

GAMBIT: Learning to Plan Continuous Multi-Robot Trajectories

Rishabh Jain¹, Akmaral Moldagalieva², Lorenzo Magnino¹, Michael Amir¹,
Keisuke Okumura^{1,3}, Ajay Shankar¹, Wolfgang Hönig² and Amanda Prorok¹

Abstract—GAMBIT is an opening chess move in which a player sacrifices a piece, typically a pawn, to gain a positional advantage later in the game. Analogously, in multi-robot coordination, individual robots may need to forgo locally reward-maximising behaviours to improve overall team performance. Such self-sacrificial behaviours are difficult to capture with manually designed heuristics, particularly in dense, interaction-rich environments. Focusing on double-integrator continuous dynamics, this work studies how to learn such coordinated heuristics over motion primitives for multi-robot trajectory execution. Our framework, GAMBIT, first learns coordinated motion-primitive selection through imitation learning and subsequently fine-tunes the policy through reinforcement learning. We further introduce a safeguarded rollout mechanism with backup trajectories that guarantees collision-free execution at all times. Experiments demonstrate that GAMBIT substantially outperforms a range of baselines, including centralised motion planners and decentralised reactive planners, while exhibiting strong scalability. In particular, it coordinates over a thousand robots with planning latency below a few hundred milliseconds in continuous domains.

I. INTRODUCTION

Generating trajectory plans using *motion primitives* has been a successful paradigm for deriving potentially long-horizon, feasible motions to steer a robot—with applications ranging from autonomous driving to manoeuvres of aerial robots [1]–[3]. While its multi-robot counterpart has also been studied [4], [5], a critical challenge in achieving fast and scalable planning remains the design of effective *heuristics* that evaluate how “good” each primitive is. Designing such a heuristic is challenging even in the single-robot case, as accurate cost-to-go estimates become difficult to obtain for higher-order dynamics beyond simple point-mass models; the multi-robot case further requires reasoning about coordination among agents. One could design a trivial heuristic based on an interaction-agnostic single-robot heuristic, but such approximations can get significantly distorted in interaction-rich multi-agent environments, where effective coordination may require non-greedy motions, such as temporarily moving away from the goal to accommodate other agents.

Using naïve or inaccurate heuristics for motion primitives can misguide the planning process, resulting in either con-

siderable computational effort or highly suboptimal robot trajectories. Manually capturing such coordination requires substantial engineering effort that scales poorly with the complexity of interactions. This makes automatically *learning* these heuristics particularly attractive. This study investigates how to realise this idea.

Learning agent-wise heuristics to synthesise coordination has proven effective in the discrete-state, *grid-world multi-agent pathfinding (MAPF)* domain [6]–[8], where the objective is to find conflict-free paths for multiple agents on a four-connected grid. Recent work has developed high-performing heuristics using decentralised neural MAPF policies [9], where each agent receives an observation and outputs an action distribution, trained using either reinforcement learning (RL) or imitation learning (IL) from expert data. A natural question then arises: can we effectively transfer this knowledge to *continuous* state spaces?

With some adaptation, this transfer appears possible using action synthesis based on motion primitives; however, the transition to continuous state spaces introduces fundamental challenges. In the context of IL, acquiring a substantial amount of expert data for scenarios involving many agents is considerably more difficult than in MAPF, where established and reliable planners are readily available [10], a challenge we hereinafter refer to as **data scarcity**. The **multi-modality** of solutions poses another challenge: trajectories belonging to different homotopy classes can have comparable costs, making it difficult to assign a unique target action to imitate. The **safeguarding gap** in rollouts—i.e., the lack of mechanisms that clamp collision-causing actions and facilitate learning in strong neural MAPF policies [11]–[13]—weakens the reward signal, making training more difficult, particularly for RL. Along with difficulties in **scenario generation** for providing informative training signals, these challenges collectively make it difficult to design advanced heuristics for multi-robot motion primitives.

We address these challenges with a framework we term GAMBIT: *Graph-Attentive Motion primitives with Backup for Interactive Trajectories*. Following the success of neural MAPF policies, our backbone architecture is based on a graph neural network (GNN) that outputs a categorical action distribution [8], [14], exploiting geometric locality to enable training on small-scale instances and deployment to larger teams. The model is first pretrained via IL on instances with a small number of agents, using solutions generated by a centralised kinodynamic multi-robot motion planner [5], as sufficient data are attainable at this scale to overcome **data scarcity**. The model is then fine-tuned via RL to mitigate

This study was partially supported by Trinity College, Cambridge, European Research Council (ERC) Project 949940 (gAIA), JST PRESTO (JPMJPR2513) and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 448549715.

¹University of Cambridge, UK {rj412, lm2183, ko393, ma2151, as3233, asp45}@cst.cam.ac.uk

²Technical University of Berlin, Germany {moldagalieva, hoenig}@tu-berlin.de

³National Institute of Advanced Industrial Science and Technology (AIST), Japan

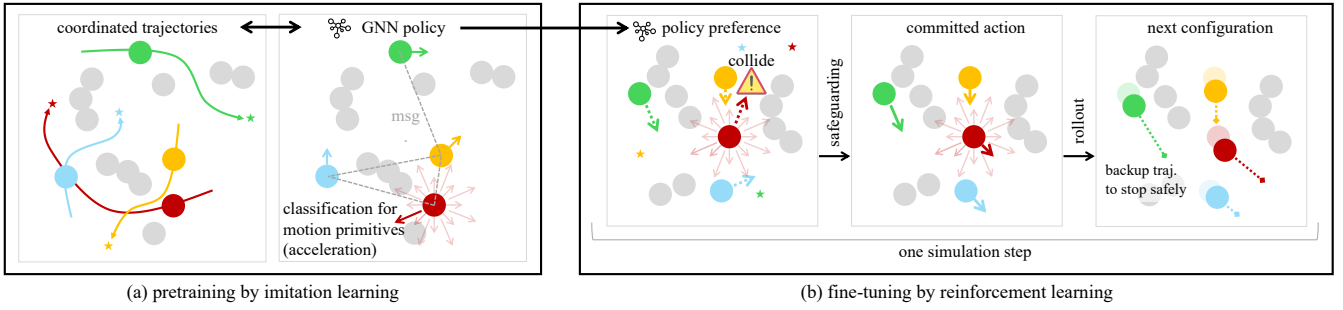


Fig. 1. GAMBIT framework overview. We aim to learn a coordination-aware GNN policy that provides heuristic preferences over motion primitives. The framework consists of (a) imitation pretraining and (b) RL fine-tuning. The former step learns from demonstration trajectories by treating motion-primitive selection as a classification task. The latter uses multi-agent RL to maximise multi-robot navigation performance through our safeguarded rollout mechanism. The resulting policy, combined with the safeguarded rollout, enables collision-free execution while retaining high scalability.

the **multi-modality** problem by biasing the policy towards consistent action choices. RL training uses newly developed safety-preserving rollouts with backup trajectories that overcome the **safeguarding gap** and stabilise the learning process. Effective learning is further facilitated by carefully designed **scenario generation** that ensures agent interactions in every training instance. Figure 1 illustrates this pipeline.

