Where to go next

Learning a Subgoal Recommendation Policy for Navigation in Dynamic Environments

Brito, Bruno; Everett, Michael; How, Jonathan Patrick; Alonso-Mora, Javier

DOI
10.1109/LRA.2021.3068662

Publication date
2021

Document Version
Final published version

Published in
IEEE Robotics and Automation Letters

Where to go Next: Learning a Subgoal Recommendation Policy for Navigation in Dynamic Environments

Bruno Brito, Michael Everett, Jonathan P. How, and Javier Alonso-Mora

***Abstract*—Robotic navigation in environments shared with other robots or humans remains challenging because the intentions of the surrounding agents are not directly observable and the environment conditions are continuously changing. Local trajectory optimization methods, such as model predictive control (MPC), can deal with those changes but require global guidance, which is not trivial to obtain in crowded scenarios. This letter proposes to learn, via deep Reinforcement Learning (RL), an interaction-aware policy that provides long-term guidance to the local planner. In particular, in simulations with cooperative and non-cooperative agents, we train a deep network to recommend a subgoal for the MPC planner. The recommended subgoal is expected to help the robot in making progress towards its goal and accounts for the expected interaction with other agents. Based on the recommended subgoal, the MPC planner then optimizes the inputs for the robot satisfying its kinodynamic and collision avoidance constraints. Our approach is shown to substantially improve the navigation performance in terms of number of collisions as compared to prior MPC frameworks, and in terms of both travel time and number of collisions compared to deep RL methods in cooperative, competitive and mixed multiagent scenarios.**

***Index Terms*—Deep reinforcement learning, motion and path planning in dynamic environments or for multi-robot systems.**

I. Introduction

AUTONOMOUS robot navigation in crowds remains difficult due to the interaction effects among navigating agents. Unlike multi-robot environments, robots operating among pedestrians require decentralized algorithms that can handle a mixture of other agents' behaviors without depending on explicit communication between agents.

Manuscript received October 15, 2020; accepted February 14, 2021. Date of publication March 24, 2021; date of current version April 13, 2021. This letter was recommended for publication by Associate Editor S. J. Guy and Editor N. Amato upon evaluation of the reviewer' comments. This work was supported in part by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement 101017008, in part by the Amsterdam Institute for Advanced Metropolitan Solutions, in part by the Netherlands Organisation for Scientific Research (NWO) domain Applied Sciences (Veni 15916), and in part by Ford Motor Company. *(Corresponding author: Bruno Brito.)*

Bruno Brito and Javier Alonso-Mora are with the Cognitive Robotics (CoR) Department, Delft University of Technology, Delft 2628 CD, The Netherlands (e-mail: bruno.debrito@tudelft.nl; j.alonsomora@tudelft.nl).

Michael Everett and Jonathan P. How are with the Aerospace Controls Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: mfe@mit.edu; jhow@mit.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2021.3068662, provided by the authors.

Digital Object Identifier 10.1109/LRA.2021.3068662

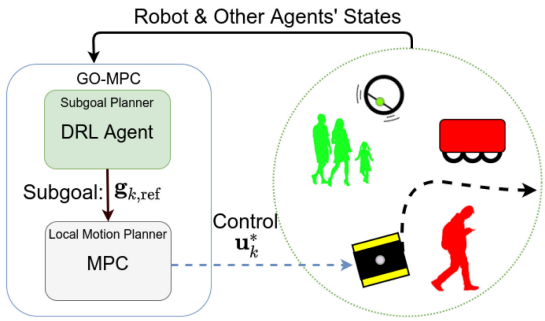


Fig. 1. Proposed navigation architecture. The subgoal planner observes the environment and suggests the next subgoal position to the local motion planner, the MPC. The MPC then computes a local trajectory and the robot executes the next optimal control command, which minimizes the distance to the provided position reference while respecting collision and kinodynamic constraints.

Several state-of-the-art collision avoidance methods employ model-predictive control (MPC) with online optimization to compute motion plans that are guaranteed to respect important constraints [1]. These constraints could include the robot's nonlinear kino-dynamics model or collision avoidance of static obstacles and other dynamic, decision-making agents (e.g., pedestrians). Although online optimization becomes less computationally practical for extremely dense scenarios, modern solvers enable real-time motion planning in many situations of interest [2].

A key challenge is that the robot's global goal is often located far beyond the planning horizon, meaning that a local subgoal or cost-to-go heuristic must be specified instead. This is straightforward in a static environment (e.g., using euclidean/diffusion [3] distance), but the presence interactive agents makes it difficult to quantify which subgoals will lead to the global goal quickest. A body of work addresses this challenge with deep reinforcement learning (RL), in which agents learn a model of the long-term cost of actions in an offline training phase (usually in simulation) [4]–[7]. The learned model is fast-to-query during online execution, but the way learned costs/policies have been used to date does not provide guarantees on collision avoidance or feasibility with respect to the robot dynamics.

In this letter, we introduce Goal Oriented Model Predictive Control (GO-MPC), which enhances state-of-art online optimization-based planners with a learned global guidance policy. In an offline RL training phase, an agent learns a policy that uses the current world configuration (the states of the robot and

other agents, and a global goal) to recommend a local subgoal for the MPC, as depicted in Fig. 1. Then, the MPC generates control commands ensuring that the robot and collision avoidance constraints are satisfied (if a feasible solution is found) while making progress towards the suggested subgoal. Our approach maintains the kino-dynamic feasibility and collision avoidance guarantees inherent in an MPC formulation, while improving the average time-to-goal and success rate by leveraging past experience in crowded situations.

The main contributions of this work are:

- A goal-oriented Model Predictive Control method (GO-MPC) for navigation among interacting agents, which utilizes a learned global guidance policy (recommended subgoal) in the cost function and ensures that dynamic feasibility and collision avoidance constraints are satisfied when a feasible solution to the optimization problem is found;
- An algorithm to train an RL agent jointly with an optimization-based controller in mixed environments, which is directly applicable to real-hardware, reducing the sim to real gap.

Finally, we present simulation results demonstrating an improvement over several state-of-art methods in challenging scenarios with realistic robot dynamics and a mixture of cooperative and non-cooperative neighboring agents. Our approach shows different navigation behaviors: navigating through the crowd when interacting with cooperative agents, avoiding congestion areas when non-cooperative agents are present and enabling communication-free decentralized multi-robot collision avoidance.

