ReCoDe: Reinforcement Learning-based Dynamic Constraint Design for Multi-Agent Coordination

Michael Amir, Guang Yang, Zhan Gao, Keisuke Okumura, Heedo Woo, Amanda Prorok

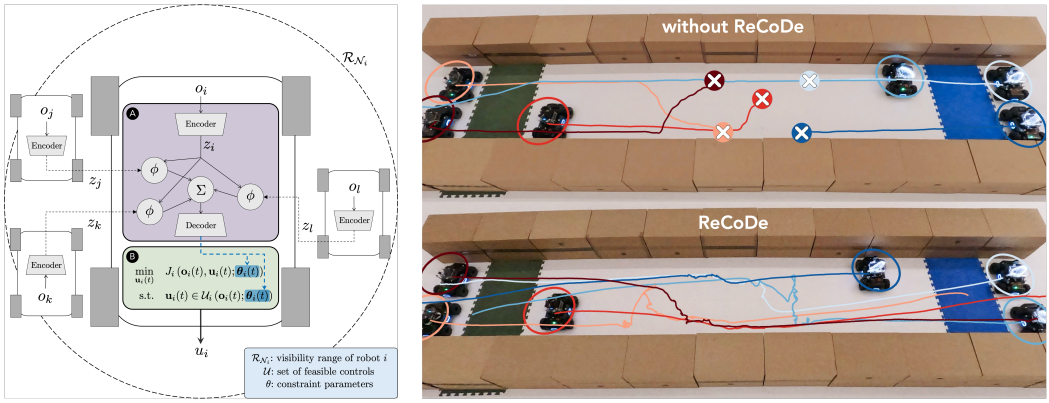


Figure 1: **Left:** An overview of the proposed ReCoDe method. A GNN policy (A) aggregates the encoded observations of neighboring agents within a visibility range $\mathcal{R}_{\mathcal{N}}$ and generates constraint parameters θ that influence the feasible set $\mathcal{U}$ of an optimization-based controller (B). **Right**: Real-robot position-swap in a 90 cm-wide, 6.4 m-long corridor. **Top:** baseline QP controller dead-locks. **Bottom:** the *same* controller augmented with ReCoDe succeeds. Six holonomic ground robots (Blumenkamp et al.) communicate only within 1.5 m.

Abstract: Constraint-based optimization is a cornerstone of robotics, enabling the design of controllers that reliably encode task and safety requirements such as collision avoidance or formation adherence. However, handcrafted constraints can fail in multi-agent settings that demand complex coordination. We introduce ReCoDe—Reinforcement-based Constraint Design—a decentralized, hybrid framework that merges the reliability of optimization-based controllers with the adaptability of multi-agent reinforcement learning. Rather than discarding expert controllers, ReCoDe improves them by learning additional, dynamic constraints that capture subtler behaviors, for example, by constraining agent movements to prevent congestion in cluttered scenarios. Through local communication, agents collectively constrain their allowed actions to coordinate more effectively under changing conditions. In this work, we focus on applications of ReCoDe to multi-agent navigation tasks requiring intricate, context-based movements and consensus, where we show that it outperforms purely handcrafted controllers, other hybrid approaches, and standard MARL baselines. We give empirical (real robot) and theoretical evidence that retaining a user-defined controller, even when it is imperfect, is more efficient than learning from scratch, especially because ReCoDe can dynamically change the degree to which it relies on this controller.

1 Introduction

Ensuring that multiple autonomous agents, such as fleets of autonomous vehicles and warehouse robots, can safely and efficiently coordinate in a shared environment is a long-standing challenge in robotics [2, 3]. Classical approaches rely heavily on optimization-based controllers, which encode mission objectives and constraints into a tractable optimization problem [4]. By carefully designing these constraints, we can ensure collision avoidance, enforce kinematic limitations, and guide agents

All authors are with the University of Cambridge. KO is also with National Institute of Advanced Industrial Science and Technology (AIST). Email: {ma2151,gy268,zg292,,ko393,hw527, asp45}@cst.cam.ac.uk.

Supplementary video: https://www.youtube.com/watch?v=SgCE0TFOz2c

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

toward their goals. Yet, as scenarios grow more complex, handcrafted constraints become insufficient: they cannot easily adapt to evolving conditions or fully exploit coordination opportunities [5]. At the opposite extreme, multi-agent reinforcement learning (MARL) is a highly adaptive paradigm that shapes agent behavior through experience without task-specific design [6, 7, 8]. However, MARL lacks the analytic structure and safety assurances of optimization-based methods, making its decisions harder to predict, verify, or trust in critical applications.

This paper presents ReCoDe (Reinforcement-based Constraint Design), a hybrid, decentralized framework combining optimization-based control with adaptation provided by MARL. ReCoDe operates by *augmenting* a given expert controller[1]: it keeps the original controller (and its safety constraints) but learns additional, situation-dependent constraints via MARL. These learned constraints dynamically modify each agent's feasible action set, allowing for finer control and improved coordination beyond what the fixed controller permits. This approach preserves the desirable properties of the expert controller—like safety guarantees and interpretability—while enabling adaptation to complex scenarios. ReCoDe is inherently multi-agent, as agents learn to collectively shape their constraints for better overall performance. Agents integrate information from neighbors via local communication when deciding how to adjust their constraints at each point in time. This design is facilitated by a Graph Neural Network (GNN)-based policy (Figure 1).

Why learn additional constraints? While handcrafted optimization-based controllers excel at encoding single-agent safety requirements and tasks, they often fall short when dealing with more intricate multi-agent interactions. Consider a two-way street where vehicles want to travel either up or down: although handcrafted constraints can ensure safety (e.g., avoiding collisions), they typically do not address higher-level challenges such as congestion or mutual blocking. To deal with such challenges, human drivers impose on themselves additional *situational* constraints—like staying strictly within a lane or yielding to unblock traffic. These more nuanced behaviors hinge on communication among agents and online adaptation to changing conditions. Starting with only basic constraints (e.g., collision avoidance), ReCoDe learns such higher-level coordination rules as additional constraints, based on context and communication with other agents. The result is a hybrid controller that keeps formal guarantees, anchors learning with expert structure, and still adapts online to multi-agent interactions by letting agents collectively shape stricter constraints that help them coordinate.

Why learn constraints rather than the optimization objective? Some existing hybrid approaches work by learning the optimization objective [9, 10]. We posit that retaining the handcrafted objective and refining decision-making with learned constraints is more data-efficient, because the default objective, even if imperfect, embeds expert knowledge of the task. Moreover, while hard constraints can, in principle, be mimicked by infinite-cost penalties in the objective, thus learning the objective is *theoretically* more general–RL rarely discovers such penalties, and its learned costs remain soft and opaque. Concretely, in ReCoDe, each agent's *learned* policy $\pi : \mathcal{O} \to \Theta$ maps its local observation $\mathbf{o}(t)$ to the parameters $\theta(t) = (\mathbf{a}(t), b(t))$ of a *single quadratic constraint* $\| \mathbf{u}(t) - \mathbf{a}(t) \|_2 \leq b(t)$, added to the expert controller at every timestep. Here $\mathbf{u}(t) \in \mathbb{R}^m$ is the constrained optimization's decision variable, $\mathbf{a}(t) \in \mathbb{R}^m$ is a *reference action* suggested by the policy, and the *uncertainty radius* $b(t) \in \mathbb{R}_{\geq 0}$ dictates how tightly the solver must follow that reference—the larger $b(t)$, the more the agent defers to the original controller. Thus the action space of the policy consists of the constraint parameters $(\mathbf{a}(t), b(t))$, rather than the actual control input $\mathbf{u}$, which is outputted by the solver.

Scope and Results. We investigate ReCoDe's mechanism theoretically and empirically. On the theory side we show that enlarging the uncertainty radius $b(t)$–thus increasing the agent's reliance on the default controller–can improve performance in some situations (Proposition 3.2), while shrinking it enables precise learned control (Propostion 3.1). Empirically we confirm that ReCoDe learns to exploit this: it tightens its quadratic constraint in congested situations requiring precision and relaxes it when the path is clear, shifting authority between the learned policy and the default controller based on the situation (Figures 3d–3e). This supports our claims about the benefits of retaining the default controller (and in particular, its optimization objective). We perform an ablation to further

[1] We will refer to this controller, interchangeably, as the default/expert/handcrafted controller.

compare learning only constraints, learning only the objective, and learning both, and find that learning *just* the constraint parameters yields the highest return and fastest convergence (Figure 3f).