The resulting GAMBIT policy, combined with motion primitives and our specialised rollout, synthesises safe continuous actions for hundreds of robots at sub-hundred-millisecond latency, while producing substantially better-coordinated trajectories in densely populated environments than reactive planners [15], [16], computationally intensive centralised planners [5], [17], and existing learning-based approaches [18]. We further validate GAMBIT through a physical deployment of eight multirotors [19] with closed-loop control. Across both simulated and real-robot deployments, our backup mechanism guarantees safe and feasible execution at all times.

As a motion-primitive heuristic constitutes a lightweight and modular planning component, we view GAMBIT as a promising building block for designing capable planners beyond what we present here. Furthermore, as policy rollout relies only on local interactions between robots, this scheme is also amenable to decentralised deployment. In what follows, we present the problem formulation, training pipeline, safeguarding rollout, and experimental evaluation, in order.

II. PRELIMINARIES AND RELATED WORK

A. Problem Formulation

We consider point-to-point navigation for a team of homogeneous spherical robots $A = \{1, \dots, n\}$ with radius $R_{\text{robot}} \in \mathbb{R}^+$ under double-integrator dynamics, operating in a 2D or 3D workspace $W \subseteq \mathbb{R}^d$, $d \in \{2, 3\}$, containing obstacles. Each robot $i \in A$ has its own start and goal positions $\mathbf{s}_i, \mathbf{g}_i \in W$. Its state is given by its position and velocity, $\mathbf{x}_i = [\mathbf{p}_i, \mathbf{v}_i]$, while its control input is its acceleration, $\mathbf{u}_i = \mathbf{a}_i$. The robot dynamics are given by

$$\mathbf{x}_i^{k+1} = \begin{bmatrix} \mathbf{p}_i^k + \Delta t \mathbf{v}_i^k + \frac{1}{2} \Delta t^2 \mathbf{a}_i^k \\ \mathbf{v}_i^k + \Delta t \mathbf{a}_i^k \end{bmatrix}, \quad (1)$$

where Δt denotes the timestep. This corresponds to the exact discretisation of the double-integrator dynamics under constant acceleration over each timestep.

Our objective is then to derive a kinodynamically consistent state-control trajectory of a priori unknown length $T \in \mathbb{N}^+$, $[\langle \mathbf{x}_i^0, \mathbf{u}_i^0 \rangle, \dots, \langle \mathbf{x}_i^T, \mathbf{u}_i^T \rangle]$ for each robot to navigate from its start to goal without collisions with either obstacles or other robots, satisfying:

- $\mathbf{x}_i^0 = [\mathbf{s}_i, \mathbf{0}]$;
- $\mathbf{x}_i^T = [\mathbf{p}_i^T, \mathbf{v}_i^T]$, with $\|\mathbf{p}_i^T - \mathbf{g}_i\| \leq R_{\text{goal}}$ and $\|\mathbf{v}_i^T\| \leq \epsilon$;
- $\|\mathbf{p}_i^k - \mathbf{p}_j^k\| > 2R_{\text{robot}}$ for all $i \neq j \in A$; and
- $\mathbf{p}_i^k$ does not collide with obstacles.

Here, $R_{\text{goal}} \in \mathbb{R}^+$ denotes the goal radius, $\mathbf{p}_i^k$ denotes the positional component of state $\mathbf{x}_i^k$, and ϵ denotes a sufficiently small number for the velocity threshold. The solution quality is assessed by the *sum-of-costs*, defined as the sum of travel times over all robots until each robot reaches its goal and remains there indefinitely.

Herein, we use ‘agent’ and ‘robot’ interchangeably. A *configuration* $Q = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ refers to a list of states for all agents. We assume access to a collision-checking function *collide* that determines whether a given state is in collision, following a standard assumption in motion planning [20].

B. Planning with Motion Primitives

A common approach to deriving such control sequences is via numerical optimisation, e.g., formulating trajectory optimisation as a non-linear program subject to dynamics and model constraints over a continuous control space, and solving it in a potentially decentralised manner [16], [21]. In contrast, this work follows a planning approach based on motion primitives, which restricts the infinitely many possible controls to a finite set, thus facilitating rapid discovery of coordinated actions. Motion primitives have been explored for multi-robot navigation [5], [22], [23], demonstrating their potential for computationally efficient coordination, but they typically rely on manually designed mechanisms for selecting the right primitive. **Our primary objective is to automate this selection process through learning.** Accordingly, we distinguish GAMBIT from learning-based multi-agent control approaches that directly output continuous

controls (e.g., [24], [25]); instead, we formulate the learning problem as one of ranking the available primitives, making the learned heuristic naturally compatible with numerous global search-based planning methods. Meanwhile, we also provide a safeguarded rollout scheme with motion primitives tailored to double integrator dynamics, demonstrating how the learned heuristic can be integrated into a practical deployment framework.

We define a *motion primitive* as a finite sequence of control inputs $m = [\mathbf{u}^0, \mathbf{u}^1, \dots, \mathbf{u}^{l-1}]$. When applied to an initial robot state $\mathbf{x}^0$, a motion primitive induces a rollout state trajectory $[\mathbf{x}^0, \mathbf{x}^1, \dots, \mathbf{x}^l]$. In this study, for simplicity, we set $l = 1$; i.e., each primitive represents a single acceleration command $\mathbf{a} \in \mathbb{R}^d$, although the proposed framework can be extended to larger values of l . We denote the set of available motion primitives by M . Our objective is then to learn a *ranking function* that ranks the primitives $m \in M$ for robot i given the current configuration Q ; we interchangeably refer to this function as a *heuristic*, as it virtually assesses the quality of each motion primitive.

