Game-theoretic Objective Space Planning

Hongrui Zheng
University of Pennsylvania
Philadelphia, PA, USA
hongruiz@seas.upenn.edu

Zhijun Zhuang
University of Pennsylvania
Philadelphia, PA, USA
zhijunz@seas.upenn.edu

Johannes Betz
Technical University of Munich
Munich, Germany
johannes.betz@tum.de

Rahul Mangharam
University of Pennsylvania
Philadelphia, PA, USA
rahulm@seas.upenn.edu

Abstract: Generating competitive strategies and performing continuous motion planning simultaneously in an adversarial setting is a challenging problem. In addition, understanding the intent of other agents is crucial to deploying autonomous systems in adversarial multi-agent environments. Existing approaches either discretize agent action by grouping similar control inputs, sacrificing performance in motion planning, or plan in uninterpretable latent spaces, producing hard-to-understand agent behaviors. Furthermore, the most popular policy optimization frameworks do not recognize the long-term effect of actions and become myopic. This paper proposes an agent action discretization method via abstraction that provides clear intentions of agent actions, an efficient offline pipeline of agent population synthesis, and a planning strategy using counterfactual regret minimization with function approximation. Finally, we experimentally validate our findings on scaled autonomous vehicles in a head-to-head racing setting. We demonstrate that using the proposed framework significantly improves learning, improves the win rate against different opponents, and the improvements can be transferred to unseen opponents in an unseen environment.

1 Introduction

Motion planning for autonomous agents in adversarial settings remains a challenging problem, especially for systems with continuous dynamics, where the state, action, and observation spaces are uncountably infinite. Consider, for example, a head-to-head race between two autonomous race cars. In order to be competitive, an agent not only needs to perform in the highly dynamically challenging task when there are no other agents but also has to adjust their strategy in the presence of another equally competitive agent.

Usually, autonomous agents in these settings are studied as a partially observed Markov decision process (POMDP). A POMDP is represented by the tuple $(\mathcal{S}, \mathcal{A}, P_{\mathcal{SA}}, \mathcal{O}, r, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, state-action transition probabilities $P_{\mathcal{SA}}$, observation space $\mathcal{O}$, rewards $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and discount factor γ. The objective of each agent in the setting is to find the optimal policy $\pi : \mathcal{O} \to \mathcal{A}$ with parameterization θ by maximizing the cumulative discounted expected rewards.

$$\theta^* = \underset{\theta}{\text{argmax}} \sum_t \gamma^t \mathop{\mathbb{E}}_{P_{\mathcal{SA}}} [r(\pi_\theta(\mathcal{O}(t)))] \tag{1}$$

However, several challenges arise when the state, action, and observation space of the agents are continuous and uncountably infinite.

1. Continuous action space creates additional complexity when optimizing for a policy. Traditional motion planners and optimal controllers are excellent at dealing with this, but it is challenging to

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

optimize them to perform well in an adversarial setting. The first choice for existing solutions is to model the policy $\pi_\lambda(s, a)$ as a distribution with a neural network that predicts the corresponding parameters λ. The actions are then sampled from the distribution. Alternatively, methods discretize the action space of agents, including binning the control input into intervals or using a bang-bang control. In the first case, bang-bang control-like behavior has been observed in learned policies that use distributions [1, 2, 3]. It has been shown that replacing Gaussian distribution-based policies with bang-bang controllers retains the same level of performance [4]. Thus, it is clear that even when continuous actions are used, existing approaches generally fall into grouping action into similar control inputs for agents and provide little control over explainable agent intention.

2. The POMDP setting uses a discount factor γ on the expected rewards. The consequence is that the rewards obtained towards the end will always be weighted less than the rewards obtained in the beginning. In adversarial settings that stretch over longer episodes, for example, head-to-head racing, chess, and Go, most of the reward is not obtained until the very end of the game. The POMDP formulation is myopic and becomes less suitable for such problems.

3. When considering an adversarial setting in which the outcome of the competition is heavily influenced by the interaction between two agents, existing approaches, for example, Sinha et al. [5], maintain a belief vector over parameterization of the opponent agent. However, the performance of such approaches does not generalize when the opponent's strategy goes out of distribution.

To address these issues, we propose an alternative framework, specifically an alternative space, dubbed the ***Objective Space***, in which policies operate and are optimized. Instead of discretizing by grouping, we propose discretizing by *abstraction* (Figure 1). i.e., representing policies by a combination of their properties, or their desired *objectives*. Back to our racing example, a racing policy is represented by how aggressive and conservative it is instead of its parameterization. Within this framework, policies generate interpretable actions by changing the characteristics of agent policies. Furthermore, instead of optimizing for aggregated discounted rewards, we aim to minimize regret where the long-term effect of actions is considered. In short, the overall goal is to formulate the agent's action selection problem into the following equation.

$$\begin{aligned} \{a'_0, a'_1, \ldots, a'_T\}^* = \underset{a'_t \sim \pi'(\theta)}{\operatorname{argmin}} \sum_t R_T(a'_t) \\ \pi'(\theta) \sim \mathcal{P}_{\mathcal{A}'} \end{aligned} \tag{2}$$

Where R_T is the regret generated by the sequence of actions against another agent, $\mathcal{P}_{\mathcal{A}'}$ is an empirical distribution over set $\mathcal{A}'$, a countable finite set of actions.

1.1 Contributions

In summary, existing frameworks do not adequately address the challenges associated with continuous space POMDPs. The primary contribution of this work is:

(i) A agent action discretization method that **encodes interactive agents via abstraction** in a new action space dubbed the ***Objective Space***.

which addresses challenges (1) and (3). The secondary contributions of this work are:

(ii) An efficient pipeline that **synthesizes a population of agents with multi-objective optimization**.

(iii) A novel **game-theoretic planning strategy using counterfactual regret minimization with function approximation** to solve an extensive form game version of the original problem.

which address challenge (2).

1.2 Related Work

1.2.1 RL in Latent Space

Our proposed framework is closely related to methods that operate in a latent space. Typically, encoder architectures are used to create implicit models of the world, agent intentions, agent interactions, or dynamics. Using a neural network to model the evolution of the world was proposed [6] and recently revisited [7]. These approaches allow the agents to train themselves inside "hallucinated dreams" generated by these models. Similarly, Hafner et al. [8] learns the latent dynamics of complex dynamic systems and trains agents in latent imagination for traditionally difficult control tasks. Schwarting et al. [9] uses latent imagination in self-play to produce interesting agent behaviors in racing games. There is also a line of work [10, 11] that combines video prediction models with latent imagination for model-based RL tasks. Finally, Xie et al. [12] uses the latent space to represent the intention of the agent in multi-agent settings. None of the above-mentioned approaches provides explainable latent spaces. In comparison, our proposed approach provides foundation for interpretable spaces where agent actions are still abstracted into a lower dimension.

1.2.2 Regret Minimization with Approximation

Value function approximation [13, 14] is widely used in reinforcement learning. Similarly, we approximate the converged counterfactual regret during the proposed approximated CFR. Jin et al. [15] approximates an advantage-like function as a proxy for regret in a single agent setting. Brown et al. [16] approximates the behavior of CFR in a multi-agent setting. Our differs from both in that we directly approximate the counterfactual regret in a multi-agent setting.

1.2.3 Value Decomposition

When constructing the new agent action space, our approach is inspired by Value Decomposition Networks (VDN) [17]. In VDN, the learned network decomposes value functions for a team of agents in MARL. Our proposed method decomposes long-term reward into surrogate objectives with multiple basis functions.

2 Methodology

2.1 Overview

We first show an overall picture of our proposed method.