While ReCoDe is a general method, we focus on studying its effectiveness in navigation and consensus settings. We design four experiments where coordination is necessary due to the small size of the environment, formation constraints, or conflicting incentives. We compare ReCoDe's performance in these experiments to several baselines: RVO [11], a gold-standard non-learning methods for collision avoidance; two different hybrid methods from the literature [5, 9]; and end-to-end MARL. Across all four scenarios, ReCoDe attains, on average, 18% better reward than the next-best method (Table 2), and trains using just 5% as many samples as end-to-end MARL (Figure 3b). We further demonstrate that ReCoDe preserves safety throughout training and deployment, and find that ReCoDe's learned constraints still provide a benefit to the handcrafted controller even in less coordination-heavy tasks (Figure 3a). Finally, we deploy ReCoDe on real robots, tasking two robot teams in a narrow corridor to swap positions. We find that the default controller often fails this task, leading robots into a deadlock, whereas ReCoDe's additional constraints let robots coordinate to avoid deadlock and successfully swap (Figure 1).

Related Work. Classical multi-robot control is rooted in *constrained optimization*, where an objective encodes the mission and constraints enforce safety, kinematics and environmental limits [4]. Linear programming and quadratic programming are the most common type of optimization [12, 13], often making use of control-barrier or Lyapunov functions for analytic collision avoidance and tracking guarantees [14, 15, 5]. Non-convex solvers can handle richer dynamics at higher computational cost [16]. In contrast, *multi-agent learning* approaches—including MARL—optimize neural policies directly from interaction data and have been applied to many coordination problems [6, 17, 18, 8]. In MARL, agents learn to make decisions by interacting with the environment and updating their policies based on the feedback. Unlike constrained optimization, these methods do not require an analytic formalization of the desired task, and are more adaptible. However, they sacrifice the analytic guarantees of model-based controllers and may converge slowly when safe, high-reward actions are sparse. We provide a more detailed survey of both strands in Appendix A.

Hybrid methods combine MARL with constrained optimization. Relevant works in this space are [5], [10], and [9]. In [5], MARL is used to optimize parameters of CBFs in constrained optimization problems for multi-agent navigation. In this work, rather than optimizing existing constraints, we learn additional, entirely new, constraints. The methods in [9] and [10] can be seen as shaping the *objective* of a constrained optimization problem, in different ways. While not applicable to our evaluated experimental scenarios, in [10], RL is used to learn the objective function of a model-predictive control system for a single agent, under analytic assumptions that enable gradient backpropagation. In [9], the *shielding* method is introduced, which intervenes when the learned policy's action violates safety constraints. One implementation of shielding uses constrained optimization whose *objective* is to find a safe action closest to the policy's. ReCoDe instead attempts to generate an action based on a user-provided objective function and safety constraints, and generates additional constraints to further guide this optimization. We compare to shielding and Online CBF, and find that ReCoDe outperforms both these methods in a variety of navigation tasks (Section 4).

2 Problem Setting and Method: Dynamic Constraint Design

Setting. We consider a multi-agent system consisting of N agents in a shared environment. The primary objective of each agent is to maximize its cumulative reward over time. The reward at time t reflects the agent's performance in the environment and may depend on factors such as task completion, efficiency, or cooperation with other agents. Agents interact with their environment through control inputs, which are obtained by solving an optimization problem whose constraints the agent itself can partially specify through choosing parameters $\theta(t)$. This optimization problem gives *instantaneous control input* at every time step (i.e., there is no MPC-like receding horizon).

Let $\mathbf{x}_i(t) \in \mathbb{R}^n$ denote the state of agent i at time t, and $\mathbf{u}_i(t) \in \mathbb{R}^m$ denote its control input. The dynamics of our agents are given by the discrete system $\mathbf{x}_i(t + dt) = f_i\left(\mathbf{x}_i(t), \mathbf{u}_i(t)\right)$, where f_i represents the dynamics of agent i and dt is a constant representing the length of a time step. Further-

more, at each time t, agent i receives an observation $\mathbf{o}_i(t) \in \mathcal{O}_i$, which depends on its own state, the states of other agents (if observed), and environmental variables: $\mathbf{o}_i(t) = h_i\left(\mathbf{x}_i(t), \mathbf{x}_{-i}(t), \mathbf{e}(t)\right)$, where h_i is the observation function, $\mathbf{x}_{-i}(t)$ denotes the states of other visible agents, and $\mathbf{e}(t)$ represents external environmental factors. Given the observation and constraint parameters $\boldsymbol{\theta}_i(t)$, agent i thus solves the constrained optimization problem $\min_{\mathbf{u}_i(t) \in \mathcal{U}_i(\mathbf{o}_i(t); \boldsymbol{\theta}_i(t))} J_i(\mathbf{o}_i(t), \mathbf{u}_i(t))$, where $\mathcal{U}_i := \{\mathbf{u}_i(t) \mid g_k(\mathbf{o}_i(t), \mathbf{u}_i(t); \boldsymbol{\theta}_i(t)) \leq 0,\ k = 1, \ldots, K\}$.

Here, $J_i\left(\mathbf{o}_i(t), \mathbf{u}_i(t)\right)$ is a strictly convex in u, quadratic cost function, possibly representing factors like energy expenditure or deviation from a desired trajectory. $\mathcal{U}_i$ is the set of admissible controls parametrized by θ, where g_i represents the constraint functions. The constraint parameters of agent i are selected based on its current observation $\mathbf{o}_i(t)$ and its policy: $\boldsymbol{\theta}_i(t) = \pi_i\left(\mathbf{o}_i(t)\right)$, where $\pi_i : \mathcal{O}_i \to \Theta_i$ is a policy mapping observations to parameters.

Method. To enable agents to design constraints, we propose (Re)inforcement-based (Co)nstraint (De)sign. ReCoDe trains agents in simulation using MAPPO, an actor-critic MARL algorithm [19]. Each agent i seeks to learn a policy π_i that maps observations to constraint parameters, aiming to maximize the agent's reward. We design each agent's policy network to leverage relational information in the multi-agent system through a mechanism that aggregates nearby agents' messages. Although any such mechanism could work in principle, we elect to use Graph Neural Networks (GNNs). GNN architectures enable decentralized execution in inference time through message passing, where each agent computes messages m_{ij} to send to neighbors; aggregates incoming messages $m_i = \sum_{j \in \mathcal{N}_i} m_{ij}$; and updates its state $\mathbf{n}'_i = \sigma(m_i)$. This local computation ensures that each agent is decentralized, relying only on its own and neighbors' information. The unique structure of GNNs make the agent's perception of its neighborhood both *permutation invariant* and *dynamic*. That is, the order of neighboring agents does not affect the computation, and the neighborhood can adapt to external constraints, such as a limited sensing range. An overview of our method is shown in Figure 1. Further implementation details are available in Appendix B.

Centralized Training, Decentralized Execution. We adopt a CTDE setup to speed up data collection during training [20]. Specifically, we run M environment instances in parallel, each with N agents, and aggregate the resulting $\mathcal{O}(M \times N)$ optimization problems into a *single batched program*. However, if just one agent learns a parameter configuration that makes its constraints infeasible, the entire batch solver can fail, making it difficult to identify which agent caused the issue. To circumvent this, we introduce a *slack variable* s_k only in *learned* constraints (thus safety is unaffected), transforming $\mathcal{U}_i$ into $\mathcal{U}_i^s := \left\{ \mathbf{u}_i(t) \,\middle|\, g_k\left(\mathbf{o}_i(t), \mathbf{u}_i(t); \boldsymbol{\theta}_i(t)\right) \leq s_k, k \in [1, K] \right\}$. During training, each agent tries to minimize $J_i\left(\mathbf{o}_i(t), \mathbf{u}_i(t)\right) + \sum_{k=1}^{|\mathcal{U}_i^s|} \lambda_k s_k$ where each slack $s_k \geq 0$ is heavily penalized ($\lambda_k \gg 0$), so it is only nonzero when no feasible solution exists. This setup also identifies infeasible programs by flagging the agents whose s_k is nonzero. This centralization takes place in *training*; at deployment, each agent is fully decentralized, and solves its own local optimization problem with the final learned constraints based only on local observations and communication.

Constraint Form. In ReCoDe, agents augment a default controller with additional constraints for improved performance. We focus on learning a single, quadratic constraint $\|\mathbf{u}_i(t) - \mathbf{a}_i(t)\|_2 \leq b_i(t) + s_0$ parametrized by $\boldsymbol{\theta}_i(t) = (\mathbf{a}_i(t), b_i(t))$, where $\mathbf{a}_i(t) \in \mathbb{R}^m$ is a *reference action* suggested by the policy, $b_i(t) \in \mathbb{R}_{\geq 0}$ is a radius that decides how much the handcrafted controller can steer away from $\mathbf{a}(t)$, and s_0 is a slack variable. We explain the reasoning for this, and consider other types of constraints, in Appendices C and G. The learned agent policy $\pi_i : \mathcal{O}_i \to \Theta_i$ maps its observation $\mathbf{o}_i(t)$ to the constraint parameters $\boldsymbol{\theta}_i(t) = (\mathbf{a}_i(t), b_i(t))$, and agent i solves:

$$\min_{\mathbf{u}_i(t)} J_i\left(\mathbf{o}_i(t), \mathbf{u}_i(t)\right) + \lambda_0 s_0 \ \text{ s.t. } \ \|\mathbf{u}_i(t) - \mathbf{a}_i(t)\|_2 \leq b_i(t) + s_0, \mathbf{u}_i(t) \in \mathcal{U}_i^s\left(\mathbf{o}_i(t)\right). \quad (1)$$

Here, $\mathcal{U}_i^s\left(\mathbf{o}_i(t)\right)$ defines the constraints of our default (expert) controller, which may be parametrized by agent observations but, unlike $(\mathbf{a}(t), b(t))$, not *learned* by our algorithm. We call $b_i(t)$ the *uncertainty radius* of agent i at time t, since it controls how strict the learned constraint is. A larger value of $b_i(t)$ can be viewed as the learned policy having uncertainty about how optimal the action $\mathbf{a_i}(t)$ is, hence preferring to influence the default controller less.

