

MAMPS: Safe Multi-Agent Reinforcement Learning via Model Predictive Shielding

Wenbo Zhang¹ and Osbert Bastani²

Abstract—Reinforcement learning is a promising approach to learning control policies for performing complex multi-agent robotics tasks. However, a policy learned in simulation often fails to guarantee even simple safety properties such as obstacle avoidance. To ensure safety, we propose multi-agent model predictive shielding (MAMPS), an algorithm that provably guarantees safety for an arbitrary learned policy. In particular, it operates by using the learned policy as often as possible, but instead uses a backup policy in cases where it cannot guarantee the safety of the learned policy. Using a multi-agent simulation environment, we show how MAMPS can achieve good performance while ensuring safety.

I. INTRODUCTION

Reinforcement learning [1], [2] has been shown to be a promising technique for learning control policies for complex robotics tasks ranging from autonomous vehicles [3] to home service robots [4], or to “compress” an expensive model predictive controller (MPC) into a much faster neural network policy [5]. A major challenge in using reinforcement learning is safety [6], [7], [8], [9], [10], [11], [12], [13]—control policies learned using reinforcement learning typically do not provide any safety guarantees, even when the safety property is explicitly considered by the learning algorithm.

As a consequence, there has been much interest in algorithms that provide safety guarantees for a learned control policy $\hat{\pi}$. We are interested in the setting where $\hat{\pi}$ is learned in simulation, and we want to ensure safety after it is deployed on a robot (assuming that our model of the dynamics is correct). One approach is to formally prove that $\hat{\pi}$ is safe [14], [15], [16]. An alternative approach, called *shielding*, is to synthesize a backup controller π_{backup} that is guaranteed to be safe on some subset of states, which we call *recoverable states* [6], [7], [12], [17], [18]. Then, the *shield policy* π_{shield} uses π_{backup} whenever $\hat{\pi}$ would bring the robot to an irrecoverable state on the next step; otherwise it uses $\hat{\pi}$. A key challenge with existing approaches is that they rely on verifying either $\hat{\pi}$ or π_{backup} , which typically does not scale to high-dimensional systems. For systems with obstacles in the environment or other agents that must be encoded in the state, verifying safety quickly becomes intractable.

A promising alternative is *model predictive shielding* (MPS) [17], [18], [19], which performs shielding on-the-fly instead of ahead-of-time. The intuition is that checking whether a single state is recoverable (i.e., the next one)

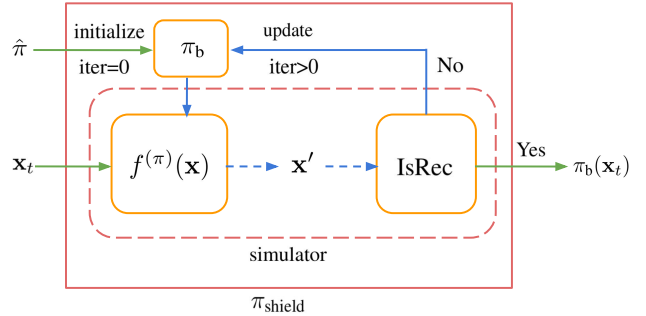


Fig. 1. **Overview of multi-agent model predictive shielding (MAMPS).** On step t , the current state of the multi-agent system is \mathbf{x}_t . The red solid box shows the entire MAMPS control policy is π_{shield} module. There are three basic components in MAMPS: the “current policy” π_b , the closed-loop dynamics $f^{(\pi)}(\mathbf{x})$, and the subroutine IsRecoverable (shorted to IsRec). The current policy is a combination of learned policies and recovery policies for different agents. First, MAMPS initializes π_b to $\hat{\pi}$ for each agent. Then, it iteratively determines whether using π_b transitions the system to a recoverable state \mathbf{x}' . If not, then it switches agents that were unsafe to π_{rec} (the blue line). Once it has found b such that $f^{(\pi_b)}(\mathbf{x})$ is recoverable, it can safely output action $\pi_b(\mathbf{x}_t)$ (the green line). The red dashed box shows the internal simulation used by MAMPS to check recoverability; note that all dynamics applications in this box are according to a simulation run by the MAMPS algorithm, not according to the true dynamics. The true dynamics are only applied once π_{shield} returns an action $\pi_b(\mathbf{x}_t)$.

is much more efficient, even on-the-fly, than exhaustively checking recoverability for all states ahead-of-time. In particular, to check whether a given state is recoverable, MPS simply simulates the closed-loop dynamics (assumed to be deterministic) with π_{backup} and checks whether it is safe. While we focus on deterministic dynamics, there has been recent work extending MPS to stochastic dynamics [19].

In this paper, we study the safety problem for multi-agent systems [20], [21]. In this setting, approaches to safety have been proposed based on restricting the velocities of the agents [22], [23], [24]; however, these approaches only apply to systems that are holonomic [23], [24] or approximately holonomic [22]. Instead, we propose an approach based on MPS, which applies to general robot dynamics. We can in principle treat the multi-agent system as a high-dimensional single-agent system, and then apply MPS to ensure safety. However, this approach can achieve very suboptimal results since even if a single robot is about to be irrecoverable, then the shield uses π_{backup} for every single robot in the system.

We propose an approach, called multi-agent model predictive shielding (MAMPS), summarized in Fig. 1, which avoids this problem by incrementally switching each agent from using $\hat{\pi}$ to using π_{backup} . The iterative process is needed

¹Wenbo Zhang is with the GRASP Lab, University of Pennsylvania, USA zwenzbo@seas.upenn.edu

²Osbert Bastani is with the Department of Computer and Information Science, University of Pennsylvania, USA obastani@seas.upenn.edu

since switching one agent to using π_{backup} can cause other agents to subsequently become unsafe. The general workflow is shown in Fig. 1. We prove that this modification preserves the key safety guarantees provided by shielding—i.e., if the system starts in a recoverable state, then using π_{shield} keeps the system safe over an infinite horizon.