While learning for motion-primitive-based planning has been studied in single-robot settings [26], [27], our primary interest lies in multi-robot systems, which present challenges that stem from the interplay between agents.

C. Neural MAPF Policies

Learning heuristics for multi-agent action selection has been studied extensively in the context of grid-world MAPF [9], under highly simplified discrete dynamics. Typically, this line of work aims to design a policy that receives a local observation o_i for agent i and outputs a distribution over five discrete actions: NORTH, WEST, SOUTH, EAST, and STAY. To obtain effective policies, established learning pipelines use either imitation learning (IL) or reinforcement learning (RL) [14], [28]. We aim to leverage these insights beyond grid-world dynamics; however, several challenges hinder their direct adaptation to our setting, as highlighted in Sec. I; that is, **data scarcity**, **multi-modality**, **safeguarding gap**, and **scenario generation**. Below, we provide our GAMBIT approach to address these.

III. GAMBIT PIPELINE

The GAMBIT framework centres around a GNN-based policy, together with its training pipeline, comprising IL pretraining and RL fine-tuning, and a safety-preserving rollout scheme. This section introduces the backbone policy architecture and then describes each component in turn.

A. Policy Architecture

While our pipeline is not restricted to a specific model architecture, we follow a successful paradigm adopted by GNN-based neural MAPF policies [8], [14]. The geometric locality of GNNs allows us to train the policy on instances with only a small number of robots while effectively capturing interactions between robots, and to deploy the learned policy to substantially larger teams at test time. Specifically,

the policy π consists of (i) an encoder ϕ_{enc} , (ii) graph neural network layers ϕ_{comm} that encode interactions between agents, and (iii) an MLP decoder ϕ_{dec} that outputs an action distribution over the set of motion primitives M .

1) *Observation Encoding*: An observation for robot i , denoted by o_i , is the concatenation of (a) its velocity, (b) the relative goal vector, (c) a field-of-view (FOV) representation of its surroundings f_i , and (d) a motion-primitive embedding ξ_i . The FOV f_i is represented as a three-channel d -dimensional grid with N_{obs} cells along each dimension. The first two channels encode obstacle and agent occupancy, respectively, while the third encodes the goal location. When the goal lies outside the FOV, its location is clipped to the FOV boundary, following prior learning-based MAPF approaches [14], [28]. The motion-primitive embedding ξ_i encodes the outcomes of the available primitives. For each $m \in M$, we compute the relative displacement after executing m followed by stopping, together with the corresponding relative goal vector. Their concatenation is processed by an MLP and averaged over all primitives to obtain ξ_i . Finally, the complete observation is encoded as $z_i^0 = \phi_{\text{enc}}(o_i)$.

2) *Interaction Modelling and Decoding*: For a configuration Q , we introduce a communication graph G_{comm} , where each vertex represents a robot $i \in A$ and an edge $i-j$ exists if robots i and j are sufficiently close in position, i.e., within a communication radius R_{comm} . The communication module ϕ_{comm} propagates the robot-wise embeddings z_i^0 through L GNN message-passing layers over G_{comm} using scaled dot-product attention [29], with pairwise relative positions as edge features. Finally, based on the resulting representation z_i^L , the decoder ϕ_{dec} outputs an action distribution over M .

B. Pretraining by Imitation Learning

Pretraining by IL followed by RL fine-tuning has emerged as a successful strategy for learning capable policies across domains, ranging from language modelling [30] to robot learning [31]; we follow this strategy. For expert data collection, we employ an adapted version of db-LaCAM [5], an offline centralised planner, in which its low-level configuration generation is replaced with our enhanced PIBT procedure described in Sec. III-D, enabling long-horizon kinodynamic multi-robot motion planning as defined in Sec. II-A. We generate demonstration trajectories by running the planner on instances involving a few agents. We limit the team size because collecting demonstrations for large teams is computationally expensive even with leading planners in this domain, making the construction of a large corpus prohibitively costly. Thanks to the locality of GNNs, we instead generate small instances that require tight local coordination, allowing the learned interactions to generalise to larger teams to overcome **data scarcity**.

Scenario generation is therefore a key component for successful learning. In particular, we prepare two families of instances, (i) *Blocked Random* and (ii) *Forest*, visualised in Fig. 2. Given the number of robots, the former first places start-goal pairs at random positions and then creates a small wall between these positions by placing obstacles

perpendicularly to the line segments connecting each start-goal pair, with some random perturbations. Unlike purely random instance generation, this structure is more likely to require coordination. The latter contains regularly placed obstacles that form numerous narrow passages, while start-goal pairs are sampled randomly, exposing the policy to obstacle avoidance and potential deadlocks with other robots.

Our IL formulation treats motion primitive selection as a classification problem, rather than directly regressing continuous actions. Accordingly, for each continuous expert action $\mathbf{u}$, we construct its discrete target as the closest motion primitive $m^* \in M$ and train the policy to predict m^* using cross-entropy loss. This formulation offers two advantages over directly regressing continuous actions followed by quantisation: (i) more stable learning, as we observed empirically, and (ii) a categorical distribution over M , which can be directly exploited by our safeguarded rollout scheme.

C. Fine-tuning by Reinforcement Learning

While IL provides the basic skills required for multi-robot navigation, IL alone does not necessarily yield a skilful policy. In addition to the solution **multi-modality**, this limitation is amplified by the fact that computing high-quality trajectories for motion planning is itself computationally challenging [32], and we therefore rely on suboptimal solutions obtained by expert planners within a practical time budget. Consequently, the demonstration dataset can contain suboptimal behaviours, which are inherited by the policy through imitation. We address this limitation through RL fine-tuning, allowing the policy to directly optimise solution quality. Empirically, we find that training with RL from scratch is unstable, whereas the two-stage IL-RL procedure yields stable training and a capable policy.