We *assume* that in (1) the objective J_i is *strictly convex* in $\mathbf{u}$ and that, for every observation $\mathbf{o}$ and parameter vector $\boldsymbol{\theta} \in \Theta$, the feasible set is *non-empty and convex*. We further assume that the problem (1) admits a *unique* minimizer $\mathbf{u}^\star(\mathbf{o}, \boldsymbol{\theta})$, and that the solution mapping $\boldsymbol{\theta} \mapsto \mathbf{u}^\star(\mathbf{o}, \boldsymbol{\theta})$ is continuously differentiable in a neighborhood of each $\boldsymbol{\theta}$ ([21] lists mild regularity conditions under which this property holds). Because $J_i(\mathbf{o}, \mathbf{u})$ is strictly convex quadratic in $\mathbf{u}$ and constraints are convex, (1) is a *convex QCQP*. Such problems are known to be efficiently solvable [22]. This is important for inference and training, as we must deploy many instances of (1) during data collection.

3 Analysis

Our analysis shows that tightening the uncertainty radius $b(t)$ enables *adaptability*, letting ReCoDe track any safe trajectory, whereas enlarging the radius lets the handcrafted controller take over and raise the reward when the policy is uncertain—thereby validating ReCoDe's design of dynamically balancing learned and expert control (proofs in Appendices D–E).

Adaptability. Since ReCoDe uses a constrained-optimization framework, user-defined safety constraints are never violated. We show here that as long as it remains within these constraints, and the slack penalty λ_0 is sufficiently large, ReCoDe is *precise and adaptable*: the agent can choose constraints that force its controller to track any safe, feasible trajectory with arbitrarily small error. Formally, let $T \in \mathbb{N}$ be a finite time horizon. Consider an agent in our system whose initial state is $x^*(1)$, and a feasible desired trajectory of actions and states $(x^*(1), u^*(1)), (x^*(2), u^*(2)), \ldots, (x^*(T), u^*(T))$ where executing $u^*(t)$ in state $x^*(t)$ takes the agent to state $x^*(t+1)$. Let $B_\varepsilon(p) = \{y \in \mathbb{R}^m : \|y - p\| < \epsilon\}$ be the ε-neighbourhood of a point p. Assume that, for each time t, the desired action $u^*(t)$ is strictly feasible under the handcrafted constraints without requiring any slack, i.e., with $s_0(t) = 0$. More formally, there exists $\eta > 0$ such that for all t, $B_\eta(u^*(t)) \subseteq \mathcal{U}_i^s(\mathbf{o}_i(t))$, where $\mathcal{U}_i^s(\cdot)$ denotes the feasible set defined by the hand-crafted constraints with slack parameters, and at $u^*(t)$ we have $s_0(t) = 0$. Then we have:

Proposition 3.1 (ε-Close Trajectory Tracking)**.** *For any $\varepsilon > 0$, there exists a sufficiently large penalty factor λ_0 and a sequence of learned constraint parameters $\{(\mathbf{a}(t), b(t))\}_{t=1}^T$, with $\mathbf{a}(t) \in \mathbb{R}^m$ and $b(t) \in \mathbb{R}_{>0}$, such that the unique optimal solutions $u^{opt}(t)$ of the optimization problem (1) satisfy $\|u^{opt}(t) - u^*(t)\| \le \varepsilon, \quad \forall t = 1, \ldots, T$. Moreover, the resulting state trajectory $\{x(t)\}_{t=1}^T$ satisfies: $\|x(t) - x^*(t)\| \le \varepsilon, \quad \forall t = 1, \ldots, T$.*

Uncertainty Mitigation. In many RL algorithms, such as actor-critic, an agent wants to pick a control input u to maximize the *true* expected reward $Q_i^*(\mathbf{o}, u)$ given observation $\mathbf{o}$, but must optimize some imperfect learned proxy of Q_i^*–a critic–which we denote $Q_i^l(\mathbf{o}, u)$. ReCoDe, instead, has the agent policy output $\mathbf{a}(\mathbf{o})$ and an *uncertainty radius* $b(\mathbf{o})$ that define a ball (constraint) where u should lie $\|u - \mathbf{a}(\mathbf{o})\|_2 \le b(\mathbf{o})$. We make the somewhat simplifying *interpretation* that the agent chooses $\mathbf{a}(\mathbf{o})$ because it maximizes $Q_i^l(\mathbf{o}, u)$ for some learned proxy Q_i^l. Next, the handcrafted optimization objective $J_i\big(\mathbf{o}, u\big)$ picks u *inside* the ball; the larger the radius $b(\mathbf{o})$, the stronger the expert controller's influence. We will show it is beneficial to enlarge $b(\mathbf{o})$, i.e., to mix the expert controller with the learned policy, when $Q_i^l(\mathbf{o}, u)$ is more *uncertain* than J_i given observation $\mathbf{o}$. By uncertain we mean that $Q_i^l(\mathbf{o}, u)$'s gradient is locally bounded by a small δ and therefore it assigns roughly the same value to all actions locally[2]. The result requires that for some $c_1(\mathbf{o}), c_2(\mathbf{o}) > 0$, in a neighborhood of $\mathbf{a}(\mathbf{o})$, a weighted combination $c_1 Q_i^l - c_2 J_i$ is a good approximation of Q_i^*.[3]

Proposition 3.2. *Assume there is $r > 0$ such that for all actions $u \in B_r\big(a(\mathbf{o})\big)$ we have $\big|Q^*(\mathbf{o}, u) - \big[c_1 Q_i^l(\mathbf{o}, u) - c_2 J_i(\mathbf{o}, u)\big]\big| \le \varepsilon$ and $\|\nabla_u Q_i^l(\mathbf{o}, u)\|_2 \le \delta_1$. Assume also that there exists a* unit *direction $\mathbf{d}$ and a constant $\delta_2 > \delta_1$ such that $\mathbf{d}^\top \nabla_u \big[-J_i\big]\big(\mathbf{o}, \mathbf{a}(\mathbf{o}) + x\mathbf{d}\big) \ge \delta_2 \quad \forall\, x \in [0, r]$. Let $c_2\delta_2 - c_1\delta_1 = \Delta$. If every action $u \in B_r\big(a(\mathbf{o})\big)$ is strictly in problem (1)'s feasible action set without slack and λ_0 (the slack penalty on s_0) is sufficiently large, then $Q^*(\mathbf{o}, u^{\mathrm{opt}}(\mathbf{o})) \ge Q^*(\mathbf{o}, a(\mathbf{o})) + r\Delta - 2\varepsilon$, where u^{opt} is the solution to (1) (i.e., ReCoDe's output) given $b(\mathbf{o}) = r$.*

[2]Strictly speaking, flatness does not necessarily imply uncertainty; we use it as an imperfect proxy.

[3]If $c_1 Q_i^l - c_2 J_i$ locally approximates $Q_i^* + c_3$ for some constant $c_3(\mathbf{o})$, a similar result holds. Note: this approximation assumption is weaker than assuming Q_i^l or J_i on their own can be used to approximate Q_i^*.

How to read the bound. $r = b(\mathbf{o})$ is the *uncertainty radius*; enlarging it lets the handcrafted objective J_i shape the solver's choice inside a wider ball. The constant δ_1 upper-bounds the gradient of Q_i^l in that ball, so it quantifies how *flat*—hence how uncertain—the learned policy is locally. By contrast $\delta_2 > \delta_1$ lower-bounds the directional derivative of $-J_i$ along *at least one* direction, certifying that the expert objective is less flat than Q_i^l. If $c_2\delta_2 > c_1\delta_1$, then $\Delta = c_2\delta_2 - c_1\delta_1 > 0$; consequently the term $r\Delta$ in the bound is positive and the solver finds an action that beats the critic's own maximizer by at least $r\Delta - 2\varepsilon$. In other words, when, locally, the critic is flat but the expert objective is decisive, enlarging the uncertainty radius *mitigates uncertainty* by mixing expert and learned knowledge.