Contributions. Our key contributions are: (i) the multi-agent model predictive shielding (MAMPS) algorithm for ensuring safety of a learned policy (Section II & III), (ii) a theoretical guarantee regarding the safety of this algorithm (Section III), and (iii) an experimental evaluation demonstrating how this algorithm outperforms the standard MPS algorithm in the multi-agent setting (Section IV).

II. PROBLEM FORMULATION

In this section, we formulate the problem of shielding a learned policy to ensure safety.

Dynamical system. We consider a deterministic, discrete time dynamical system with continuous states $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$, continuous action $\mathbf{u}^i \in \mathcal{U}^i \subseteq \mathbb{R}^{n_u}$ (for each agent $i \in [N] = \{1, 2, \dots, N\}$), dynamics $f : \mathcal{X} \times \mathcal{U}^1 \times \dots \times \mathcal{U}^N \rightarrow \mathcal{X}$, and probability distribution d_0 over initial states. We assume that $\mathbf{x} \in \mathcal{X}$ encodes the state of all the agents as well as goals and obstacles in the environment.

As a running example, consider a system of non-holonomic robots with acceleration control and steering control—i.e., each robot $i \in [N]$ has state $\mathbf{x}^i = (x^i, y^i, v^i, \theta^i)$ and action $\mathbf{u}^i = (a^i, \delta^i)$, where (x^i, y^i) is the position of the robot in \mathbb{R}^2 , v^i is its velocity, θ^i is its heading, a^i is its acceleration, and δ^i is its steering angle. In addition, we consider N goals with positions $\mathbf{g}^i = (g^i, h^i) \in \mathbb{R}^2$ for each $i \in [N]$, as well as M obstacles with positions $\mathbf{z}^i = (z^i, w^i) \in \mathbb{R}^2$ for each $i \in [M]$. The goal is for agent i to reach goal i . Together, the multi-agent state is $\mathbf{x} = (\mathbf{x}^1, \dots, \mathbf{x}^N, \mathbf{g}^1, \dots, \mathbf{g}^N, \mathbf{z}^1, \dots, \mathbf{z}^M)$.

Control policy. Given a vector of control policies $\pi = (\pi^1, \dots, \pi^N)$, where $\pi^i : \mathcal{X} \rightarrow \mathcal{U}^i$ for each $i \in [N]$, we use $f^{(\pi)}(x, u) = f(x, \pi^1(x), \dots, \pi^N(x))$ to denote the closed-loop dynamics. An infinite-horizon trajectory generated using π from initial state $\mathbf{x}_0 \in \mathcal{X}$ is the sequence $\zeta = (\mathbf{x}_0, \mathbf{x}_1, \dots)$, where $\mathbf{x}_{t+1} = f^{(\pi)}(\mathbf{x}_t)$. Similarly, given a finite horizon $T \in \mathbb{N}$, a finite-horizon trajectory generated using π from initial state $\mathbf{x}_0 \in \mathcal{X}$ is the sequence $\zeta = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{T-1})$.

Safe states. We assume given sets $\mathcal{X}_{\text{safe}} \subseteq \mathcal{X}$ indicating that agent $i \in [N]$ is safe. Then, the system as a whole is safe if every agent is safe—i.e.,

$$\mathcal{X}_{\text{safe}} = \bigcap_{i=1}^N \mathcal{X}_{\text{safe}}^i.$$

The goal is to ensure that the system never transitions to an unsafe state $\mathbf{x} \notin \mathcal{X}_{\text{safe}}$. Given a trajectory $\zeta = (\mathbf{x}_0, \mathbf{x}_1, \dots)$ we say ζ is safe if $\mathbf{x}_t \in \mathcal{X}_{\text{safe}}$ for all $t \geq 0$.

In our example, an agent $i \in [N]$ is safe if it has not collided with an obstacle or any other robots. In particular,

we have $\mathbf{x} \in \mathcal{X}_{\text{safe}}^i$ if it satisfies the following constraints:

$$\begin{aligned} d(\mathbf{x}^i, \mathbf{x}^j) &\geq 2r_{\text{robot}} + m \quad (\forall j \in [N], i \neq j) \\ d(\mathbf{x}^i, \mathbf{z}^j) &\geq r_{\text{robot}} + r_{\text{obstacle}} + m \quad (\forall j \in [M]), \end{aligned}$$

where r_{robot} is the robot radius, r_{obstacle} is the obstacle radius, m is a safety margin, and d is the Euclidean distance. As a consequence, the overall system is safe if the above constraints are satisfied for all $i \in [N]$.

Stable controller and stable states. To ensure safety for an infinite horizon, we assume that we are given a control policy π_{stable} and a subset $\mathcal{X}_{\text{stable}} \subseteq \mathcal{X}_{\text{safe}}$ of states such that using π_{stable} guarantees safety indefinitely. In particular, we assume given sets $\mathcal{X}_{\text{stable}}^i \subseteq \mathcal{X}_{\text{safe}}^i$ indicating that agent $i \in [N]$ is stable. Then, the system as a whole is stable if every agent is stable—i.e.,

$$\mathcal{X}_{\text{stable}} = \bigcap_{i=1}^N \mathcal{X}_{\text{stable}}^i.$$

Our key assumption is that for any $\mathbf{x} \in \mathcal{X}_{\text{stable}}$, the trajectory ζ generated using π_{stable} from initial state $\mathbf{x}_0 = \mathbf{x}$ is safe.

In our running example, we have $\mathbf{x} \in \mathcal{X}_{\text{stable}}^i$ if $v^i = 0$ and $\mathbf{x} \in \mathcal{X}_{\text{safe}}$ —i.e., agent i is at rest and the overall system is safe. Furthermore, the backup control policy is $\pi^i(x) = (0, 0)$ —i.e., the steering angle and acceleration are both zero. As a consequence, we have $\mathbf{x} \in \mathcal{X}_{\text{stable}}$ if $v^i = 0$ for every agent $i \in [N]$ and furthermore $\mathbf{x} \in \mathcal{X}_{\text{safe}}$. In other words, in a stable state, all agents are at rest, and the backup control policy keeps them at rest.