First, we discretize agent actions by decomposing agent optimization objectives into a function space with basis functions describing surrogate agent characteristics. For example, for an autonomous race car, how aggressive or conservative the driving policies are. Then we generate the inverse of the function space by generating populations of agents using population-based multi-objective optimization. The inverse is used to define the new discretized agent actions where each action increases or decreases the function value of a single basis function of the function space. Lastly, we form the original problem into a two-player, finite, extensive form game, and solve the game with counterfactual regret minimization (CFR) with function approximation. In the following sections, we will describe details of each module.

2.2 Agent Action Discretization

We take inspiration from the value decomposition [17]. We can think of general policy optimization as "moving" a policy's characteristics towards being performant on a reward defined by the task. Our approach decomposes this single metric by defining multiple characterization functions $f_k : \mathcal{S} \times \mathcal{O} \times \mathcal{A} \to \mathbb{R}$, where each function $f_k, k \in \{1, 2, \ldots, n\}$ models some property k of the outcome (generated trajectory) of each agent policy. For example, in a car racing task, we can model the aggressiveness and restraint of the racing policy. We then form a function space $\mathcal{F} \subseteq \mathbb{R}^n$ using

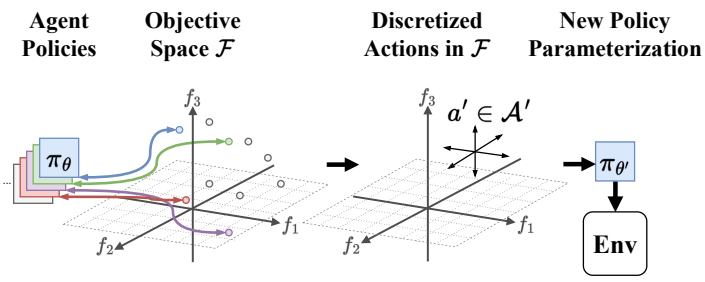

Figure 1: Agent Action Discretization. In the proposed method, agent policies are encoded into the Objective Space via designed basis functions. The new action set corresponds to moving in the Objective Space. When an action is chosen, a new policy parameterization is selected.

these characteristic functions as basis functions. We refer to this function space as the ***Objective Space*** in the following discussion. Let the vector $V_{\mathcal{F}} = [f_1(\theta), f_2(\theta), \ldots, f_n(\theta)]$ denote a point in the Objective Space. This new function space can be related to the original policies of the POMDP as follows.

$$\pi(\theta) = \mathcal{F}^{-1}(V_{\mathcal{F}}) \tag{3}$$

For a point $V_{\mathcal{F}}$ in $\mathcal{F}$, the inverse is defined as:

$$\mathcal{F}^{-1}(V_{\mathcal{F}}) = \underset{\theta}{\text{argmin}} ||V_{\mathcal{F}} - \mathcal{F}(\theta)||_2 \tag{4}$$

We describe how a population of θs are generated for the inverse in Section 2.3. Next, we define a legal action in the new action space $\mathcal{A}'$. For a point $V_{\mathcal{F}} \in \mathbb{R}^n$ in $\mathcal{F}$, the new point induced by an action $a'(V_{\mathcal{F}}), a' \in \mathcal{A}'$ will have the following properties.

$$\begin{aligned} \exists k \in \{1, 2, \ldots, n\}, \quad & |V_{\mathcal{F}}[k] - a'(V_{\mathcal{F}})[k]| = \epsilon, \epsilon > 0 \\ \forall j \neq k, \quad & V_{\mathcal{F}}[j] = a'(V_{\mathcal{F}})[j] \end{aligned} \tag{5}$$

This means that in the new action space $\mathcal{A}'$, actions are increasing or decreasing the value of only one of basis functions. Thus, such a transformation reduces the motion planning space from infinitely large to 2^n, and also provides an explanation for agent actions, depending on how these characterization functions are defined. This addresses challenge one. Furthermore, the input of the characterization functions are state space trajectories, or production of agent policies in the state space. In a POMDP setting, state space trajectories of an adversarial agent are usually assumed to be observable. Thus, modeling opponent agent actions in this discretization setting is fairly straightforward. An overview of the method can be found in Figure 1. This addresses challenge two.

2.3 Agent Population Synthesis

In order to define the inverse, we need to create a discrete population of θs in Equation 4. Denote the set of all possible policy parameterizations as Θ. Since it might not be defined at every point in the function space $\mathcal{F}$, we define the inverse loosely by generating a population of θs using population-based multi-objective optimization where the objectives are the previously defined characterization functions:

$$\min_{\theta \in \Theta}(f_1(\theta), f_2(\theta), \ldots, f_n(\theta)) \tag{6}$$

Note that here the functions could be negated in the minimization depending on the specific characteristic function. Since a θ that minimizes all objectives simultaneously usually does not exist, we use the Pareto Front Θ^* as the population.

$$\begin{aligned} \theta^* \in \Theta^* \iff & \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, |\Theta|\} \quad f_i(\theta^*) \leq f_i(\theta_j) \\ \text{And} \quad & \exists i \in \{1, \ldots, n\}, f_i(\theta^*) < f_i(\theta_j) \end{aligned} \tag{7}$$

The population synthesis process is depicted in Figure 2. Since multi-objective optimization is a well studied research area and not a core contribution of this paper, we refer the readers to the literature [18, 19] for more technical discussions on the subject.

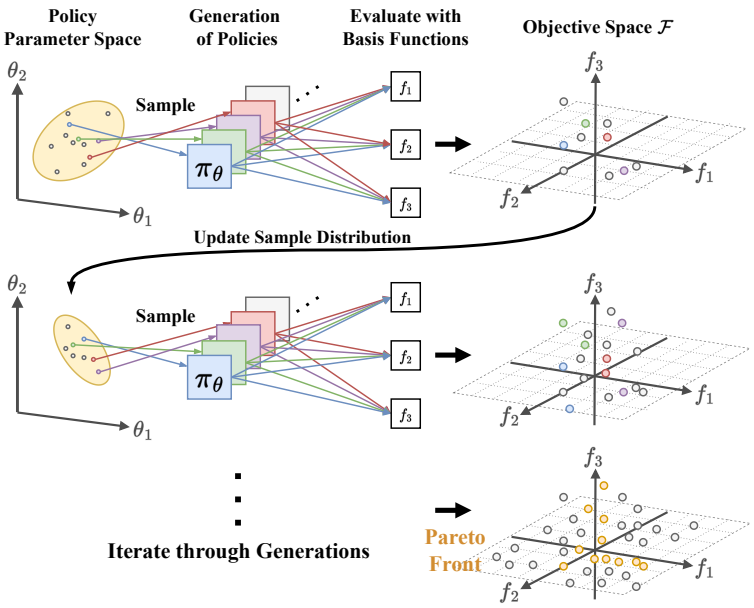

Figure 2: Population Synthesis. Population-based optimization is an iterative process that maintains a sampling distribution. At each iteration, a generation of genomes is sampled using the distribution, then put through the basis functions into the Objective Space. Based on the performance of each genome, the sampling distribution is updated. The figure depicts 2D parameter space and 3D Objective Space, but in practice there are no limits in their dimensions.

2.4 Game Model and Notation

After the discretization of the actions, one could choose their policy optimization method based on the specific application. In this paper, we specifically study a two-player, zero-sum, finite game. Thus, we model it as an Extensive Form Game, which has a tree-like structure. We also make a connection to the original POMDP by also discretizing it in time. We define each transition between nodes on the game tree as a fixed-length sub-episode in the original POMDP. We partition the original POMDP episode $T_{0,\dots,T}$ into m segments $[T_{0,\dots,T/m}, T_{T/m,\dots,2T/m}, \dots, T_{(m-1)T/m,\dots,T}]$. At each transition point between segments, an agent is allowed to perform an action defined in Equation 5. This means that the game tree will have depth m. An important observation here is that even with this discretization, since the state space of the original POMDP is continuous, the new game's state space is still infinite, i.e. there are infinitely many types of nodes in the game tree. We will discuss how to address this in Section 2.5.