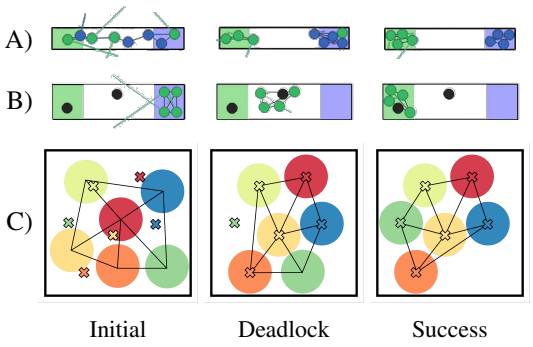
The proposition does not directly imply a strategy, as $c_1(\mathbf{o})$ and $c_2(\mathbf{o})$ are *unknown*. Instead, it explains *why* giving the policy control over the uncertainty radius can be useful. To properly determine $b(\mathbf{o})$, the an agent should have some notion of how to weigh the handcrafted vs. learned optimization goal locally. Does ReCoDe tune $b(\mathbf{o})$ like this in practice? In Section 4 we give empirical evidence for this: $b(\mathbf{o})$ *shrinks* in crowded, high-interaction states where the learned policy is more reliable than the expert, but *expands* once the path is clear, leaning on the handcrafted controller.

4 Evaluation

We evaluate ReCoDe in four *multi-agent navigation and consensus* tasks that are designed to expose two common failure modes in multi-robot control. The first mode appears when safe, reward-producing actions are sparse; in such settings pure reinforcement learning spends most of its time exploring moves that end in collisions and learns very slowly. The second mode remains even when individual safe moves are plentiful and easy to compute: with several robots in close proximity, reciprocal blocking and group-level constraints can trap a system in deadlock, something a handcrafted controller often cannot anticipate or avert.

The **Narrow Corridor** task places two teams at opposite ends of a narrow hallway and asks them to swap positions, so agents must discover when to yield in a space where successful moves are sparse. **Connectivity** requires a single team to navigate to the end of the hallway, but introduces static obstacles and and requires that every pair of robots stay within a fixed communication range, thereby preserving full connectivity throughout the task. This binds the motions of the entire group and requires them to collectively negotiate movements. The **Waypoint Navigation** scenario moves over-sized robots in a small room with random goals, frequently requiring robots to go "the long way round" to their location if deadlocks are to be avoided. Finally, the **Sensor Coverage** scenario is a multi-objective scenario that couples motion with high-level consensus about *where* to go: a fleet of sensors, each assigned to monitor different phenomena, must collectively decide where to position themselves to attain the best overall coverage of the environment while never breaking their communication graph (see Appendix F, Figure 4). In all scenarios, holonomic agents observe their own position, distance to goal position, and relative position to obstacles and other agents within their communication range. Detailed scenario definitions are available in Appendix F.

Across all tasks we benchmark ReCoDe against multiple baselines. The first is the **handcrafted controller**—the best constraint-based controller we could craft for each scenario, the details of which we give in Appendix F. This is also the controller we use as a basis for ReCoDe. The other non-learning based method is **Reciprocal Velocity Obstacle** (RVO) algorithm [11], a gold-standard method for multi-agent collision avoidance. To isolate the benefit of optimization we also include **Pure MARL**, an end-to-end MARL policy that directly controls the agents. Finally, we test against two hybrid methods, **Online-CBF** [5] and **shielding** [9], covered in our Related Work section. In our implementation of shielding, the policy outputs a target velocity and passes it through a safety filter using the same safety constraints we used for ReCoDe; in Online CBF, we learn the parameter k in the control barrier function of (2). In all experiments that make use of reinforcement learning, we used MAPPO and the same GNN-based actor-critic architecture introduced in Section 2 (such architectures are SOTA in end-to-end MARL–see, e.g., [23]). We train all baselines for 7.2 million environment steps, or in the case of *Pure MARL*, for longer until reward stabilizes.



	Narrow Corridor (a)	Connectivity (b)	Waypoint Nav. (c)	Sensor Coverage
ReCoDe	**0.55±0.03**	**0.40±0.03**	**0.73±0.06**	**0.25±0.01**
Handcrafted	0.43±0.03	0.18±0.03	0.44±0.02	0.19±0.02
Online CBF	0.38±0.04	0.23±0.05	0.53±0.05	0.19±0.02
MARL	-4.50±0.98	-3.85±3.37	-0.17±0.05	-0.45±0.06
Shielding	0.45±0.02	**0.38±0.02**	0.57±0.03	0.2±0.01
RVO	0.44±0.03	N/A	0.62±0.02	N/A

Figure 2: Experimental scenarios and results. Leftmost column: initial conditions; middle: a possible deadlock scenario where agents require coordination to proceed; rightmost: scenario success. Agents are colored circles; black edges denote communication lines. (A) Narrow Corridor: blue/green agents navigate to color-matched regions while communicating with neighbors. (B) Connectivity: agents bypass obstacles without breaking any communication links. (C) Waypoint Navigation: agents coordinate to reach color-matched goals, requiring some to leave goals despite short-term incentives temporarily. The table shows average rewards per time step for ReCoDe, handcrafted constraints, Online CBF, RVO, shielding, and Pure MARL across 75 random starting conditions (mean ± standard deviation over best 6 consecutive training steps). Maximum possible rewards are *roughly* 1 in Narrow Corridor, Connectivity, 1.5 in Waypoint, and unknown in Sensor Coverage.

4.1 Results

Our results are summarized in Table 2. ReCoDe significantly outperforms the baselines in all tested scenarios, attaining, on average, 18% greater reward than the next-best method. *Pure MARL* performed the worst, and was unable to control agents and reach to an adequate performance across all scenarios. This is because our scenarios are very sensitive to small changes in the control input (e.g., in a narrow corridor a very small change in input makes the difference between a collision and successful navigation); it is difficult to obtain a positive reward without optimization. Qualitatively, the most frequent failure modes for other methods were deadlocks stemming from lack of coordination.

MARL/Handcrafted Controller Comparison. We examined how MARL and the handcrafted controller compare to ReCoDe under easier conditions by varying each agent's radius in *Waypoint Navigation*. Smaller radii make collisions less likely, simplify navigation and require less intricate coordination. Figure 3a shows that even with smaller agents, ReCoDe outperforms both pure MARL and the handcrafted controller, and so might be beneficial even for less coordination-heavy scenarios. Furthermore, compared to MARL, ReCoDe demonstrates much faster training convergence: as shown in Figure 3b, whereas pure MARL still underperforms substantially after 500 training steps of 120k frames each, ReCoDe reaches excellent performance in 20 steps. Another critical consideration is *safety during training*. As shown in Figure 3c, ReCoDe consistently maintains near-zero collision rates during training, a key advantage of hybrid methods over end-to-end MARL.

Mechanism. Proposition 3.2 suggests that agents should shrink the uncertainty radius b when the learned policy is reliable and enlarge it when the handcrafted objective offers stronger signal. To test whether ReCoDe learns this behavior, we logged b at three training checkpoints (0.96 M, 1.92 M and 8.64 M steps) in the *Narrow Corridor* task over 100 six-robot episodes with random initial states, collecting data from agents aiming for the green region. Figures 3e–3d plot b against each robot's neighbor count and y-position (robots with $y > 1.5$ have reached the goal). Early in training b is uniformly small, indicating near-total reliance on the learned policy; as learning progresses the mean radius grows, allowing the default controller to contribute more. We find that b correlates negatively with number of nearby agents ($r \approx -0.03,\ p < 10^{-40}$) and positively with y/goal proximity ($r \approx 0.08,\ p < 10^{-200}$). Thus ReCoDe tightens b to resolve likely deadlocks (as the learned policy is better at such coordination) and relaxes it once the path is clear (as the default controller makes more efficient, greedy movements), matching the strategy predicted by our analysis.

