Neural Tree Expansion for Multi-Robot Planning in Non-Cooperative Environments

Benjamin Rivière, Wolfgang Hönig, Matthew Anderson, and Soon-Jo Chung

***Abstract*— We present a self-improving, neural tree expansion method for multi-robot online planning in non-cooperative environments, where each robot tries to maximize its cumulative reward while interacting with other self-interested robots. Our algorithm adapts the centralized, perfect information, discrete-action space method from Alpha Zero to a decentralized, partial information, continuous action space setting for multi-robot applications. Our method has three interacting components: (i) a centralized, perfect-information "expert" Monte Carlo Tree Search (MCTS) with large computation resources that provides expert demonstrations, (ii) a decentralized, partial-information "learner" MCTS with small computation resources that runs in real-time and provides self-play examples, and (iii) policy & value neural networks that are trained with the expert demonstrations and bias both the expert and the learner tree growth. Our numerical experiments demonstrate neural expansion generates compact search trees with better solution quality and 20 times less computational expense compared to MCTS without neural expansion. The resulting policies are dynamically sophisticated, demonstrate coordination between robots, and play the Reach-Target-Avoid differential game significantly better than the state-of-the-art control-theoretic baseline for multi-robot, double-integrator systems. Our hardware experiments on an aerial swarm demonstrate the computational advantage of neural tree expansion, enabling online planning at 20 Hz with effective policies in complex scenarios.**

I. Introduction

Multi-agent interactions in non-cooperative environments are ubiquitous in next-generation robotic domains such as self-driving, space exploration, and human-robot interactions. Planning, or sequential decision-making, in these settings requires a prediction model of the other agents, which can be generated through a game theoretic framework.

Recently, the success of Alpha Zero [1] at the game of Go has popularized a self-improving machine learning algorithm: bias a Monte Carlo Tree Search with value and policy neural networks, use the tree statistics to train the networks with supervised learning and then iterate over these two steps to improve the policy and value networks over time. However, this algorithm is designed for classical artificial intelligence tasks (e.g. chess or Go), and applications in multi-robot domains require different assumptions: continuous state-action, decentralized evaluation, partial information, and limited computational resources. To the best of our knowledge, this is the first work to provide a complete multi-robot adaption from algorithm design to hardware experiment of this powerful method.

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

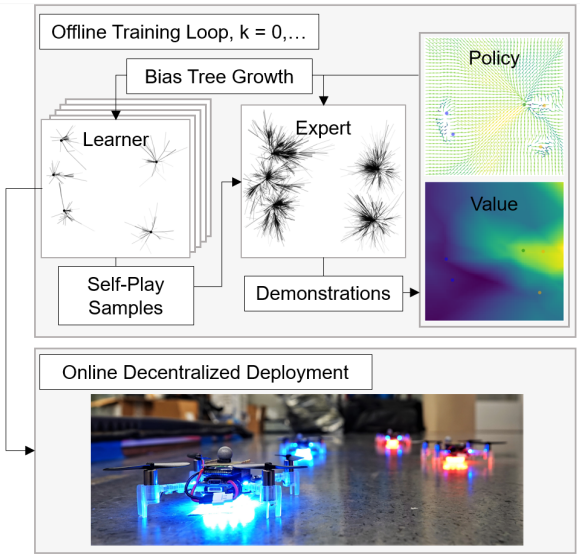


Fig. 1. Our method learns to play the multi-robot Reach-Target-Avoid game. The learner trees query relevant states for demonstration from the expert trees, and the data is used to train policy and value neural networks, which are used to bias both tree growths. At runtime, the learner is queried at each robot to generate an action with local information and little computational resources.

The overview of our algorithm is shown in Fig. 1. The key algorithmic innovation of our approach is to create two distinct MCTS policies to bridge the gap between high-performance simulation and real-world robotic application: the "expert" tree search is centralized and has access to perfect information and large computational resources, whereas the "learner" tree search is decentralized and has access to partial information and limited computational resources. During the offline phase, the neural networks are trained in an imitation learning style using the self-play states of the learner and the high-quality demonstrations of the expert. The expert's high-quality demonstrations enable policy improvement through iterations and incrementally improve the policy and value networks. The learner samples states that should appear more frequently at runtime. For deployment, each robot uses the learner tree search to effectively plan online with partial information and limited computational budget. Our contributions are extending Biased Neural Monte Carlo Tree Search to (i) decentralized evaluation with local information, (ii) continuous state-action domain, and (iii) limited computational resources.

We validate our method in simulation and experiment. We demonstrate numerically that our approach generates

compact trees of similar or better performance with 20 times fewer nodes, and the resulting policies play the Reach-Target-Avoid differential game with double-integrator dynamics significantly better than the current state of the art. We show the generalizability of the approach with a numerical simulation extension with 3D Dubin's vehicle dynamics. Our hardware experiments demonstrate that the solutions are robust to the sim-to-real gap and neural expansion generates compact search trees to run effective policies in real-time.

Related Work: Our work relates to multiple communities: planning, machine learning, and game theory. Planning, or sequential decision-making, problems can be formalized with a Partially Observable Markov Decision Process (POMDP). The conventional solution to solving POMDPs in an online setting is Monte Carlo Tree Search (MCTS) [2]. MCTS searches through the large decision-making space by rolling out simulated trajectories and biasing the tree growth towards areas of high reward [3]. MCTS frequently uses the Upper Confidence Bound for Trees algorithm [4] that uses a discrete-action, multi-armed bandit solution [5] to balance exploration and exploitation in node selection. Recent work uses a non-stationary bandit analysis to propose a polynomial, rather than logarithmic, exploration term [6].

Application of MCTS to a dynamically-constrained robot planning setting requires extending the theoretical foundations to a continuous state and action space. In general, the recent advances in this area answer two principal questions: i) how to select an action, and ii) when is a node fully expanded. Regarding the former question, it is possible to use the solution of the χ-armed bandit problem [7], which is the extension of the multi-armed bandit to continuous domains [8]. Our approach is to use a policy network to generate actions in the tree search. Regarding the latter question, a popular method is to use progressive widening and variants; we adapt the Polynomial Upper Continuous Trees (PUCT) algorithm [9] that uses the double progressive widening method. Despite the advance in theory for continuous action spaces, there have been relatively few studies of learning-based extensions of biasing continuous MCTS with data-driven methods [10].