Formally, we define the simplified game as a *zero sum extensive form game* with *partial information* and *perfect recall*. The agents have partial information since the original formulation is a POMDP, perfect recall since the agents have unlimited memory. We denote a player as $i \in P$. History $h \in H$ is all current state information, including private information known to subsets of players, and the history of actions taken. The empty set and all prefixes of h are also in H. $Z \subseteq H$ denotes the set of terminal histories. Actions are denoted as $\mathcal{A}'(h) = a'$ where h is nonterminal. Note that this is the same set of actions as defined in Equation 5. The information set, or infoset, $I_i \in \mathcal{I}_i$ for the player i is similar to the history, but only contains information visible to the player i. The strategy of the player i is denoted as $\sigma_i \in \Sigma_i$, which is a distribution over actions $\mathcal{A}'(I_i)$, and Σ_i is the set of all strategies for the player i. Furthermore, the strategy is the probability of taking action a' given the information set I. σ is the strategy profile that comprises all the strategies of the players. σ_{-i} is strategies of all players except i. The probability of reaching history h with the strategy profile σ is denoted by $p^\sigma(h)$. And $p^\sigma(h)_{-i}$ denotes the probability of reaching h without the contribution of the player i. The terminal utility, or payoff, of a player i for a terminal history $h \in Z$ is $u_i(h)$.

2.5 Counterfactual Regret Minimization

Following the definition of the game model in the previous section, it is natural to optimize the strategy of our agents in a regret-minimizing framework. We choose Counterfactual Regret Minimization (CFR). CFR is an iterative algorithm that has a convergence bound of $O\left(\frac{1}{\sqrt{T}}\right)$. CFR minimizes overall regret by minimizing counterfactual regret, and therefore can compute a Nash equilibrium in self-play [20]. However, we still have not addressed the issue of infinite game states. Inspired by value function approximation in RL approaches [13, 14], we address this by approximating the counterfactual regret. After approximating counterfactual regret, we use regret matching (RM) [21] as the strategy of an iteration since it does not require parameters. We next describe the procedure of CFR with approximate counterfactual regret.

The counterfactual value $\mathbf{v}_i^\sigma(I)$ of the player i is the expected utility of the player i reaching I with probability one.

$$\mathbf{v}_i^\sigma(I) = \sum_{z \in Z_I} p_{-i}^\sigma(z[I]) p^\sigma(z[I] \to z) u_i(z) \tag{8}$$

Where Z_I is the set of terminal histories reachable from I and $z[I]$ is the prefix of z up to I. $p^\sigma(z[I] \to z)$ is the probability of reaching z from $z[I]$. And $\mathbf{v}_i^\sigma(I, a')$ follows the same calculation and assumes that player i takes action a' on the information set I with probability one. The immediate or instantaneous counterfactual regret is

$$r^t(I, a') = \mathbf{v}_i^{\sigma^t}(I, a') - \mathbf{v}_i^{\sigma^t}(I) \tag{9}$$

The *counterfactual regret* for information set I and action a at iteration t is

$$R^t(I, a') = \sum_{\tau=1}^{t} r^\tau(I, a') \tag{10}$$

Up to this point, we've condensed the calculation of counterfactual regret into a very compact form. Here, we introduce the function approximator g_ϕ parameterized by ϕ, where we approximate the counter factual regret at iteration t as:

$$R^t(I, a') \approx g_\phi(I, a') \tag{11}$$

Additionally, we clip the counterfactual regret by using $R_+^t(I, a') = \max\{R^t(I, a'), 0\}$. Regret Matching is used to pick the next action. In RM, the strategy for iteration $t + 1$ is:

$$\sigma^{t+1}(I, a') = \frac{R_+^t(I, a')}{\sum_{a' \in \mathcal{A}(I)} R_+^t(I, a')} \tag{12}$$

If the sum of the counterfactual regret of all actions at an iteration is zero, then any arbitrary strategy may be chosen. Finally, to better cope with approximation errors [16], we choose the action with the highest approximate counterfactual regret with probability one:

$$a'_{t+1} = \underset{a'_{t+1} \in \mathcal{A}'}{\operatorname{argmax}} \; R_+^{t+1}(I, a'_{t+1}) = \underset{a'_{t+1} \in \mathcal{A}'}{\operatorname{argmax}} \; g_\phi(I, a'_{t+1}) \tag{13}$$

Since we have the possibility to query the game using many possible combinations of action for both agents, we exploit the convergent behavior of CFR in self play. Therefore, we set the approximation target for g_ϕ not to be the iterative behavior of CFR, but the "converged" counterfactual regrets. In a traditional CFR setup, the set of possible infosets is countable and finite, but the tree depth might be immense; hence, the iterative structure. In contrast, since we have partitioned the POMDP episode into equal length segments, our tree depth can be small by choice. However, our set of possible infosets is uncountable and infinite. Thus, instead of approximating the iterative behavior of CFR, we collect limited samples of full games, then directly predict the "converged" counterfactual regret. The next section will describe the self-play structure and how training samples are collected.

2.6 Collecting Game Samples

The objective of the function approximator g_ϕ is to approximate the counterfactual regret at the final iteration of CFR. Therefore, the input of the approximator is the set of information and the action being evaluated. We set up the two player games as follows. First, we set the depth of the game tree to a fixed integer m. Thus, an agent will take m actions in total for the rollout. The initial starting points for both agents are chosen as random points of $V_\mathcal{F}$ from the Pareto Front Θ^*. Then, we traverse every single branch on the game tree by taking all combinations of action at each node for both agents. At the terminal nodes on the tree, the final utilities are calculated for each game, and the corresponding counterfactual regret is also calculated for every action at every node. Since the number of actions is determined by the number of basis functions f_k chosen, we know $|\mathcal{A}'| = 2^k$. With a tree depth of m, the total number of branches in each game tree is $2^{k(m-1)}$. The total number of games played between two agents using all possible combinations of actions will then be $2^{2k(m-1)}$. If we select N_{init} initial starting points $V_\mathcal{F}$ for both agents, there will be N_{init}^2 number of trees with different root nodes for each agent. Thus, in total, the number of data points collected to train the approximator is $N_{\text{init}}^2 2^{2k(m-1)}$. An information set I at a specific node includes: the history of nodes traversed and the actions taken before arriving at this node, the Objective Space values $V_\mathcal{F}^{\text{ego}}$, $V_\mathcal{F}^{\text{opp}}$ of each agent of the corresponding nodes in the history, More details on network architecture design and the training procedure will be provided in Appendix C.

In summary, we started with a POMDP with continuous states, action, and observation spaces. Then, through abstraction, we discretized the agent's action space. And through partitioning the POMDP into segments of equal duration in time, we were able to optimize for the original POMDP objective in Equation 1 using CFR with counterfactual regret approximation in an extensive form game, regret minimizing setting, giving us the final action selection policy in Equation 13.

3 Experiments

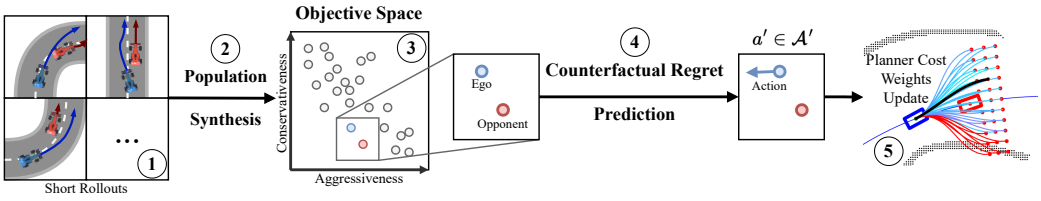

Figure 3: Overview of experiment pipeline. Simulated racing between two autonomous race cars is used as a case study (Marker 1, Section 3.1). A competitive agent population is synthesized offline (Marker 2, Section 3.3). These agents are used to build the Objective Space using basis functions that model the aggressiveness and conservativeness of policies (Marker 3, Section 3.3). Then online, an approximate CFR where counterfactual regret is predicted with a learned model is used to find the best action in the Objective Space (Marker 4, Section 3.4). Lastly, with the updated motion planner parameterization, control inputs for the autonomous race car are produced (Marker 5, Section 3.2).

