

RAPID: Autonomous Multi-Agent Racing using Constrained Potential Dynamic Games

Yixuan Jia¹, Maulik Bhatt², and Negar Mehr²

Abstract—In this work, we consider the problem of autonomous racing with multiple agents where agents must interact closely and influence each other to compete. We model interactions among agents through a game-theoretical framework and propose an efficient algorithm for tractably solving the resulting game in real time. More specifically, we capture interactions among multiple agents through a constrained dynamic game. We show that the resulting dynamic game is an instance of a simple-to-analyze class of games. Namely, we show that our racing game is an instance of a constrained dynamic potential game. An important and appealing property of dynamic potential games is that a generalized Nash equilibrium of the underlying game can be computed by solving a single constrained optimal control problem instead of multiple coupled constrained optimal control problems. Leveraging this property, we show that the problem of autonomous racing is greatly simplified and develop RAPID (autonomous multi-agent Racing using constrained Potential Dynamic games), a racing algorithm that can be solved tractably in real-time. Through simulation studies, we demonstrate that our algorithm outperforms the state-of-the-art approach. We further show the real-time capabilities of our algorithm in hardware experiments.

I. INTRODUCTION

Autonomous racing is gaining popularity because of its broad applicability in various competitive and non-cooperative motion planning scenarios. Multi-agent autonomous racing is a highly challenging motion planning task, requiring multiple nonlinear agents to plan their motions in real time while operating at their limits. Furthermore, they must account for other agents with conflicting objectives and ensure safety constraints, such as avoiding collisions and staying on the track. This results in a set of coupled motion planning problems that are highly nonlinear and complex. Such complexities require efficient motion planning algorithms for real-time capabilities and to ensure safety while generating competitive trajectories.

One of the earlier approaches to autonomous racing in the context of RC cars was [1], where the authors employed an optimization-based model predictive controller to maximize progress on the track, subject to safety requirements. However, due to the reactive nature of such approaches, they do not generate competitive trajectories. Learning-based

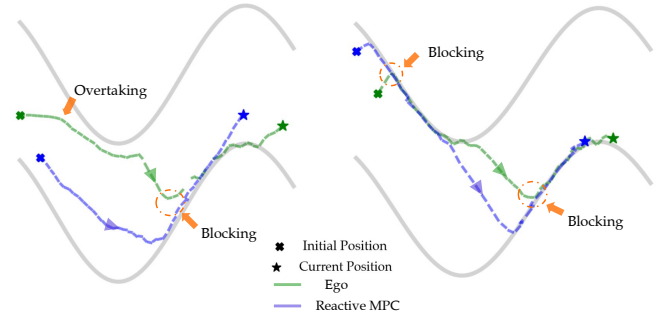


Fig. 1: Visualizations of two of the hardware experiments using two Crazyflies. As it can be seen in the left figure, even when the ego drone starts behind, it overtakes the opponent and ultimately blocks the opponent to avoid being overtaken. On the right, it can be seen that when the ego drone starts ahead of the opponent, it generates blocking behavior twice to avoid being overtaken. No collisions occurred in the hardware experiments and the ego-drone won the race in both instances.

approaches for autonomous racing were studied in [2], [3], [4], [5]. However, these methods mainly focus on the path-planning aspect of racing instead of interactions among the agents and may fail to generate competitive behaviors such as blocking and overtaking.

Due to the interactive nature of racing, where each agent has to account for other agents' decisions, interactions among agents can naturally be captured in a game-theoretic framework. Game-theoretic planning has been extensively used in non-cooperative motion planning for multiple agents [6], [7], [8], [9], [10], [11]. Due to its success in motion planning, game theory has also recently been applied in autonomous racing. Non-cooperative game theoretic planners for autonomous racing involving two agents were studied in [12], [13]. However, these works apply to only two-agent settings. Going beyond two agents, it is often computationally challenging to account for interactions among multiple agents due to the nonlinearities inherent in racing.

In [14], a game-theoretic planner was proposed for racing among multiple agents using sensitivity-based analysis. However, the proposed algorithm does not scale well with the number of agents. A game-theoretic planner with data-driven identification of vehicle models was developed for head-to-head autonomous racing in [15]. This work uses Stackelberg strategies which are not generally suitable for multi-agent racing as they normally involve a leader-follower structure.

In this work, we present RAPID, an autonomous multi-

This work is supported by the National Science Foundation, under grants ECCS-2145134 CAREER Award, CNS-2218759, and CCF-2211542.