The key of Alpha Zero [1] is using MCTS as a policy improvement operator; i.e. given a policy network to help guide MCTS, MCTS's own action will be closer to the optimal action than the action generated by the original policy network. Then, the policy network is trained with supervised learning to imitate the superior MCTS policy, matching the quality of the network to that of MCTS over the samples. By iterating over these two steps, the model improves over time. The first theoretical analysis of this powerful method is recently shown for single-agent discrete action space problems [6]. In comparison, our method is applied to a continuous state-action, multi-agent setting. Whereas Alpha Zero methods use the policy network to bias the node selection process, i.e. given a list of actions, select the best one, our policy network is an action generator for the expansion process to create edges to children, i.e. given a state, generate an action. A neural expansion operator has previously been explored in robot motion planning [11], but not robot decision-making. In addition, our method's supervised learning step is closer to imitation learning, as used in DAgger [12], because the learner benefits from an adaptive dataset generation of a learner using self-play to query from an expert.

Although the Alpha Zero family of methods use a form of supervised learning to train the networks, they can be classified as a reinforcement learning method because the networks are trained without a pre-existing labelled dataset. Policy gradient [13] is the conventional reinforcement learning solution and there are many recent advances in this area [14]. In contrast, our method has a higher degree of interpretability provided by the tree structure.

In contrast to data-driven methods, traditional analytical solutions can be studied and derived through differential game theory. The game we study, Reach-Target-Avoid, was first introduced and solved by [15] for simple-motion, 1 vs. 1 systems. Later, multi-robot, single-integrator solutions have been proposed by [16, 17]. Solutions considering multi-robots with non-trivial dynamics, such as the double-integrator [18], are an active area of research.

II. Problem Formulation

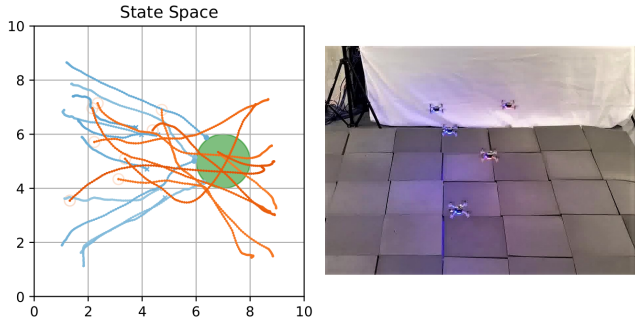
Notation: We denote the learning iteration with k and the physical timestep with a subscript t and, unless necessary, we suppress the time dependency for notation simplicity. Robot-specific quantities are denoted with i or j superscript, and, in context, the absence of superscript denotes a joint-space quantity, e.g. the joint state vector is the vertical stack of all individual robot vectors, $\mathbf{s}_t = [\mathbf{s}_t^1; \dots; \mathbf{s}_t^N]$ where N is the number of robots.

Definition 1: A partially observable stochastic game (POSG) is defined by a tuple: $\mathcal{G} = \langle \mathcal{I}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \mathcal{O}, H \rangle$ where: $\mathcal{I} = \{1, ..., N\}$ is the set of robot indices, $\mathcal{S}$ is the set of joint robot states, $\mathcal{A}$ is the set of joint robot actions, $\mathcal{T}$ is the joint robot transition function where $\mathcal{T}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) = \mathbb{P}(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$ is the probability of transitioning from joint state $\mathbf{s}_t$ to $\mathbf{s}_{t+1}$ under joint action $\mathbf{a}_t$, $\mathcal{R}$ is the set of joint robot rewards functions where $\mathcal{R}^i(\mathbf{s}, \mathbf{a}^i)$ is the immediate reward of robot i for taking local action $\mathbf{a}^i$ in joint state $\mathbf{s}$, $\mathcal{Z}$ is the set of joint robot observations, $\mathcal{O}$ is the set of joint robot observation probabilities where $\mathcal{O}(\mathbf{z}, \mathbf{s}, \mathbf{a}) = \mathbb{P}(\mathbf{z} | \mathbf{s}, \mathbf{a})$ is the probability of observing joint observation $\mathbf{z}$ conditioning on the joint state and action, and H is the planning horizon. Each of these joint-quantities can be appropriately constructed from the robot-specific quantity. The solution of a POSG is a sequence of actions that maximizes the expected reward over time and is often characterized by a policy or value function.

Assumption 1: A deterministic transition function is assumed, thereby permitting rewriting $\mathcal{T}$ with a deterministic dynamic function, f through:

$$\mathcal{T}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) = \mathbb{I}(f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}) \tag{1}$$

where $\mathbb{I}$ is an indicator function. Similarly, rewriting $\mathcal{O}$ with a deterministic observation function results in $\mathbf{z} = h(\mathbf{s}, \mathbf{a})$.



(a) 10 vs. 10 Simulation (b) 3 vs. 2 Experiment

Fig. 2. Example state space: blue robots, indexed by $\mathcal{I}_A$, try to reach the green goal region, and the red robots, indexed by $\mathcal{I}_B$, tries to prevent that from happening by tagging the blue robots first. An x on the trajectory indicates tagged state, and a o indicates reached goal.

This assumption can be relaxed by considering specialized variants that are not the focus of this work. For example, the stochastic transition can be handled with the double-progressive-widening algorithm [9] (which we only use to handle continuous action space) and measurement uncertainty can be handled with observation widening [19].

Problem Statement: At time t, each robot i makes a local observation, $\mathbf{z}^i$, uses it to formulate an action, $\mathbf{a}^i$, and updates its state, $\mathbf{s}^i$, according to the dynamical model. Our goal is to find policies for each robot, $\pi^i : \mathcal{Z}^i \to \mathcal{A}^i$ that synthesizes actions from local observations through:

$$\mathbf{z}_t^i = h^i(\mathbf{s}_t), \quad \mathbf{a}_t^i = \pi^i(\mathbf{z}_t^i), \tag{2}$$

to approximate the solution to the general-sum, game theoretic optimization problem:

$$\begin{aligned} \mathbf{a}_t^{i^*} &= \underset{\{\mathbf{a}_\tau^i | \forall \tau\}}{\arg\max} \sum_{\tau=t}^{t+H} \mathcal{R}^i(\mathbf{s}_\tau, \mathbf{a}_\tau^i) \quad \text{s.t.} \\ \mathbf{s}_{\tau+1} &= f(\mathbf{s}_\tau, \mathbf{a}_\tau),\ \mathbf{s}_\tau^i \in \mathcal{X}^i,\ \mathbf{a}_\tau^i \in \mathcal{U}^i,\ \mathbf{s}_{\tau_0}^i = \mathbf{s}_0^i,\ \forall i, \tau \end{aligned} \tag{3}$$

where $\mathcal{U}^i \subseteq \mathcal{A}^i$ is the set of available actions (e.g. bounded control authority constraints), and $\mathcal{X}^i \subseteq \mathcal{S}^i$ is the set of safe states (e.g. collision avoidance) and $\mathbf{s}_0^i$ is the initial state condition. The optimization problems for each robot i are simultaneously coupled through the evolution of the global state vector $\mathbf{s}$, where each robot attempts to maximize its own reward function $\mathcal{R}^i$.