Reward. We consider a reward function $r : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$ that we seek to maximize. Then, given a finite time horizon T , initial states \mathcal{X} and an initial state distribution d_0 , our goal is to find a policy $\hat{\pi}$ that maximizes

$$\mathbb{E}_{\mathbf{x}_0 \sim d_0} \left[\sum_{t=0}^{T-1} r(\mathbf{x}_t, \mathbf{u}_t) \right],$$

where $\mathbf{x}_{t+1} = f^{(\pi)}(\mathbf{x}_t, \mathbf{u}_t)$ and $\mathbf{u}_t = \pi(\mathbf{x}_t)$. In our example, the reward might be to minimize the distance of each robot to a goal state:

$$r(\mathbf{x}, \mathbf{u}) = - \sum_{i=1}^N d(\mathbf{x}^i, \mathbf{g}^i),$$

where $\mathbf{g}^i \in \mathbb{R}^2$ is a goal state.¹ The rewards can also encode soft constraints that encourage the robot to remain in $\mathcal{X}_{\text{safe}}$.

Learned policy. We assume given a vector of policies $\hat{\pi} = (\hat{\pi}^1, \dots, \hat{\pi}^N)$, where each policy can be arbitrary. For example, $\hat{\pi}$ can be learned using the multi-agent deep deterministic policy gradient algorithm (MADDPG) [25], [26].

Shielding problem. A policy π is safe if for any trajectory $\zeta = (\mathbf{x}_0, \mathbf{x}_1, \dots)$ starting from $\mathbf{x}_0 \in \mathcal{X}_{\text{stable}}$, ζ is safe.² Then,

¹For our experiments, we use a more complex reward function; see Section IV for details.

²We restrict to trajectories starting from stable states; it is impossible in general to guarantee safety for trajectories starting from unstable states without leveraging specific aspects of the dynamical system in question.

Algorithm 1 Compute the MAMPS policy for state \mathbf{x} .

function MAMPS(\mathbf{x}):

```

1:  $b \leftarrow (\text{true}, \dots, \text{true})$ 
2: while true do
3:    $\eta_{\text{rec}} \leftarrow \text{IsRecoverable}(f^{(\pi_b)}(\mathbf{x}))$ 
4:   if  $\eta_{\text{rec}} = (\text{true}, \dots, \text{true})$  then
5:     return  $f^{(\pi_b)}(\mathbf{x})$ 
6:   end if
7:   for  $i \in [N]$  do
8:      $b'_i \leftarrow b_i \wedge \eta_{\text{rec}}^i$ 
9:   end for
10:  if  $b' = b$  then
11:     $b \leftarrow (\text{false}, \dots, \text{false})$ 
12:  else
13:     $b \leftarrow b'$ 
14:  end if
15: end while

```

Algorithm 2 Check if each agent is recoverable at state \mathbf{x} .

function IsRecoverable(\mathbf{x}):

```

1:  $\eta_{\text{safe}} \leftarrow (\text{true}, \dots, \text{true})$ 
2:  $\eta_{\text{rec}} \leftarrow (\text{false}, \dots, \text{false})$ 
3: for  $t \in \{0, 1, \dots, T_{\text{max}} - 1\}$  do
4:   for  $i \in [N]$  do
5:      $\eta_{\text{safe}}^i \leftarrow \eta_{\text{safe}}^i \wedge (\mathbf{x} \in \mathcal{X}_{\text{safe}}^i)$ 
6:      $\eta_{\text{rec}}^i \leftarrow \eta_{\text{rec}}^i \vee (\eta_{\text{safe}}^i \wedge (\mathbf{x} \in \mathcal{X}_{\text{stable}}^i))$ 
7:   end for
8:    $\mathbf{x} \leftarrow f^{(\pi_{\text{backup}})}(\mathbf{x})$ 
9: end for
10: return  $\eta_{\text{rec}}$ 

```

our goal is to construct a vector of control policies π_{shield} that is safe. To try and maximize reward, our construction of π_{shield} leverages $\hat{\pi}$ —in particular, it tries to maximize the number of states $\mathbf{x} \in \mathcal{X}$ for which $\pi_{\text{shield}}(\mathbf{x}) = \hat{\pi}(\mathbf{x})$.

III. MULTI-AGENT MODEL PREDICTIVE SHIELDING

We propose an extension of the model-predictive shielding (MPS) algorithm to the setting of multi-agent systems, which we call multi-agent model predictive shielding (MAMPS).

Background on MPS. Recall that π_{stable} can guarantee safety for an infinite horizon starting from any stable state $\mathbf{x} \in \mathcal{X}_{\text{stable}}$. Thus, we can use the shielding approach using π_{stable} . In particular, suppose we start at a state $\mathbf{x} \in \mathcal{X}_{\text{stable}}$. To decide whether to use $\hat{\pi}$ or π_{stable} , we check if $\mathbf{x}' = f^{(\hat{\pi})}(\mathbf{x}) \in \mathcal{X}_{\text{stable}}$. If so, then it is safe to use $\hat{\pi}(\mathbf{x})$, since π_{stable} is guaranteed to be safe starting from \mathbf{x}' , so we can continue to guarantee safety. Otherwise, we use $\pi_{\text{stable}}(\mathbf{x})$ (which is guaranteed to be safe).

Constructing a stable controller π_{stable} along with stable states $\mathcal{X}_{\text{stable}}$ that is large can be a challenging problem. In our example, the proposed set of stable states (i.e., states where all robots are at rest) has zero volume in the state space. Thus, using shielding with π_{stable} as the backup policy will result in poor performance; indeed, in our example, the robots will