¹Yixuan Jia is with the Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign, 306 N Wright St, Urbana, IL 61801, USA yixuanj3@illinois.edu

²Maulik Bhatt and Negar Mehr are with the Department of Aerospace Engineering, University of Illinois Urbana-Champaign, 104 S Wright St, Urbana, IL 61801, USA mcbhatt2@illinois.edu, negar@illinois.edu

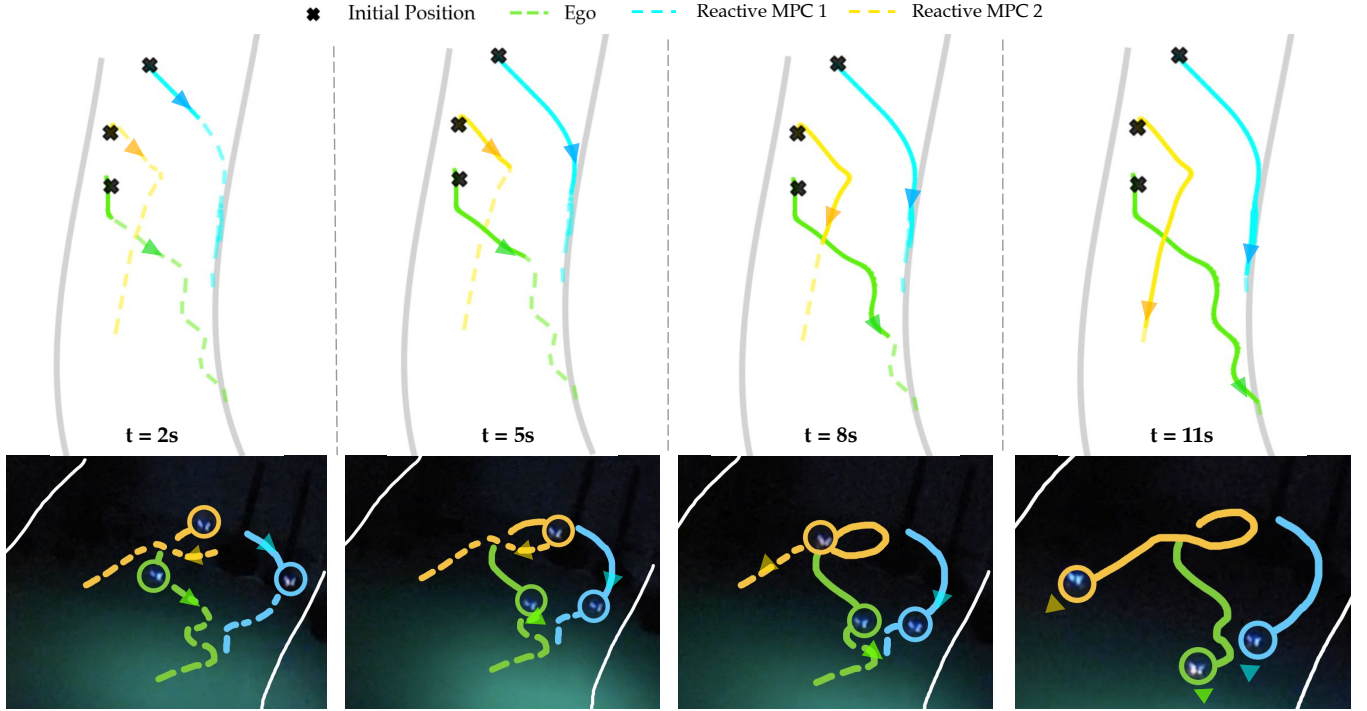


Fig. 2: Experiment snapshots of the race among three quadrotors. Green color is used to represent the ego quadrotor which uses our trajectory planner RAPID and exhibits an overtaking maneuver around the two quadrotors that use reactive MPC. The top row consists of top views that are plotted with the actual hardware experiment data. The second row shows the snapshots of the actual hardware experiment. For visualization purposes, the approximated boundaries of the track are marked by the white lines on the second row. We see that the ego (green) is trying to block the agent marked with the blue color. Videos can be found at: <https://youtu.be/85PYCj6vUd4>.

agent racing algorithm that uses constrained dynamic potential games. We pose the racing problem as a non-cooperative constrained general-sum dynamic game and seek the Nash equilibria of the game. We show that although, in general, finding generalized Nash equilibria — Nash equilibria of constrained dynamic games — is very challenging, the equilibria of our racing game can be found tractably and efficiently. Our key insight is that our racing game is an instance of a *dynamic potential game* for which equilibria always exist and can be found by solving a single constrained optimal control problem. The advantage of formulating the problem as a dynamic potential game is that it is generally more efficient to solve the underlying multivariate optimal control problem than to solve a set of coupled optimal control problems [16], [17]. We leverage this property and develop a tractable planning algorithm for racing among multiple agents. We compare our method with the state-of-the-art and demonstrate that our method beats the existing work in terms of both the computation time and the quality of the generated trajectories. We further demonstrate the real-time capabilities of our method in a hardware racing experiment involving three quadcopters.