For this RL phase, we use the same instance distribution as in the IL phase and employ MAPPO [33] with a simple reward directly focused on optimising the sum-of-costs and an additional reward for all robots reaching their goals. Let Q denote the rollout configuration obtained by applying the actions selected by the policy to the input configuration. For each robot i , we define the binary goal indicator, $\mathbb{I}_i(Q) := \mathbb{I}[i \text{ is at its goal in } Q]$. The reward is then defined as

$$w_1 \prod_{i \in A} \mathbb{I}_i(Q) - w_2 \sum_{i \in A} (1 - \mathbb{I}_i(Q)), \quad (2)$$

where $w_1, w_2 > 0$ are weighting coefficients. The first term provides a team-level bonus when all robots reach their goals, encouraging successful planning as a team. The second term penalises each robot that has not yet reached its goal at every timestep, thereby encouraging lower sum-of-costs.

Notably, Eq. (2) contains no collision penalty for either obstacles or other robots. Incorporating such penalties into the reward can dilute the goal-directed learning signal; instead, we address the **safeguarding gap** through a specialised rollout scheme for double-integrator dynamics that prevents collisions algorithmically.

D. Safeguarded Rollout with Backup Trajectories

Recall that the policy outputs an action distribution $\pi(o_i)$ for robot i over the set of motion primitives M , which by itself does not determine the action to execute. A straightforward strategy is to sample a primitive $m \in M$ according to $\pi(o_i)$; however, the resulting rollout under Eq. (1) is not necessarily collision-free. Rather than penalising unsafe primitives during learning, we develop a provably safe rollout scheme by exploiting backup trajectories, analogous in spirit to control barrier functions for continuous control [34].

1) *Safe Execution with Static Obstacles*: We first illustrate the basic idea for robot i in isolation. Given a state $\mathbf{x}_i$ and a motion primitive $m \in M$, we define a *stopping trajectory* $\tau_{\text{stop}}(\mathbf{x}_i, m)$ by first applying m to $\mathbf{x}_i$ and subsequently applying maximum acceleration opposite to the velocity until the robot comes to a complete stop. This results in a state-control trajectory $[\langle \mathbf{x}_i^0 (= \mathbf{x}_i), m \rangle, \langle \mathbf{x}_i^1, -\mathbf{v}/|\mathbf{v}| \cdot \min\{a_{\text{max}}, \|\mathbf{v}\|/\Delta t\} \rangle, \dots, \langle \mathbf{x}_{\text{stop}}, \mathbf{0} \rangle]$, where $\mathbf{v}$ represents the velocity component of $\mathbf{x}_i$ and $\mathbf{x}_{\text{stop}}$ has zero velocity. We filter out motion primitives whose corresponding stopping trajectories collide with obstacles, yielding a *safe primitive set* $M^{\text{safe}}(\mathbf{x}_i) \subseteq M$, abbreviated as M_i^{safe} when the state $\mathbf{x}_i$ is clear from context. This set is safe in the sense that, for any primitive selected from M_i^{safe} , a collision-free trajectory that brings the robot to a complete stop is always available.

Now, suppose that at timestep $t-1$, robot i at state $\mathbf{x}_i^{t-1}$ executes a primitive $m \in M^{\text{safe}}(\mathbf{x}_i^{t-1})$, resulting in the next state $\mathbf{x}_i^t$. If $M^{\text{safe}}(\mathbf{x}_i^t)$ is non-empty, selecting a primitive from this set again ensures the availability of a collision-free stopping trajectory. If $M^{\text{safe}}(\mathbf{x}_i^t)$ is empty, however, the robot can continue executing the remaining portion of $\tau_{\text{stop}}(\mathbf{x}_i^{t-1}, m)$, which is guaranteed to bring the robot to a stop without collision. We refer to this stopping trajectory retained from the previously selected primitive as the *backup trajectory*. This rollout strategy, which switches to the backup trajectory if no safe primitive is available, prevents the robot from entering a dead-end state, a state from which no collision-free action sequence exists. Given that $\mathbf{x}_i^0$ is collision-free and has zero velocity, simple induction shows that this execution scheme guarantees collision-free motion with respect to obstacles.

2) *Safe Execution with Other Robots*: We now leverage the backup trajectory scheme for safe multi-robot execution by integrating it with *PIBT collision shielding* [11], [12], a successful paradigm for substantially improving the performance of neural MAPF policies. In the context of grid-world MAPF, PIBT receives an agent-wise ordered list of one-step actions, called a *preference*, and greedily assigns actions to construct a collision-free configuration from the current one. Such preferences can be constructed, for example, by ordering actions according to their cost-to-go values to individual goals or directly from the outputs of neural policies. PIBT guarantees that a feasible action assignment can always be found, with this feasibility relying on the availability of the ‘stay’ action as a safe fallback when preferred actions cannot be assigned, provided that interacting agents can likewise fall

Algorithm 1 Safeguarded rollout with PIBT and backups

input: config $(x_1, \dots, x_n)$, backup actions $(b_1, \dots, b_n)$
output: action m_i for each robot $i \in A$, initialised with $\perp$

```
1: for  $i \in A$  do if  $m_i = \perp$  then PIBT( $i$ )
2: procedure PIBT( $i$ )
3:   sort  $M_i^{\text{safe}}$   $\triangleright$  based on  $\pi$  in GAMBIT
4:   for  $m \in M_i^{\text{safe}}$  do
5:     if collide_stop( $i, m$ ) then continue
6:      $m_i \leftarrow m$ 
7:     while  $\exists j \in A, m_j = \perp, \text{collide\_stop}(j, b_j)$  do
8:       if PIBT( $j$ ) = invalid then break
9:       if no invalid received then return valid
10:   $m_i \leftarrow b_i$ 
11: return invalid
```

back to their stay actions.

In kinodynamic settings, however, such a stay action is not always available due to non-zero velocity. PIBT has been extended to general motion primitives [5]; while this variant guarantees collision-free outcomes, it does not guarantee that a feasible action assignment can always be found, which is particularly critical in dense scenarios. Our key observation is that, under double-integrator dynamics, the backup trajectories introduced above can play a role analogous to the ‘stay’ action, thereby providing both safety, i.e., collision-free execution, and feasibility, i.e., a feasible action assignment can always be found in finite time.