A. Related Work

1) Navigation Among Crowds: Past work on navigation in cluttered environments often focuses on *interaction models* using geometry [8], [9], physics [10], topologies [11], [12], handcrafted functions [13], and cost functions [14], [14] or joint probability distributions [15] learned from data. While accurate interaction models are critical for collision avoidance, this work emphasizes that the robot's performance (time-to-goal) is highly dependent on the quality of its *cost-to-go model* (i.e., the module that recommends a subgoal for the local planner). Designing a useful cost-to-go model in this problem remains challenging, as it requires quantifying how "good" a robot's configuration is with respect to dynamic, decision-making agents. In [4], deep RL was introduced as a way of modeling cost-to-go through an offline training phase; the online execution used simple vehicle and interaction models for collision-checking. Subsequent works incorporated other interactions to generate more socially compliant behavior within the same framework [5], [16]. To relax the need for simple online models, [6] moved the collision-checking to the offline training phase. While these approaches use pre-processed information typically available from perception pipelines (e.g., pedestrian detection, tracking systems), other works proposed to learn end-to-end policies [7], [17]. Although all of these RL-based approaches learn to estimate the cost-to-go, the online implementations do not provide guarantees that the recommended actions will satisfy realistic vehicle dynamics or collision avoidance constraints. Thus, this work builds on the promising idea of learning a cost-to-go model, but we start from an inherently safe MPC formulation.

2) Learning-Enhanced MPC: Outside the context of crowd navigation, numerous recent works have proposed learning-based solutions to overcome some of the known limitations of optimization-based methods (e.g., nonlinear MPC) [18]. For example, solvers are often sensitive to the quality of the *initial guess* hence, [19] proposes to learn a policy from data that efficiently "warm-starts" a MPC. *Model inaccuracies* can lead to sub-optimal MPC solution quality; [20] proposes to learn a policy by choosing between two actions with the best expected reward at each timestep: one from model-free RL and one from a model-based trajectory optimizer. Alternatively, RL can be used to optimize the weights of an MPC-based Q-function approximator or to update a robust MPC parametrization [21]. When the model is completely unknown, [22] shows a way of learning a dynamics model to be used in MPC. *Computation time* is another key challenge: [23] learns a cost-to-go estimator to enable shortening of the planning horizons without sacrificing much solution quality, although their approach differs from this work as it uses local and linear function approximators which limits its applicability to high-dimensional state spaces. The closest related works address *cost tuning* with learning. MPC's cost functions are replaced with a value function learned via RL offline in [24] (terminal cost) and [25] (stage cost). [26] deployed value function learning on a real robot outperforming an expert-tuned MPC. While these ideas also use RL for a better cost-to-go model, this work focuses on the technical challenge of learning a subgoal policy required for navigation through crowds avoiding the approximation issues and extrapolation issues to unseen events. Moreover, this work learns to set terminal constraints rather than setting a cost with a value function.

3) Combining MPC with RL: Recently, there is increasing interest on approaches combining the strengths of MPC and RL as suggested in [27]. For instance, optimization-based planning has been used to explore high-reward regions and distill the knowledge into a policy neural network, rather than a neural network policy to improve an optimization. [28]–[30].

Similar to our approach, [31] utilizes the RL policy during training to ensure exploration and employs a MPC to optimize sampled trajectories from the learned policy at test time. Moreover, policy networks have be used to generate proposals for a sampling-based MPC [32], or to select goal positions from a predefined set [33].

Nevertheless, to the extent of our knowledge, approaches combining the benefits of both optimization and learning-based methods were not explored in the context of crowd navigation. Moreover, the works exploring a similar idea of learning a cost-to-go model do not allow to explicitly define collision constraints and ensure safety. To overcome the previous issues, in this letter, we explore the idea of learning a cost-to-go model to directly generate subgoal positions, which lead to higher long-term rewards and too give the role of local collision avoidance and kinematics constraints to an optimization-based planner.

Such cost-to-go information can be formulated as learning a value function for the ego-agent state-space providing information which states are more valuable [25]. In contrast, we propose to learn a policy directly informing which actions lead to higher

rewards allowing to directly incorporate the MPC controller in the training phase.

II. Preliminaries

Throughout this letter, vectors are denoted in bold lowercase letters, $\mathbf{x}$, matrices in capital, M, and sets in calligraphic uppercase, $\mathcal{S}$. $\|\mathbf{x}\|$ denotes the Euclidean norm of $\mathbf{x}$ and $\|\mathbf{x}\|_Q = \mathbf{x}^T Q \mathbf{x}$ denotes the weighted squared norm. The variables $\{\mathbf{s}, \mathbf{a}\}$ denote the state and action variables used in the RL formulation, and $\{\mathbf{x}, \mathbf{u}\}$ denote the control state and action commands used in the optimization problem.

A. Problem Formulation

Consider a scenario where a robot must navigate from an initial position $\mathbf{p}_0$ to a goal position $\mathbf{g}$ on the plane $\mathbb{R}^2$, surrounded by n non-communicating agents. At each time-step t, the robot first observes its state $\mathbf{s}_t$ (defined in Section III-A2) and the set of the other agents states $\mathbf{S}_t = \bigcup_{i\in\{1,\dots,n\}} \mathbf{s}_t^i$, then takes action $\mathbf{a}_t$, leading to the immediate reward $R(\mathbf{s}_t, \mathbf{a}_t)$ and next state $\mathbf{s}_{t+1} = h(\mathbf{s}_t, \mathbf{a}_t)$, under the transition model h.