II. PROBLEM FORMULATION

A. Notations

We consider the problem of autonomous racing in discrete time. Let $T \in \mathbb{N}$ denote the number of time steps in the problem, where \mathbb{N} denotes the set of natural numbers. For

any natural number n , let $[n] := \{1, \dots, n\}$ be the set of all natural numbers smaller than or equal to n . Let $[N]$, where $N \in \mathbb{N}$, denote the set of agents' indices, where each index corresponds to an agent. For each agent $i \in [N]$, agent i 's state is denoted as $x^i \in \mathcal{X}^i \subset \mathbb{R}^{n_i}$ where \mathcal{X}^i is the state space for agent i and $n_i \in \mathbb{N}$ is the dimension of state space of agent i . Similarly, the control action for each agent $i \in [N]$ is denoted as $u^i \in \mathcal{U}^i \subset \mathbb{R}^{m_i}$, where \mathcal{U}^i is the action space for agent i and $m_i \in \mathbb{N}$ is the dimension of the control space of agent i .

Let $\mathcal{X} := \mathcal{X}^1 \times \dots \times \mathcal{X}^N$ denote the entire state space of the system and let $\mathcal{U} := \mathcal{U}^1 \times \dots \times \mathcal{U}^N$ be the action space of all agents participating in the race. We denote the state of all agents at time step t as $x_t := (x_t^1, \dots, x_t^N)^\top \in \mathcal{X} \subset \mathbb{R}^n$, and the control action of all agents at time t can be denoted as $u_t := (u_t^1, \dots, u_t^N)^\top \in \mathcal{U} \subset \mathbb{R}^m$, where $n := \sum_{i \in [N]} n_i$ and $m := \sum_{i \in [N]} m_i$. Furthermore, we use $u_t^{-i} := (u_t^1, \dots, u_t^{i-1}, u_t^{i+1}, \dots, u_t^N)^\top \in \mathcal{U}^{-i}$, where $\mathcal{U}^{-i} := \prod_{j \neq i} \mathcal{U}^j$, to denote the control actions of all agents at time step t except for agent i . With a slight abuse of notations, we can write $u_t = (u_t^i, u_t^{-i})^\top$.

We assume that for each agent $i \in [N]$, its state at every time step x_t^i evolves according to the discrete dynamics $x_{t+1}^i = f^i(x_t^i, u_t^i, t)$, where $f^i : \mathcal{X}^i \times \mathcal{U}^i \times [N] \rightarrow \mathbb{R}^{n_i}$, and x_t^i, u_t^i denote the state and action of agent i at time step t . We define the overall system dynamics as $f := (f^1, \dots, f^N)$. Then, the evolution of the joint state of the system is

described by:

$$x_{t+1} = f(x_t, u_t, t). \quad (1)$$

While racing against each other, each agent has to satisfy some constraints, such as staying on the race track and avoiding collisions with obstacles and other agents. We denote the constraints of the race track through the function,

$$h(x_t^i) \leq 0, \forall i \in [N], \forall t \in \{0, 1, \dots, T-1\}. \quad (2)$$

We consider collision avoidance constraints which are defined as follows for every pair of agents:

$$d(x_t^i, x_t^j) > d_{min}, \quad \forall i < j, i, j \in [N], \quad (3)$$

where $d(x_t^i, x_t^j)$ is the Euclidean distance between agents i and j at time step t and d_{min} is the minimum distance between before which two agents are considered collided. We also consider upper and lower bounds on the control inputs of each agent. For each agent $i \in [N]$, we have

$$u_{min}^i \leq u_t^i \leq u_{max}^i, \forall t \in \{0, 1, \dots, T-1\} \quad (4)$$

where u_{min}^i and u_{max}^i are the vectors of minimum and maximum control inputs for agent i , respectively. Let $g^i(x, u, t) \leq 0$ denote the concatenated vector of constraints in (2), (3) and (4) for agent i , where $g^i : \mathcal{X} \times \mathcal{U} \times [N] \rightarrow \mathbb{R}^{c_i}$ is a vector-valued function and $c_i \in \mathbb{N}$ is the number of constraints of agent i . Note that the inequality is to be understood element-wise. It should also be noted that the constraint function of agent i , in general, depends on the states and control actions of *all agents*. This results in a coupling between agents' decisions since one agent cannot satisfy their constraint function in isolation. Hence, agents must account for such couplings while choosing their control actions. We can collect constraints for all agents and define $g := (g^1, \dots, g^N)^\top : \mathcal{X} \times \mathcal{U} \times [N] \rightarrow \mathbb{R}^c$, where $c := \sum_{i \in [N]} c_i$. Then the combined constraint function for the entire system is:

$$g(x_t, u_t, t) \leq 0. \quad (5)$$

We define \mathcal{C}_t to be the set of all feasible states and actions at time t , i.e., the constrained subset of $\mathcal{X} \times \mathcal{U}$ which satisfies all the constraints (5) at time step t . We can define $\mathcal{C}_0 = \mathcal{X} \times \mathcal{U} \cap \{(x_0, u_0) : g(x_0, u_0) \leq 0\}$ and $\mathcal{C}_t = \{\{\mathcal{X} \cap \{x_t : x_t = f(x_{t-1}, u_{t-1}, t-1)\}\} \times \mathcal{U} \cap \{(x_t, u_t) : g(x_t, u_t) \leq 0\}\}$ for $t \in \{1, \dots, T-1\}$. At the terminal time step T , no action is being taken, and; hence, the constraint function will only be a function of states ($g(x_T, (\cdot), T)$) and the corresponding constraint set will be $\mathcal{C}_T = \mathcal{X} \cap \{x_T : g(x_T, (\cdot), T) \leq 0\}$.

B. Agents' Strategies and Objectives

Each agent wants to choose actions sequentially to maximize their chances of winning the race. The policy through which each agent chooses its actions is called its strategy. A strategy can have various forms. In the present scenario, we consider open-loop strategies for the agents, which means that the strategy of each agent depends only on the initial

state of the system and time.¹ Let $\mathcal{T} = \{0, 1, \dots, T\}$. For any agent $i \in [N]$, we define the open-loop strategy of agent i , $\gamma^i : \mathcal{X} \times \mathcal{T} \rightarrow \mathcal{U}^i$ as follows:

$$\gamma^i(x_0, t) := u_t^i.$$

In other words, at each time step t , given the initial state of the system x_0 , $\gamma^i(x_0, t)$ would be the control action chosen by agent i . We use Γ^i to denote the space of all possible strategies for agent i . We define $\gamma := (\gamma^1, \dots, \gamma^N)^\top \in \Gamma$, where $\Gamma := \Gamma^1 \times \dots \times \Gamma^N$ is the combined strategy space of the system. Let $\Gamma^{-i} := \prod_{j \neq i} \Gamma^j$ denote the strategy space of all agents except agent i . Same as before, we use $\gamma^{-i} := (\gamma^1, \dots, \gamma^{i-1}, \gamma^{i+1}, \dots, \gamma^N)^\top \in \Gamma^{-i}$ to denote the strategy and strategy space of all agents except the agent i . It should be noted that open-loop strategies provide equivalence between strategy and control actions at all time instants ($\gamma \equiv \{u_t\}_{\{1, \dots, T-1\}}$), which in turn determines the states of the whole system at all time steps given the initial state.

When racing, each agent has their own objective of finishing the race before the opponents while satisfying some constraints such as being on the race track and avoiding collisions with other agents. As discussed before, let these constraints be denoted by $g(x_t, u_t, t) \leq 0$. The agents' objectives should incentivize behaviors such as moving as fast as possible along the race track and blocking the other agents if they are trying to overtake. Let the objective function of each agent be denoted by a function $J^i : \mathcal{X} \times \Gamma \rightarrow \mathbb{R}$ which is a function of the initial state of the game and the strategies of all the agents. Taking inspiration from [14], for each agent i , we choose J^i to be of the form

$$J^i(x_0, \gamma) = -r(x_T^i) + \alpha \sum_{t=0}^{T-1} \sum_{\substack{j \in [N] \\ j \neq i}} d(x_t^i, x_t^j)^2, \quad (6)$$

where $r(\cdot)$ is the path covered along the race track, and α is a hyperparameter that determines the amount of weight given to generating blocking behavior with respect to moving along the track as fast as possible. We choose the objective functions to be of the form in (6) with the following motivations:

- For a fixed planning horizon T , each agent wants to maximize the path covered along the race track, which corresponds to minimizing $-r(x_T^i)$.
- We take motivation from the sensitivity terms introduced in [14] to incentivize competitive behaviors. Each agent wants to prevent other agents from overtaking themselves and remain in close proximity to their opponents who are ahead of them. Our method incentivizes blocking behaviors if the agent using our method is

¹We acknowledge that considering open-loop strategies may generate different trajectories in contrast to choosing closed-loop strategies which can be a function of the system's state at every time step. However, we implement the open-loop strategies in a receding horizon fashion to account for the new information at each time step and mimic feedback strategies. We show empirically that this is a reasonable approximation for practical purposes in motion planning

ahead of other agents. This is because, while the first term in (6) encourages the agent to progress along the track, the second term encourages being in proximity to other agents, which generates blocking behavior if the agent using our method is ahead. On the other hand, if the opponents are ahead, minimizing the distance to opponents is equivalent to catching up. Furthermore, in such cases, due to the first term, our method will incentivize the agent not only to catch up but also to overtake.