Algorithm 1 provides a concrete procedure for this safeguarded rollout, which takes as input the current state x_i and the first action b_i of the backup trajectory for each robot i . Its overall structure follows the original PIBT algorithm, where a recursive subprocedure PIBT (Line 2–) attempts to assign an action greedily and is invoked sequentially for each agent (Line 1). The preference over actions is determined at Line 3 and can in principle be defined arbitrarily; in this study, we stochastically sample the order of motion primitives, based on the probabilities assigned by the policy π . The subprocedure returns *valid* if robot i successfully secures a non-colliding action; otherwise, it returns *invalid*, forcing the interacting agent to replan. The function `collide_stop( $i, m$ )` determines whether robot i , by executing primitive m from its current state x_i , would collide with robots whose actions have already been assigned. Our key modification to the original PIBT is that this collision check extends beyond the immediate execution of m to the corresponding stopping trajectories; this ensures that robots can remain collision-free in the future, provided that they execute their corresponding backup trajectories. Algorithm 1 falls back to the backup trajectory if no primitive can be safely assigned (Line 10).

Theorem 1: In finite time, Alg. 1 finds a safe action assignment together with collision-free subsequent stopping trajectories, provided that the input backup trajectories are jointly collision-free.

Proof: (sketch) The collision-free property follows from the analysis of the original PIBT and its motion-primitive-based variant [5], [11]. In short, the subprocedure PIBT is invoked only for a robot whose action has not yet been

assigned, and each robot finalises its action at most once. A robot i finalises its action m_i only if the corresponding stopping trajectory does not collide with those of robots whose actions have already been assigned. We guarantee that unassigned robots have a safe action available by ensuring that robot i ’s actions do not conflict with the backup trajectories of these unassigned robots; otherwise, PIBT is recursively invoked for the interacting robot j to resolve the conflict before the action of i is finalised. If no safe primitive can be assigned through this recursion, the involved robots fall back to their input backup trajectories, which are collision-free by assumption. Hence, the recursion terminates after finitely many assignments while preserving collision-free stopping trajectories for all robots. ■

Theorem 2: Consider a multi-robot execution in which Alg. 1 is invoked every Δt , the resulting actions are executed, and the corresponding stopping trajectories are retained as backup trajectories for the next invocation. Then, the resulting multi-robot trajectories remain collision-free for all time.

Proof: By the previous theorem, each PIBT invocation preserves collision-freeness and produces collision-free stopping trajectories, which are used as the backup trajectories for the next invocation. The claim therefore follows by induction over timesteps. ■

3) *Auxiliary Reward for RL:* With the safeguarded rollout, an ordered list of motion primitives is sampled from the policy output $\pi(o_i)$, after which the selected actions may be overridden by PIBT collision shielding. While safety is ensured, such interventions can weaken the learning signal for the policy itself to synthesise coordinated actions. Therefore, we augment the reward defined in Eq. (2) with an auxiliary term based on whether the policy’s actions are consistent with the PIBT-processed actions, penalising the overriding of the actions by the PIBT collision shielding.

E. Implementation Details

In 2D, the motion-primitive set M comprises 8 acceleration vectors uniformly distributed across a single acceleration magnitude, with $a_{\text{max}}=2 \text{ m s}^{-2}$, together with zero acceleration, yielding 9 primitives in total. Similarly, in 3D, we use 26 directions at the max acceleration magnitude, with $a_{\text{max}}=5 \text{ m s}^{-2}$, together with zero acceleration, yielding 27 primitives in total. The robot radius R_{robot} is set to 0.25 m, the communication radius R_{comm} to 3 m, and the FOV N_{obs} to 11 cells along each dimension, with a cell size of 0.25 m. The workspace is typically 6 m along each dimension and Δt is taken to be 0.1 s.

For IL, we use 60K demonstrations, with half generated on *Blocked Random* instances and the other half on *Forest* instances, containing four agents. Data collection requires approximately 22 hours of wall-clock time on Intel Xeon Platinum 8452Y CPUs, with 24 runs executed in parallel. The policy contains approximately 1.4M trainable parameters and is trained using the AdamW optimiser [35]. IL training requires approximately 40 GPU hours on NVIDIA L40S GPUs. The RL phase requires an additional 25 GPU hours, with rollouts using the PIBT-based safeguarded rollout