3.1 Simulation Setup

We study a two-player head-to-head autonomous race scenario as the case study. The simulation environment [22] is a gym [23] environment with a dynamic bicycle model [24] that considers side slip. The objective of the ego in this game is to progress further along the track than the opponent in the given amount of time without crashing into the environment or the other agent. The state space of an agent in the environment is $\mathbf{x} = [x, y, \psi, s]$ where x, y is the position of the agent in the world, ψ is the heading angle, and s is the progress indicator in the Frenet coordinate system [25]. The control input space of an agent is $u = [\delta, v]$ where δ is the steering angle and v is the desired longitudinal velocity. The observation space of an agent is $\mathbf{r} \in \mathbb{R}^q$, where r is the range measurement vector produced by a ray-marching LiDAR simulation, and q is the number of laser beams. Based on the

simulation, the utility of the winner of the game is the scalar value $(s_{\text{winner}} - s_{\text{loser}})$, and the loser is the negative of that to keep the game zero-sum. If there are collisions that end the game prematurely, both agents get zero utility. Since the agents will plan its motion in a receding-horizon fashion, we need to make a clear distinction between the motion planner time steps and the POMDP time steps. Following the notation of the previous sections, the POMDP time steps will use t and T, and the motion planning time steps will use τ and $\mathcal{T}$ in the following discussions.

3.2 Motion Planning and Agent Parameterization

We parameterize the agent policy $\pi(\theta)$ by parameterizing the specific motion planner used by the agent. The motion planner is the link between mapping our defined agent actions in Equation 5 to continuous control input to the agent's dynamic system. We use a sampling-based motion planner that samples local goals for the vehicle in a lattice pattern, as shown in Figure 4. Then, a set of dynamically feasible trajectories Υ is generated [26] that takes the vehicle from the current state to the sampled goal state. The predefined cost functions c_j are then evaluated on all the trajectories sampled. θ in our case is a vector of weights that indicates how much the motion planner considers a certain property (maximum curvature, collision with opponent, etc.) of the proposed trajectories. The trajectory with the lowest weighted sum of the cost function is selected as the final trajectory (Equation 14). More details on trajectory generation and cost function designs can be found in Appendix A.

$$\Upsilon^* = \underset{i \in \{1,2,\ldots,n\}}{\operatorname{argmin}} \sum_{j=1}^{n} \theta_j c_j(\Upsilon_i) \tag{14}$$

After selecting a trajectory, control inputs are generated using Pure Pursuit [27]. We then denote the instantaneous control input given a desired trajectory as u_t defined below.

$$u_\tau = \text{PP}(\Upsilon^*_\tau) = [\delta_\tau, v_\tau] \tag{15}$$

For a sequence of control inputs over a horizon, we denote it as $U_{0,\mathcal{T}} = [u_0, u_1, \ldots, u_\tau]$. As empirical evidence, we can see the different behavior of the agent induced by having different parameterizations in Figure 4. By changing how each cost function weighs in the cost calculation, the motion planner produces very different trajectories. The parameterization in the first figure weighs the penalty on deviating from a predetermined raceline less, thus generating a path that keeps a higher velocity around the opponent vehicle. The parameterization in the second figure weights this penalty more, thus generating a path that slows down and follows the opponent in front, while keeping a trajectory closely matching the raceline.

3.3 Basis Function Definition and Population Synthesis

We next define the basis functions that form the new objective space in which we plan. In this case study, we experiment with a two-dimensional Objective Space. We define two basis functions, one

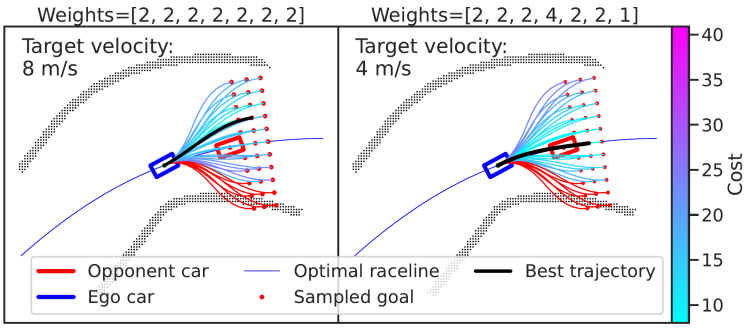

Figure 4: Effect of the different weighting of cost functions on agent behavior. A detailed description of all cost functions can be found in Appendix A. The red trajectories are in collision with the track, thus assigned infinite cost.

modeling the aggressiveness and the other modeling the restraint of the agent. Again, each of the basis functions takes in state space trajectories and observation histories produced by agents, and produces a scalar value. Recall from Section 2.4, we have segmented the original POMDP episode into m sub-episodes. To keep consistency, each basis function will be evaluated on trajectories generated during the same duration with these sub-episodes. For aggressiveness, we design a function that measures how much an agent has made progress on the track more than its opponent. Thus, for a given trajectory, the aggressiveness is as follows.

$$f_{\text{agg}}(\theta_{\text{ego}}) = f_{\text{agg}}(\Upsilon(\theta_{\text{ego}}), \Upsilon(\theta_{\text{opp}})) = s_{\text{ego}} - s_{\text{opp}} \tag{16}$$

For restraint, we design a function that measures how much an agent tries to avoid collision with average minimum instantaneous time-to-collision (iTTC). And the restraint is as follows.

$$f_{\text{res}}(\theta_{\text{ego}}) = f_{\text{res}}(\mathbf{r}_{\text{ego}}) = -\frac{1}{(T/m)} \sum_{t=0}^{T/m} \min_{q} \left[\frac{\mathbf{r}_{t,q}}{\dot{\mathbf{r}}_{t,q}} \right]_{+\infty} \tag{17}$$

Where $\dot{\mathbf{r}}$ is the range rate and is calculated as projections of the longitudinal velocity of the vehicle to the corresponding scan angles of the LiDAR rays. The operator $[]_{+\infty}$ sets the negative elements of the vector to infinity. Note that both outputs of these functions depend either on the opponent agent's trajectory or on the specific track segment that the agents traveled on. We will discuss how these are selected in the population synthesis discussions. Note that for both of these basis functions, the higher the function value, the more aggressive or restraint the agent policy is. Therefore, when synthesizing for a population, we minimize the negated value of these functions (Equation 6).

We synthesize the population of agents using population-based optimization. These optimization algorithms generally have the same recipe. First, a distribution of optimization variables is initialized. Then, a predetermined number (population size) of genomes are sampled from the distribution. Then each genome is evaluated for the given objectives. After collecting all the evaluation results, the sampling distribution is updated based on the results. For example, some algorithms maintain a high-dimensional Gaussian distribution, and update the mean and covariance based on the top performing samples in the last iteration. Iterating through generations, we can build a population using all previously sampled genomes. During the evaluation, we select a fixed number of random sections of the race track and a set of random parameterization θ as opponents. These random selections remain the same throughout generations. Depending on the specific use case, and desired density of the Objective Space, one can choose to subsample the population. In our case, we use a near-optimal set based on the Pareto front Θ^*, denoted $\mathbb{P}_{\text{no}}$, so that the selected policies are optimized for each objective. In all experiments where a subset of initial points in the Objective Space is needed, we use a Determinantal Point Process (DPP) [28] using Euclidean distances in the Objective Space for subsampling. More details on the optimization algorithm and how we create the subsets can be found in Appendix B.