- Note that the α term in (6) indicates the aggressiveness of agents, as a larger value of α will incentivize agents to be in proximity with other agents. It should also be noted that, due to the collision constraints, the distance between each pair of agents has a lower bound, which guarantees the safety of the generated trajectories.

When racing, each agent wants to win the race, which translates to minimizing its objective. However, each agent cannot minimize its objective and satisfy its constraints in isolation, as the value of its objective depends on the actions and states of other agents as well. Therefore, none of the agents can independently minimize their objectives as the objectives and constraints are inherently coupled. Consequently, to capture the dependence of agents upon one another, they must seek equilibria of the dynamic game underlying their interactions.

We denote such a non-cooperative general-sum constrained dynamic game in a compact form as $\mathcal{G} := ([N], \{\Gamma_i\}_{i \in [N]}, \{J_i\}_{i \in [N]}, \{\mathcal{C}_k\}_{k \in \{0, \dots, T\}}, f, x_0)$. We model the outcome of this competitive racing scenario by open-loop constrained Nash equilibria, also known as generalized Nash equilibria, of the constrained dynamic game \mathcal{G} .

Definition 1: A set of strategies γ^* is a generalized Nash equilibrium of the game $\mathcal{G} = ([N], \{\Gamma_i\}_{i \in [N]}, \{J_i\}_{i \in [N]}, \{\mathcal{C}_k\}_{k \in \{0, \dots, T\}}, f, x_0)$ if the following holds for each agent $i \in [N]$:

$$\begin{aligned} J^i(x_0, \gamma^*) &\leq J^i(x_0, \gamma^i, \gamma^{-i*}) \\ \forall \left(x_t, \gamma^i(x_0, t), \gamma^{-i*}(x_0, t) \right) &\in \mathcal{C}_t, t \in \{0, \dots, T-1\}, \\ \forall x_T &\in \mathcal{C}_T. \end{aligned} \quad (7)$$

This definition essentially means that no agent would want to deviate from their equilibrium strategy to any other feasible strategy as it would result in incurring a higher cost. It is important to notice that finding a solution to (7) will require solving a set of N -coupled constrained optimal control problems, which is challenging to solve tractably in practice. In the next section, we will discuss how we can efficiently find the equilibria of the game tractably in real time.

III. DYNAMIC POTENTIAL GAMES

Dynamic potential games are a class of games for which Nash equilibria can be found by solving a single optimal control problem instead of having to solve several coupled optimal control problems. If a game is a potential game, there exists a potential function P , and the Nash equilibria

of the game can be computed by minimizing this potential function. This largely simplifies the computation of Nash equilibria. This property of potential games has been recently leveraged in the context of multi-agent navigation and has been shown to simplify the problem of trajectory planning significantly [18], [19]. We formally define dynamic potential games as follows:

Definition 2: A non-cooperative constrained dynamic game $\mathcal{G} := ([N], \{\Gamma_i\}_{i \in [N]}, \{J_i\}_{i \in [N]}, \{\mathcal{C}_k\}_{k \in \{0, \dots, T\}}, f, x_0)$ is a constrained potential dynamic game if there exists a potential function $P : \mathcal{X} \times \Gamma \rightarrow \mathbb{R}$ such that for every agent $i \in [N]$, and every pair of strategies $\gamma^i \in \Gamma^i, \gamma^{-i} \in \Gamma^{-i}$, once we fix the set of strategies $\gamma^{-i} \in \Gamma^{-i}$, we have:

$$\begin{aligned} J^i(x_0, \gamma^i, \gamma^{-i}) - J^i(x_0, \gamma^i, \gamma^{-i}) \\ = P(x_0, \gamma^i, \gamma^{-i}) - P(x_0, \gamma^i, \gamma^{-i}). \end{aligned} \quad (8)$$

Definition 2 essentially states that a game \mathcal{G} is a constrained dynamic potential game if there exists a global potential function that captures the change in the cost of all agents when they change their strategy while other agents keep their strategies fixed.