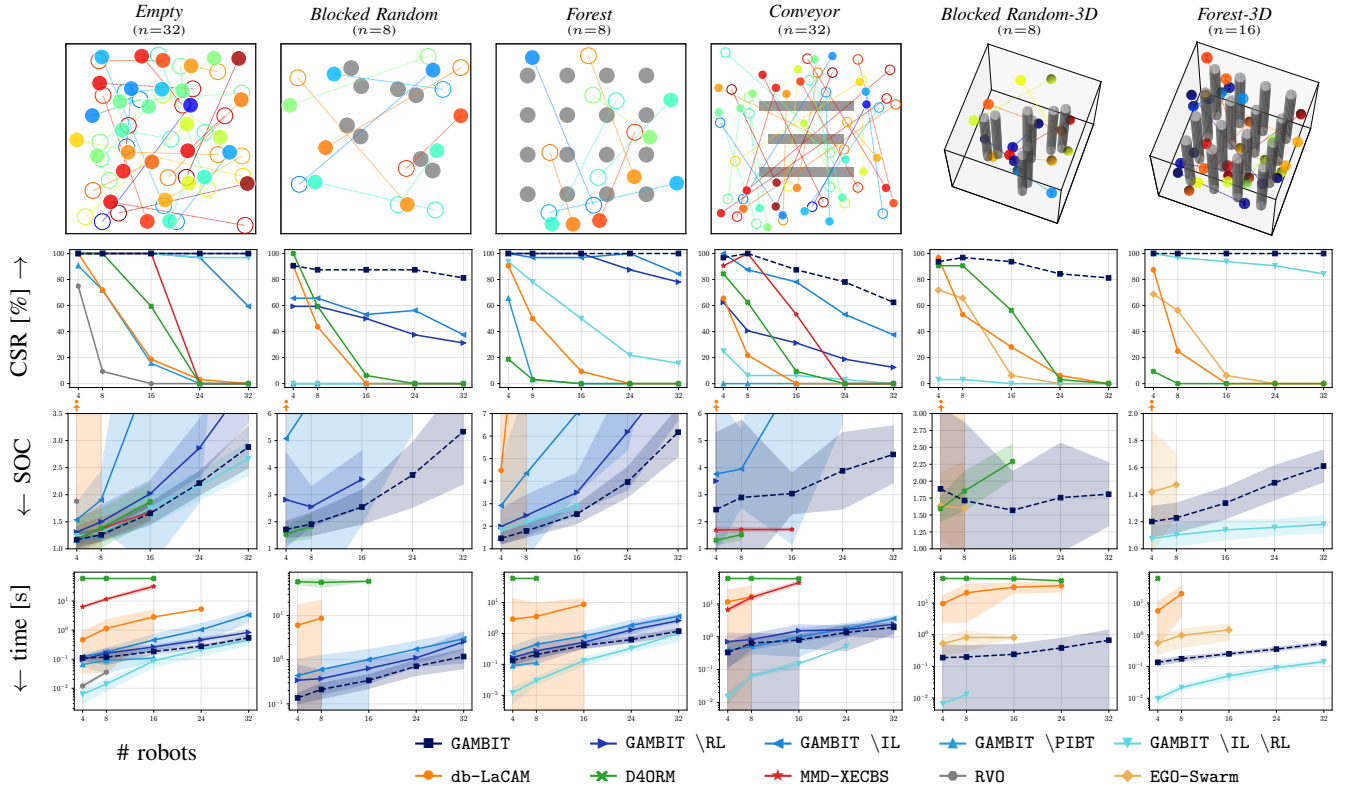


Fig. 2. Main result. For each scenario, an example instance is shown at the top, where filled circles represent robots and unfilled circles connected by lines indicate their corresponding goals. Grey circles, rectangles, and cylinders represent obstacles. The remaining plots show, from top to bottom, CSR (complete success rate), SOC (sum-of-costs; normalised by lower bounds based on start-goal distances), and computation time to derive the entire trajectories. SOC and computation time are averaged over successful instances for each planner, with shaded regions indicating one standard deviation; to ensure meaningful comparisons, we report these metrics only when the corresponding success rate exceeds 50%. db-LaCAM’s SOC often exceeds the displayed range; we use an orange dot and arrow to indicate the clipped value to preserve readability of the remaining curves.

described above, with the reward parameters in Eq. (2) set to $w_1=5.0$ and $w_2=0.01$. The policy is trained in Python and subsequently exported for C++ inference to reduce runtime overhead. The 3D setup is configured similarly.

IV. EXPERIMENTS

We evaluate GAMBIT as a reactive planner that repeatedly selects motion primitives from local observations, combined with the safeguarded rollout (Alg. 1). Visualisations of both the simulation and real-robot deployment are provided in the supplementary video. All methods are evaluated using the compute infrastructure described in Sec. III-E.

A. Simulation Setup

1) *Instances*: We prepare 32 instances with varying numbers of robots from 4 to 32, based on the following scenarios: (i) *Empty* (2D), representing a basic scenario without obstacles. Start and goal positions are sampled uniformly at random; (ii) *Blocked Random* (2D, 3D) and (iii) *Forest* (2D, 3D) described in Sec. III-B, representing in-distribution scenarios. The test instances are distinct from those used during training; (iv) *Conveyor* (2D), adapted from [18], representing out-of-distribution scenarios.

2) *Baselines*: On these instances, we contrast GAMBIT against the following representative baselines spanning different planning and optimisation paradigms:

- RVO [15], a widely used reactive collision-avoidance algorithm. We use its double-integrator formulation [36].
- db-LaCAM [5], a centralised, full-horizon kinodynamic multi-robot motion planner that computes complete trajectories from start to goal. A variant of this planner is also used to generate the dataset for IL.
- MMD-XECBS [18], a centralised, full-horizon planner based on diffusion models trained for the target maps. We include this method as a learning-based alternative.
- D4ORM [17], a GPU-accelerated population-based trajectory smoother for multi-robot systems.
- EGO-Swarm [16], a decentralised reactive trajectory planner designed for 3D multi-drone navigation.

For all baselines, we use the publicly available codebases provided by the respective authors. Due to their implementation constraints, RVO supports only 2D obstacle-free scenarios (i.e. *Empty*), whereas EGO-Swarm is evaluated only in 3D. As MMD requires training on the target scenario, we evaluate it only on *Empty* and *Conveyor*. All planners are allocated a time limit of 1 min.

We additionally evaluate the following variants of GAMBIT to assess the contribution of each technical component:

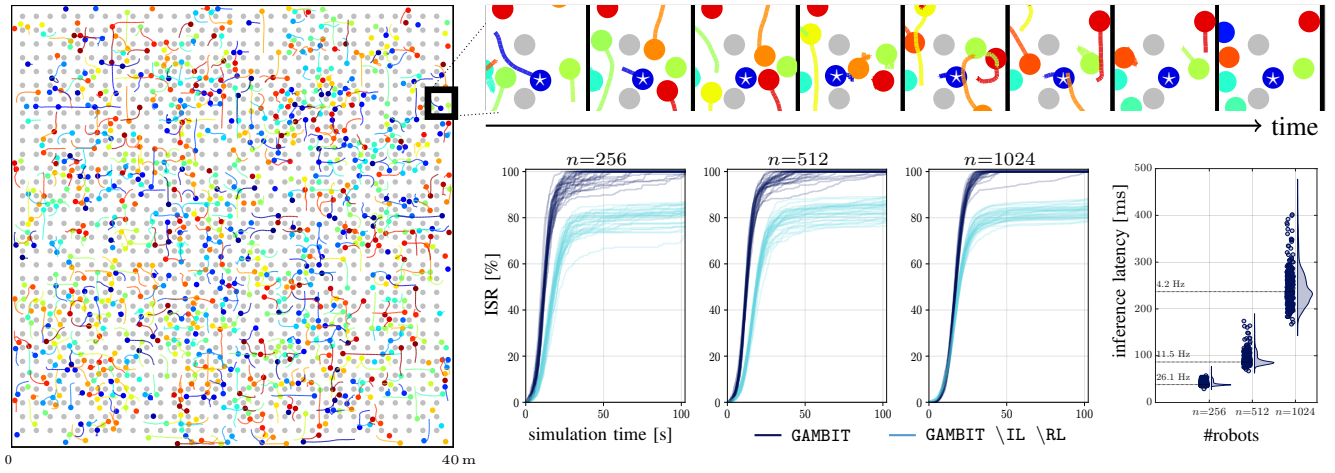


Fig. 3. Scalability assessment with up to 1,024 robots in the *Forest* scenario. The left panel shows a snapshot of the entire workspace at a certain timestep. Circles represent robots, with solid lines indicating their trajectories as tails. The upper-right panel depicts the trajectory evolution within a specific region, highlighting coordinated behaviours produced by GAMBIT; e.g., the $\star$ robot temporarily moves away from its goal to let others pass. The lower-right panel presents individual success rate (ISR) evolution for $n = \{256, 512, 1,024\}$; i.e., the percentage of robots that have reached their goals, as a function of elapsed simulation time. We run the planners on 32 instances, with each curve representing the solution over a different instance. The bottom-right panel shows the distribution around the median inference time/step (and the corresponding loop frequency) for computing actions for the entire team.

- GAMBIT \ IL \ RL removes the learning components entirely, and instead sorts motion primitives in Line 3 using a manually designed heuristic. This heuristic considers the positional distance to the goal from the rollout state together with the resulting velocity, representing greedy behaviour towards the goal.
- \ RL ablates the RL fine-tuning stage, using the policy trained solely by IL.
- \ IL removes IL, with the policy trained purely by RL until convergence (≈ 65 h).
- \ PIBT removes the safeguarded rollout, directly samples a motion primitive from the action distribution, and executes it according to Eq. (1). For this variant, we modify the RL reward to penalise collisions.

3) *Metrics*: The following metrics assess the performance of each planner: *complete success rate (CSR)* measures the fraction of instances in which *all* agents reach their goals; *sum-of-costs (SOC)* assesses trajectory quality; and *computation time (Time)* measures the wall-clock time required to compute the complete trajectories. We additionally use *individual success rate (ISR)*, measuring the fraction of agents that reach their goals within each instance.

B. Main Results

Figure 2 summarises our main results. GAMBIT achieves high success rates across all tested scenarios, including both in-distribution and out-of-distribution settings, at scale while maintaining highly competitive solution quality, delivering the strongest overall performance among the evaluated methods. The computation time remains moderate; even with 32 robots, GAMBIT requires only about 1s to derive feasible trajectories, suggesting real-time planning ability, which we investigate further in Sec. IV-C and IV-D. A closer look reveals that learning is a key component (see

GAMBIT \ IL \ RL), with both IL and RL providing substantial improvements in success rate. The performance of \ PIBT further demonstrates that the safeguarded rollout is critical to achieving strong performance; i.e., direct execution of the learned policy is not performant. We conjecture that this is because multi-robot navigation policies generally need to learn multiple objectives jointly, including goal reaching and collision avoidance, whereas the safeguarded rollout largely relieves the policy of the latter, simplifying the learning problem and allowing the policy to focus its learning capacity on goal-directed coordination.

Among the baselines, the domain-independent planners db-LaCAM and D4ORM struggle to scale, while MMD-XECBS, which uses learning to adapt to the target domain, achieves superior planning performance but at a high computational cost. Decentralised local collision-avoidance planners such as RVO and EGO-Swarm struggle with interaction-heavy, dense instances where many robots are packed into a limited workspace, as these scenarios require coordination beyond mere collision avoidance. While remaining reactive at execution time, GAMBIT incorporates such long-horizon reasoning through offline learning.

While GAMBIT is capable of dense coordination, some failures remain. Our qualitative observation suggests that these failures are primarily caused by livelocks, where one or more robots repeatedly execute the same actions without making progress. Livelocks occur primarily near obstacles, e.g., when a robot must pass through a narrow gap between obstacles but safety margins prevent the motion, or when it must traverse around an obstacle to reach a goal on the opposite side. We envision two promising directions for overcoming this limitation: stronger policies through improved architectures and training pipelines, and integrating the learned motion-primitive heuristic into long-horizon

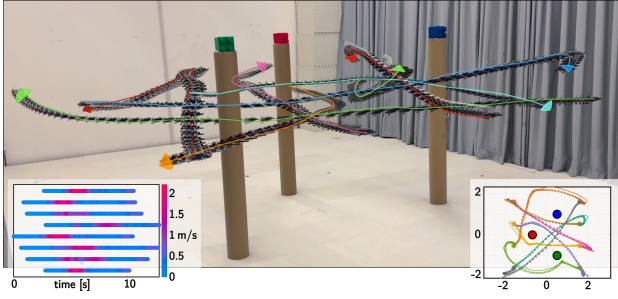


Fig. 4. Snapshot from a GAMBIT deployment with eight multirotors planning at 10 Hz. **(left-inset)** Speed profile (m/s) for the robots in this instance. **(right-inset)** Top-down view of the arena and trajectories.

search [5], [17]. Importantly, none of the observed failures are caused by collisions, owing to the safeguarded rollout.

C. Scalability Assessment

Using the *Forest* scenario with an enlarged workspace, we evaluate the scalability of GAMBIT on instances with up to 1,024 robots. At this scale, none of the baseline methods successfully solve the instances, and thus we evaluate GAMBIT and its non-learning variant (i.e., $\backslash$ IL $\backslash$ RL). Figure 3 demonstrates that GAMBIT provides a highly scalable planning framework, maintaining high individual success rates even in densely packed settings. This scale is significantly beyond the reach of existing methods for computing full-horizon continuous multi-robot trajectories. Moreover, GAMBIT retains real-time responsiveness at scale: with 512 robots, it synthesises safe coordinated actions in < 100 ms/step, enabling real-time control at scale. While the non-learning, greedy variant still allows a portion of the robots to reach their goals (see the ISR evolution plot), some robots remain unsuccessful due to its coordination-unaware heuristic, emphasising the importance of learning coordination-aware heuristics.

D. Multirotor Deployment

We finally deploy GAMBIT on physical multirotors [19]. Figure 4 shows overlaid trajectories from one demonstration in which eight multirotors swap their start–goal locations in a motion-capture arena containing obstacles. We run GAMBIT at 10 Hz to synthesise motion online, yielding collision-free execution under closed-loop control even at a peak speed of 2.2 m/s in the instance shown.