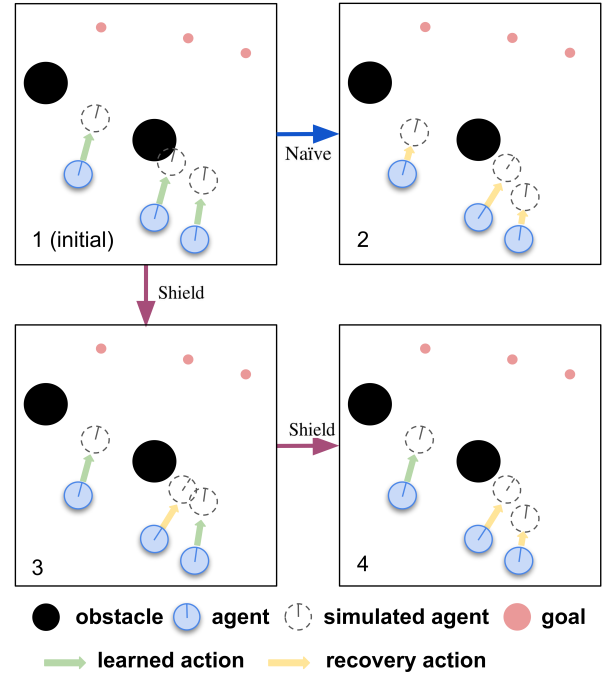


Fig. 2. **Comparison of MAMPS to the naïve approach.** Image 1 is the initial state. In this state, the middle agent will collide with the obstacle if it uses the learned policy $\hat{\pi}$. The naïve approach treats the system as a single-agent system and uses MPS [18]. In this approach, all agents will switch to using the recovery policy π_{rec} (Image 2). In contrast, when MAMPS iteratively checks whether successively more conservative configurations can transition the system to a recoverable state. In this example, MAMPS first switches the middle agent to using the recovery policy (Image 3). However, this change causes a new problem—the middle agent will now collide with the right-most agent. Thus, in the second iteration, MAMPS switches the right-most agent to using the recovery policy (Image 4). This choice transitions the system to a recoverable state, so MAMPS returns these actions. In this example, MAMPS allows the left-most agent to use the learned policy, whereas the naïve approach switches it to the recovery policy. Thus, MAMPS can achieve a significantly higher reward.

never be able to move. The idea behind MPS is to expand the set of states for which we can guarantee safety for an infinite horizon by using a *recovery policy* π_{rec} to try and transition the system to a state $\mathbf{x}' \in \mathcal{X}_{\text{stable}}$. More precisely, given a state \mathbf{x} , suppose that there exists T such that the finite-horizon trajectory $\zeta = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{T-1})$ generated using π_{rec} from $\mathbf{x}_0 = \mathbf{x}$ is safe and reaches $\mathcal{X}_{\text{stable}}$ (i.e., $\mathbf{x}_{T-1} \in \mathcal{X}_{\text{stable}}$). Then, we can guarantee safety starting from \mathbf{x} by first using π_{rec} for T steps, and then using π_{stable} afterwards; we call such a state \mathbf{x} *recoverable*, since it can be recovered to a stable state using π_{rec} . Thus, using this combination of π_{rec} and π_{stable} as the backup policy substantially expands the set of states where we can use $\hat{\pi}$.

We can in principle apply this approach to multi-agent systems (with centralized control), where we treat the system as a single high-dimensional system. However, this approach can work poorly—if even a single agent needs to switch to the recovery policy, then every agent must be switched to using the recovery policy. In contrast, our proposed algorithm, multi-agent MPS (MAMPS), considers different choices of policy for different robots. For example, Fig. 2

shows an instance where the naïve approach of treating the system as a single-agent system and using MPS causes all the agents to switch to the recovery policy, whereas our MAMPS algorithm avoids this failure.

Recovery policy. As with MPS, our approach uses a recovery policy $\pi_{\text{rec}} = (\pi_{\text{rec}}^1, \dots, \pi_{\text{rec}}^N)$ that tries to transition the system from any state $\mathbf{x} \in \mathcal{X}$ to a stable state $\mathbf{x}' \in \mathcal{X}_{\text{stable}}$. This policy can be manually specified or learned using reinforcement learning methods. Like $\hat{\pi}$, our algorithm works with any choice of recovery policies. In our example, we can use the policy π_{rec}^i that decelerates the agent as fast as possible until it reaches a stop. Once all robots come to a stop (assuming no robot hits an obstacle or another robot), then the system is in a stable state.

Backup policy. The backup policy π_{backup} is a combination of the recovery policy and the stable policy. In particular, for agent $i \in [N]$, it uses π_{stable}^i if agent i is stable (i.e., $\mathbf{x} \in \mathcal{X}_{\text{stable}}^i$), and uses π_{rec}^i otherwise:

$$\pi_{\text{backup}}^i(\mathbf{x}) = \begin{cases} \pi_{\text{stable}}^i(\mathbf{x}) & \text{if } \mathbf{x} \in \mathcal{X}_{\text{stable}}^i \\ \pi_{\text{rec}}^i(\mathbf{x}) & \text{otherwise,} \end{cases}$$

for each $i \in [N]$.

Recoverable states. Specific to the multi-agent setting, we can decompose the notion of recoverability and define recoverability for an individual agent. In particular, a state $\mathbf{x} \in \mathcal{X}$ is recoverable for agent $i \in [N]$, denoted $\mathbf{x} \in \mathcal{X}_{\text{rec}}^i$, if there exists T such that the finite-horizon trajectory $\zeta = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{T-1})$ generated from $\mathbf{x}_0 = \mathbf{x}$ using π_{backup} is (i) safe for agent i (i.e., $\mathbf{x}_t \in \mathcal{X}_{\text{safe}}^i$ for all $t \in \{0, 1, \dots, T-1\}$), and (ii) reaches a stable state for agent i (i.e., $\mathbf{x}_{T-1} \in \mathcal{X}_{\text{stable}}^i$). Then, the set of recoverable states is

$$\mathcal{X}_{\text{rec}} = \bigcap_{i=1}^N \mathcal{X}_{\text{rec}}^i.$$

We say a state $\mathbf{x} \in \mathcal{X}_{\text{rec}}$ is *recoverable*. It is easy to see that if \mathbf{x} is recoverable, then there exists $T \leq T_{\text{max}}$ such that the finite-horizon trajectory $\zeta = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{T-1})$ generated from $\mathbf{x}_0 = \mathbf{x}$ using π_{backup} is (i) safe (i.e., $\mathbf{x}_t \in \mathcal{X}_{\text{safe}}$ for all $t \in \{0, 1, \dots, T-1\}$), and (ii) reaches a stable state (i.e., $\mathbf{x}_{T-1} \in \mathcal{X}_{\text{stable}}$). In this definition, $T_{\text{max}} \in \mathbb{N}$ is a hyperparameter that bounds the length of the trajectory we need to check if recoverability holds, making it feasible to check recoverability in simulation.