3.4 Regret Prediction Model

When collecting training samples for the approximator g_ϕ, we employ the self-play structure described in Section 2.6. First, two subsets of the near-optimal set $\mathbb{P}_{\text{DPP1}}$ and $\mathbb{P}_{\text{DPP2}}$, where $\mathbb{P}_{\text{DPP1}} \cap \mathbb{P}_{\text{DPP2}} = \varnothing$, and $|\mathbb{P}_{\text{DPP1}}| = |\mathbb{P}_{\text{DPP2}}|$. We construct game trees using each pair in the Cartesian product $\mathbb{P}_{\text{DPP1}} \times \mathbb{P}_{\text{DPP2}}$ as the initial starting points in the Objective Space of the two agents. We then play through the entire game tree and collect the necessary data points for training. We used a multilayer perceptron (MLP) with one hidden layer of size 2048 and leaky ReLU activation as g_ϕ. The network is trained using L1 loss on the prediction targets, and optimized with Adam with adaptive learning rate for 2000 epochs. More details on the number of each chosen subset and the total number of data points can be found in Appendix C.

To put everything together, the game-theoretic planner works in the following order. First, the ego selects a random starting point on the Pareto Front in the Objective Space. Then, the approximated CFR observes the opponent's trajectory to locate it in the Objective Space, and predicts the counterfactual regret for each available action. Next the action with highest approximate counterfactual

regret is taken, and moves the ego's current position in the Objective Space to a new point. The corresponding cost weights for the motion planner are used to update the motion planner. Lastly, the motion planner generates the control input for the ego agent.

3.5 Simulated Racing

In the experiments, our aim is to answer the following four questions.

1. Does the agent action discretization aid our agent in learning a more competitive and general policy?
2. Does being game-theoretic improve an agent's win rate against a competitive opponent?
3. Does the proposed agent action discretization provide interpretable explanations for agent actions?
4. Does the proposed approach generalize to unseen environments and unseen opponents?

3.5.1 Action Discretization in Learning

To answer the first question, we compare the results of a single agent environment where the objective is to finish two laps as fast as possible on the race track without crashing. We compare three agents in this experiment. The first is PPO [29] with continuous actions on both steering and throttle. The second is PPO with discretized actions: turn left, turn right, and stay straight. The third is our proposed approach without the CFR updates. The agents under both PPO settings are rewarded by a small keep alive reward for every time step the car is not in collision and a large terminal reward when finishing two laps under the time limit. They are also penalized by a value scaled with lap time. The PPO policies use the range measurement vector from the LiDAR scan as the observation. During training, the PPO agent with discrete actions converges to a policy that can finish two laps. In comparison, the PPO agent with continuous actions does not. We set up the experiments as follows. For PPO agents, the same trained agent is used in all rollouts, and the starting positions of the agents are slightly different in each rollout. For agents using the proposed approach, we again sample a subset of the near-optimal set using DPP. The number of DPP samples here matches the number of random starts for the PPO agents. We record the success rate and lap times over 20 trials for each agent both on the seen map and on the unseen map. More details on the implementation of PPO agents can be found in Appendix D.

Table 1: Success rate and elapsed times of different agents finishing two laps in a single agent setting.

Agent	Success Rate	Avg. Elapsed Time (s)
On Seen Map		
PPO-continuous	0.0	N/A
PPO-discrete	0.3	63.782 ± 0.225
Ours (w/o CFR)	**1.0**	48.533 ± 1.398
On Unseen Map		
PPO-continuous	0.0	N/A
PPO-discrete	0.25	67.292 ± 0.156
Ours (w/o CFR)	**1.0**	50.814 ± 1.343

The recorded results are shown in Table 1. First, we compare the two PPO agents to see the effect of using discretized actions. Our experiment confirms that agents with discretized actions are easier to train. At the end of training, the continuous PPO agent, although turning in the correct direction when encountering corners, was unable to complete the entire lap. Then we compare the discrete PPO with our proposed method. Although able to complete the two laps during training, simply changing the starting position by a small amount drops the success rate to only 30% even when evaluations are performed on the same map the agent saw during training. When moved to an unseen map, the success rate drops further to only 25%. In comparison, the proposed method maintains a

perfect success rate throughout the evaluations, even when moved to an unseen map. Although not the main objective of this experiment, we see that the proposed method was able to achieve a lower lap time across the board. Although PPO agents are awarded for having shorter lap times, they cannot compete with the proposed method.

3.5.2 Effect of being Game-theoretic

Table 2: Win-rates in head-to-head racing experiments with mean win rate differences and p-values.

Opponent	Ego		Δ_μ	p-value
	Win-rate Non-GT	Win-rate GT		
On Seen Map				
Non-GT	0.515 ± 0.251	0.569 ± 0.213	0.054	0.0142
Random	0.624 ± 0.225	0.670 ± 0.199	0.046	0.00370
Unseen	0.586 ± 0.101	0.597 ± 0.089	0.011	0.0863
On Unseen Map				
Non-GT	0.553 ± 0.256	0.628 ± 0.180	0.075	0.0124
Random	0.625 ± 0.278	0.738 ± 0.172	0.113	0.00276
Unseen	0.556 ± 0.101	0.565 ± 0.098	0.009	0.147

To answer the second question, and in order to show the effectiveness of game-theoretic planning, we race the policies from our framework against various opponents in different environments. Each experiment is a single head-to-head race with a fixed duration. We designate the winner of the race as the agent who progressed further down the track at the end of the duration. To ensure fairness, the agents start side by side at the same starting line on track, and alternate starting positions. The starting line is also randomized five times for one pair of agents. There are three types of agents in the experiment. The GT agent is produced under the proposed population synthesis framework with CFR and counterfactual regret approximation. The non-GT agent is also produced in the same population synthesis framework but without CFR. The random agent is a random selection from all the explored parameterizations from the population synthesis framework without CFR. Lastly, the unseen agent is a competitive motion planner under a completely different parameterization. The ego agents in the experiments are GT and non-GT agents, while the opponents are non-GT, random, and unseen (Table 2). We choose 20 different variants of each agent. Thus, each table cell in Table 2 consists of statistics from $20^2 * 2 * 5 = 4000$ head-to-head racing games. Each agent is allowed $m = 4$ actions, with each sub-episode lasting $T/m = 8$ seconds. The total length of the games is 40 seconds, with the first 8 seconds as the initial sub-episode to observe the opponent. We report the results of paired t-tests against all opponents with the null hypothesis that the use of the proposed CFR process does not change the win rate. As shown in Table 2, the main result of this experiment is that the p-value is small enough to reject the null hypothesis in most cases. Across all pairings of ego and opponent, there is an increase in average win rate by using our proposed approximated CFR, significantly in most cases. Although not as significant when playing an unseen opponent, the trend is still clearly present. This finding validates our counterfactual regret approximator and the effect of CFR by showing significant improvement.

3.5.3 Agent Characterization

To answer the third question, we examine selected rollouts of the racing games to investigate whether the actions of the agents are interpretable. In the first segment of the track (before the decision point) in Figure 5, the ego agent uses a random choice of starting parameterization to plan and observe the opponent for T/m seconds. The ego and the opponent remained side by side until the second corner. Then at the decision point, which is the moment we select an action and transition from one node on the game tree to the next, the ego observed that the opponent's strategy produced a trajectory that is positioned in the lower right quadrant of the Objective Space (more conservative than aggressive). The counterfactual regret approximator then predicted that increasing aggressiveness has the highest estimated counterfactual regret. The planner switches to the new cost weights corresponding to the

new point in the objective space (left subplot). The selected action's effect is immediately evident since the ego slowed down less than the opponent in the chicane (after the decision point) and overtook the opponent by the end of the second sub-episode.