Ablation. To test whether performance can be improved by learning the objective, we compared three variants of ReCoDe in *Narrow Corridor*: *standard* ReCoDe, which learns only the quadratic-constraint parameters; a variant that keeps this constraint fixed but lets the policy learn the *goal*

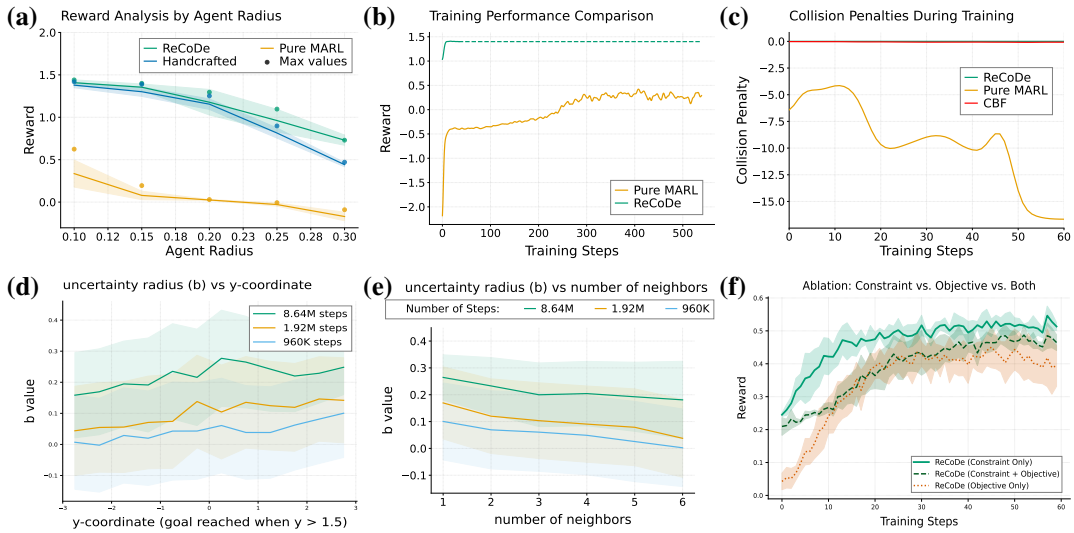


Figure 3: **(a)** Complexity vs. Reward in Waypoint Navigation: ReCoDe consistently outperforms both pure MARL and the handcrafted controller across different agent radii, demonstrating robustness in both high- and low-task complexities. Shaded regions indicate standard deviations. **(b)** Sample Efficiency: In Waypoint Navigation with agent radius = 0.1, ReCoDe quickly converges to near-optimal reward, whereas pure MARL remains suboptimal even after 500 steps. **(c)** Collision penalties during training of the Narrow Corridor scenario are near-zero for ReCoDe (-0.0001 on average), Online CBF (-0.06) and pure RL (-9.4). **(d)–(e)** Plots of the learned constraint radius (b) vs. the agent's y-position and number of neighbors at various stages of training in the Narrow Corridor scenario. Shaded regions indicate $\frac{1}{2}$ std. **(f)** Ablation: Learning constraints vs. objective vs. both in the Narrow Corridor setting (4 random seeds).

position of the objective; and a variant that learns both the constraint and the target position simultaneously. Although the third variant is the most expressive, the constraint-only version reached its peak sooner and achieved the highest return, while the two objective-learning variants converged more slowly and plateaued at lower scores (Fig. 3f), supporting our claim that the fixed, expert-designed objective provides a valuable inductive bias that is diluted once it becomes a moving target.

Robot Demonstration. We transferred the policy trained in the *Narrow-Corridor* simulation directly to six holonomic ground robots [1]. The physical arena matches the simulation scale: a 90 cm-wide, 6.4 m-long corridor. Robots exchange messages only when separated by ≤ 1.5 m to mimic the limited communication range used during training. In our trials, the handcrafted controller dead-locked in every run, typically when two teams met near the midpoint. With ReCoDe's learned quadratic constraint active, all six robots consistently complete the swap without violating safety margins. These results corroborate the simulation study: ReCoDe's online constraint generation resolved reciprocal blocking despite real-world noise stemming from positional tracking errors, communication delays, and actuator limitations in precisely executing intended commands. We further experiment with learning *linear* constraints, and find this encourages the robots to avoid partial solutions, but does not affect reward–find more details in App. G. Figure 1 overlays the trajectories; a video is available at `https://www.youtube.com/watch?v=SgCE0TF0z2c`.

5 Discussion

We presented ReCoDe, a hybrid control framework that combines expert knowledge with reinforcement learning to augment constrained optimization-based controllers. Our experiments, focusing on applications of ReCoDe to navigation and consensus, show that this combination outperforms both of its components in isolation: handcrafted controllers respect safety yet coordinate poorly, and pure MARL explores freely but lacks local precision and safety guarantees. ReCoDe overcomes both weaknesses by constraining each agent in response to its neighbors' intents, enabling adaptability when the handcrafted controller would stall, but falling back on that controller when the learned policy is uncertain. As a result it consistently outperforms other methods in our tested scenarios, requiring fewer samples than pure learning while not violating the user-defined safety constraints.

6 Limitations

(i) Our experiments demonstrate the performance improvements of ReCoDe in intricate navigation tasks, and complex tasks involving both navigation and multi-objective consensus. However, due to the significant work involved in setting up relevant experimental scenarios, we have not yet studied ReCoDe in non-navigation settings such as multi-agent manipulation, leaving open the question of how it compares to existing baselines in such settings. This will be a topic for future work.

(ii) ReCoDe currently assumes the underlying optimization problem is convex. In principle, ReCoDe can be extended to nonconvex optimization, but the user must provide a method of solving the optimization problem that is efficient enough to collect data for. If the user is fine with sub-optimal solutions, there are many methods (e.g., gradient descent) for solving nonconvex optimization problems, thus enabling us to extend ReCoDe to such settings. We note that because we are modifying the user-provided controller anyway, it is likely not important for the user to use a solver that finds the global optimum, and so this is not necessarily a fundamental limitation. However, as we did not explore nonconvex, sub-optimal solvers in this work, we cannot decisively comment on how well ReCoDe works in such settings.

(iii) Finally, data collection in our setting is computationally demanding to scale up to a great number of agents due to the need to many solve optimization problems. These optimization problems are challenging to parallelize efficiently, as most solvers use complex, branching control flows better suited for CPUs rather than GPUs, potentially limiting the scalability of our method. To address this limitation, we explored GPU-compatible solvers such as qpth [24] and JAXOpt [25]. While these recently developed solvers were specifically designed to address such scalability challenges, we observed minimal computational advantages over CPU-based solvers given our relatively inexpensive RL experiments. However, tests we performed indicate that these GPU-based solvers demonstrate significantly better scaling properties when handling larger numbers of problems, suggesting they should be preferred over CPU-based alternatives like CVXPYLayers [26] for larger-scale experiments. Scaling up our framework using such GPU-compatible solvers can expand the scope of our method to larger systems of agents, and further exploration is required to find the optimal setup.

7 Acknowledgements

This work was supported by ERC Project 949940 (gAIa).

References

[1] J. Blumenkamp, A. Shankar, M. Bettini, J. Bird, and A. Prorok. The Cambridge RoboMaster: An Agile Multi-Robot Research Platform. In *International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2024.

[2] J. Alonso-Mora, T. Naegeli, R. Siegwart, and P. Beardsley. Collision avoidance for aerial vehicles in multi-agent scenarios. *Autonomous Robots*, 39:101–121, 2015.

[3] W. Merkt, V. Ivan, and S. Vijayakumar. Continuous-time collision avoidance for trajectory optimization in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[4] J. Gregory. *Constrained optimization in the calculus of variations and optimal control theory*. Chapman and Hall/CRC, 2018.

[5] Z. Gao, G. Yang, and A. Prorok. Online control barrier functions for decentralized multi-agent navigation. In *IEEE International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2023.

[6] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[7] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11):4948, 2021.

[8] S. Gronauer and K. Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2):895–943, 2022.

[9] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[10] A. R., E. A., Y. S., and D. S. Actor-Critic Model Predictive Control: Differentiable Optimization meets Reinforcement Learning, 2024.

[11] J. Van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE international conference on robotics and automation*, pages 1928–1935. Ieee, 2008.

[12] M. G. Earl and R. D'Andrea. Modeling and control of a multi-agent system using mixed integer linear programming. In *IEEE Conference on Decision and Control (CDC)*, 2002.

[13] A. Fallahi, J. M. Rosenberger, V. C. P. Chen, W. Lee, and S. Wang. Linear programming for multi-agent demand response. *IEEE Access*, 7:181479–181490, 2019.

[14] J. Nocedal and S. J. Wright. Quadratic programming. *Numerical Optimization*, pages 448–492, 2006.

[15] Q. Nguyen and K. Sreenath. Exponential control barrier functions for enforcing high relative-degree safety-critical constraints. In *IEEE American Control Conference (ACC)*, 2016.

[16] N. Q. H. Tran, I. Prodan, and L. Lefèvre. Nonlinear optimization for multi-agent motion planning in a multi-obstacle environment. In *IEEE International Conference on System Theory, Control and Computing (ICSTCC)*, 2017.

[17] T. Chu, J. Wang, L. Codecà, and Z. Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3): 1086–1095, 2019.

[18] Y. Xue and W. Chen. Multi-agent deep reinforcement learning for UAVs navigation in unknown complex environment. *IEEE Transactions on Intelligent Vehicles*, 2023.

[19] Z. Ning and L. Xie. A survey on multi-agent reinforcement learning and its application. *Journal of Automation and Intelligence*, 2024.

[20] C. Amato. An introduction to centralized training for decentralized execution in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2409.03052*, 2024.

[21] A. V. Fiacco. Introduction to sensitivity and stability analysis in non linear programming. 1983.

[22] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear algebra and its applications*, 284(1-3):193–228, 1998.