In particular, we can check whether a state is recoverable (either for an individual agent or for overall) by simulating π_{backup} . This check concludes that \mathbf{x} is recoverable if and only if \mathbf{x} is actually recoverable, then \mathbf{x} is guaranteed to be recoverable. Algorithm 2 performs this check, and returns a vector $\rho = (\rho_1, \dots, \rho_N) \in \{0, 1\}^N$, where $\rho_i \in \{0, 1\}$ indicates whether $\mathbf{x} \in \mathcal{X}_{\text{rec}}^i$ ($\rho_i = 1$) or not ($\rho_i = 0$).

Multi-agent model predictive shielding. Our algorithm, multi-agent model predictive shielding (MAMPS),³ chooses whether to use the learned policy $\hat{\pi}^i$ or the recovery policy

π_{rec}^i for each agent $i \in [N]$. In contrast to the MPS approach described above, which either uses $\hat{\pi}^i$ for every agent i or uses π_{rec}^i for every agent i , MAMPS considers different choices of learned policy or recovery policy for different agents. We represent the possible choices by a *configuration* $b = (b_1, \dots, b_N) \in \mathcal{B} = \{0, 1\}^N$. In particular, $b_i \in \{0, 1\}$ indicates whether to use $\hat{\pi}^i$ ($b_i = 1$) or π_{rec}^i ($b_i = 0$). For any $b \in \mathcal{B}$, we use π_b to denote the corresponding combination of policies for each agent—i.e.,

$$\pi_b^i(\mathbf{x}) = \begin{cases} \hat{\pi}^i(\mathbf{x}) & \text{if } b_i = 1 \\ \pi_{\text{backup}}^i(\mathbf{x}) & \text{otherwise.} \end{cases}$$

Our key insight is that we can use π_b for *any* configuration $b \in \mathcal{B}$ as long as $f^{(\pi_b)}(\mathbf{x}) \in \mathcal{X}_{\text{rec}}$. Since we can check recoverability of any state \mathbf{x} using simulation, we can simply enumerate over all configurations $b \in \mathcal{B}$ to find one that satisfies this condition. If there are multiple choices of b , then we want to choose the one that maximizes $\|b\|_1$ —i.e., the one that maximizes the number of agents $\hat{\pi}$ using their learned control policy. Thus, we want to compute

$$b^* = \arg \max_{b \in \mathcal{B}} \|b\|_1 \cdot \mathbb{I}[f^{(\pi_b)}(\mathbf{x}) \in \mathcal{X}_{\text{rec}}], \quad (1)$$

where \mathbb{I} is the indicator function (taking values in $\{0, 1\}$).

For systems with many agents, iterating over all combinations $b \in \mathcal{B}$ can become very expensive, since $|\mathcal{B}| = 2^N$ is exponential in the number of agents. Especially for systems with limited computational resources, computing b^* can be intractable. Thus, MAMPS instead solves (1) approximately using a greedy iterative search strategy. In particular, MAMPS starts off by considering the best possible candidate configuration $b = (1, 1, \dots, 1)$ —i.e., every agent uses the learned policy. Then, it checks whether $f^{(\pi_b)}(\mathbf{x}) \in \mathcal{X}_{\text{rec}}$. If so, then we can use π_b . Otherwise, there are agents for which \mathbf{x} is irrecoverable—i.e.,

$$\mathcal{I} = \{i \in [N] \mid \mathbf{x}^i \notin \mathcal{X}_{\text{rec}}^i\} \neq \emptyset.$$

For agents $i \in \mathcal{I}$, MAMPS switches to using the backup policy—i.e., $b_i = 0$.

Note that switching an agent from $b_i = 1$ to $b_i = 0$ may cause a different agent that was previously recoverable to become irrecoverable. As an example, the illustration of the MAMPS policy in Fig. 2 shows a case where switching one agent causes another to become irrecoverable. Thus, we have to again check for additional agents that have become irrecoverable. We iteratively perform this process until we find a configuration b such that $f^{(\pi_b)}(\mathbf{x}) \in \mathcal{X}_{\text{rec}}$. Thus, we know that $f^{(\pi_{\text{backup}})}(\mathbf{x}) \in \mathcal{X}_{\text{rec}}$, so we can safely use π_b .

The full MAMPS algorithm is shown in Algorithm 1. For convenience, when representing b , this algorithm uses true instead of 1, and false instead of 0. One subtlety is that while $\|b\|_1$ is monotonically decreasing in this algorithm, but it nevertheless get “stuck” at some point b that is not guaranteed to be safe, but there are also no agents switching from $b_i = 1$ to $b_i = 0$. Thus, the algorithm includes a check to see whether b converges. In this case, it sets $b = (0, \dots, 0)$,

³In this name, “model predictive” refers to the fact that we are simulating trajectories according to the dynamics model to determine whether to use the learned policy or the backup policy.