Reach-Target-Avoid Game: An instance of the above formulation is the Reach-Target-Avoid game [15] for two teams of robots, where team A gets points for robots that reach the goal region, and team B gets points for defending the goal by tagging the invading robots first. The teams are parameterized by index sets $\mathcal{I}_A$ and $\mathcal{I}_B$, respectively, where the union of the two teams represents all robots, $\mathcal{I}_A \cup \mathcal{I}_B = \mathcal{I}$. A state-space example of the Reach-Target-Avoid game is shown in Fig. 2.

Dynamics: We consider the discrete-time double-integrator system for the i^{th} robot as a motivating example:

$$\mathbf{s}_{t+1}^i = \begin{bmatrix} \mathbf{p}_t^i \\ \mathbf{v}_t^i \end{bmatrix} + \begin{bmatrix} \mathbf{v}_t^i \\ \mathbf{a}_t^i \end{bmatrix} \Delta_t \tag{4}$$

where $\mathbf{p}^i$ and $\mathbf{v}^i$ denote position and velocity and Δ_t denotes the simulation timestep. We use a simultaneous turn game formulation where at a given timestep, each team's action is chosen without knowledge of the other team's action.

Admissible State and Action Space: The admissible state space for each robot is defined by the following constraints: remain inside the position and velocity bounds, $\overline{p}, \overline{v}$, and avoid collisions within the physical robot radius, r_p:

$$\|\mathbf{p}^i\|_\infty \leq \overline{p},\ \|\mathbf{v}^i\|_2 \leq \overline{v}, \quad \|\mathbf{p}^j - \mathbf{p}^i\| > r_p,\ \forall i, j \in \mathcal{I} \tag{5}$$

For robots on team A, the admissible state space has an additional constraint: avoid the robots on team B by at least the tag radius:

$$\|\mathbf{p}^j - \mathbf{p}^i\| > r_t, \quad \forall j \in \mathcal{I}_B,\ \forall i \in \mathcal{I}_A \tag{6}$$

Then, the admissible state space for each team can be written compactly, e.g. $\mathcal{X}^i = \{\mathbf{s}^i \in \mathcal{S}^i \mid \text{ s.t. (5), (6)}\}, \quad \forall i \in \mathcal{I}_A$. The admissible action space for each robot is constrained by its acceleration limit: $\mathcal{U}^i = \{\mathbf{a}^i \in \mathcal{A}^i \mid \|\mathbf{a}^i\|_2 \leq \overline{a}\},\ \forall i \in \mathcal{I}$, where $\overline{a}$ is the robot's acceleration limit. When a robot exits the admissible state or action space, or a robot on team A's position is within an r_g radius about the goal position $\mathbf{p}_g$, it becomes inactive.

Observation Model: Under Assumption 1, for each robot i, we define a measurement model that is similar to visual relative navigation $h^i : \mathcal{S} \to \mathcal{Z}^i$, which measures the relative state measurement between neighboring robots, as well as relative state to the goal. Specifically, an observation is defined as:

$$\mathbf{z}^i = \left[\mathbf{g} - \mathbf{s}^i, \{\mathbf{s}^j - \mathbf{s}^i\}_{j \in \mathcal{N}_A}, \{\mathbf{s}^j - \mathbf{s}^i\}_{j \in \mathcal{N}_B}\right], \tag{7}$$

where $\mathbf{g}$ is the goal position embedded in the state space, e.g. for 2D double-integrator $\mathbf{g} = [\mathbf{p}_g; 0; 0]$. Then, $\mathcal{N}_A$ and $\mathcal{N}_B$ denote the neighboring sets of robots on team A and B, respectively. These sets are defined by each robot's sensing radius, r_{sense}, e.g.,

$$\mathcal{N}_A = \{j \in \mathcal{I}_A \mid \|\mathbf{p}^j - \mathbf{p}^i\|_2 \leq r_{\text{sense}}\}. \tag{8}$$

Reward: The robot behavior is driven by the reward function; the inter-team cooperation behavior is incentivized by sharing the reward function and intra-team adversarial behavior is incentivized by assigning complementary reward functions only dependent on global state, $\mathcal{R}^i(\mathbf{s}) = -\mathcal{R}^j(\mathbf{s}),\ \forall i \in \mathcal{I}_A, j \in \mathcal{I}_B$. The reward can be defined by a single, robot-agnostic game reward, $\mathcal{R}(\mathbf{s})$ that team A tries to maximize and team B tries to minimize. The game reward is 0 until the terminal state where the game reward and traditional value function are identical and defined as:

$$V(\mathbf{s}_t) = \sum_{\forall i \in \mathcal{I}_A} \mathbb{I}(\|\mathbf{p}_t^i - \mathbf{p}_g\|_2 \leq r_g), \tag{9}$$

i.e. the value is the number of team A robots in the goal region. The indicator function payoff is known to be sparse and makes traditional search and reinforcement learning techniques ineffective [20]. The game termination occurs when all robots on team A are inactive; typically when they have reached the goal or been tagged by a robot on team B.

TABLE I
SUMMARY OF POLICIES

Symbol	Name	Computation	Input Domain
π^*	True Optimal	N/A	Joint State
$\overline{\pi}_k$	Expert	Biased MCTS	Joint State
$\hat{\pi}_k^i$	Learner	Biased MCTS	Joint Estimated State
$\tilde{\pi}_k$	Joint Policy	Composition of $\tilde{\pi}^i$	Joint Observation
$\tilde{\pi}_k^i$	Policy	Neural Network	Local Observation

III. ALGORITHM DESCRIPTION

We present the meta-algorithm, the biased MCTS variant, and then each of the components: expert, learner, and policy and value networks. We summarize the symbol, notation, and computation of our policies in Table I.