[23] S. Nayak, K. Choi, W. Ding, S. Dolan, K. Gopalakrishnan, and H. Balakrishnan. Scalable multi-agent reinforcement learning through intelligent information aggregation. In *International Conference on Machine Learning*, pages 25817–25833. PMLR, 2023.

[24] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning (ICML)*, 2017.

[25] M. Blondel, Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-López, F.and Pedregosa, and J. Vert. Efficient and Modular Implicit Differentiation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[26] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable Convex Optimization Layers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[27] S. Dong. methods for constrained optimization. *Massachusetts Institute of Technology*, 2006.

[28] J. Zhang, F. Lin, S. Ding, and W. Xing. Linear Programming-Based Consensus of Positive Continuous-Time Multi-Agent Systems. *IEEE/CAA Journal of Automatica Sinica*, 11(12): 2519–2521, 2024.

[29] N. Motee and A. Jadbabaie. Distributed multi-parametric quadratic programming. *IEEE Transactions on Automatic Control*, 54(10):2279–2289, 2009.

[30] M. Endo, T. Ibuki, and M. Sampei. Collision-free formation control for quadrotor networks based on distributed quadratic programs. In *IEEE American Control Conference (ACC)*, 2019.

[31] A. Romero, Y. Song, and D. Scaramuzza. Actor-critic model predictive control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[32] X. Sun and C. G. Cassandras. Optimal dynamic formation control of multi-agent systems in constrained environments. *Automatica*, 73:169–179, 2016.

[33] J. Chai and J. K. Hodgins. Constraint-based motion optimization using a statistical dynamic model. In *ACM SIGGRAPH papers*, pages 8–es. 2007.

[34] J. Lin, N. Somani, B. Hu, M. Rickert, and A. Knoll. An efficient and time-optimal trajectory generation approach for waypoints under kinematic constraints and error bounds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[35] A. Muñoyerro, A. Hernández, M. Urizar, and O. Altuzarra. A general automatic method for mechanism optimization based on kinematic constraints and analytical Jacobian matrix. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 237(14):3181–3197, 2023.

[36] A. González-Briones, F. De La Prieta, M. S. Mohamad, S. Omatu, and J. M. Corchado. Multi-agent systems applications in energy optimization problems: A state-of-the-art review. *Energies*, 11(8):1928, 2018.

[37] Z. Gao and A. Prorok. Environment optimization for multi-agent navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[38] Z. Gao and A. Prorok. Constrained environment optimization for prioritized multi-agent navigation. *IEEE Open Journal of Control Systems*, 2023.

[39] S. Kornienko, O. Kornienko, and J. Priese. Application of multi-agent planning to the assignment problem. *Computers in Industry*, 54(3):273–290, 2004.

[40] A. Nedic, A. Ozdaglar, and P. A. Parrilo. Constrained consensus and optimization in multi-agent networks. *IEEE Transactions on Automatic Control*, 55(4):922–938, 2010.

[41] X. Zheng and L. Wang. A multi-agent optimization algorithm for resource constrained project scheduling problem. *Expert Systems with Applications*, 42(15-16):6039–6049, 2015.

[42] L. Buşoniu, R. Babuška, and B. De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in Multi-Agent Systems and Applications-1*, pages 183–221, 2010.

[43] Z. Gao, G. Yang, and A. Prorok. Co-Optimization of Environment and Policies for Decentralized Multi-Agent Navigation. *arXiv preprint arXiv:2403.14583*, 2024.

[44] S. Damadam, M. Zourbakhsh, R. Javidan, and A. Faroughi. An intelligent IoT based traffic light management system: deep reinforcement learning. *Smart Cities*, 5(4):1293–1311, 2022.

[45] X. Pan, M. Liu, F. Zhong, Y. Yang, S.-C. Zhu, and Y. Wang. Mate: Benchmarking multi-agent reinforcement learning in distributed target coverage control. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[46] F. Aydemir and A. Cetin. Multi-agent dynamic area coverage based on reinforcement learning with connected agents. *Computer Systems Science and Engineering*, 45(1), 2023.

[47] C. S. De Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. S. Torr, M. Sun, and S. Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

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

[49] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations (ICLR)*, 2022.

[50] M. Bettini, R. Kortvelesy, J. Blumenkamp, and A. Prorok. VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning. *International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2022.

[51] M. Bettini, A. Prorok, and V. Moens. Benchmarl: Benchmarking multi-agent reinforcement learning. *Journal of Machine Learning Research*, 25(217):1–10, 2024.

[52] J. Blumenkamp, A. Shankar, M. Bettini, J. Bird, and A. Prorok. The Cambridge RoboMaster: An Agile Multi-Robot Research Platform. In *IEEE International Symposium on Distributed Robotic Systems (DARS)*, 2024.

[53] A. Shankar, S. Elbaum, and C. Detweiler. Freyja: A full multirotor system for agile & precise outdoor flights. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

Appendix

A Further Related Work on Constrained Optimization and Multi-Agent RL

Constrained Optimization: Constrained optimization is one of the main tools to address multi-agent motion control problems with system dynamics in the continuous domain. It provides a structured methodology for choosing agent actions, where the objective function for optimization encodes the mission goal and the constraints adhere to certain operational, physical or environmental constraints [4]. Depending on the problem's complexity, various algorithms have been used in the literature [27]. When objective functions and constraints are linear, linear programming efficiently coordinates multi-agent movements [12, 13, 28]. For quadratic objectives (e.g., minimizing energy or path deviation), quadratic programming (QP) and Second-order Cone Programming (SOCP) are widely used [14, 29, 21]. They are often combined with control barrier functions (CBFs) and control Lyapunov functions (CLFs) for collision avoidance and target tracking [15, 5], and find applications in formation control [30] and model-predictive control [31]. These methods integrate system dynamics and constraints to provide solutions in convex settings. When dealing with complex dynamics, nonlinear programming remains a possibility [16, 32], at higher computational cost.

Constraints: In (multi-)agent control, constrained optimization problems generally consider four types of constraints: (i) collision avoidance constraints; (ii) kinematic constraints; (iii) environmental constraints and (iv) task-specific constraints. In particular, collision avoidance is critical for multi-agent motion control to ensure safe and efficient operation in shared spaces, which is a fundamental requirement in many existing works [2, 3]. Kinematic constraints describe limitations on agent movements given their physical configuration and properties, which is important to ensure physically feasible trajectories for multi-agent systems and reduces sim-to-real gap compared to traditional discrete multi-agent path finding methods [33, 34, 35]. Similarly, environmental constraints impose restrictions on multi-agent systems based on characteristics and conditions of the environment in which multi-agent systems operate, including physical constraints such as walls or obstacles and operational constraints such as energy limitations. These constraints play a critical role in ensuring safe, efficient and robust operations of multi-agent systems in real-world environments [36, 37, 38]. Finally, task-specific constraints arise from specific objectives or missions that agents need to accomplish, which are fundamental to align motion planning and control strategies with functional goals of the system, such as goal constraints, time constraints and assignment constraints [39, 40, 41]. These constraints must be carefully designed to avoid conflicting with each other, which could make the optimization problem infeasible.

Learning-based methods for multi-agent control have attracted significant attention in recent years [8]. These methods parameterize the control policies of multi-agent systems with neural networks and train their parameters with gradient-based algorithms, among which multi-agent reinforcement learning (MARL) is highly notable. In MARL, agents learn to make decisions by interacting with the environment and optimizing their policies based on environmental feedback and communication. Agents can be cooperative, competitive or a mix of both [6, 42, 7]. MARL has been applied to multi-agent navigation [18, 43], traffic management [17, 44], coverage control [45, 46], among other domains. Learning-based methods do not need to solve optimization problems and are computationally efficient. However, they lose theoretical guarantees compared to optimization-based methods and do not always converge to a good policy.

B Implementation Details

In our setting, we train agents using Multi-Agent Proximal Policy Optimization (MAPPO) [47], a multi-agent variant of PPO [48] where each agent optimizes its policy using shared experience from the environment while maintaining a centralized value function to improve coordination and stability during training. We find that IPPO–a simpler, more decentralized multi-agent learning version of PPO–also works well.

Both our agents' actor policy and critic policy consist of a Graph Attention Network v2 (`GATv2Conv`) [49] layer that processes agent observations as node and edge features (e.g., node

features might be the agent's goal point, and edge features might consist of relative positions and velocities). The `GATv2Conv` layer computes attention coefficients between connected agents to dynamically weigh the importance of neighboring information, producing updated node. The output of the GNN is decoded by an MLP with one hidden layer (128 units, `Tanh` activation) that maps the GNN embeddings to constraint parameters $\boldsymbol{\theta}_i(t)$.

To solve optimization problems we use CVXPYLayers [26], which has some built-in support for parallelization that enables us to solve large batches of problems faster. We use VMAS [50] as the backbone of our vectorized multi-agent environment.