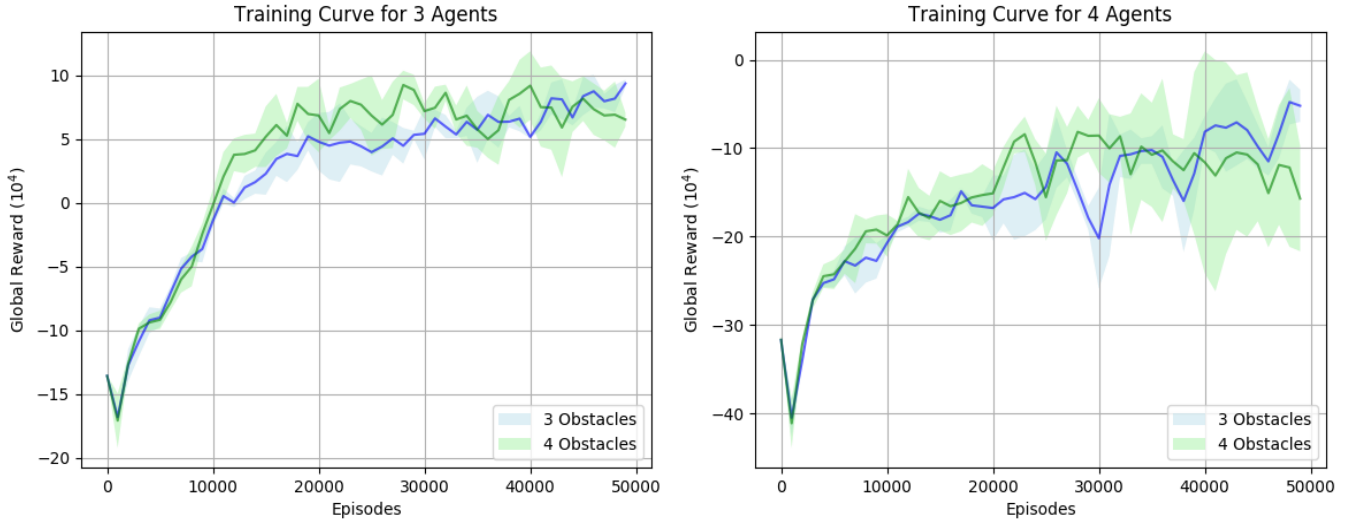


Fig. 3. **Training curves for the MADDPG algorithm.** We show the cases of 3 agents (left) and 4 agents (right).

so $\pi_b = \pi_{\text{backup}}$ is guaranteed to be safe. We have the following guarantee (proved in Appendix V-A):

Theorem 1: The MAMPS policy π_{MAMPS} is safe.

IV. EXPERIMENTS

In our experiments, we aim to answer the following research questions:

- How does MAMPS compare to using the learned policy without any shield (in terms of reward and safety)?
- How does MAMPS compare to the naïve shield (i.e., treat the system as a single-agent system and use MPS)?
- How does the performance of MAMPS vary with respect to the number of agents or obstacles?

All experiments are performed on a server with an Intel Xeon Gold 6148 CPU and Nvidia RTX 2080 GPUs.

A. Setup

We perform our experiments using the multi-agent particle environment [26]. This environment consists of a multi-agent system in which a set of N agents is trying to reach a set of N goals. In our setup, each agent is assigned a specific goal that they are trying to reach. There is also a collection of M obstacles that the agents must avoid; in addition, the agents must avoid colliding with one another.

The learned policy $\hat{\pi}$ is trained using multi-agent deep deterministic policy gradients (MADDPG) [25], which uses decentralized actors and a centralized critic. Our reward function is

$$r(\mathbf{x}, \mathbf{u}) = r_{\text{goal}}(\mathbf{x}, \mathbf{u}) + \lambda \cdot r_{\text{bonus}}(\mathbf{x}, \mathbf{u}) + \lambda' \cdot r_{\text{safe}}(\mathbf{x}, \mathbf{u}),$$

where the reward for approaching goals is

$$r_{\text{goal}}(\mathbf{x}, \mathbf{u}) = - \sum_{i=1}^N d(\mathbf{x}^i, \mathbf{g}^i),$$

the bonus reward for arriving at a goal is ⁴

$$r_{\text{bonus}}(\mathbf{x}, \mathbf{u}) = \sum_{i=1}^N \mathbb{I}(d(\mathbf{x}^i, \mathbf{g}^i) \leq \epsilon),$$

the penalty for collisions is

$$r_{\text{safe}} = - \sum_{i=1}^N \sum_{j=i+1}^N \mathbb{I}(d(\mathbf{x}^i, \mathbf{x}^j) < 2 \cdot r_{\text{robot}} + m) \\ - \sum_{i=1}^N \sum_{j=1}^M \mathbb{I}(d(\mathbf{x}^i, \mathbf{z}^j) < r_{\text{robot}} + r_{\text{obstacle}} + m),$$

$\lambda, \lambda', \epsilon \in \mathbb{R}_{>0}$ are hyperparameters, and $\mathbb{I}(C) \in \{0, 1\}$ is the indicator function, which indicates whether condition C holds. Parameter choices are shown in Appendix V-B.

For the recovery policy π_{rec} , we use the policy that applies the maximum possible acceleration to decrease the robot's velocity (i.e., apply the brakes as hard as possible). Note that if the robot uses recovery policy while it is at rest ($v = 0$), then it will stay in the same state. Better choices may be possible, but we find that this simple choice is very effective.

B. Results

In Fig. 3, we show learning curves for the MADDPG algorithm. As can be seen, it successfully learns how to control each system. In Fig. 4, we compare our approach, MADDPG with MAMPS (MADDPG+MAMPS), with two baselines: (i) using just the learned policy (MADDPG), and (ii) using MADDPG with the naïve shield (MADDPG+Naïve). For completeness, we have also shown the naïve algorithm in Appendix V-C. Our results are averaged over 500 episodes. We compare two metrics: probability of safety (left) and probability of reaching the goal (right). For safety, we measure the fraction of the agent/episodes pairs for which

⁴We find that MADDPG does not perform well without this term.

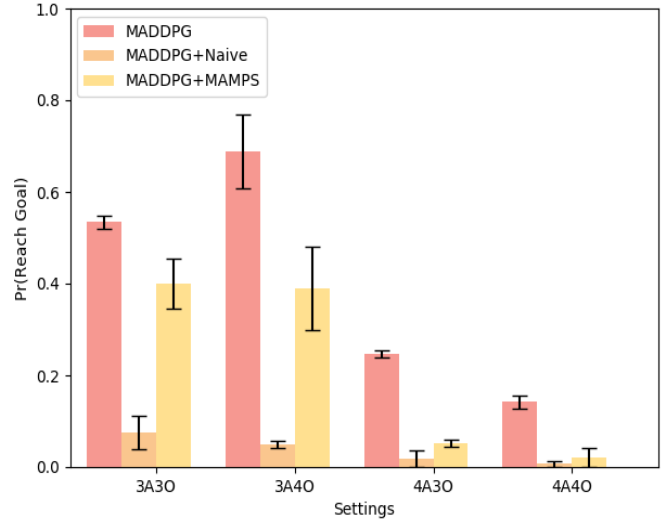
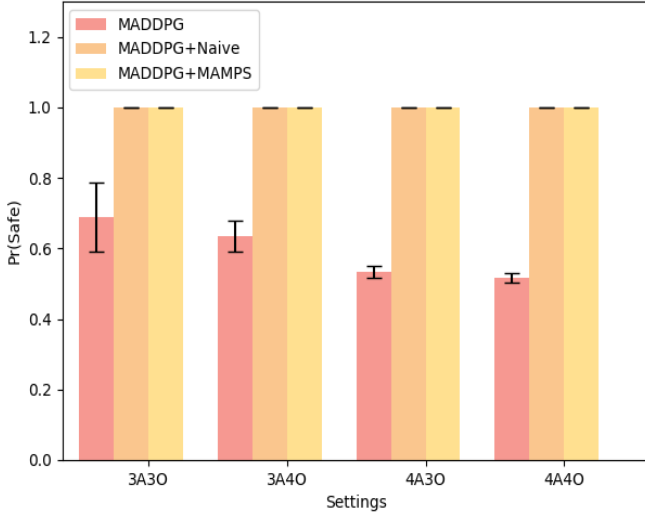