A. Meta Self-Improving Algorithm

The input of the meta-learning is the POSG game described in Sec. II, and the outputs are two neural networks for policy and value distributions. The goal of the meta-learning is to improve the models across learning iterations, especially in relevant state domains, such that at runtime, the robots can evaluate the learner policy. The desired model improvement can be expressed by decreasing some general probability distribution distance between the policy network and the optimal policy function:

$$\text{dist}(\tilde{\pi}_k(\mathbf{z}), \pi^*(\mathbf{s})) \leq \text{dist}(\tilde{\pi}_l(\mathbf{z}), \pi^*(\mathbf{s})),\ \forall k > l,\ \forall \mathbf{s} \quad (10)$$

where π^* is the optimal policy function that inputs a joint state and returns a joint action, $\tilde{\pi}_k$ is the policy network we train, and k, l are learning iteration indices. Specifically, we train robot-specific policies $\tilde{\pi}_k^i$ that map local observation to local action and compose them together to create the joint policy $\tilde{\pi}_k = [\tilde{\pi}_k^1; \ldots; \tilde{\pi}_k^{|\mathcal{I}|}]$ that maps joint observation, $\mathbf{z}$, to joint action, $\mathbf{a}$. Adapting the proof concept in [6] to our setting, the policy improvement can be shown by validating the following two properties and then iterating: (i) bootstrap

$$\text{dist}(\pi^*(\mathbf{s}), \overline{\pi}_k(\mathbf{s}, \tilde{\pi}_k, \tilde{V}_k)) \leq \text{dist}(\pi^*(\mathbf{s}), \tilde{\pi}_k(\mathbf{z})),\ \forall \mathbf{s} \quad (11)$$

and, after generating an appropriate dataset, (ii) learning

$$\mathcal{D} = \{\mathbf{s}, \mathbf{a}\} \text{ with } \mathbf{a} = \overline{\pi}_k(\mathbf{s}, \tilde{\pi}_k, \tilde{V}_k) \quad (12)$$

$$\text{dist}(\tilde{\pi}_{k+1}(\mathbf{z}), \pi^*(\mathbf{s})) \approx \text{dist}(\mathbf{a}, \pi^*(\mathbf{s})),\ \forall (\mathbf{s}, \mathbf{a}) \in \mathcal{D} \quad (13)$$

where $\tilde{V}_k$ is the value network and $\overline{\pi}$ is the expert that maps state, policy and value networks to joint action. Intuitively, the bootstrap property of MCTS (11) generates a dataset of policy samples superior to that of the policy network, and then the supervised learning property (13) matches the quality of the policy network to the quality of the new dataset.

Validating these two properties drives the design of our meta-learning algorithm in Algorithm 1. At each learning iteration k, each robot's policy network is trained by generating a set of states from self-play of the learner policy, $\hat{\pi}_k^i$. Then, the expert policy $\overline{\pi}_k$ evaluates its policy on these states to create a dataset of learning targets for the supervised learning of each robot's policy network $\tilde{\pi}_k^i$. The value network, $\tilde{V}_k$ is then trained to predict the outcome of the game if each team was to play the joint policy $\tilde{\pi}_k$. Because the centralized expert has perfect information, coordination and large computational resources, the bootstrap property is more likely to hold. Furthermore, as the learner generates state space samples through self-play, the dataset is dense in frequently visited areas of the state space, and the models will be more accurate there, validating the learning property.

In addition, we specify a simple POSG generator in Line 4 of Algorithm 1 to select opponent policies and game parameters for self-play. We select the most recent learning iteration networks for self policies, and we uniformly sample the other robot policies across all learning iterations. We sample the number of robots to train on a range of branching factors through the varying dimension of the joint-space action. We also sample the environment size to train on a range of maximum depths of the tree, as planning to the terminal state takes more timesteps in larger environments.

Algorithm 1: Meta Self-Improving Learning

```
1  def Meta-Learning(G*):
2      π̃_0, Ṽ_0 = None, None
3      for k = 0, ..., K do
4          {G}_k = makePOSG(G*, k)
5          for i ∈ I do
6              {s} = selfPlay(π̂((.), π̃_k, Ṽ_k), {G}_k)
7              D^π = {s, π̄_k(s, π̃_k, Ṽ_k))}
8              π̃^i_{k+1} = trainPolicy(D^π)
9          end
10         D^V = {s, selfPlay(π̃_k, {G}_k)}
11         Ṽ_{k+1} = trainValue(D^V)
12     end
```

B. Biased Neural Monte Carlo Tree Search (MCTS)

In order to specify the expert and learner policies, we first explain their common search tree algorithm, shown in Algorithm 2 and adapted from [3] to our setting. For a complete treatment of MCTS, we refer the reader to [3].

The biased MCTS algorithm begins at some start state $\mathbf{s}$ and grows the tree until its computational budget is exhausted, typically measured by number of nodes in the tree, L. Each node in the tree, n, is initialized with a state vector and an action edge to its parent node, i.e. $\mathrm{n}' = \textit{Node}(\mathbf{s}, (\mathrm{n}, \mathbf{a}))$. Each node stores the state vector, $S(\mathrm{n})$, the number of visits to the node, $N(\mathrm{n})$, its children set, $C(\mathrm{n})$, and its action set, $A(\mathrm{n}, \mathrm{n}'),\ \forall \mathrm{n}' \in C(\mathrm{n})$. The growth iteration in the main function, *Search*, has four steps: (i) node selection, *Select*, selects a node to balance exploration of space and exploitation of rewards (ii) node expansion, *Expand*, creates a child node by forward propagating the selected node with an action either constructed by the neural network or by random sampling, (iii) *DefaultPolicy* collects terminal reward statistics by either sampling the value neural network or by rolling out a simulated state trajectory from the new node, and (iv) *Backpropagate* updates the number of

visits and cumulative reward up the tree. The action returned by the search is the child of the root node with the most visits. The primary changes we make from standard MCTS are the integration of neural networks shown in the highlighted sections of Algorithm 2 (called Biased Neural MCTS).