We use the superscript $i \in \{1, \dots, n\}$ to denote the i-th nearby agent and omit the superscript when referring to the robot. For each agent $i \in \{0, n\}$, $\mathbf{p}_t^i \in \mathbb{R}^2$ denotes its position, $\mathbf{v}_t^i \in \mathbb{R}^2$ its velocity at step t relative to a inertial frame, and r_i the agent radius. We assume that each agent's current position and velocity are observed (e.g., with on-board sensing) while other agents' motion intentions (e.g., goal positions) are unknown. Finally, O_t denotes the area occupied by the robot and O_t^i by each surrounding agent, at time-step t. The goal is to learn a policy π for the robot that minimizes time to goal while ensuring collision-free motions, defined as:

$$\begin{aligned}
\pi^* = \underset{\pi}{\operatorname{argmax}} \;\; & \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t, \mathbf{S}_t))\right] \\
\text{s.t.} \;\; & \mathbf{x}_{t+1} = f(\mathbf{x}_t, \boldsymbol{u}_t), && \text{(1a)} \\
& \boldsymbol{s}_T = \mathbf{g}, && \text{(1b)} \\
& O_t(\mathbf{x}_t) \cap O_t^i = \emptyset && \text{(1c)} \\
& \mathbf{u}_t \in \mathcal{U},\ \mathbf{s}_t \in \mathcal{S},\ \mathbf{x}_t \in \mathcal{X}, \\
& \forall t \in [0, T], \quad \forall i \in \{1, \dots, n\} && \text{(1d)}
\end{aligned}$$

where (1a) are the transition dynamic constraints considering the dynamic model f, (1b) the terminal constraints, (1c) the collision avoidance constraints and $\mathcal{S}, \mathcal{U}$ and $\mathcal{X}$ are the set of admissible states, inputs (e.g., to limit the robot's maximum speed) and the set of admissible control states, respectively. Note that we only constraint the control states of the robot. Moreover, we assume other agents have various behaviors (e.g., cooperative or non-cooperative): each agent samples a policy from a closed set $\mathcal{P} = \{\pi_1, \dots, \pi_m\}$ (defined in Section II-C) at the beginning of each episode.

B. Agent Dynamics

Real robotic systems' inertia imposes limits on linear and angular acceleration. Thus, we assume a second-order unicycle model for the robot [34]:

$$\begin{aligned}
\dot{x} &= v\cos\psi \quad \dot{v} = u_a \\
\dot{y} &= v\sin\psi \quad \dot{\omega} = u_\alpha \\
\dot{\psi} &= \omega
\end{aligned} \tag{2}$$

where x and y are the agent position coordinates and ψ is the heading angle in a global frame. v is the agent forward velocity, ω denotes the angular velocity and, u_a the linear and u_α angular acceleration, respectively.

C. Modeling Other Agents' Behaviors

In a real scenario, agents may follow different policies and show different levels of cooperation. Hence, in contrast to previous approaches, we do not consider all the agents to follow the same policy [6], [35]. At the beginning of an episode, each non-ego agent either follows a cooperative or a non-cooperative policy. For the cooperative policy, we employ the Reciprocal Velocity Obstacle (RVO) [36] model with a random cooperation coefficient[1] $c^i \sim \mathcal{N}(0.1, 1)$ sampled at the beginning of the episode. The "reciprocal" in RVO means that all agents follow the same policy and use the cooperation coefficient to split the collision avoidance effort among the agents (e.g., a coefficient of 0.5 means that each agent will apply half of the effort to avoid the other). In this work, for the non-cooperative agents, we consider both constant velocity (CV) and non-CV policies. The agents following a CV model drive straight in the direction of their goal position with constant velocity. The agents following a non-CV policy either move in sinusoids towards their final goal position or circular motion around their initial position.

III. Method

Learning a sequence of intermediate goal states that lead an agent toward a final goal destination can be formulated as a single-agent sequential decision making problem. Because parts of the environment can be difficult to model explicitly, the problem can be solved with a reinforcement learning framework. Hence, we propose a two-level planning architecture, as depicted in Fig. 1, consisting of a subgoal recommender (Section III-A2) and an optimization-based motion planner (Section II-C). We start by defining the RL framework and our's policy architecture (Section III-A2). Then, we formulate the MPC to execute the policy's actions and ensure local collision avoidance (Section III-B).

A. Learning a Subgoal Recommender Policy

We aim to develop a decision-making algorithm to provide an estimate of the cost-to-go in dynamic environments with mixed-agents. In this letter, we propose to learn a policy directly informing which actions lead to higher rewards.

1) RL Formulation: As in [4], the observation vector is composed by the ego-agent and the surrounding agents states, defined as:

$$\begin{aligned}
\mathbf{s}_t &= [d_{\mathbf{g}}, \mathbf{p}_t - \mathbf{g}, v_{\text{ref}}, \psi, r] \quad \text{(Ego-agent)} \\
\mathbf{s}_t^i &= \left[\mathbf{p}_t^i, \mathbf{v}_t^i, r^i, d_t^i, r^i + r\right] \;\; \forall i \in \{1, n\} \quad \text{(Other agents)}
\end{aligned} \tag{3}$$

[1]This coefficient is denoted as α_A^B in [8]

where $\mathbf{s}_t$ is the ego-agent state and $\mathbf{s}_t^i$ the i-th agent state at step t. Moreover, $d_\mathbf{g} = \|\mathbf{p}_t - \mathbf{g}\|$ is the ego-agent's distance to goal and $d_t^i = \|\mathbf{p}_t - \mathbf{p}_t^i\|$ is the distance to the i-th agent.

Here, we seek to learn the optimal policy for the ego-agent $\pi : (\mathbf{s}_t, \mathbf{S}_t) \rightarrow \mathbf{a}_t$ mapping the ego-agent's observation of the environment to a probability distribution of actions. We consider a continuous action space $\mathcal{A} \subset \mathbb{R}^2$ and define an action as position increments providing the direction maximizing the ego-agent rewards, defined as:

$$\mathbf{p}_t^{\text{ref}} = \mathbf{p}_t + \boldsymbol{\delta}_t \tag{4a}$$

$$\pi_{\theta^\pi}(\mathbf{s}_t, \mathbf{S}_t) = \boldsymbol{\delta}_t = [\boldsymbol{\delta}_{t,x}, \boldsymbol{\delta}_{t,y}] \tag{4b}$$

$$\|\boldsymbol{\delta}_t\| \leq N v_{\max}, \tag{4c}$$

where $\boldsymbol{\delta}_{k,x}, \boldsymbol{\delta}_{k,y}$ are the $(x,\ y)$ position increments, $v_{\max}$ the maximum linear velocity and θ^π are the network policy parameters. Moreover, to ensure that the next sub-goal position is within the planning horizon of the ego-agent, we bound the action space according with the planning horizon N of the optimization-based planner and its dynamic constraints, as represented in (4b).