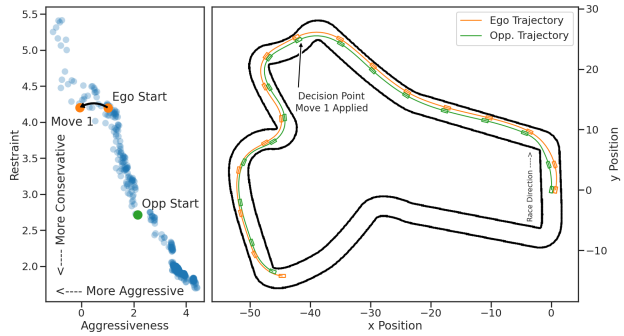

Figure 5: Effect of taking an action in the Objective Space. The left subplot shows what the action looks like in the Objective Space. The right subplot shows that after taking an action, the ego overtakes the opponent.

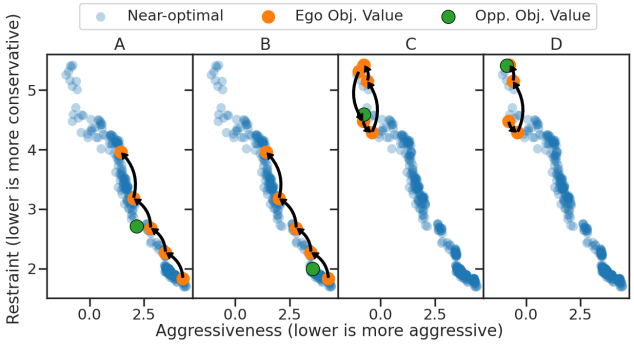

Figure 6: Trajectories of ego's actions in Objective Space. In subplots A and B, the opponent is more conservative and the ego decides to increase aggressiveness right away. In subplots C and D, the opponent is more aggressive and the ego decides to increase in restraint until there is an opportunity to increase in aggressiveness and overtake.

We then take a step back and look at a bigger picture which shows agent decisions of the entire rollout in the Objective Space. In Figure 6, four different rollouts are shown in which the ego wins at the end. In subfigures A and B, the opponent agent is observed to be in the lower right quadrant of the Objective Space, meaning that these agents value safety more than progress. In these scenarios, the ego agent, starting in the lower right quadrant, became more aggressive than conservative with each action. In subfigures C and D, the opponent agent is observed to be in the upper left quadrant of the objective space, meaning that these agents value progress more than safety. In these scenarios, the ego decides to stay conservative and only increases aggressiveness later on when there is an opportunity to overtake.

From these two closer examinations of agent actions, using interpretable basis functions to construct the Objective Space has a clear benefit. Discretizing agent actions by abstraction provides a much clearer context for explaining agent decision-making.

3.5.4 Facing Unknown Opponents

And lastly, to answer the fourth question, we examine the results when faced with unknown environments. Since we defined the Objective Space basis function to take the outcome (generated state space trajectories) of a policy as input, agents deal with unseen adversaries in unseen environments

better. In the adversarial multi-agent environment from rows 3 and 6 in Table 2, we see that the increase in the win rate is retained, though not as significant as other scenarios.

4 Limitations and Future Work

First, we chose to partition the original POMDP episode into equal duration segments. This choice led to a distinct game tree structure with depth m, and kept the input space of the counterfactual regret approximator g_ϕ manageable. If agent decision making, or strategy switching, is desired at a higher frequency, one might employ a receding horizon update scheme. However, this might lead to too large a tree depth m when designing the function approximator. Future work should focus on addressing this issue by finding the balance point for this trade-off.

Second, experiments should investigate biases in the evaluation scenarios during the optimization and experiments. Randomness affects the selection of the opponent set and the track sections during optimization. The biases in selecting evaluation scenarios are also present in the first column of Table 2. When a non-GT agent plays against a non-GT agent, the ego win rate should be close to 50%. Future work can focus on designing experiments to evaluate whether this deviation from the 50% win rate is significant. Furthermore, biases must be eliminated when selecting random subsets of the agent population and evaluating scenarios.

Third, instead of playing against randomized opponents during population-based optimization, the opponent set should become more and more competitive as the optimization iterates. The authors experimented with mixing agents on the Pareto front periodically into the opponent set, resulting in premature convergence to less competitive agents. An alternative is to incorporate self-play into the acceptance of new genomes. Instead of accepting all new genomes in each generation, new genomes are compared against the ones from the previous generation. Only winning genomes are passed on to the next generation.

Fourth, opponents in the experiments do not adapt their planner cost weights like ego agents. Using an adaptive planner would increase competition for a fairer two-sided interaction.

Lastly, the basis functions used in the case study are chosen by hand with expert knowledge of the subject matter. On the basis of the proposed formulation, this should not be a hard requirement for improvement in agent action discretization. Discovering these basis functions automatically and selecting the metrics most different from each other to construct the Objective Space will create an interesting research problem where hidden mechanics of agent interaction could be discovered.

5 Conclusion

To overcome the challenges that come with continuous-space POMDPs, we propose an agent action discretization that encodes policy characteristics into the Objective Space. Agent actions produced in this space are interpretable and help with generalization. The central hypotheses of this paper are that using the proposed discretization of agent action and CFR with counterfactual regret approximation not only significantly improves the win rate against different opponents, improves interpretability when explaining agent decisions, but also transfers to unseen opponents in an unseen environment. We first define the Objective Space and legal actions in this space, then perform agent population synthesis via multi-objective optimization. Next, we train a counterfactual regret approximator and implement an online planning pipeline that uses CFR to maximize the win rate. Lastly, we provide statistical evidence showing significant improvements to the win rate that are generalized to unseen environments. Moreover, we provide an examination on how the agent action discretization method improves interpretability when explaining agent decisions. We significantly (with a p-value less than 0.05) improved the win rate by 5.4% on the seen map and 7.5% on the unseen map on average against non-game-theoretic opponents; the win rate by 4.6% on the seen map and 11.3% on the unseen map on average against random opponents. Lastly, we improved the win rate by 1.1% on the seen map and 0.9% on the unseen map against unseen opponents.

6 Societal Impact

Understanding the intent of a learning-based agent and its peers in the environment is crucial to deploying autonomous systems in adversarial multi-agent settings. Real-world applications, such as autonomous vehicles, financial decision making algorithms, and recommendation algorithms, are examples of such autonomous systems. Without a proper system that supports explaining decisions made by such systems in a human-interpretable way, it is impossible to assign blame when malfunctions occur. Especially in life-critical applications, these requirements become even more important. This work provides a preliminary framework for interpretable agent actions in autonomous systems.

References

[1] S. H. Huang, M. Zambelli, J. Kay, M. F. Martins, Y. Tassa, P. M. Pilarski, and R. Hadsell. Learning Gentle Object Manipulation with Curiosity-Driven Deep Reinforcement Learning, Mar. 2019. URL http://arxiv.org/abs/1903.08542. arXiv:1903.08542 [cs].

[2] G. Novati and P. Koumoutsakos. Remember and Forget for Experience Replay, May 2019. URL http://arxiv.org/abs/1807.05827. arXiv:1807.05827 [cs, stat].

[3] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi. Model-Based Reinforcement Learning for Closed-Loop Dynamic Control of Soft Robotic Manipulators. *IEEE Transactions on Robotics*, 35(1):124–134, Feb. 2019. ISSN 1552-3098, 1941-0468. doi:10.1109/TRO.2018.2878318. URL https://ieeexplore.ieee.org/document/8531756/.

[4] T. Seyde, I. Gilitschenski, W. Schwarting, B. Stellato, M. Riedmiller, M. Wulfmeier, and D. Rus. Is Bang-Bang Control All You Need? Solving Continuous Control with Bernoulli Policies, Nov. 2021. URL http://arxiv.org/abs/2111.02552. arXiv:2111.02552 [cs].

[5] A. Sinha, M. O'Kelly, H. Zheng, R. Mangharam, J. Duchi, and R. Tedrake. FormulaZero: Distributionally Robust Online Adaptation via Offline Population Synthesis. In *Proceedings of the 37th International Conference on Machine Learning*, pages 8992–9004. PMLR, Nov. 2020. URL https://proceedings.mlr.press/v119/sinha20a.html. ISSN: 2640-3498.