We find that ReCoDe's performance is not particularly sensitive to choice of hyperparameters and use the defaults of BenchMARL [51] to set the parameters of MAPPO and the GNN layers in all experiments.

C Benefits of the Quadratic Constraint

When considering what type of constraint to augment the handcrafted controller with, one might be tempted to use general linear constraints, expressed as $\mathbf{a}_i^\top \mathbf{u}_i(t) \leq b_i$, where $\mathbf{a}_i \in \mathbb{R}^m$ and $b_i \in \mathbb{R}$ are parameters that the agent can adjust, as these are the simplest and most efficient types of constraints to solve for. While these constraints are beneficial in specific scenarios, and we do find them useful in our ground robot demonstration (see Appendix G), they have an important drawback: if an agent is allowed to parametrize k such constraints, the total number of parameters becomes $k(m+1)$, growing quickly with k and m. However, linear constraints have a serious drawback: a small value of k severely limits the agent's ability to control the outcome of its optimization problem. Specifically, with fewer than $m+1$ linear constraints, it is impossible to specify a bounded region in $\mathbb{R}^m$ that contains an ϵ-ball around a point, which is necessary for fine-grained control. This suggests setting $k \geq m+1$, making the dimension of our action space at least $(m+1)^2$. However, such an action space scales poorly with m, requiring more samples for the agent to explore and learn effective policies, and making it difficult to generalize from limited data. We conclude that this choice of constraints is suboptimal.

We instead augment the handcrafted controller with a single quadratic constraint $\|\mathbf{u}_i(t) - \mathbf{a}_i(t)\|_2 \leq b_i(t) + s$ defining the center of a ball and its radius, as described in the paper. The action space dimension of this constraint is $\dim(\theta_i) = m+1$–significantly smaller than $(m+1)^2$ when using multiple linear constraints. By adjusting $\mathbf{a}_i(t)$ and $b_i(t)$, the agent can position the quadratic constraint's feasible region anywhere in the control input space and adjust its size, allowing for a wide range of control actions. This makes the quadratic constraint a flexible choice.

In spite of the above, there are some situations where we find that learning linear constraints sometimes produces desirable behavior. Please see Appendix G for details.

D Proof of Proposition 3.1

Proof. Consider a feasible desired trajectory $\{(x^*(t), u^*(t))\}_{t=1}^T$. To prove the claim, we must show that we can choose $(a(t), b(t))$ and λ_0 so that the optimal $u^{\text{opt}}(t)$ never strays more than ε from $u^*(t)$. The main subtlety is that $u^{\text{opt}}(t)$ affects $x(t+1)$ and thus $\mathbf{o}_i(t+1)$, potentially changing feasibility at future steps. We handle this by leveraging the continuity of the dynamics and observations. We prove the claim by induction, assuming it holds for any $\varepsilon > 0$ up to time t, and showing it also holds at time $t+1$.

First we show the base case $t = 1$. The agent's state is $x^*(1)$, trivially satisfying $\|x(1) - x^*(1)\| = 0 \leq \varepsilon$. Choose $a(1) = u^*(1)$ and $b(1) = \varepsilon$. This introduces the constraint $\|u^{\text{opt}}(1) - u^*(1)\| \leq \varepsilon + s_0$ to the optimization problem (1), where s_0 is a slack variable. Note that as $\lambda_0 \to \infty$, any positive slack $s_0 > 0$ becomes increasingly costly for the optimization problem. For large enough λ_0, if a solution with $s_0 = 0$ exists, the solver will prefer it over any with $s_0 > 0$. Thus, by assumption, $u^*(1)$ is feasible (it lies within $\mathcal{U}_i^0(\mathbf{o}_i(1))$), and there exists a sufficiently large λ_0 for which our construction will ensure $s_0 = 0$, implying that $u^{\text{opt}}(1)$ must be within the ε-ball around $u^*(1)$ to remain optimal. Hence, $\|u^{\text{opt}}(1) - u^*(1)\| \leq \varepsilon$. This establishes the inductive claim for $t = 1$.

Now assume the induction holds up to t, and we shall show it holds for $t+1 \leq T$. By assumption, for any ε_0, there exists a sequence of quadratic constraint parameters that results in our agent having state $\|x(t) - x^*(t)\| < \varepsilon_0$ at time t. Moreover, there is a quadratic constraint such that the unique optimal solution $u^{\text{opt}}(t)$ of the optimization problem (1) satisfies $\|u^{\text{opt}}(t) - u^*(t)\| \leq \varepsilon_0$. We know executing action $u^*(t)$ from state $x^*(t)$ results in our agent having state $x^*(t+1)$. By continuity, for any $\epsilon_1 > 0$ there exists small enough ε_0 such that executing $u^{\text{opt}}(t)$ results in our agent having state $x(t+1)$ satisfying $\|x(t+1) - x^*(t+1)\| < \varepsilon_1$. By assumption, $u^*(t+1)$ is strictly feasible when our agent's state is $x^*(t+1)$. By our continuity assumptions regarding (1), we may choose $\varepsilon_1 > 0$ to be small enough such that $u^*(t+1)$ is strictly feasible when our agent's state is $x(t+1)$. Given such a ε_1, as in the base case, we can set $a(t+1) = u^*(t+1)$ and $b(t+1) = \varepsilon$ and fix λ_0 large enough to ensure $\|u^{\text{opt}}(t+1) - u^*(t+1)\| \leq \varepsilon$. This establishes the inductive claim for $t+1$. □

E Proof of Proposition 3.2

Proof. By the directional–derivative assumption, integrating along $x \mapsto a(\mathbf{o}) + x\mathbf{d}$ for $x \in [0, r]$ gives $-J_i\big(\mathbf{o}, a(\mathbf{o}) + r\mathbf{d}\big) \;\geq\; -J_i\big(\mathbf{o}, a(\mathbf{o})\big) + r\delta_2$. A sufficiently large λ_0 forces $u^{\text{opt}} \in B_r\big(a(\mathbf{o})\big)$; since the solver *minimises* J_i, hence *maximises* $-J_i$ on that ball,

$$-J_i\big(\mathbf{o}, u^{\text{opt}}(\mathbf{o})\big) \;\geq\; -J_i\big(\mathbf{o}, a(\mathbf{o}) + r\mathbf{d}\big) \;\geq\; -J_i\big(\mathbf{o}, a(\mathbf{o})\big) + r\delta_2.$$

Because $\|\nabla_u Q_i^l(\mathbf{o}, u)\|_2 \leq \delta_1$ in the same ball, $Q_i^l\big(\mathbf{o}, u^{\text{opt}}(\mathbf{o})\big) \;\geq\; Q_i^l\big(\mathbf{o}, a(\mathbf{o})\big) - r\delta_1$. Combining the two bounds, $c_1 Q_i^l(\mathbf{o}, u^{\text{opt}}) - c_2 J_i(\mathbf{o}, u^{\text{opt}}) \;\geq\; c_1 Q_i^l(\mathbf{o}, a) - c_2 J_i(\mathbf{o}, a) + r\Delta$. Finally, since $\big|Q^*(\mathbf{o}, u) - [c_1 Q_i^l(\mathbf{o}, u) - c_2 J_i(\mathbf{o}, u)]\big| \leq \varepsilon$ throughout the ball, we obtain

$$Q^*(\mathbf{o}, u^{\text{opt}}) + \varepsilon \;\geq\; c_1 Q_i^l(\mathbf{o}, u^{\text{opt}}) - c_2 J_i(\mathbf{o}, u^{\text{opt}}) \;\geq\; c_1 Q_i^l(\mathbf{o}, a) - c_2 J_i(\mathbf{o}, a) + r\Delta \;\geq\; Q^*(\mathbf{o}, a) - \varepsilon + r\Delta.$$

Rearranging yields the stated inequality. □

F Evaluation Scenarios in Detail

Experiment: Narrow Corridor. In this experiment, agents initiated at random locations in a narrow corridor seek to either get to the blue or green region. Agents receive a reward of -10 for bumping into each other or the corridor boundaries, 1 for every time step they spend in the correct region, and a small reward whenever they take a step that brings them closer to this region.