We design the reward function to motivate the ego-agent to reach the goal position while penalizing collisions:

$$R(\mathbf{s}, \mathbf{a}) = \begin{cases} r_{\text{goal}} & \text{if } \mathbf{p} = \mathbf{p}_g \\ r_{\text{collision}} & \text{if } d_{\min} < r + r^i \ \ \forall i \in \{1, n\} \\ r_t & \text{otherwise} \end{cases} \tag{5}$$

where $d_{min} = \min_i \|\mathbf{p} - \mathbf{p}^i\|$ is the distance to the closest surrounding agent. r_t allows to adapt the reward function as shown in the ablation study (Section IV-C), r_{goal} rewards the agent if reaches the goal $r_{\text{collision}}$ penalizes if it collides with any other agents. In Section. IV-C we analyze its influence in the behavior of the learned policy.

2) Policy Network Architecture: A key challenge in collision avoidance among pedestrians is that the number of nearby agents can vary between timesteps. Because feed-forward NNs require a fixed input vector size, prior work [6] proposed the use of Recurrent Neural Networks (RNNs) to compress the n agent states into a fixed size vector at each time-step. Yet, that approach discarded time-dependencies of successive observations (i.e., hidden states of recurrent cells).

Here, we use the "store-state" strategy, as proposed in [37]. During the rollout phase, at each time-step we store the hidden-state of the RNN together with the current state and other agents state, immediate reward and next state, $(\mathbf{s}_k, \mathbf{S}_k, \mathbf{h}_k, \mathbf{r}_k, \mathbf{s}_{k+1})$. Moreover, the previous hidden-state is feed back to warm-start the RNN in the next step, as depicted in Fig. 2. During the training phase, we use the stored hidden-states to initialize the network. Our policy architecture is depicted in Fig. 2. We employ a RNN to encode a variable sequence of the other agents states $\mathbf{S}_k$ and model the existing time-dependencies. Then, we concatenate the fixed-length representation of the other agent's states with the ego-agent's state to create a join state representation. This representation vector is fed to two fully-connected layers (FCL). The network has two output heads: one estimates the probability distribution parameters $\pi_{\theta^\pi}(\mathbf{s}, \mathbf{S}) \sim \mathcal{N}(\mu, \sigma)$ of the policy's action space and the other estimates the state-value function $V^\pi(s_t) := \mathbb{E}_{s_{t+1:\infty},}[\sum_{l=0}^{\infty} r_{t+l}]$. μ and σ are the mean and variance of the policy's distribution, respectively.

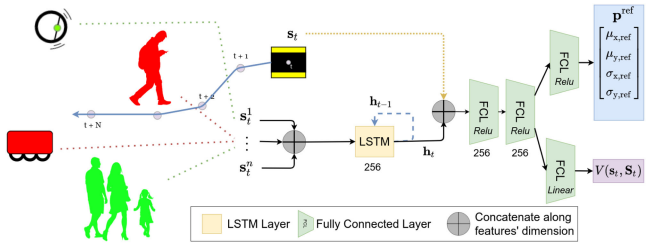


Fig. 2. Proposed network policy architecture.

B. Local Collision Avoidance: Model Predictive Control

Here, we employ MPC to generate locally optimal commands respecting the kino-dynamics and collision avoidance constraints. To simplify the notation used, hereafter, we assume the current time-step t as zero.

1) State and Control Inputs: We define the ego-agent control input vector as $\mathbf{u} = [u_a, u_\alpha]$ and the control state as $\mathbf{x} = [x, y, \psi, v, w] \in \mathbb{R}^5$ following the dynamics model defined in Section II-B.

2) Dynamic Collision Avoidance: We define a set of nonlinear constraints to ensure that the MPC generates collision-free control commands for the ego-agent (if a feasible solution exists). To limit the problem complexity and ensure to find a solution in real-time, we consider a limited number of surrounding agents $\mathcal{X}^m$, with $m \leq n$. Consider $\mathcal{X}^n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ as the set of all surrounding agent states, than the set of the m-th closest agents is:

Definition **1** *A* set $\mathcal{X}^m \subseteq \mathcal{X}^n$ is the set of the m-th closest agents if the euclidean distance $\forall \mathbf{x}_j \in \mathcal{X}^m, \forall \mathbf{x}_i \in \mathcal{X}^n \setminus \mathcal{X}^m : \|\mathbf{x}_j, \mathbf{x}\| \leq \|\mathbf{x}_i, \mathbf{x}\|$.

We represent the area occupied by each agent O^i as a circle with radius r_i. To ensure collision-free motions, we impose that each circle $i \in \{1, \ldots, n\}$ i does not intersect with the area occupied by the ego-agent resulting in the following set of inequality constraints:

$$c_k^i(\mathbf{x}_k, \mathbf{x}_k^i) = \left\|\mathbf{p}_k, \mathbf{p}_k^i\right\| \geq r + r_i, \tag{6}$$

for each planning step k. This formulation can be extended for agents with general quadratic shapes, as in [2].

3) Cost function: The subgoal recommender provides a reference position $\mathbf{p}_0^{\text{ref}}$ guiding the ego-agent toward the final goal position $\mathbf{g}$ and minimizing the cost-to-go while accounting for the other agents. The terminal cost is defined as the normalized distance between the ego-agent's terminal position (after a planning horizon N) and the reference position (with weight coefficient Q_N):

$$J_N(\mathbf{p}_N, \pi(\mathbf{x}, \mathbf{X})) = \left\| \frac{\mathbf{p}_N - \mathbf{p}_0^{\text{ref}}}{\mathbf{p}_0 - \mathbf{p}_0^{\text{ref}}} \right\|_{Q_N}, \tag{7}$$

To ensure smooth trajectories, we define the stage cost as a quadratic penalty on the ego-agent control commands

$$J_k^{\mathbf{u}}(\mathbf{u}_k) = \|\mathbf{u}_k\|_{Q_u}, \quad k = \{0, 1, \ldots, N-1\}, \tag{8}$$

where Q_u is the weight coefficient.

4) MPC Formulation: The MPC is then defined as a non-convex optimization problem

$$\begin{aligned}\min_{\mathbf{x}_{1:N},\mathbf{u}_{0:N-1}} \; & J_N(\mathbf{x}_N,\mathbf{p}_0^{\text{ref}}) + \sum_{k=0}^{N-1} J_k^u(\mathbf{u}_k) \\ \text{s.t.} \; & \mathbf{x}_0 = \mathbf{x}(0), \quad \text{(1d), (2)}, \\ & c_k^i(\mathbf{x}_k,\mathbf{x}_k^i) > r + r_i, \\ & \mathbf{u}_k \in \mathcal{U}, \quad \mathbf{x}_k \in \mathcal{S}, \\ & \forall i \in \{1,\dots,n\};\, \forall k \in \{0,\dots,N-1\}.\end{aligned} \tag{9}$$

In this work, we assume a constant velocity model estimate of the other agents' future positions, as in [2].