[6] J. Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 253–258 vol.2, San Diego, CA, USA, 1990. IEEE. doi:10.1109/IJCNN.1990.137723. URL http://ieeexplore.ieee.org/document/5726682/.

[7] D. Ha and J. Schmidhuber. World Models. Mar. 2018. doi:10.5281/zenodo.1207631. URL http://arxiv.org/abs/1803.10122. arXiv:1803.10122 [cs, stat].

[8] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to Control: Learning Behaviors by Latent Imagination, Mar. 2020. URL http://arxiv.org/abs/1912.01603. arXiv:1912.01603 [cs].

[9] W. Schwarting, T. Seyde, I. Gilitschenski, L. Liebenwein, R. Sander, S. Karaman, and D. Rus. Deep Latent Competition: Learning to Race Using Visual Control Policies in Latent Space, Feb. 2021. URL http://arxiv.org/abs/2102.09812. arXiv:2102.09812 [cs].

[10] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski. Model-Based Reinforcement Learning for Atari, Feb. 2020. URL http://arxiv.org/abs/1903.00374. arXiv:1903.00374 [cs, stat].

[11] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine. SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 7444–7453. PMLR, May 2019. URL https://proceedings.mlr.press/v97/zhang19m.html. ISSN: 2640-3498.

[12] A. Xie, D. Losey, R. Tolsma, C. Finn, and D. Sadigh. Learning Latent Representations to Influence Multi-Agent Interaction. In *Proceedings of the 2020 Conference on Robot Learning*, pages 575–588. PMLR, Oct. 2021. URL https://proceedings.mlr.press/v155/xie21a.html. ISSN: 2640-3498.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning, second edition: An Introduction*. MIT Press, Nov. 2018. ISBN 978-0-262-35270-3. Google-Books-ID: uWV0DwAAQBAJ.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning, Dec. 2013. URL http://arxiv.org/abs/1312.5602. arXiv:1312.5602 [cs].

[15] P. Jin, K. Keutzer, and S. Levine. Regret Minimization for Partially Observable Deep Reinforcement Learning, Oct. 2018. URL http://arxiv.org/abs/1710.11424. arXiv:1710.11424 [cs].

[16] N. Brown, A. Lerer, S. Gross, and T. Sandholm. Deep Counterfactual Regret Minimization. Nov. 2018. doi:10.48550/arXiv.1811.00164. URL https://arxiv.org/abs/1811.00164v3.

[17] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning, June 2017. URL http://arxiv.org/abs/1706.05296. arXiv:1706.05296 [cs].

[18] R. Marler and J. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, Apr. 2004. ISSN 1615-1488. doi:10.1007/s00158-003-0368-6. URL https://doi.org/10.1007/s00158-003-0368-6.

[19] N. Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[20] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret Minimization in Games with Incomplete Information. In *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper/2007/hash/08d98638c6fcd194a4b1e6992063e944-Abstract.html.

[21] S. Hart and A. Mas-Colell. A Simple Adaptive Procedure Leading to Correlated Equilibrium. *Econometrica*, 68(5):1127–1150, 2000. ISSN 1468-0262. doi:10.1111/1468-0262.00153. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-0262.00153. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1468-0262.00153.

[22] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam. F1TENTH: An Open-source Evaluation Environment for Continuous Control and Reinforcement Learning. *Proceedings of Machine Learning Research*, 123, Apr. 2020. URL https://par.nsf.gov/biblio/10221872-f1tenth-open-source-evaluation-environment-continuous-control-reinforcement-learni

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, June 2016. URL http://arxiv.org/abs/1606.01540. arXiv:1606.01540 [cs].

[24] M. Althoff, M. Koschi, and S. Manzinger. CommonRoad: Composable benchmarks for motion planning on roads. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 719–726, June 2017. doi:10.1109/IVS.2017.7995802. URL https://ieeexplore.ieee.org/abstract/document/7995802?casa_token=44FC2OFG6RcAAAAA:k7KqKJTKPoT7qzAA_PYWEWjzNoiZR805_3hrVNbJMuke313u318YUQ4eVTiy91ZrVp6LjrRzYVQ.

[25] M. Werling, J. Ziegler, S. Kammel, and S. Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE international conference on robotics and automation*, pages 987–993. IEEE, 2010.

[26] A. Kelly and B. Nagy. Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control. *The International Journal of Robotics Research*, 22(7-8):583–601, July 2003. ISSN 0278-3649. doi:10.1177/02783649030227008. URL https://doi.org/10.1177/02783649030227008. Publisher: SAGE Publications Ltd STM.

[27] R. C. Coulter et al. *Implementation of the pure pursuit path tracking algorithm*. Carnegie Mellon University, The Robotics Institute, 1992.

[28] A. Kulesza and B. Taskar. Determinantal Point Processes for Machine Learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, Dec. 2012. ISSN 1935-8237, 1935-8245. doi:10.1561/2200000044. URL https://www.nowpublishers.com/article/Details/MAL-044. Publisher: Now Publishers, Inc.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, Aug. 2017. URL http://arxiv.org/abs/1707.06347. arXiv:1707.06347 [cs].

[30] D. Ferguson, T. M. Howard, and M. Likhachev. Motion planning in urban environments. *Journal of Field Robotics*, 25(11-12):939–960, 2008. ISSN 1556-4967. doi:10.1002/rob.20265. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20265. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20265.

[31] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann. Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics*, 58(10):1497–1527, Oct. 2020. ISSN 0042-3114. doi:10.1080/00423114.2019.1631455. URL https://www.tandfonline.com/doi/10.1080/00423114.2019.1631455. Publisher: Taylor & Francis.

[32] B. Nagy and A. Kelly. TRAJECTORY GENERATION FOR CAR-LIKE ROBOTS USING CUBIC CURVATURE POLYNOMIALS.

[33] T. M. Howard and A. Kelly. Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots. *The International Journal of Robotics Research*, 26(2):141–166, Feb. 2007. ISSN 0278-3649. doi:10.1177/0278364906075328. URL https://doi.org/10.1177/0278364906075328. Publisher: SAGE Publications Ltd STM.

[34] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE International Conference on Robotics and Automation*, pages 4889–4895, May 2011. doi:10.1109/ICRA.2011.5980223. URL https://ieeexplore.ieee.org/abstract/document/5980223?casa_token=6oERxpLXAXAAAAAA:26DT28GkkAlWWvnMTWPBdMeuNu1f0bs5T6nT_3E2wAPPTeoT5O8tEVIKOowSqzxlhcCEgcNiCtQ. ISSN: 1050-4729.

[35] T. M. Howard. *Adaptive model-predictive motion planning for navigation in complex environments*. Carnegie Mellon University, 2009.

[36] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

A Motion Planner

We use a sampling-based hierarchical motion planner similar to that of [30]. At the top level, the planner receives information on the current poses and velocities of the ego and opponent, as well as an optimal race line generated using [31] for the current map. The race line consists of waypoints as tuples of (x, y, θ, v), which are the desired position, heading, and velocity of the vehicle. Then n uniform grid points representing local goals are sampled around the race line (see Figure 4). Then dynamically feasible trajectories are generated from the current pose of the car to the sampled goals using third-order clothoids combining methods from [32, 26, 33, 34, 35]. Additionally, for each generated trajectory, m velocity scaling factors are assigned to generate different velocity profiles for the same path. Hence, the planner samples $n \times m$ trajectories at one planning step.