Algorithm 2: Biased Neural MCTS

def *Search*$(\mathbf{s}, \tilde{\pi}, \tilde{V})$**:**
  $\mathrm{n}_0 \leftarrow$ *Node*$(\mathbf{s}, \mathrm{None})$
  for $l = 1, \ldots, L$ **do**
    $\mathrm{n}_l \leftarrow$ *Expand*(*Select*$(\mathrm{n}_0, \tilde{\pi})$)
    $v \leftarrow$ *DefaultPolicy*$(S(\mathrm{n}_l))$
    Backpropagate(n_l, v)
  return $A(\mathrm{n}_0, \arg\max_{\mathrm{n}' \in C(\mathrm{n}_0)} N(\mathrm{n}'))$
def *Expand*$(\mathrm{n}, \tilde{\pi})$**:**
  $\alpha \sim \mathbb{U}(0,1)$
  if $\alpha < \beta_\pi$ **then**
    $\mathbf{a} \leftarrow [\mathbf{a}^1, \ldots, \mathbf{a}^{|\mathcal{I}|}], \quad \mathbf{a}^i \sim \tilde{\pi}^i(h^i(\mathbf{s})), \quad \forall i \in \mathcal{I}$
  else
    $\mathbf{a} \leftarrow [\mathbf{a}^1, \ldots, \mathbf{a}^{|\mathcal{I}|}], \quad \mathbf{a}^i \sim \mathcal{U}^i, \quad \forall i \in \mathcal{I}$
  $\mathrm{n}' \leftarrow$ *Node*$(f(\mathbf{s}, \mathbf{a}), (\mathrm{n}, \mathbf{a}))$
  return n'
def *DefaultPolicy*$(\mathbf{s}, \tilde{V})$**:**
  $\alpha \sim \mathbb{U}(0,1)$
  if $\alpha < \beta_V$ **then**
    $v \sim \tilde{V}(h_y(\mathbf{s}))$
  else
    while $\mathbf{s}$ *is not terminal* **do**
      $\mathbf{a} \leftarrow [\mathbf{a}^1, \ldots, \mathbf{a}^{|\mathcal{I}|}], \quad \mathbf{a}^i \sim \mathcal{U}^i, \quad \forall i \in \mathcal{I}$
      $\mathbf{s} \leftarrow f(\mathbf{s}, \mathbf{a})$
    $v \leftarrow V(\mathbf{s})$
  return v

C. Expert Biased Neural MCTS

The expert, $\overline{\pi}$, is a function from joint state $\mathbf{s}$ to joint action $\mathbf{a}$. The expert computes the action by calling *Search* in Algorithm 2 with a large number of nodes L_{expert}. The expert's perfect information, centralized response, and large computational budget is necessary to guarantee the bootstrap property (11). We found that if the expert is given less computational resources, the learning process is not stable and the quality of the policy and value networks deteriorates over learning iterations. Many of the desirable properties of the expert for theoretical performance make it an infeasible solution for multi-robot applications, motivating the learner.

D. Learner Biased Neural MCTS

The learner for robot i, $\hat{\pi}^i$, is a function from local observation $\mathbf{z}^i$ to local action $\mathbf{a}^i$. The learner computes the action by reconstructing the state from its local observation naively: $\tilde{\mathbf{s}}(\mathbf{z}) = \{\tilde{\mathbf{s}}^j\}, \ \forall j \in \mathcal{N}_A \cup \mathcal{N}_B$, where we assume that the learner has prior knowledge of the absolute goal location. The estimated state could be of a different dimension to the true state if there are robots outside of robot i's sensing radius. Then, the learner calls *Search* in Algorithm 2 with the estimated state and a small number of nodes L_{learner}. The final action $\mathbf{a}^i$ is selected from the appropriate index of the joint-space action returned by *Search*. Both expert and learner predict the actions of the opponent team, but the learner also predicts the actions of other robots on its own team. This communication-less implicit coordination enables operation in bandwidth-limited or communication-denied environments.

E. Policy and Value Neural Networks

We introduce each neural network with its dataset generation and training in Algorithm 1, and its effect on tree growth via integration into Algorithm 2.

Policy Network: The policy network for robot i is an action generator used to create children nodes, mapping observations to action distribution for a single robot. The desired behavior of the policy network is to generate individual robot actions with a high probability of being near-optimal expansions given the current observation, i.e. generate edges to nodes with a high number of visits in the expert search.

The dataset for each robot i's policy network is composed of observation action pairs as computed in Line 7 of Algorithm 1. Each datapoint pair is generated by querying the expert π at some state, $\mathbf{s}_l$, considering the i^{th} robot's action to each of the root node's children, $A^i(\mathrm{n}_0, \mathrm{n}')$ and calculating the action label as the first moment of the action distribution, weighted by the relative number of visits:

$$\mathbf{a}_l^i = \sum_{\mathrm{n}' \in C(\mathrm{n}_0)} \frac{N(\mathrm{n}')}{N(\mathrm{n}_0)} A^i(\mathrm{n}_0, \mathrm{n}') \tag{14}$$

Next, we change the input from state to observation by applying robot i's observation model, $\mathbf{z}_l^i = h^i(\mathbf{s}_l)$. This is a global-to-local learning technique to automatically synthesize local policies from centralized examples [21]. The collection of observation-action samples can be written in a dataset as $\mathcal{D}_\pi^i = \{(\mathbf{z}_l^i, \mathbf{a}_l^i) |\ l = 1, \ldots\}$.

The policy network training in Line 8 in Algorithm 1 is cast as a multivariate Gaussian learning problem, i.e. the output of the neural network is a mean, μ and variance $\mathbf{\Sigma}$. An action sample $\mathbf{a}_l^i \sim \hat{\pi}^i(\mathbf{z}_l^i)$ can then be computed by sampling ϵ and transforming it by the neural network output:

$$\mathbf{a}_l^i = \mu(\mathbf{z}_l^i) + \mathbf{\Sigma}(\mathbf{z}_l^i)\epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{15}$$

The variable size of the input, $\mathbf{z}_l^i$ is handled with a DeepSet [22] feedforward architecture similar to [21]. The solution to the multivariate Gaussian problem [23] is found by minimizing the following loss function:

$$\mathcal{L} = \mathbb{E} \sum_l (\mathbf{a}_l^i - \mu)^T \mathbf{\Sigma}^{-1} (\mathbf{a}_l^i - \mu) + \frac{1}{2} \ln |\mathbf{\Sigma}| \tag{16}$$

$$\tilde{\pi}^i = \arg\min_{\tilde{\pi}^i \in \Pi^i} \mathbb{E}\ \mathcal{L}(\mu(\mathbf{z}_l^i), \mathbf{\Sigma}(\mathbf{z}_l^i), \mathbf{a}_l^i) \tag{17}$$

where $\mu, \mathbf{\Sigma}$ are generated by the neural network given $\mathbf{z}_l^i$, and $\mathbf{a}_l^i$ is the target. To challenge the model assumption

of multivariate Gaussian, we experimented with conditional variational autoencoder techniques (CVAE) [24] capable of generating arbitrary distributions. We found that the empirical results were similar and the CVAE approach needed more data and more epochs to train stable models.