C. PPO-MPC

In this work, we train the policy using a state-of-art method, Proximal Policy Optimization (PPO) [38], but the overall framework is agnostic to the specific RL training algorithm. We propose to jointly train the guidance policy π_{θ^π} and value function $V_{\theta^V}(\mathbf{s})$ *with* the MPC, as opposed to prior works [6] that use an idealized low-level controller during policy training (that cannot be implemented on a real robot). Algorithm 1 describes the proposed training strategy and has two main phases: supervised and RL training. First, we randomly initialize the policy and value function parameters $\{\theta^\pi, \theta^V\}$. Then, at the beginning of each episode we randomly select the number of surrounding agents between $[1, n_{\text{agents}}]$, the training scenario and the surrounding agents policy. More details about the different training scenarios and n_{agents} considered is given in Section IV-B.

An initial RL policy is unlikely to lead an agent to a goal position. Hence, during the warm-start phase, we use the MPC as an expert and perform supervised training to train the policy and value function parameters for n_{MPC} steps. By setting the MPC goal state as the ego-agent final goal state $\mathbf{p}^{\text{ref}} = \mathbf{g}$ and solving the MPC problem, we obtain a locally optimal sequence of control states $\mathbf{x}^*_{1:N}$. For each step, we define $\mathbf{a}_t^* = \mathbf{x}^*_{t,N}$ and store the tuple containing the network hidden-state, state, next state, and reward in a buffer $\mathcal{B} \leftarrow \{\mathbf{s}_k, \mathbf{a}_t^*, r_k, \mathbf{h}_k, \mathbf{s}_{k+1}\}$. Then, we compute advantage estimates [39] and perform a supervised training step

$$\theta^V_{k+1} = \arg\min_{\theta^V} \mathbb{E}_{(\mathbf{a}_k,\mathbf{s}_k,r_k)\sim\mathcal{D}_{\text{MPC}}}[\left\| V_\theta(\mathbf{s}_k) - V_k^{\text{targ}} \right\|] \tag{10}$$

$$\theta^\pi_{k+1} = \arg\min_{\theta} \mathbb{E}_{(\mathbf{a}_k^*,\mathbf{s}_k)\sim\mathcal{D}_{\text{MPC}}}[\|\mathbf{a}_k^* - \pi_\theta(\mathbf{s}_k)\|] \tag{11}$$

where θ^V, θ^π are the value function and policy parameters, respectively. Note that θ^V and θ^π share the same parameter except for the final layer, as depicted in Fig. 2. Afterwards, we use Proximal Policy Optimization (PPO) [38] with clipped gradients for training the policy. PPO is a on-policy method addressing the high-variance issue of policy gradient methods for continuous control problems. We refer the reader to [38] for more details about the method's equations. Please note that our approach is agnostic to which RL algorithm we use. Moreover, to increase the learning speed during training, we gradually increase the number of agents in the training environments (curriculum learning [40]).

Algorithm 1: PPO-MPC Training.

Inputs: planning horizon H, value fn. and policy parameters $\{\theta^V, \theta^\pi\}$, number of supervised and RL training episodes $\{n_{\text{MPC}}, n_{\text{episodes}}\}$, number of agents n, $n_{\text{mini-batch}}$, and reward function $R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{a}_{t+1})$
Initialize states: $\{\mathbf{s}_0^0, \dots, \mathbf{s}_0^n\} \sim \mathcal{S}$, $\{\mathbf{g}^0, \dots, \mathbf{g}^n\} \sim \mathcal{S}$
while $episode < n_{\text{episodes}}$ **do**
  Initialize $\mathcal{B} \leftarrow \emptyset$ and $h_0 \leftarrow \emptyset$
  for $k = 0, \dots, n_{\text{mini-batch}}$ **do**
    if $episode \leq n_{\text{MPC}}$ **then**
      Solve Eq.9 considering $\mathbf{p}^{\text{ref}} = \mathbf{g}$
      Set $\mathbf{a}_t^* = \mathbf{x}_N^*$
    else
      $\mathbf{p}^{\text{ref}} = \pi_\theta(\mathbf{s}_t, \mathbf{S}_t)$
    end if
    $\{\mathbf{s}_k, \mathbf{a}_k, r_k, \mathbf{h}_{k+1}, \mathbf{s}_{k+1}, \text{done}\} = \text{Step}(\mathbf{s}_t^*, \mathbf{a}_t^*, \mathbf{h}_t)$
    Store $\mathcal{B} \leftarrow \{\mathbf{s}_k, \mathbf{a}_k, r_k, \mathbf{h}_{k+1}, \mathbf{s}_{k+1}, \text{done}\}$
    if done **then**
      $episode += 1$
      Reset hidden-state: $h_t \leftarrow \emptyset$
      Initialize: $\{\mathbf{s}_0^0, \dots, \mathbf{s}_0^n\} \sim \mathcal{S}, \{\mathbf{g}^0, \dots, \mathbf{g}^n\} \sim \mathcal{S}$
    end if
  end for
  if $episode \leq n_{\text{MPC}}$ **then**
    Supervised training: Eq.10 and Eq.11
  else
    PPO training [38]
  end if
end while
return $\{\theta^V, \theta^\pi\}$

IV. Results

This section quantifies the performance throughout the training procedure, provides an ablation study, and compares the proposed method (sample trajectories and numerically) against the following baseline approaches:

- MPC: Model Predictive Controller from Section III-B with final goal position as position reference, $\mathbf{p}_{\text{ref}} = \mathbf{g}$;
- DRL [6]: state-of-the-art Deep Reinforcement Learning approach for multi-agent collision avoidance

To analyze the impact of a realistic kinematic model during training, we consider two variants of the DRL method [6]: the same RL algorithm [6] was used to train a policy under a first-order unicycle model, referred to as DRL, and a second-order unicycle model (2), referred to as DRL-2. All experiments use a second-order unicycle model (2) in environments with cooperative and non-cooperative agents to represent realistic robot/pedestrian behavior. Animations of sample trajectories accompany the letter.