Fig. 4. **Probability of safety (left) and probability of reaching the goal (right) for multi-agent systems.** Here, “3A3O”, “3A4O”, “4A3O”, and “4A4O” represent “3 agents and 3 obstacles”, “3 agents and 4 obstacles”, “4 agents and 3 obstacles”, and “4 agents and 4 obstacles”, respectively.

safety is ensured for the entire duration of the episode—i.e.,

$$\Pr(\text{safe}) \approx \frac{1}{N \cdot K} \sum_{i=1}^K \sum_{j=1}^N \delta_{ij}^{\text{safe}},$$

where $K = 500$ is the number of episodes, $\delta_{ij}^{\text{safe}} \in \{0, 1\}$ indicates whether agent j is safe during the entirety of episode i (i.e., $\mathbf{x}_t^i \in \mathcal{X}_{\text{safe}}^i$). For reaching the goal, we count the fraction of agent/episode pairs for which the agent reaches the goal at any point during the episode—i.e.,

$$\Pr(\text{reach goal}) \approx \frac{1}{N \cdot K} \sum_{i=1}^K \sum_{j=1}^N \delta_{ij}^{\text{goal}},$$

where $\delta_{ij}^{\text{goal}} \in \{0, 1\}$ indicates whether agent j reaches its goal at any point in episode i .

Ensuring safety. Note that MADDPG alone performs quite poorly in terms of safety—it can guarantee safety less than 50% of the time when there are 4 agents and 4 obstacles. The poor performance in terms of safety happens even though MADDPG includes a penalty for collisions. The difficulty is that there are a huge number of possible configurations of the state space, and it is not possible to ensure that MADDPG trains the neural network to account for all of them. As expected, for MADDPG, the probability of safety decreases as there are more agents or more obstacles. Finally, both MADDPG+MAMPS and MADDPG+Naive guarantee the safety of multi-agent for all agents in all episodes.

Reaching goals. As expected, MADDPG+MAMPS achieves its goals less frequently than MADDPG since MADDPG is allowed to have unsafe collisions without affecting this metric. More interestingly, MADDPG+MAMPS substantially outperforms MADDPG+Naive in terms of performance, often by an order of magnitude. Furthermore, the performance of MADDPG+MAMPS is quite close to the performance of MADDPG alone in settings where there are 3 agents. The relative performance degrades substantially when

there are 4 agents, likely because agent-agent collisions increase significantly, which causes agents to use the recovery policy, and therefore fail to reach their goals.

Discussion. Overall, these results demonstrate the substantial promise of the MAMPS approach. In some settings, it is able to guarantee safety without sacrificing very much performance. There is inevitably some tradeoff between safety and good performance. Nevertheless, we believe that there is much potential to improve MAMPS and reduce how much performance must be sacrificed to ensure safety.

V. CONCLUSION

We have proposed a novel algorithm, MAMPS, for ensuring the safety of a learned control policy for multi-agent systems. Our algorithm comes with strong theoretical guarantees on safety. Furthermore, our experimental results show how MAMPS can ensure safety without sacrificing much performance, and that MAMPS can substantially outperform a naïve approach. There is much room for future work—e.g., allowing for partially observed environments and closing the gap in performance between MAMPS and the learned policy.

APPENDIX

A. Proof of Theorem 1

Note that to prove the theorem statement, it suffices to prove that $\mathbf{x} \in \mathcal{X}_{\text{rec}}$, then $f^{(\pi_{\text{shield}})}(\mathbf{x}) \in \mathcal{X}_{\text{rec}}$ as well. In particular, by induction, this claim implies that a trajectory $(\mathbf{x}_0, \mathbf{x}_1, \dots)$ generated using π_{shield} from initial state $\mathbf{x}_0 \in \mathcal{X}_{\text{rec}}$ satisfies $\mathbf{x}_t \in \mathcal{X}_{\text{rec}}$ for all $t \geq 0$. Furthermore, by definition we have $\mathcal{X}_{\text{stable}} \subseteq \mathcal{X}_{\text{rec}} \subseteq \mathcal{X}_{\text{safe}}$. Thus, a trajectory $(\mathbf{x}_0, \mathbf{x}_1, \dots)$ generated using π_{shield} starting from $\mathbf{x}_0 \in \mathcal{X}_{\text{stable}}$ satisfies $\mathbf{x}(t) \in \mathcal{X}_{\text{safe}}$ —i.e., π_{shield} is safe.

Next, we prove the remaining claim. Consider the action $\mathbf{u} = \pi_b(\mathbf{x})$ returned by Algorithm 1. Since Algorithm 1 checks that $\mathbf{x}' = f^{(\pi_b)}(\mathbf{x}) \in \mathcal{X}_{\text{rec}}$, we are guaranteed that

Algorithm 3 Control policy using the naïve approach.

function NaïveShield(\mathbf{x}):

```
1: if IsRecoverable( $f^{(\hat{\pi})}(\mathbf{x})$ ) then  
2:   return  $\hat{\pi}(\mathbf{x})$   
3: else  
4:   return  $\pi_{\text{rec}}(\mathbf{x})$   
5: end for
```

\mathbf{x}' is recoverable. The challenge is proving that Algorithm 1 actually returns an action.