Proposition 1: If a game is a constrained dynamic potential game, it can be shown that generalized open-loop Nash equilibria of the dynamic potential game \mathcal{G} can be computed by solving the following multivariate optimal control problem:

$$\begin{aligned} &\text{minimize}_{\gamma \in \Gamma} P(x_0, \gamma) \\ &\text{subject to } x_{k+1} = f(x_k, u_k, k), x_0 \text{ given} \\ &g(x_k, u_k, k) \leq 0. \end{aligned} \quad (9)$$

Proof: Proof can be found in Theorem-1 from [20]. ■

IV. RAPID: SCALABLE PLANNER FOR RACING

In this section, we show that our proposed multi-agent racing game is an instance of a dynamic constrained potential game. We will prove that under cost structures (6), the resulting constrained dynamic game will be a dynamic potential game whose generalized Nash equilibria can be found by solving one single constrained optimal control problem. The following theorem characterizes the cost structures under which the game \mathcal{G} will be a constrained dynamic potential game.

Theorem 1: A game \mathcal{G} where every agent optimizes (6) with constraints given in (2), (3) and (4), is a constrained dynamic potential game with the potential function

$$P(x_0, \gamma) = - \sum_{i=1}^N r(x_T^i) + \alpha \sum_{t=0}^{T-1} \sum_{i=1}^N \sum_{j=i+1}^N d(x_t^i, x_t^j)^2. \quad (10)$$

A generalized Nash equilibrium of \mathcal{G} can be computed by solving the following single optimal control problem

$$\begin{aligned} &\text{minimize}_{\gamma \in \Gamma} P(x_0, \gamma) \\ &\text{subject to } u_{\min} \leq u_t^i \leq u_{\max}, \quad \forall i \in [N], \forall t \in [T-1]; \\ &d(x_t^i, x_t^j) > d_{\min}, \quad \forall i \neq j \in [N], \forall t \in [T]; \\ &h(x_t^i) \leq 0, \quad \forall i \in [N], \forall t \in [T]. \end{aligned} \quad (11)$$

Proof: See Appendix A. ■

Note that Theorem 1 indicates that to find the equilibria of the games, it suffices to solve (11). Since (11) is a single constrained optimal control problem, one can use any existing constrained trajectory optimizer to solve it.

Our overall algorithm is described in Algorithm 1, where ego represents the agent that uses our algorithm.

Note that while the α term in Algorithm 1 stays the same during each planning horizon, its value can be different between planning horizons. Since the α value indicates the aggressiveness of agents, we introduce the idea of *adaptive aggressiveness scaling*, which scales α depending on the average distance of ego to other agents.

In line 1 of Algorithm 1, we sum over the squared distance of the ego agent to other agents. When the average squared distance of ego to other agents is above some threshold D , we use a smaller α value ($\alpha_{inactive}$ in line 2). Otherwise, a larger value (α_{active} in line 4) is used when ego is in proximity with other agents. When ego is far from others, the sense of being “aggressive” is not of relevance and thus the aggressiveness term is inactive. When ego is in proximity with other agents, having a larger α value indicates the aggressiveness term is active and ego will try to catch up with the agent ahead or block the agent behind, depending on the relative positions, which generates overtaking and blocking maneuvers.

Algorithm 1 RAPID with Adaptive Aggressiveness Scaling

Input: all agents’ current states $\{x^i\}_{i \in [N]}$, active distance threshold D , alpha inactive $\alpha_{inactive}$, alpha active α_{active} .

- 1: **if** $\sum_{i \in [N]} \|d(x^{ego}, x^i)\|^2 > (N - 1)D$ **then**
 - 2: $\alpha = \alpha_{inactive}$
 - 3: **else**
 - 4: $\alpha = \alpha_{active}$
 - 5: **end if**
 - 6: solve (11) with α .
-

V. SIMULATION RESULTS

In this section, we demonstrate the performance of our algorithm by considering two agents racing against each other.

We use a discrete variation of the Dubins car model for each agent. The state vector of each agent $i \in [N]$ at time step t is:

$$x_t^i = [p_{x,t}^i, p_{y,t}^i, v_t^i, \theta_t^i] \quad (13)$$

where $p_{x,t}^i, p_{y,t}^i$ represent the x, y coordinates of the agent in the world frame and v_t^i, θ_t^i represent its speed and orientation. Each agent’s dynamics is given by:

$$x_{t+1}^i = \begin{bmatrix} p_{x,t}^i + v_t^i \cos(\theta_t^i) \delta t \\ p_{y,t}^i + v_t^i \sin(\theta_t^i) \delta t \\ v_t^i + a_t^i \delta t \\ \theta_t^i + \omega_t^i \delta t \end{bmatrix}$$

where a_t^i, ω_t^i are the acceleration and angular velocity (control inputs) of agent i at time step t , and δt represents the

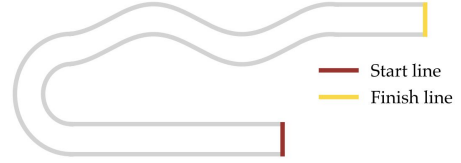


Fig. 3: The top view of the customized track. The brown line is the starting line, and the gold line is the finish line.

time step length. During the start of each planning horizon, all agents have access to the ground truth of their own current states as well as the current states of other agents. For simplicity, we call the agent that uses our algorithm the *ego*. We compare our algorithm with the racing algorithm developed in [14], which will be called the *baseline*. The baseline algorithm used iterative best responses and was shown to be able to generate competitive behaviors.