A. Experimental Setup

The proposed training algorithm builds upon the open-source PPO implementation provided in the Stable-Baselines [41] package. We used a laptop with an Intel Core i7 and 32 GB of RAM for training. To solve the non-linear and non-convex MPC problem of (9), we used the ForcesPro [42] solver. If no feasible

TABLE I
HYPER-PARAMETERS

Planning Horizon N	2 s	Num. mini batches	2048
Number of Stages	20	r_{goal}	3
γ	0.99	$r_{\text{collision}}$	-10
Clip factor	0.1	Learning rate	10^{-4}

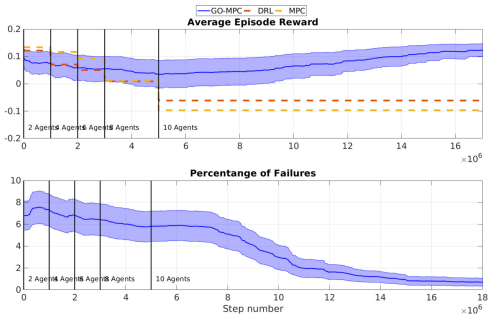


Fig. 3. Moving average rewards and percentage of failure episodes during training. The top plot shows our method average episode reward vs DRL [6] and simple MPC.

solution is found within the maximum number of iterations, then the robot decelerates. All MPC methods used in this work consider collision constraints with up to the closest six agents so that the optimization problem can be solved in less than 20 ms. Moreover, our policy's network has an average computation time of 2ms with a variance of 0.4ms for all experiments. Hyperparameter values are summarized in Table I.

B. Training Procedure

To train and evaluate our method we have selected four navigation scenarios, similar to [5]–[7]:

- **Symmetric swapping**: Each agent's position is randomly initialized in different quadrants of the $\mathbb{R}^2$ x-y plane, where all agents have the same distance to the origin. Each agent's goal is to swap positions with an agent from the opposite quadrant.
- **Asymmetric swapping**: As before, but all agents are located at different distances to the origin.
- **Pair-wise swapping**: Random initial positions; pairs of agents' goals are each other's intial positions
- **Random**: Random initial & goal positions

Each training episode consists of a random number of agents and a random scenario. At the start of each episode, each other agent's policy is sampled from a binomial distribution (80% cooperative, 20% non-cooperative). Moreover, for the cooperative agents we randomly sample a cooperation coefficient $c^i \sim U(0.1, 1)$ and for the non-cooperative agents is randomly assigned a CV or non-CV policy (i.e., sinusoid or circular). Fig. 3 shows the evolution of the robot average reward and the percentage of failure episodes. The top sub-plot compares our method average reward with the two baseline methods: DRL (with pre-trained weights) and MPC. The average reward for the baseline methods (orange, yellow) drops as the number of agents increases (each vertical bar). In contrast, our method (blue) improves with training and eventually achieves higher average reward for 10-agent scenarios than baseline methods achieve for 2-agent scenarios. The bottom plot demonstrates

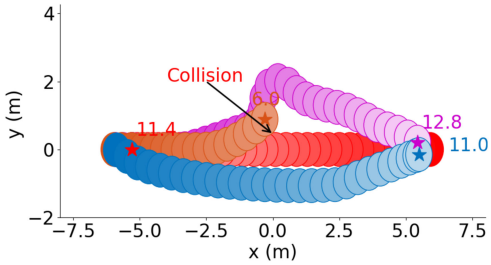


Fig. 4. Two agents swapping scenario. In blue is depicted the trajectory of robot, in red the non-cooperative agent, in purple the DRL agent and, in orange the MPC.

that the percentage of collisions decreases throughout training despite the number of agents increasing.

C. Ablation Study

A key design choice in RL is the reward function; here, we study the impact on policy performance of three variants of reward. The *sparse* reward uses $r_t = 0$ (only non-zero reward upon reaching goal/colliding). The *time* reward uses $r_t = -0.01$ (penalize every step until reaching goal). The *progress* reward uses $r_t = 0.01 * (\|\mathbf{s}_t - \mathbf{g}\| - \|\mathbf{s}_{t+1} - \mathbf{g}\|)$ (encourage motion toward goal). Aggregated results in Table II show that the resulting policy trained with a time reward function allows the robot to reach the goal with minimum time, to travel the smallest distance, and achieve the lowest percentage of failure cases. Based on these results, we selected the policy trained with the time reward function for the subsequent experiments.

D. Qualitative Analysis

This section compares and analyzes trajectories for different scenarios. Fig. 4 shows that our method resolves a failure mode of both RL and MPC baselines. The robot has to swap position with a non-cooperative agent (red, moving right-to-left) and avoid a collision. We overlap the trajectories (moving left-to-right) performed by the robot following our method (blue) versus the baseline policies (orange, magenta). The MPC policy (orange) causes a collision due to the dynamic constraints and limited planning horizon. The DRL policy avoids the non-cooperative agent, but due to its reactive nature, only avoids the non-cooperative agent when very close, resulting in larger travel time. Finally, when using our approach, the robot initiates a collision avoidance maneuver early enough to lead to a smooth trajectory and faster arrival at the goal.

We present results for mixed settings in Fig. 5 and homogeneous settings in Fig. 6 with $n \in \{6, 8, 10\}$ agents. In mixed settings, the robot follows our proposed policy while the other agents either follow an RVO [36] or a non-cooperative policy (same distribution as in training). Fig. 5 demonstrates that our navigation policy behaves differently when dealing with only cooperative agents or both cooperative and non-cooperative. Whereas in Fig. 5(a) the robot navigates through the crowd, Fig. 5(b) shows that the robot takes a longer path to avoid the congestion.

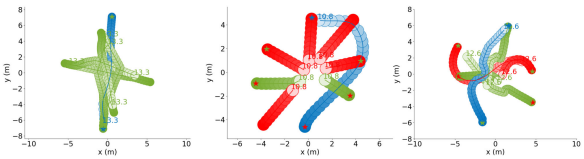
In the homogeneous setting, all agents follow our proposed policy. Fig. 6 shows that our method achieves faster time-to-goal than two DRL baselines. Note that this scenario was never

TABLE II
ABLATION STUDY: DISCRETE REWARD FUNCTION LEADS TO BETTER POLICY THAN SPARSE, DENSE REWARD FUNCTIONS. RESULTS ARE AGGREGATED OVER 200 RANDOM SCENARIOS WITH $n \in \{6, 8, 10\}$ AGENTS.

