-greedy policy, where ϵ decays linearly from 0.5 to 0.1 in the first 400 training episodes, and remains 0.1 thereafter. The RL step took approximately 2.5 hours to complete 1,000 training episodes. The entire training process was repeated on three sets of training trajectories generated by ORCA, and the value network converged in all three trials. The paths generated by the value networks from the three trials were very similar, as indicated by a less than 5% difference in time to reach goal on all test cases in the evaluation set.

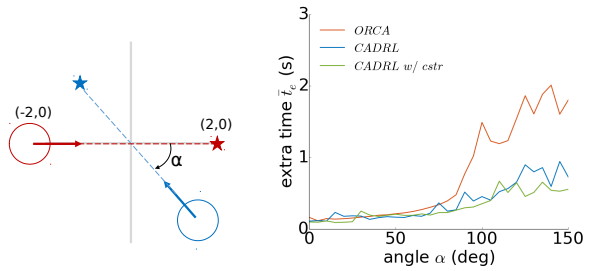
B. *Performance Comparison on a Crossing Scenario*

To evaluate the performance of CADRL over a variety of test cases, we compute the average extra time spent to reach goals, that is

$$\bar{t}_e = \frac{1}{n}\sum_{i=1}^{n}\left[t_{i,g} - \frac{||\mathbf{p}_{i,0} - \mathbf{p}_{i,g}||_2}{v_{i,pref}}\right], \qquad (12)$$

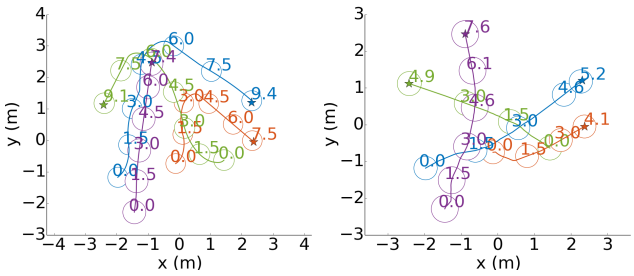
where $t_{i,g}$ is the ith agent's time to reach its goal, and the second term is a lower bound of $t_{i,g}$ (to go straight toward goal at the preferred speed). This metric removes the effects due to variability in the number of agents and the nominal distance to goals.

A crossing scenario is shown in Fig. 7a, where two identical agents with goals along collision courses are run into each other at different angles. Thus, cooperation is required for avoiding collision at the origin. Figure 7a shows that over a wide range of angles ($\alpha \in [90, 150]$ deg), agents following CADRL reached their goals much faster than that of ORCA. Recall a minimum separation of 0.2m is specified for CADRL in the reward function (5). For this reason, similar to the bottom row of Fig. 4, CADRL finds paths that are slightly slower than ORCA around $\alpha = 0$. It is also interesting to note that CADRL with rotational constraints (CADRL w/ cstr) performs slightly better than the unconstrained. This is



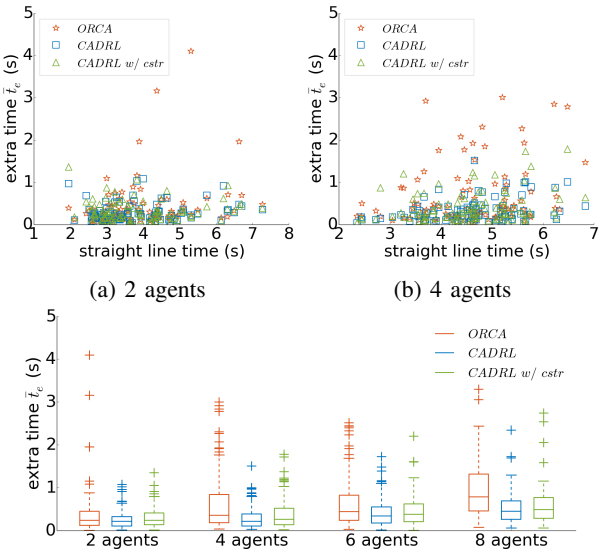
(a) Crossing configuration (b) Time to reach goal

Fig. 7: Performance comparison on a crossing scenario. (a) shows the crossing configuration, where a red agent travels from left to right, and a blue agent travels at a diagonal angled at α. Both agents have a radius of 0.3m and a preferred speed of 1.0m/s, and they would collide at the origin if both travel in a straight line. (b) compares the extra time spent to reach goal (12) using different collision avoidance strategies. CADRL performs significantly better than ORCA on a wide range of angles $\alpha \in [90, 150]$ deg.