The policy neural network is integrated in Line 9–11 in Algorithm 2 in the expansion operation by constructing a joint-space action from decentralized evaluations of the policy network for all the agents, and then forward propagating that action. We found that using a neural expansion, rather than neural selection as in Alpha Zero, is necessary for planning with small number of nodes in environments with many robots. For example, in a 10 vs. 10 game such as that shown in Fig. 2, the probability of uniform randomly sampling a control action that steers each robot towards the goal within 90 degrees is $(1/4)^{10}$. If the learner policy is evaluated with standard parameters (see Sec. IV-A), it will generate 5 children, which collectively are not likely to contain the desired joint action. Using the neural network expansion operator will overcome this limitation by immediately generating promising child nodes. The policy network is not used at every expand operation to maintain some pure exploration [25]; the hyperparameter $\beta_\pi \in [0, 1]$ determines the relative frequency of querying the network and sampling a uniform random action.

Value Network: The value network is used to gather reward statistics in place of a policy rollout, and is called in the *DefaultPolicy* in Lines 17–19 of Algorithm 2. The value network uses an alternative state representation to be compatible with the estimated state for local computation:

$$\mathbf{y} = h_y(\mathbf{s}) = \left[\{\mathbf{s}^j - \mathbf{g}\}_{j\in\mathcal{N}_A}, \{\mathbf{s}^j - \mathbf{g}\}_{j\in\mathcal{N}_B}, n_{rg}\right] \quad (18)$$

where $h_y(\mathbf{s})$ is the alternative observation function and n_{rg} is the number of robots that have already reached the goal. The value network maps this alternative state representation to the parameters of a multi-variate Gaussian distribution. The desired behavior of the value network is to predict the outcome of games if they were rolled out with the current policy network. The value function implementation in *DefaultPolicy* is the same as Alpha Zero methods.

The value network dataset in Line 10 of Algorithm 1 is generated for all robots at the same time and is composed of alternative state-value pairs. The $\mathbf{y}_l$ state can be generated from $\mathbf{s}$, and the value label, v_l is generated by self-play with the current policy network. The dataset, $\mathcal{D}_V$ can then be written as $\mathcal{D}_V = \{(\mathbf{y}_l, v_l) | \, \forall l = 1, \ldots\}$. Because Alpha Zero methods use a policy selector rather than a generator, the dataset for the value network has to be made by rolling out entire games with MCTS. Instead, we generate the dataset by rolling out the policy network, which is much faster per sample, resulting in less total training time.

The value network is trained in Line 11 of Algorithm 1 with a similar loss function (16) as the policy network, using a learning target of the value labels, v_l, instead of the action $\mathbf{a}_l^i$. The value is also queried from the neural network in a similar fashion (15). The value network uses a similar model architecture as the policy network, permitting variable input size of $\mathbf{y}$. Integration of the value network in *DefaultPolicy* uses the same probabilistic scheme as the policy network with parameter β_V.

IV. Experimental Validation

A. MCTS and Learning Implementation

We implement Algorithm 1 in Python and Algorithm 2 in C++ with Python bindings. For the meta-algorithm, we only train the inner robot loop in Line 5 of Algorithm 1 once per team because we use homogeneous robots and policies. Our MCTS variant uses the following hyperparameters: $L_{\text{expert}} = 10\,000$, $L_{\text{learner}} = 500$, $C_p = 2.0$, $C_{pw} = 1.0$, $\alpha_{pw} = 0.25$ and $\alpha_d = (1-3/(100-10d))/20$ where d is the depth of the node. The neural frequency hyperparameters are $\beta_\pi = \beta_V = 0.5$. The double-integrator game parameters are chosen to match the hardware used in the physical experiments (see Sec. IV-E). We use position bounds $\overline{p} = 1, 2, 3$ m and constant velocity $\overline{v} = 1.0\,\mathrm{m/s}$ and acceleration $\overline{a} = 2.0\,\mathrm{m/s^2}$ bounds. The tag, collision, and sensing radii are: $r_t = 0.2\,\mathrm{m}$, $r_p = 0.1\,\mathrm{m}$, $r_{\text{sense}} = 2.0\,\mathrm{m}$. We train for up to 5 agents on each team. We use a simulation and planning timestep of $\Delta = 0.1\,\mathrm{s}$. Each team starts at opposite sides of the environment, and the goal is placed slightly closer to the defenders' starting position.

We implement the machine learning components in PyTorch [26]. The datasets are of size $80\,000$ points per iteration for both value and policy datasets. The meta learning algorithm is trained over $K = 12$ iterations. The policy and value network models both use DeepSet [22] neural network architecture, (e.g. [21]) with inner, ϕ, and outer, ρ, networks with one hidden layer with 16 neurons and appropriate input and output dimensions. All networks are of feedforward structure with ReLU activation functions, batch size of 1028, and are trained over 300 epochs.

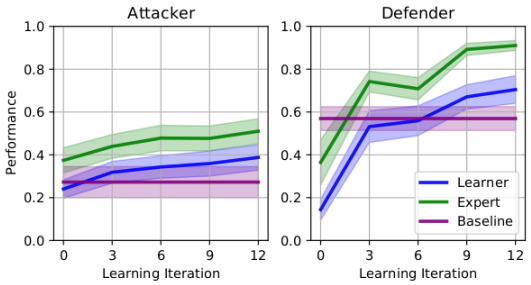
B. Variants and Baseline

In order to evaluate our method, we test the multiple learners and expert policies, each equipped with networks after $k = 0, 3, 6, 9, 12$ learning iterations. To isolate the effect of the neural expansion, we consider the $k = 0$ case for both learner and expert as an unbiased MCTS baseline solution.