We solve both the ego’s optimization problem and the baseline’s optimization problem with do-mpc [21], which models the problems symbolically with CasADi [22] and solves them with IPOPT. [23]. We choose the planning horizon to be 5 time steps for each agent, where each time step corresponds to 0.1s.

The race track that we used is shown in Fig 3. The start and finish lines are marked with brown and gold, respectively. We run the experiments between the ego and the baseline for 50 random initial positions. The initial positions are randomly sampled within a fixed area around the starting line so that the initial distance between the two agents is within $[1.0, 1.5] m$. The agent with a leading starting position is given a maximum speed of 2.4 m/s, and the other agent is given a maximum speed of 2.5 m/s. Note that we chose this setup to ensure enough interactions between the two agents.

The results are demonstrated in Table I. Note that the solve time here represents the time that it takes the agent to solve its action for one time step. As shown in Table I, our algorithm has a significant advantage over the baseline in terms of both the solve time and the quality of the generated trajectories. Our algorithm is about 4 times faster than the baseline while winning the race more frequently. We find that the ego is able to push the baseline against the outer boundary of the track during the first corner, and the baseline would be stuck at the boundary until the ego passes. This blocking behavior is plotted in Fig 4.

	Average Solve Time (s)	Win
Ego	0.051 ± 0.010	36
Baseline	0.215 ± 0.084	14

TABLE I: Simulation Results for the Customized Track

To test the scalability of our approach, we ran our algorithm and the baseline algorithm against the reactive MPC algorithm, respectively, where the reactive MPC algorithm is described in Equation (15). We ran simulations for various numbers of agents, where one of them used our algorithm or the baseline algorithm, while the others used the reactive MPC algorithm. The results are recorded in Table II. As

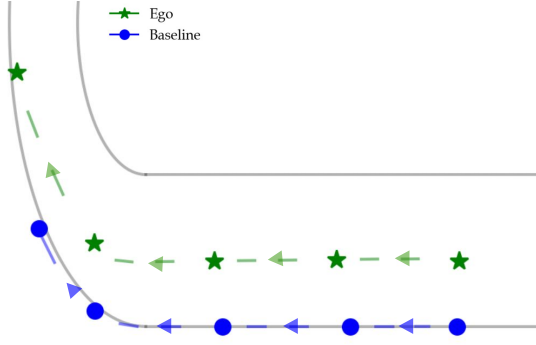


Fig. 4: The ego (plotted with green color) pushes the baseline (plotted with blue color) against the boundary.

indicated by the results, our algorithm is able to scale better than the baseline.

Number of Agents	2	5	10
RAPID	0.078 ± 0.046	0.229 ± 0.535	0.548 ± 0.400
Baseline	0.217 ± 0.061	0.637 ± 0.110	1.456 ± 0.187

TABLE II: Mean Solve Time (in seconds) Comparisons for Different Numbers of Agents.

VI. HARDWARE EXPERIMENTS

To further evaluate the real-time capabilities of our framework, we set up experiments in hardware using Crazyflie 2.0 quadcopters within the Robot Operating System (ROS) framework [24]. We used a Vicon motion capture system to collect the state information of the agents. We also used Crazyswarm [25] to send waypoint commands to the Crazyflies. We then used a low-level controller to follow the waypoints.

The state vector of each quadrotor $i \in [N]$, at each time step t , is:

$$x_t^i = [p_{x,t}^i, p_{y,t}^i, p_{z,t}^i, \phi_t^i, \theta_t^i, \psi_t^i], \quad (14)$$

which consists of its position ($[p_{x,t}^i, p_{y,t}^i, p_{z,t}^i]$) and orientation ($[\phi_t^i, \theta_t^i, \psi_t^i]$). We model the quadrotor dynamics as given by $\dot{x}_t^i = u_t^i$, where $u_t^i = [u_{1,t}^i, u_{2,t}^i, u_{3,t}^i, u_{4,t}^i, u_{5,t}^i, u_{6,t}^i]$ is the control input. The first three entries of u_k^i correspond to the linear velocities and the last three entries correspond to the angular velocities. We use this simplified model since we can send waypoints directly as commands to the quadcopters through the Crazyswarm framework [25], and an in-place low-level controller will be used to track the waypoints.