(a) ORCA (b) CADRL

Fig. 8: Four-agent test case. (a) shows agents following ORCA traversed long arcs before reaching their goals, which reflects the similar two-agent problem shown in Fig. 4a. (b) shows agents following CADRL were able to reach their goal much faster.



(a) 2 agents (b) 4 agents

(c) Box-and-whisker plot

Fig. 9: Performance comparison on randomly generated test cases. The extra time to goal t_e is computed on one hundred random test cases for each configuration listed in Table I. (a) and (b) show scatter plots of the raw data, and (c) shows a box whisker plot. CADRL is seen to perform similarly (slightly better) to ORCA on the easier test cases (median), and significantly better on the more difficult test cases (>75 percentile).

because CADRL w/ cstr is more conservative (yielding) early on, which is coincidentally good for the crossing scenario. More specifically, if the other agent has stopped (reached goal) or turned before it reached the origin, unconstrained CADRL would have performed better. In short, as will be shown later in Fig. 9, unconstrained CADRL is better on average (randomized test cases), but can be slightly worse than CADRL w/ cstr on particular test cases.

C. Performance Comparison on Random Test Cases

In addition to showing that CADRL can handle some difficult test cases that fared poorly for ORCA (Figs. 4 and 7), a more thorough evaluation is performed on randomly generated test cases. In particular, within square shaped domains specified in Table I, agents are generated with randomly sampled speed, radius, initial positions and goal positions. This work chooses $v_{pref} \in [0.5, 1.5]$m/s, $r \in [0.3, 0.5]$m, which are parameters similar to that of typical pedestrian motion. Also, the agents' goals are projected to the boundary of the room to avoid accidentally creating a trap formed by multiple stationary agents. A sample four-agent test case is illustrated in Fig. 8, where agents following CADRL were able to reach their goal much faster than that of ORCA. For each configuration in Table I, one hundred test cases are generated as described above. ORCA, CADRL, and CADRL w/cstr are employed to solve for these test cases. The average extra time to reach goal, $\bar{t}_e$, is computed for each set of generated trajectories and plotted in Fig. 9. Key statistics are computed and listed in Table I, and it can be seen that CADRL performs similarly (slightly better) than ORCA on the easier test cases (median), and more than 26% better on the hard test cases (>75 percentile). Also, performance difference is more clear on test cases with more agents, which could be a result of more frequent interactions.

D. Navigating around Non-cooperative Agents

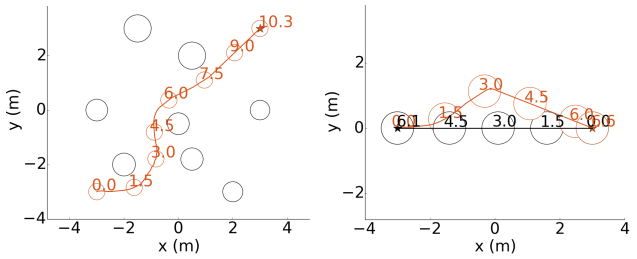
Recall CADRL's value network is trained with both agents following the same collision avoidance strategy, which is the reciprocity assumption common to many existing works [11], [12]. Figure 10 shows that the proposed method can also navigate efficiently around non-CADRL agents. Figure 10a shows a CADRL agent navigating static obstacles modeled as stationary agents ($\tilde{\mathbf{v}}_i = 0$). We acknowledge that CADRL could get stuck in a dense obstacle field, where traps/dead-ends could form due to positioning of multiple obstacles. Recall CADRL is a collision avoidance (not path planning) algorithm not designed for such scenarios. Figure 10b shows a CADRL agent navigating around a non-cooperative agent (black), who traveled in a straight line from right to left. In comparison with the cooperative case shown in Fig. 4d, here the red CADRL agent chooses to veer more to its left to avoid collision.