Our handcrafted controller seeks to prevent collisions while sending agents to their target region. To avoid collisions, the ego agent's controller introduces a CBF defined for each agent j: $h_j(\mathbf{p}_{\text{ego}}, \mathbf{p}_j) = k(\|\mathbf{p}_{\text{ego}} - \mathbf{p}_j\|_2^2 - d_{\min}^2)$, where $d_{\min} > 0$ is a safe distance threshold, $\mathbf{p}$ denotes position, and k is a tuneable constant. This function h_j is positive if the ego agent is at a distance greater than $d_{\min}$ from agent j, and negative if too close. The collision avoidance constraints use h_j to ensure that, by properly choosing $\mathbf{u}$, the ego agent moves in a direction that maintains or increases this safety margin, thereby preventing collisions. At each time step, the ego agent solves (omitting some low-level technical details):

$$\begin{aligned} \max_{\mathbf{u}, s} \quad & d_i \cdot u_y \\ \text{s.t.} \quad & \forall j, 2\big((x_{\text{ego}} - x_j)u_x + (y_{\text{ego}} - y_j)u_y\big) + h_j(\mathbf{p}_{\text{ego}}, \mathbf{p}_j) \geq 0 \\ & (\mathbf{p}_{\text{ego}} + \mathbf{u}) \in \mathcal{B}, \|\mathbf{u}\|_2 \leq M \end{aligned} \tag{2}$$

Here, $\mathcal{B} = [-X_{\max}, X_{\max}] \times [-Y_{\max}, Y_{\max}]$ denotes the boundaries of the environment; $d \in \{-1, 1\}$ indicates the target direction of the agent; $\mathbf{u}$ is a decision variable denoting the agent's velocity control input; the first constraint handles agent-agent collisions through the control barrier function; the second constraint prevents agents from going out of bounds; and the third constraint is a velocity limit. To use ReCoDe in this controller, all we need to do is introduce a slack decision variable $s \geq 0$ and a quadratic constraint on top of existing constraints, as shown in (1). Both the handcrafted controller and the ReCoDe modification are efficiently solvable [22].

Navigating this scenario demands sophisticated agent coordination, as robots must not only avoid collisions but also work in concert to prevent gridlock. While traditional handcrafted controllers

can manage basic collision avoidance, they frequently succumb to deadlocks due to their inability to facilitate inter-agent coordination, as illustrated in Figure 2. These limitations expose a fundamental weakness in conventional control approaches that rely on constrained optimization: without a mechanism for coordination, agents cannot make cooperative decisions like yielding to resolve impasses. In contrast, our ReCoDe framework overcomes these challenges.

Experiment: Connectivity. In this experiment, agents in a narrow corridor must move to the green region, while never breaking communication links: every pair of agents needs to always stay within each others' communication range. We introduce static obstacles into the environment that the agents must learn to bypass while maintaining this connectivity requirement. We prevent agent actions that would break this connectivity and give negative rewards to actions that attempt this. Reward structure is otherwise the same as the previous experiment: a positive reward for reaching the green region, and a negative reward for colliding.

Our handcrafted controller is based on (2), with the main difference being the introduction of a quadratic constraint on distance: $\|\mathbf{p}_{\text{ego}} - \mathbf{p}_j\|_2^2 \leq d_{\min}^2$.

This problem requires coordination between the agents: an agent greedily attempting to move to the green region might result in a deadlock, as it leaves too little leeway for other agents to bypass obstacles–see Figure 2. While our handcrafted controller struggles in such situations, ReCoDe handles them successfully (see Table 2).

Experiment: Waypoint Navigation. In this experiment, large agents in a small environment must navigate to their respective goal points (Figure 2). The agents and goal points are initiated randomly. At each time step, agents receive a reward proportional to $d_{prev} - d_{current}$ (their previous distance to their goal minus their current distance), as well as a large, discrete reward when they get sufficiently close. A penalty reward of -10 is given when agents collide. The scenario is similar to Narrow Corridor in that agents must navigate to a goal point, but unlike Narrow Corridor, agents have a greater diversity of possible goal points and strategies, enabling us to test whether ReCoDe can explore diverse strategies. Our handcrafted controller is based on (2), with only the objective function changed to minimize an agent's distance to its respective goal point.

Experiment: Sensor Coverage. *Sensor Coverage* is a multi-objective task that poses a consensus challenge alongside a navigation challenge. In it, decentralized mobile sensors monitor different phenomena (e.g. wildlife or pollution) distributed across different locations–see Figure 4. The sensors are placed in an environment with obstacles. Each sensor aims to maximize accuracy by getting as close as possible to its assigned sensing target (which is unique to it), but the same formation constraints as in the *Connectivity* scenario apply: sensors must maintain all communication links by staying together. Due to this formation constraint, the sensors must choose which targets to prioritize and decide on the best location for overall coverage, which might, e.g., be directly on top of some assigned targets, or somewhere near the center of the targets' convex hull, not reaching any target but weakly covering all targets, thus maximizing cumulative reward. The agents must also all coordinate about how to move in formation to their desired location.

At every time-step the total reward r_i for sensor i is $r_i = r_i^{\text{prox}} + r_i^{\text{safety}}$. Here, $r_i^{\text{prox}} = e^{-\lambda_{prox}\left\|\mathbf{p}_i - \mathbf{g}_i\right\|^2}$, where λ_{prox} is a hyperparameter, $\mathbf{p}_i$ is the sensor's position and $\mathbf{g}_i$ the centre of its sensing target. r_i^{safety} denotes a reward penalty for unsafe behavior–collisions, going out of bounds, and breaking formation constraints, as in the *Connectivity* scenario.

The handcrafted QP is identical to (2) *except* that the objective minimizes the distance-to-goal $\|\mathbf{p}_i + \mathbf{u}_i - \mathbf{g}_i\|_2^2$. This kind of objective is suboptimal since it pulls agents in different directions (due to having different goals), but they must stay connected, hence the entire cloud of agents can get stuck in a deadlock. However, it is the best we could find for a quadratic constrained optimization program, and it performs on par with other baselines except ReCoDe–see Table 2.

This is a multi-objective problem requiring decentralized consensus where the sensors must determine *where* to move and *how* to get there effectively. Discovering, and agreeing, on a solution to pursue along the Pareto frontier of optimal trajectories and final locations is hard. Shielding and

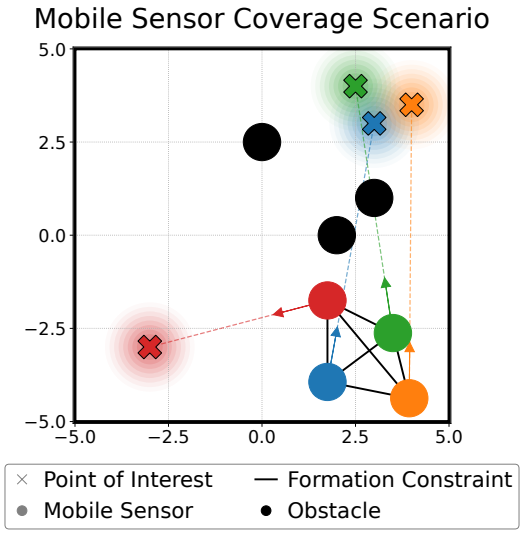


Figure 4: Visual depiction of the *Sensor Coverage* experiment. Sensors (the Os) increase the reward by approaching their color-matched Point of Interest (the Xs) but are constrained by the need to maintain formation with other sensors.

Pure MARL have a difficulty optimizing this delicate trade-off, and the fixed QP stalls when two attractive targets pull the team in opposite directions. ReCoDe's learned constraints let neighboring sensors negotiate their trajectory, while locally relying on the default QP's (imperfect) objective function to further improve performance–thus achieving superior reward on this task.

G Additional Details on Robot Demonstrations

The platform we used is the Cambridge Robomaster [52]. We used the Freyja library [53] for executing velocity commands. Our simulator was implemented using the `robomaster` branch of the BenchMARL repository [51] as a base. We implemented our Narrow Corridor scenario in this repository, and manually calibrated the robot controller parameters and arena dimensions to attain sim2real. The default QP used for the robots is the one defined for the Narrow Corridor scenario in Appendix F, lightly calibrated to attain better sim2real performance. We used motion capture for localization, and messages were only passed between robots within each others' communication range ($1.5\,\mathrm{m}$). However, robots were only made aware of *relative spatial information* about neighboring robots and their own coordinates (relative velocity and position).

What if we learn linear constraints instead of quadratic constraints? We experimented with having robots learn a single linear constraint of the form $\mathbf{a}_i^\top \mathbf{u}_i(t) \leq b_i$, where $\mathbf{a}_i \in \mathbb{R}^m$ and $b_i \in \mathbb{R}$ are parameters that the agent can adjust. In Appendix C we explain why these kind of constraints are generally inefficient. However, because the default QP we used for the Narrow Corridor scenario already introduces other constraints (control barrier function and boundary constraints to prevent collisions), linear constraints become more expressive, as they can *intersect* these other constraints and bound various kinds of volumes. Thus, we speculated that they will perform well in this scenario.

We found that using the linear constraint, although it is less principled, neither decreases nor increases average reward or robomaster performance. However, it seems to enforce different kinds of behaviors: while the quadratic constraint sometimes drove robots to attain *partial* solutions of the Narrow Corridor environment (by having only some of them reach their goal), the linear constraint-trained robots seemed to adapt an "all or nothing" strategy, where either all robots reach their goal, or none of them do. It is interesting that both strategies led to the same average reward despite these different behavioral modes. In future work, we are interested in exploring the effects of different kinds of constraints on the agents' learned policy.