	Time to Goal [s]			% failures (% collisions / % timeout)			Traveled distance Mean [m]		
# agents	6	8	10	6	8	10	6	8	10
Sparse Reward	8.00	8.51	8.52	0 (0 / 0)	1 (0 / 1)	2 (1 / 1)	13.90	14.34	14.31
Progress Reward	8.9	8.79	9.01	2 (1 / 1)	3 (3 / 0)	1 (1 / 0)	14.75	14.57	14.63
Time Reward	**7.69**	**8.03**	**8.12**	**0 (0 / 0)**	**0 (0 / 0)**	**0 (0 / 0)**	**13.25**	**14.01**	**14.06**

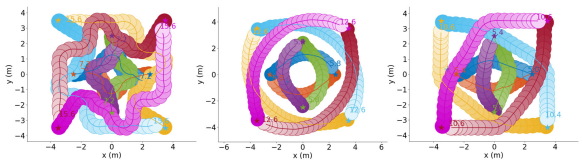
TABLE III
STATISTICS FOR 200 RUNS OF PROPOSED METHOD (GO-MPC) COMPARED TO BASELINES (MPC, DRL [6] AND DRL-2, AN EXTENSION OF [6]): TIME TO GOAL AND TRAVELED DISTANCE FOR THE SUCCESSFUL EPISODES, AND NUMBER OF EPISODES RESULTING IN COLLISION FOR $n \in \{6, 8, 10\}$ AGENTS. FOR THE MIXED SETTING, 80% OF AGENTS ARE COOPERATIVE, AND 20% ARE NON-COOPERATIVE.

	Time to Goal (mean $\pm$ std) [s]			% failures (% collisions / % deadlocks)			Traveled Distance (mean $\pm$ std) [m]		
# agents	6	8	10	6	8	10	6	8	10
Mixed Agents									
MPC	11.2 $\pm$ 2.2	11.3 $\pm$ 2.4	11.0 $\pm$ 2.2	13 (0 / 0)	22 (0 / 0)	22 (22 / 0)	12.24 $\pm$ 2.3	12.40 $\pm$ 2.5	12.13 $\pm$ 2.3
DRL [6]	13.7 $\pm$ 3.0	13.7 $\pm$ 3.1	14.4 $\pm$ 3.3	17 (17 / 0)	23 (23 / 0)	29 (29 / 0)	13.75 $\pm$ 3.3	13.80 $\pm$ 4.0	14.40 $\pm$ 3.3
DRL-2 [6]+	15.3 $\pm$ 2.3	16.1 $\pm$ 2.2	16.7 $\pm$ 2.2	6 (6 / 0)	10 (10 / 0)	13 (13 / 0)	14.86 $\pm$ 2.3	16.05 $\pm$ 2.2	16.66 $\pm$ 2.2
GO-MPC	12.7 $\pm$ 2.7	12.9 $\pm$ 2.8	13.3 $\pm$ 2.8	**0 (0 / 0)**	**0 (0 / 0)**	**0 (0 / 0)**	13.65 $\pm$ 2.7	13.77 $\pm$ 2.8	14.29 $\pm$ 2.8
Homogeneous									
MPC	17.37 $\pm$ 2.9	16.38 $\pm$ 1.5	16.64 $\pm$ 1.7	30 (29 / 1)	36 (25 / 11)	35 (28 / 7)	11.34 $\pm$ 2.1	10.86 $\pm$ 2.3	10.62 $\pm$ 2.8
DRL [6]	14.18 $\pm$ 2.4	14.40 $\pm$ 2.7	14.64 $\pm$ 3.3	16 (14 / 2)	20 (18 / 2)	20 (20 / 0)	12.81 $\pm$ 2.3	12.23 $\pm$ 2.3	12.23 $\pm$ 3.2
DRL-2 [6]+	15.96 $\pm$ 3.1	17.47 $\pm$ 4.2	15.96 $\pm$ 4.5	17 (11 / 6)	29 (21 / 8)	28 (24 / 4)	15.17 $\pm$ 3.0	15.85 $\pm$ 4.2	15.40 $\pm$ 4.5
GO-MPC	**13.77 $\pm$ 2.9**	**14.30 $\pm$ 3.3**	**14.63 $\pm$ 2.9**	**0 (0 / 0)**	**0 (0 / 0)**	**2 (1 / 1)**	14.67 $\pm$ 2.9	15.09 $\pm$ 3.3	15.12 $\pm$ 2.9



(a) (b) (c)

Fig. 5. Sample trajectories with mixed agent policies (robot: blue, cooperative: green, non-cooperative: red). In (a), all agents are cooperative; in (b), two are cooperative and five non-cooperative (const. vel.); in (c), three are cooperative and two non-cooperative (sinusoidal). The GO-MPC agent avoids non-cooperative agents differently than cooperative agents.



(a) DRL [6] (b) DRL-2 (ext. of [6]) (c) GO-MPC

Fig. 6. 8 agents swapping positions. To simulate a multi-robot environment, all agents follow the same policy.

introduced during the training phase, nor have the agents ever experienced other agents with the same policy before. Following the DRL policy (Fig. 6(a)), all agents navigate straight to their goal positions leading to congestion in the center with reactive avoidance. The trajectories from the DRL-2 approach (Fig. 6(b)) are more conservative, due to the limited acceleration available. In contrast, the trajectories generated by our approach (Fig. 6(c)), present a balance between going straight to the goal and avoiding congestion in the center, allowing the agents to reach their goals faster and with smaller distance traveled.

E. Performance Results

This section aggregates performance of the various methods across 200 random scenarios. Performance is quantified by average time to reach the goal position, percentage of episodes that end in failures (either collision or timeout), and the average distance traveled.

The numerical results are summarized in Table III. Our method outperforms each baseline for both mixed and homogeneous scenarios. To evaluate the statistical significance, we performed pairwise MannWhitney U-tests between GO-MPC and each baseline (95% confidence). GO-MPC shows statistically significant performance improvements over the DRL-2 baseline in terms of travel time and distance, and the DRL baseline in term of travel time for six agents and travel distance for ten agents. . For homogeneous scenarios, GO-MPC is more conservative than DRL and MPC baselines resulting in a larger average traveled distance. Nevertheless, GO-MPC is reaches the goals faster than each baseline and is less conservative than DRL-2, as measured by a significantly lower average distance traveled.

Finally, considering higher-order dynamics when training DRL agents (DRL-2) improves the collision avoidance performance. However, it also increases the average time to goal and traveled distance, meaning a more conservative policy that still under-performs GO-MPC in each metric.