V. Conclusion

This work developed a decentralized multiagent collision avoidance algorithm based on a novel application of deep

TABLE I: Extra time to reach goal ($\bar{t}_e$) statistics on the random test cases shown in Fig. 9. CADRL finds paths that on average, reach goals much faster than that of ORCA. The improvement is more clear on hard test cases (>75 percentile) and in multiagent ($n > 2$) scenarios that require more interactions.

Test case configuration		Extra time to goal $\bar{t}_e$ (s) [Avg / 75th / 90th percentile]			Average min separation dist. (m)		
num agents	domain size (m)	ORCA	CADRL	CADRL w/ cstr	OCRA	CADRL	CADRL w/ cstr
2	4.0 × 4.0	0.46 / 0.45 / 0.73	0.27 / 0.33 / 0.56	0.31 / 0.42 / 0.60	0.122	0.199	0.198
4	5.0 × 5.0	0.69 / 0.85 / 1.85	0.31 / 0.40 / 0.76	0.39 / 0.53 / 0.86	0.120	0.192	0.191
6	6.0 × 6.0	0.65 / 0.83 / 1.50	0.44 / 0.56 / 0.87	0.48 / 0.63 / 1.02	0.118	0.117	0.180
8	7.0 × 7.0	0.96 / 1.33 / 1.84	0.54 / 0.70 / 1.01	0.59 / 0.77 / 1.09	0.110	0.171	0.170



(a) Static obstacles (b) Non-cooperative agent

Fig. 10: Navigating around non-CADRL agents. (a) shows a red agent navigating around a series of static obstacles. (b) shows a red CADRL agent avoids collision with a non-cooperative black agent, who traveled in a straight line from right to left.

reinforcement learning. In particular, a pair of agents were simulated to navigate around each other to learn a value network that encodes the expected time to goal. The solution (value network) to the two-agent collision avoidance RL problem was generalized in a principled way to handle multiagent ($n > 2$) scenarios. The proposed method was shown to be real-time implementable for a decentralized ten-agent system. Simulation results show more than 26% improvement in paths quality when compared with ORCA.

ACKNOWLEDGMENT

This work is supported by Ford Motor Company.

REFERENCES

[1] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 477–483.

[2] S. Choi, E. Kim, and S. Oh, "Real-time navigation in crowded dynamic environments using Gaussian process motion control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3221–3226.

[3] S. Kim, S. J. Guy, W. Liu, D. Wilkie, R. W. Lau, M. C. Lin, and D. Manocha, "BRVO: predicting pedestrian trajectories using velocity-space reasoning," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 201–217, Feb. 2015.

[4] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey, "ClearPath: Highly parallel collision avoidance for multi-agent simulation," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '09. New York, NY, USA: ACM, 2009, pp. 177–187.

[5] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 1917–1922.

[6] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," in *proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5954–5961.

[7] O. Purwin, R. D'Andrea, and J.-W. Lee, "Theory and implementation of path planning by negotiation for decentralized agents," *Robotics and Autonomous Systems*, vol. 56, no. 5, pp. 422–436, May 2008.

[8] V. R. Desaraju and J. P. How, "Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 4956–4961.

[9] J. Snape, J. Van den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, Aug. 2011.

[10] G. Ferrer, A. Garrell, and A. Sanfeliu, "Social-aware robot navigation in urban environments," in *2013 European Conference on Mobile Robots (ECMR)*, Sept. 2013, pp. 331–336.

[11] J. Van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, ser. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2011, no. 70, pp. 3–19.

[12] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation (ICRA)*, 2008, pp. 1928–1935.

[13] P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2010, pp. 797–803.

[14] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 5628–5635.

[15] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How, "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns," *Autonomous Robots*, vol. 35, no. 1, pp. 51–76, May 2013.

[16] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *The International Journal of Robotics Research*, Jan. 2016.

[17] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: the case for cooperation," in *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, May 2013, pp. 2153–2160.

[18] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard, "Feature-based prediction of trajectories for socially compliant navigation," in *Robotics:Science and Systems*, 2012.

[19] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[21] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[22] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel, "Learning deep neural network policies with continuous memory states," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 520–527.

[23] A. Bera and D. Manocha, "Realtime multilevel crowd tracking using reciprocal velocity obstacles," in *2014 22nd International Conference on Pattern Recognition (ICPR)*, Aug. 2014, pp. 4164–4169.