First, we show that for the choice $b = (0, \dots, 0)$, we have $f^{(\pi_b)}(\mathbf{x}) \in \mathcal{X}_{\text{rec}}$. To this end, note that in this case, we have $\pi_b = \pi_{\text{backup}}$. Then, note that since \mathbf{x} is recoverable, we know that there exists $T \in \{0, 1, \dots, T_{\text{max}} - 1\}$ such that the trajectory $(\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_T)$ generated using π_{backup} from $\mathbf{x}_0 = \mathbf{x}$ satisfies the following: (i) it is safe (i.e., $\mathbf{x}_t \in \mathcal{X}_{\text{safe}}$ for all $t \in \{0, 1, \dots, T\}$), and (ii) it reaches $\mathcal{X}_{\text{stable}}$ (i.e., $\mathbf{x}_T \in \mathcal{X}_{\text{stable}}$). Thus, for $\mathbf{x}' = f^{(\pi_{\text{backup}})}(\mathbf{x})$, consider the trajectory $(\mathbf{x}_1, \dots, \mathbf{x}_T)$ generated using π_{backup} from $\mathbf{x}_1 = \mathbf{x}'$. Trivially, this trajectory is also safe and reaches $\mathcal{X}_{\text{stable}}$. Thus, \mathbf{x}' is recoverable, as claimed. Finally, the check in Algorithm 1 eventually considers $b = (0, \dots, 0)$; thus, it is guaranteed to terminate. The claim follows. ■

B. Parameters

Environment parameters. Maximum episode length: 300. Total size of the environment: $[-1, 1]^2$. Object dimensions: $r_{\text{robot}} = 0.1$, $r_{\text{obstacle}} = 0.1$, $r_{\text{goal}} = 0.05$. Allowed accelerations: $[-1, 1]$. Allowed velocities: $[0, 3]$. Time step: 0.025.

MADDPG parameters. Learning rate: 10^{-3} . Discount factor γ : 0.95. Size of minibatch sample: 1024. Actor and critic networks: 8 fully connected layers with 128 hidden units each for three-agent settings and 10 fully connected layers with 128 hidden units each for four-agent settings.

MAMPS parameters. Maximum trajectory length T_{max} for recovery check: 120.

C. Naïve Approach

Algorithm 3 shows the naïve approach of treating the system as a single-agent system, and then using MPS.

REFERENCES

- [1] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.
- [2] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al., “Learning dexterous in-hand manipulation,” *arXiv preprint arXiv:1808.00177*, 2018.
- [3] H. Tanner, A. Jadbabaie, and G. J. Pappas, “Coordination of multiple autonomous vehicles,” in *IEEE Mediterranean Conference on Control and Automation*, 2003, pp. 869–876.
- [4] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe, “Herb: a home exploring robotic butler,” *Autonomous Robots*, vol. 28, no. 1, p. 5, 2010.
- [5] S. Levine and V. Koltun, “Guided policy search,” in *International Conference on Machine Learning*, 2013, pp. 1–9.
- [6] J. H. Gillula and C. J. Tomlin, “Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor,” in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2723–2730.
- [7] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin, “Reachability-based safe learning with gaussian processes,” in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 1424–1431.
- [8] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, “Automated composition of motion primitives for multi-robot systems from safe ltl specifications,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 1525–1532.
- [9] J. F. Fisac, N. F. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. J. Tomlin, “Bridging hamilton-jacobi safety analysis and reinforcement learning,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8550–8556.
- [10] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, “Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments,” *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.
- [11] W. Schwarting, J. Alonso-Mora, L. Pauli, S. Karaman, and D. Rus, “Parallel autonomy in automated vehicles: Safe motion generation with minimal intervention,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1928–1935.
- [12] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, “Safe reinforcement learning via shielding,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [13] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, “Concrete problems in ai safety,” *arXiv preprint arXiv:1606.06565*, 2016.
- [14] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” in *Advances in neural information processing systems*, 2017, pp. 908–918.
- [15] O. Bastani, Y. Pu, and A. Solar-Lezama, “Verifiable reinforcement learning via policy extraction,” in *Advances in Neural Information Processing Systems*, 2018, pp. 2494–2504.
- [16] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, “Verisig: verifying safety properties of hybrid systems with neural network controllers,” in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. ACM, 2019, pp. 169–178.
- [17] K. P. Wabersich and M. N. Zeilinger, “Linear model predictive safety certification for learning-based control,” *CoRR*, vol. abs/1803.08552, 2018. [Online]. Available: <http://arxiv.org/abs/1803.08552>
- [18] O. Bastani, “Safe reinforcement learning via online shielding,” *arXiv preprint arXiv:1905.10691*, 2019.
- [19] S. Li and O. Bastani, “Robust model predictive shielding for safe reinforcement learning with stochastic dynamics,” [Online]. Available: <https://obastani.github.io/docs/rmps.pdf>
- [20] J. P. Desai, J. P. Ostrowski, and V. Kumar, “Modeling and control of formations of nonholonomic mobile robots,” 2001.
- [21] J. Alonso-Mora, S. Baker, and D. Rus, “Multi-robot navigation in formation via sequential convex programming,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4634–4641.
- [22] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. A. Beardsley, and R. Siegwart, “Optimal reciprocal collision avoidance for multiple non-holonomic robots,” in *DARS*, 2010.
- [23] J. van den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” 05 2008, pp. 1928–1935.
- [24] A. Khan, C. Zhang, S. Li, J. Wu, B. Schlotfeldt, S. Y. Tang, A. Ribeiro, O. Bastani, and V. Kumar, “Learning safe unlabeled multi-robot planning with motion constraints,” *arXiv preprint arXiv:1907.05300*, 2019.
- [25] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *Neural Information Processing Systems (NIPS)*, 2017.
- [26] I. Mordatch and P. Abbeel, “Emergence of grounded compositional language in multi-agent populations,” *arXiv preprint arXiv:1703.04908*, 2017.