V. CONCLUSION

We presented GAMBIT, a framework for learning motion-primitive heuristics for dense multi-robot coordination. While the resulting policy combined with our safeguarded rollout already achieves strong planning performance and exceptional scalability, we view GAMBIT more broadly as a powerful toolkit for building more capable multi-robot planners. Indeed, learned components have increasingly demonstrated their potential to enhance multi-agent planning [8], [9], [18]. Promising directions include extending it to more general dynamics beyond the double-integrator model.

REFERENCES

- [1] M. Likhachev and D. Ferguson, “Planning long dynamically feasible maneuvers for autonomous vehicles,” *Int. J. Robot. Res.*, 2009.
- [2] M. Pivtoraiko and A. Kelly, “Kinodynamic motion planning with state lattice motion primitives,” in *IROS*, 2011.
- [3] J. Ortiz-Haro, W. Hönig, V. N. Hartmann, and M. Toussaint, “idb-a*: Iterative search and optimization for optimal kinodynamic motion planning,” *IEEE Trans. Robot.*, 2024.
- [4] L. Wen, Y. Liu, and H. Li, “Cl-mapf: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints,” *Robotics and Autonomous Systems*, 2022.
- [5] A. Moldagalieva, K. Okumura, A. Prorok, and W. Hönig, “db-lacm: Fast and scalable multi-robot kinodynamic motion planning with discontinuity-bounded search and lightweight mapf,” in *ICAPS*, 2026.
- [6] H. Jiang, Y. Wang, R. Veerapaneni, T. Duhan, G. Sartoretti, and J. Li, “Deploying ten thousand robots: Scalable imitation learning for lifelong multi-agent path finding,” in *ICRA*, 2025.
- [7] Y. Wang, T. Duhan, J. Li, and G. Sartoretti, “Lns2+ rl: Combining multi-agent reinforcement learning with large neighborhood search in multi-agent path finding,” in *AAAI*, 2025.
- [8] R. Jain, K. Okumura, M. Amir, and A. Prorok, “Graph attention-guided search for dense multi-agent pathfinding,” in *AAAI*, 2026.
- [9] J.-M. Alkazzi and K. Okumura, “A comprehensive review on leveraging machine learning for multi-agent path finding,” *IEEE Access*, 2024.
- [10] K. Okumura, “Engineering lacm*: Towards real-time, large-scale, and near-optimal multi-agent pathfinding,” in *AAMAS*, 2024.
- [11] K. Okumura, M. Machida, X. Défago, and Y. Tamura, “Priority inheritance with backtracking for iterative multi-agent path finding,” *Artif. Intell.*, 2022.
- [12] R. Veerapaneni, Q. Wang, K. Ren, A. Jakobsson, J. Li, and M. Likhachev, “Improving learnt local mapf policies with heuristic search,” in *ICAPS*, 2024.
- [13] R. Jain, K. Okumura, M. Amir, P. Lio, and A. Prorok, “Pairwise is not enough: Hypergraph neural networks for multi-agent pathfinding,” in *ICLR*, 2026.
- [14] Q. Li, W. Lin, Z. Liu, and A. Prorok, “Message-aware graph attention networks for large-scale multi-robot path planning,” *IEEE Robot. Autom. Lett.*, 2021.
- [15] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in *Proceedings of the International Symposium of Robotics Research (ISRR)*, 2011.
- [16] X. Zhou, J. Zhu, H. Zhou, C. Xu, and F. Gao, “Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments,” in *ICRA*, 2021.
- [17] Y. Zhang, K. Okumura, H. Woo, A. Shankar, and A. Prorok, “D4orm: Multi-robot trajectories with dynamics-aware diffusion denoised deformations,” in *IROS*, 2025.
- [18] Y. Shaoul, I. Mishani, S. Vats, J. Li, and M. Likhachev, “Multi-robot motion planning with diffusion models,” in *ICLR*, 2025.
- [19] H. Woo, K. R. I. Sanim, K. Okumura, G. Yang, A. Shankar, and A. Prorok, “Sanity: An agile brushless quadrotor for multi-agent experiments,” 2025, <https://openreview.net/forum?id=i5dKOANPZZ>.
- [20] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
- [21] J. Tordesillas and J. P. How, “Mader: Trajectory planner in multiagent and dynamic environments,” *IEEE Trans. Robot.*, 2021.
- [22] A. Moldagalieva, J. Ortiz-Haro, and W. Hönig, “db-ecbs: Interaction-aware multirobot kinodynamic motion planning,” *IEEE Trans. Robot.*, 2025.
- [23] J. Hou, X. Zhou, N. Pan, A. Li, Y. Guan, C. Xu, Z. Gan, and F. Gao, “Primitive-swarm: An ultra-lightweight and scalable planner for large-scale aerial swarms,” *IEEE Trans. Robot.*, 2025.
- [24] B. Riviere, W. Hönig, Y. Yue, and S.-J. Chung, “Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning,” *IEEE Robot. Autom. Lett.*, 2020.
- [25] S. Batra, Z. Huang, A. Petrenko, T. Kumar, A. Molchanov, and G. S. Sukhatme, “Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning,” in *CoRL*, 2022.
- [26] E. Camci and E. Kayacan, “Learning motion primitives for planning swift maneuvers of quadrotor,” *Auton. Robots*, 2019.
- [27] J. Franke, A. Moldagalieva, P. Hanfeld, and W. Hönig, “Accelerating db-a* for kinodynamic motion planning using diffusion,” *arXiv preprint*, 2025.

- [28] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robot. Autom. Lett.*, 2019.
- [29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.
- [30] L. Ouyang, J. Wu *et al.*, "Training language models to follow instructions with human feedback," in *NeurIPS*, 2022.
- [31] Physical Intelligence, " $\pi_{0.6}^*$: a VLA that learns from experience," in *RSS*, 2026.
- [32] K. Solovey, "Complexity of planning," *arXiv preprint*, 2020.
- [33] C. Yu, A. Velu, E. Vinitzky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative multi-agent games," in *NeurIPS*, 2022.
- [34] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *ECC*, 2019.
- [35] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR*, 2019.
- [36] J. Van Den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *ICRA*, 2011.