We define multiple cost functions (c_j in Equation 14) to evaluate the quality of each trajectory for geometric properties and velocity profiles. We use the following cost functions:

- $c_{\text{mc}} = \max_{s=0}^{s_f}\{\kappa_s\}$: maximum curvature on the trajectory
- $c_{\text{al}} =$ arc length s_f of the trajectory
- $c_{\text{hys}} = \left|\left|\mathcal{T}_t - \mathcal{T}^*_{t-1}\right|\right|_2$ hysteresis loss that measures similarity to the previously selected trajectory. Calculated as the Euclidean distance between the two trajectories. Note that the previously selected trajectory is shifted forward to compensate for the vehicle's motion between time steps.
- c_{do} distance to the optimal race line measured by lateral deviation. For every single position on the trajectory, a corresponding nearest point is found on the optimal raceline. Then the lateral is calculated in the Frenet coordinate frame.
- c_{co} fixed collision cost with the opponent discounted by relative speed to the opponent at each time step.
- c_{v1} velocity cost that encourages higher speed.
- c_{v2} velocity cost that penalizes co-occurrence of high speed and high curvature.

In addition, we include a global velocity scaling factor γ as another parameter for the agent. Therefore, each agent can be parameterized by the weight vector θ:

$$\theta = [\gamma,\ c_{mc},\ c_{al},\ c_{\text{hys}},\ c_{do},\ c_{co},\ c_{v1},\ c_{v2}]$$

In addition, trajectories in collision with the environment are assigned infinite costs.

B Population-based Agent Optimization

We use the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [19] as the population-based multi-objective optimizer. Parameterization of each genome in the ES has the seven cost weights ranging from 1.0 to 10.0 and the velocity discount factor ranging from 0.6 to 1.0. The duration of each rollout is 8 seconds in the evaluation. In the evaluation to obtain the Objective Space position of each genome, we use 120 pairs of opponents with randomized cost weights and randomized section of the race track. The average basis function values across all pairs obtained for each genome are used to put them into the Objective Space. To encourage exploration, during every rollout, if the ego overtakes the opponent, the aggressiveness value of the genome increases by 10%. If the ego crashes into the opponent, the aggressiveness value increases by 10% and the conservativeness value increases by 1.

In Figure 7, we show the progression of the two competing objectives and the crash and overtake rate during optimization. The figure shows that the overtake rate is higher when the current parameterization is more aggressive. However, the crash rate is also higher. The trend of the objective scores in the figure shows that the two objectives are competing, and we have explored a wide range of different parameterizations.

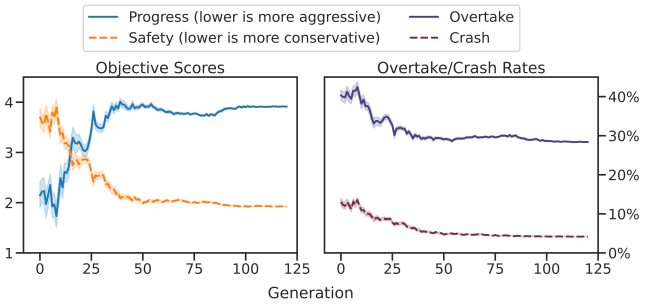

Figure 7: Progression of CMA-ES optimization. The x-axis denotes generations of 100 genomes.

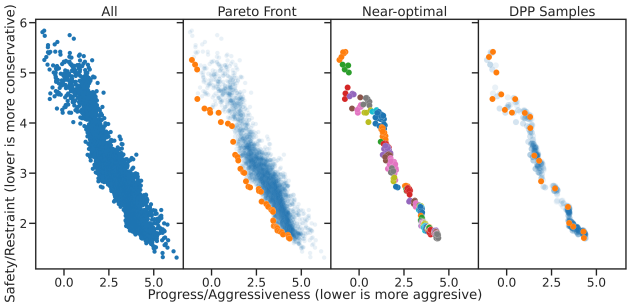

Figure 8: Position of different sets of prototypes in the Objective Space.

After obtaining the Pareto front $\mathbb{P}_{\text{pf}}$ and the set of all agents explored $\mathbb{P}_{\text{all}}$ from CMA-ES, we create several different subsets of the agents explored using the Objective Space. First, we create a near-optimal set $\mathbb{P}_{\text{no}}$ using all points in the agents explored within $d_{\text{near}} = 0.3$ away in Euclidean distance from every point in $\mathbb{P}_{\text{pf}}$. Next, we use a Determinantal Point Process (DPP) to sample $N_{\text{DPP}i} = 20$ samples from the near-optimal set into $\mathbb{P}_{\text{DPP}i}$. Multiple DPP subsets are used in our experiments.

Figure 8 shows the subsets of prototypes $\mathbb{P}_{\text{all}}$, $\mathbb{P}_{\text{pf}}$, $\mathbb{P}_{\text{no}}$, and an example DPP subset $\mathbb{P}_{\text{DPP1}}$ in the 2D Objective Space. This figure shows the coverage of explored agents in the Objective Space.

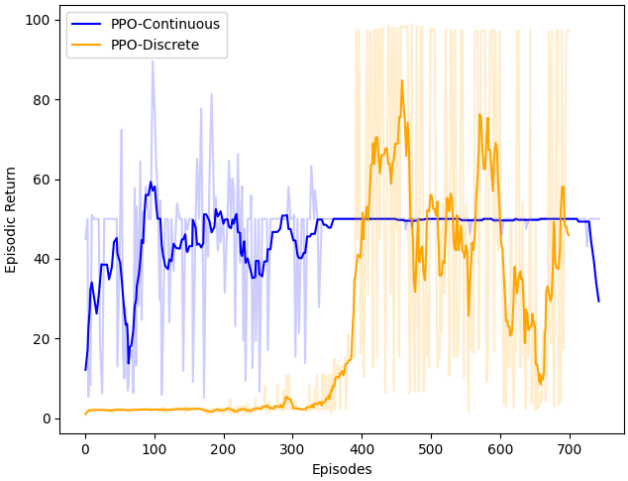

Figure 9: Episodic returns when training PPO policies. Moving average of 10 steps shown.

C Counterfactual Regret Prediction

In our experiments, we choose our game tree depth as $m = 4$ and the dimension of the Objective Space as $k = 2$. The total number of branches is $4^3 = 64$. The possible combinations of actions taken between two agents are $64^2 = 4096$ games. When $N_{\text{init}} = 20$, the total number of games played in data collection is $20^2 * 4096 = 1638400$. We calculated and collected the counterfactual regret at each data point for querying. When two nodes are on the same branch of the game tree, they have the same terminal utility.

We encode Objective Space values into fixed-length vectors with zero padding. Masks are set up so that only positions with objective values are filled with ones; otherwise, zeros. All actions are one-hot encoded. In total, the input feature size for the prediction model is 40. We use a multilayer perceptron (MLP) with one hidden layer of size 2048 and leaky ReLU activation between layers. We train the network with L1 loss, batch size of 1024, and Adam optimizer with adaptive learning rate (starting at 0.005, reduce when validation loss plateaus for ten steps) in 2000 epochs.

D PPO Agents

RL agents used for the comparison are trained using continuous and discrete PPO in CleanRL [36]. We show the episodic returns for each agent in 9. The training is capped at 1000000 total time steps, with 7000 steps between each policy update. The learning rate is annealed and starts at 3e-4 for continuous PPO and 2.5e-4 for discrete PPO. The discount factor for both is set at 0.99. The λ for the general advantage estimation for both is set at 0.95. The epochs to update is 10 for the continuous policy and 4 for the discrete policy. The surrogate clipping coefficient is 0.2 for both policies. The entropy coefficient is 0 for the continuous policy and 0.01 for the discrete policy. The coefficient for the value function is 0.5 for both policies. The maximum norm for gradient clipping is 0.5 for both policies. The continuous action is sampled from a learned Gaussian with control input limits. The discrete action is steering with 0.3, -0.3, and 0.0 steering angles, all with a velocity of 5.0 m/s. Agents get a reward of 0.01 for every time step not in collision, a reward of 40 for reaching the end of two laps without collision. The gym environment is set with a timeout to avoid infinite episodes.