V. CONCLUSIONS & FUTURE WORK

This letter introduced a subgoal planning policy for guiding a local optimization planner. We employed DRL methods to learn a subgoal policy accounting for the interaction effects among the agents. Then, we used an MPC to compute locally optimal motion plans respecting the robot dynamics and collision avoidance constraints. Learning a subgoal policy improved the collision avoidance performance among cooperative and non-cooperative agents as well as in multi-robot environments.

Moreover, our approach can reduce travel time and distance in cluttered environments. Future work could account for environment constraints.

REFERENCES

[1] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, Mar. 2016.

[2] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, "Model predictive contouring control for collision avoidance in unstructured dynamic environments," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 4459–4466, Oct. 2019.

[3] Y. F. Chen, S.-Y. Liu, M. Liu, J. Miller, and J. P. How, "Motion planning with diffusion maps," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 1423–1430.

[4] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 285–292.

[5] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 6015–6022.

[6] M. Everett, Y. F. Chen, and J. P. How, "Collision avoidance in pedestrian-rich environments with deep reinforcement learning," *IEEE Access*, vol. 9, pp. 10 357–10 377, 2021.

[7] T. Fan, P. Long, W. Liu, and J. Pan, "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," *Int. J. Robot. Res.*, vol. 39, no. 7, pp. 856–892, 2020.

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

[9] J. Van DenBerg, J.Snape, S. J.Guy, and D.Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 3475–3482.

[10] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Phys. Rev. E*, vol. 51, no. 5, 1995, Art. no. 4282.

[11] C. I. Mavrogiannis and R. A. Knepper, "Multi-agent path topology in support of socially competent navigation planning," *Int. J. Robot. Res.*, vol. 38, no. 2/3, pp. 338–356, 2019.

[12] C. I. Mavrogiannis, W. B. Thomason, and R. A. Knepper, "Social momentum: A framework for legible navigation in dynamic multi-agent environments," in *Proc. ACM/IEEE Int. Conf. Hum.-Robot Interact.*, 2018, pp. 361–369.

[13] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: Statistical models and experimental studies of human-robot cooperation," *Int. J. Robot. Res.*, vol. 34, no. 3, pp. 335–356, 2015.

[14] B. Kim and J. Pineau, "Socially adaptive path planning in human environments using inverse reinforcement learning," *Int. J. Social Robot.*, vol. 8, no. 1, pp. 51–66, 2016.

[15] A. Vemula, K. Muelling, and J. Oh, "Modeling cooperative navigation in dense human crowds," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 1685–1692.

[16] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1343–1350.

[17] L. Tai, J. Zhang, M. Liu, and W. Burgard, "Socially compliant navigation through raw depth inputs with generative adversarial imitation learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 1111–1117.

[18] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 3, pp. 269–296, 2020.

[19] N. Mansard, A. DelPrete, M. Geisert, S. Tonneau, and O. Stasse, "Using a memory of motion to efficiently warm-start a nonlinear predictive controller," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 2986–2993.

[20] G. Bellegarda and K. Byl, "An online training method for augmenting MPC with deep reinforcement learning," in *2020 IEEE/RSJ Int. Conf. Int. Robots Syst. (IROS)*, 2020, pp. 5453–5459, doi: 10.1109/IROS45743.2020.9341021.

[21] M. Zanon and S. Gros, "Safe reinforcement learninge using robust MPC," *IEEE Trans. Autom. Control*, 2020, pp. 1–1, doi: 10.1109/TAC.2020.3024161.

[22] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE Int. Conf. Robotics Automat. (ICRA)*, 2018, pp. 7559–7566, doi: 10.1109/ICRA.2018.8463189.

[23] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value function approximation and model predictive control," in *Proc. IEEE Symp. Adaptive Dyn. Program. Reinforcement Learn.*, 2013, pp. 100–107.

[24] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, "Plan online, learn offline: Efficient learning and exploration via model-based control," in *Int. Conf. Learn. Repres.*, 2019. [Online]. Available: https://openreview.net/forum?id=Byey7n05FQ

[25] F. Farshidian, D. Hoeller, and M. Hutter, "Deep value model predictive control," in *Proc. Conf. Robot Learn.*, 2020, pp. 990–1004.

[26] N. Karnchanachari, M. I. Valls, D. Hoeller, and M. Hutter, "Practical reinforcement learning of stabilizing economic MPC," in *18th Euro. Control Conf. (ECC)*, 2019, pp. 2258–2263, doi: 10.23919/ECC.2019.8795816.

[27] D. Ernst, M. Glavic, F. Capitanescu, and L. Wehenkel, "Reinforcement learning versus model predictive control: a comparison on a power system problem," *IEEE Trans. Syst., Man, Cybern., Part B*, vol. 39, no. 2, pp. 517–529, Apr. 2009.

[28] S. Levine and V. Koltun, "Guided policy search," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1–9.

[29] S. Levine and V. Koltun, "Variational policy search via trajectory optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 207–215.

[30] I. Mordatch and E. Todorov, "Combining the benefits of function approximation and trajectory optimization," in *Proc. Robot.: Sci. Syst.*, vol. 4, 2014.

[31] Z.-W. Hong, J. Pajarinen, and J. Peters, "Model-based lookahead reinforcement learning," 2019, *arXiv:1908.06012*.

[32] T. Wang and J. Ba, "Exploring model-based planning with policy networks," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=H1exf64KwH

[33] C. Greatwood and A. G. Richards, "Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control," *Auton. Robots*, vol. 43, no. 7, pp. 1681–1693, 2019.

[34] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[35] T. Fan, P. Long, W. Liu, and J. Pan, "Fully distributed multi-robot collision avoidance via deep reinforcement learning for safe and efficient navigation in complex scenarios," *Int. J. Robot. Res.*, vol. 39, no. 7, pp. 856–892, 2020.

[36] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2008, pp. 1928–1935.

[37] S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan, and R. Munos, "Recurrent experience replay in distributed reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=r1lyTjAqYX

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

[39] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. Int. Conf. Learn. Repres. (ICLR)*, 2016.

[40] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 41–48.

[41] A. Hill *et al.*, "Stable baselines," *GitHub repository*, 2018.

[42] A. Domahidi and J. Jerez, "FORCES professional," *Embotech AG* (http://embotech.com/FORCES-Pro), 2014–2019.