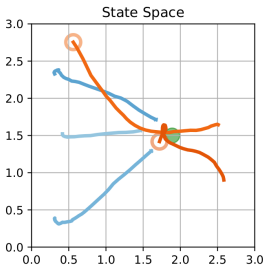
As a baseline for the double-integrator game, we use the solution from [18]. This work adapts the exact differential game solution of simple-motion, single-robot proposed in [15] to a double-integrator, multi-robot setting. However, their adapted solution is not exact because it assumes a constant acceleration magnitude input, and it does have collaborative behavior as it relies on composition of pairwise matching strategies.

C. Simulation Results

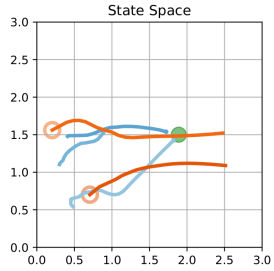
We evaluate our expert and learner by initializing 100 different initial conditions of a 3 attacker, 2 defender game in a $3\,\mathrm{m}$ space. Then, we rollout every combination of the variants and the baseline for both team A and team B policies, for a total of $12\,100$ games. The performance criteria for team A is the averaged terminal reward and



(a) Double-integrator game evaluation: the thick lines indicate the average performance and the shaded area represents the variance over 100 games. The fully-trained learner outperforms both the unbiased expert and the baseline, and approaches the performance of the biased expert.



(b) Learner defense (orange) finds emergent cooperative strategies to defend the goal (green).



(c) Baseline defense (orange) is vulnerable to learner's offensive (blue) dodge maneuver.

Fig. 3. Double-integrator performance and strategy examples.

the performance criteria for team B is one minus average terminal reward in order to have consistency of plots, i.e. higher is better. An example game with different number of agents and environment size is shown in Fig. 2 and an animation of this instance is provided in the supplemental video. The 10 vs. 10 game illustrates the natural scale-ability in number of agents of decentralized approach and the generalize-ability of the neural networks, as they were only trained with data containing up to 5 robots per team.

The statistical results of the 3 vs. 2 experiment are shown in Fig. 3a where the thick lines denote the average performance value and the shade is the performance variance. We find the expected results; for both team A and team B, the learner with no bias has the worst performance, and learner with fully trained networks surpasses the centralized and expensive unbiased expert and approaches the biased expert. In the case of attacking policies, the baseline is relatively weak. However, the baseline's defensive strategy is strong and it provides an analytical solution capable of beating the unbiased expert. In both cases, the fully-trained biased expert and learner are able to significantly outperform the baseline.

To investigate the qualitative advantages of our method, we looked at the games where our learner defense outperformed the baseline defense and found two principal advantages: first, the learner defense sometimes demonstrated emergent coordination that is more effective than pairwise matching strategy, e.g. one defender goes quickly to the goal to protect against greedy attacks while the other defender slowly approaches the goal to maintain its maneuverability, see Fig. 3b. Second, the learner attacker is sometimes able to exploit the momentum of the baseline defender and perform a dodge maneuver, e.g. the bottom left interaction in Fig. 3c, whereas the learner defense is robust to this behavior. These examples show both offensive and defensive behaviors of learner are sophisticated and effective.

D. Dynamics Extension

In order to show the generalizeability of a search tree approach, we adapt our solution to the nonlinear 3D Dubin's vehicle dynamics. Planning solutions for 3D Dubin's vehicle are applicable for fixed-wing aircraft applications [27]. We consider the state, action, and dynamics for $\mathbf{s}_t = [x_t, y_t, z_t, \psi_t, \gamma_t, \phi_t, v_t]^T$ and $\mathbf{a}_t = [\dot{\gamma}_t, \dot{\phi}_t, \dot{v}_t]^T$:

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_t + \begin{bmatrix} v_t \cos(\gamma_t) \sin(\psi_t) \\ v_t \cos(\gamma_t) \cos(\psi_t) \\ -v_t \sin(\gamma_t) \\ \frac{g}{v_t} \tan(\phi_t) \\ \mathbf{a}_t \end{bmatrix} \Delta_t \qquad (19)$$

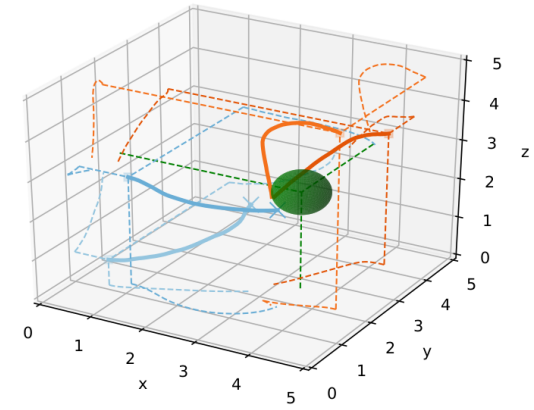
where x, y, z are inertial position, v is speed, ψ is the heading angle, γ is the flight path angle, and ϕ is the bank angle and g is the gravitational acceleration. The game is bounded to $\overline{p} = 5\,\mathrm{m}$ with a maximum linear acceleration of $2.0\,\mathrm{m/s^2}$ and maximum angular rates of $72\,\mathrm{deg/s}$, and g is set to $0.98\,\mathrm{m/s^2}$ to scale to our game length scale.

We initialize 2 attacker, 2 defender games for 100 different initial conditions in a $5\,\mathrm{m}$ region and test the policy variants, without an external baseline, for a total of 10 000 games. An example state space and the performance results are shown in Fig. 4. Again, we see the same trend that the learner with no bias has the worst performance, and the learner with fully trained networks surpasses the centralized, perfect information, and expensive unbiased expert and approaches the biased expert.

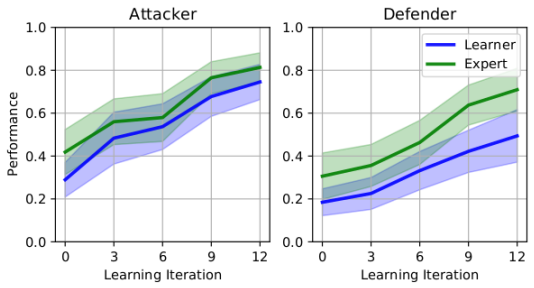
E. Hardware Validation

To test our algorithm in practice, we fly in a motion capture space, where each robot (CrazyFlie 2.x, see Fig. 1) is equipped with a single marker, and we use the Crazyswarm [28] for tracking and scripting. For a given double-integrator policy, we evaluate the learner to construct an action, forward-propagate double-integrator dynamics, and track the resulting position and velocity set-point using a nonlinear controller for full quadrotor dynamics.

We evaluate the double-integrator learner for up to 3 attacker, 2 defender games in an aerial swarm flight demonstration. We show the results of the experiments in our supplemental video. We use the same parameters as in simulation in Sec. IV-C. Our learner evaluation takes an average of $11\,\mathrm{ms}$ with a standard deviation of $6\,\mathrm{ms}$, with each robot policy process running in parallel on an Intel(R) Core(TM) i7-8665U. By comparison, the biased expert takes $329 \pm 144\,\mathrm{ms}$ to execute and the unbiased expert takes $277 \pm 260\,\mathrm{ms}$. Our computational tests show that the learner has significant (≈ 25 times) computational advantage over the baseline unbiased expert. Our physical demonstration



(a) State Space Example



(b) 3D Dubin's vehicle game evaluation: the thick lines indicate the average performance and the shaded area represents the variance over 100 games. The fully-trained learner outperforms the unbiased expert and approaches the performance of the biased expert.

Fig. 4. 3D Dubin's vehicle state space and performance.

shows that our learner is robust to the sim-to-real gap and can run in real-time on off-the-shelf hardware.

V. Conclusion

In this work, we presented a new approach for multi-robot planning in non-cooperative environments with an iterative search and learning method called neural tree expansion. Our method bridges high-performance Alpha Zero method and real-world robotics applications by introducing a learner agent with decentralized evaluation, partial information, and limited computational resources. Our method outperforms the current state-of-the-art analytical baseline for the multi-robot double-integrator Reach-Target-Avoid game with dynamically sophisticated and coordinated strategies. Furthermore, our method readily extends to the 3D Dubin's vehicle dynamics used for fixed-wing aircraft planning problems. We validate the effectiveness through hardware experimentation and show that our policies run in real-time on off-the-shelf computational resources.

Acknowledgement

The authors are thankful to Rob Fuentes of Raytheon for technical discussions.

References

[1] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play", *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[2] M. J. Kochenderfer, C. Amato, G. Chowdhary, *et al.*, "Decision Making Under Uncertainty: Theory and Application", 1st Ed. The MIT Press, 2015, ISBN: 0262029251.

[3] C. Browne, E. J. Powley, D. Whitehouse, *et al.*, "A survey of Monte Carlo tree search methods", *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, 2012.

[4] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning", in *Eur. Conf. Mach. Learn.*, vol. 4212, Springer, 2006.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem", *Mach. Learn.*, vol. 47, no. 2-3, pp. 235–256, 2002.

[6] D. Shah, Q. Xie, and Z. Xu, "Non-asymptotic analysis of Monte Carlo tree search", 2020. arXiv: 1902.05213.

[7] C. R. Mansley, A. Weinstein, and M. L. Littman, "Sample-based planning for continuous action markov decision processes, and scheduling", in *Int. Conf. on Autom. Planning and Scheduling*, 2011.

[8] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári, "*X*-armed bandits", *J. Mach. Learn. Res.*, vol. 12, pp. 1655–1695, 2011.

[9] D. Auger, A. Couëtoux, and O. Teytaud, "Continuous upper confidence trees with polynomial exploration – Consistency", in *Eur. Conf. Mach. Learn.*, vol. 8188, Springer, 2013.

[10] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "A0C: Alpha zero in continuous action space", in *Eur. Workshop on Reinforcement Learn. 14*, 2018.

[11] B. Chen, B. Dai, Q. Lin, *et al.*, "Learning to plan in high dimensions via neural exploration-exploitation trees", in *Int. Conf. on Learn. Representations*, 2020.

[12] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning", in *Proc. Int. Conf. Artif. Intell. and Statist.*, 2011.

[13] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation", in *Neural Inf. Process. Syst.*, 1999.

[14] M. Prajapat, K. Azizzadenesheli, A. Liniger, Y. Yue, and A. Anandkumar, "Competitive policy optimization", 2020. arXiv: 0902.0885.

[15] R. Isaacs, "Differential games; a mathematical theory with applications to warfare and pursuit, control and optimization". Wiley, 1965, ISBN: 0486406822.

[16] E. Garcia, D. W. Casbeer, and M. Pachter, "Optimal strategies for a class of multi-player reach-avoid differential games in 3d space", *IEEE Robotics Autom. Lett.*, vol. 5, no. 3, pp. 4257–4264, 2020.

[17] R. Yan, X. Duan, Z. Shi, Y. Zhong, and F. Bullo, "Matching-based capture strategies for 3d heterogeneous multiplayer reach-avoid differential games", 2019. arXiv: 1909.11881.

[18] M. Coon and D. Panagou, "Control strategies for multiplayer target-attacker-defender differential games with double integrator dynamics", in *2017 IEEE 56th Annual Conf. on Decis. and Control*, 2017.

[19] Z. N. Sunberg and M. J. Kochenderfer, "Online algorithms for POMDPs with continuous state, action, and observation spaces", in *Int. Conf. on Autom. Planning and Scheduling*, 2018.

[20] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning", in *Conf. on Robot Learn.*, vol. 78, 2017.

[21] B. Rivière, 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 Robotics Autom. Lett.*, vol. 5, no. 3, pp. 4249–4256, 2020.

[22] M. Zaheer, S. Kottur, S. Ravanbakhsh, *et al.*, "Deep sets", in *Neural Inf. Process. Syst.*, 2017.

[23] S. Dasgupta and D. J. Hsu, "On-line estimation with the multivariate gaussian distribution", in *Conf. on Learn. Theory*, vol. 4539, Springer, 2007.

[24] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models", in *Neural Inf. Process. Syst.*, 2015.

[25] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning", in *Proc. IEEE Int. Conf. on Robot. and Automat.*, 2018.

[26] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library", in *Neural Inf. Process. Syst.* 2019.

[27] M. Owen, R. W. Beard, and T. W. McLain, "Implementing Dubins airplane paths on fixed-wing UAVs", in *Handbook of Unmanned Aerial Vehicles*. Dordrecht: Springer Netherlands, 2015, pp. 1677–1701, ISBN: 978-90-481-9707-1.

[28] J. A. Preiss, W. Hönig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm", in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017.