We also implemented a model predictive controller (MPC) for comparison purposes. This MPC planner is in fact a reactive planner that treats other agents as obstacles that need to be avoided. We call this planner *reactive MPC*. When running reactive MPC, each agent aims to maximize its progress along the track while satisfying input constraints, collision constraints, and track constraints. The reactive MPC predicts the position of other agents by assuming other agents

Overtakings	Defendings
10/10	8/10

TABLE III: Results for Two-agent Hardware Experiments. Our method was able to overtake 10 times out of the 10 trials when it started behind. And it was able to defend its leading position 8 times out of the 10 trials when it started ahead.

move with constant velocity during each planning horizon, where the constant velocity comes from finite difference estimation. For each agent $l \in [N]$, the reactive MPC solves the following optimization problem:

$$\begin{aligned} & \underset{\gamma^l \in \Gamma^l}{\text{minimize}} && -r(x_T^l, T) \\ & \text{subject to} && u_{\min} \leq \|u_t^l\| \leq u_{\max}, \forall i \in [N], \forall t \in [T]; \\ & && d(x_t^l, x_t^j) > d_{\min}, \forall j \in [N] \setminus \{l\}, \forall t \in [T]; \\ & && h(x_t^l) \leq 0, \quad \forall t \in [T] \end{aligned} \quad (15)$$

A. Two-agent Race

We begin by considering a race between two agents, where one agent uses our algorithm while the other uses the reactive MPC algorithm. The track is shown in Fig 1. The initial positions are chosen with some randomness. The quadrotor that starts ahead is given a maximum speed of 1.5 m/s, and the one that starts behind is given a maximum speed of 1.8 m/s.

The results are displayed in Table III, where *Overtakings* records the number of times where the ego agents started behind but reached the finish line first and *Defendings* records the number of times where it started ahead and reached the finish line first. As shown by the results, the ego agent significantly outperforms the reactive MPC by winning most of the races. Fig 1 shows both agents' trajectories in two of the hardware experiments. As we can see, the ego quadrotor is able to overtake the opponent (the quadrotor that uses reactive MPC) when it starts behind and then block the opponent when the opponent has a possibility to overtake. In both cases, the ego is able to secure a leading position without causing any collision. This also demonstrates that our algorithm is able to exploit the trade-off between making progress and blocking opponents without causing constraint violations.

B. Three-agent Race

We also test our algorithm with a more complicated setting where there are three agents and the track becomes narrower. One of the agents uses our algorithm, and the other two use the reactive MPC algorithm.

The track is similar to the one used in the two-agent race but is made to be narrower in order to test our algorithm in a more competitive scenario. The initial conditions are selected such that the quadrotors engage in interactions and have enough space for maneuvering. The speed limit for the leading quadrotor was set to be 1.5m/s, the middle quadrotor 1.65m/s, and the quadrotor behind had a speed limit of 1.8m/s.

We performed the experiments 15 times where the ego quadrotor starts first 5 times, starts second 5 times, and starts third 5 times. The results are recorded in table IV. As we can see, the ego was able to win most of the races, even when it starts behind or starts ahead but given a lower speed limit. We observe that our algorithm is able to take advantage of the collision avoidance constraints of other agents and generate overtaking or blocking, and it is doing so without causing any collision. Fig 2 shows an instance of such a blocking maneuver during a three-agent race. As we can see, the ego (represented by green color) intentionally moves toward agent 1 (represented by blue color) to block it. As a result, the ego is able to make more progress than agent 1 despite being given a lower speed limit. Note that the experimental snapshots are not taken from a top view, so the relative position of agents may look different from the top view plots. For more details on the hardware experiments of three drones, we recommend that interested readers check out the videos.

	Started 1st	Started 2nd	Started 3rd
Ego Finished 1st	3/5	4/5	4/5

TABLE IV: Results for Three-agents Hardware Experiments

VII. CONCLUSION AND FUTURE WORK

In this work, we presented RAPID, an efficient algorithm for trajectory planning in autonomous racing. Our method exploits the special cost structure under which the race can be formulated as a potential game, which enables us to obtain a generalized Nash equilibrium of the game by solving a single constrained optimal control problem. We demonstrated the performance of the algorithm through both simulation studies and hardware experiments. We showed that RAPID could generate delicate interactive maneuvers such as overtaking and blocking while avoiding collisions with other agents. Currently, some parameters such as the active distance threshold and $\alpha_{inactive}, \alpha_{active}$ in Algorithm-1 used in RAPID need to be determined empirically. In future work, we wish to provide theoretical results on the effects of these parameters.

REFERENCES

- [1] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.
- [2] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 5115–5120.
- [3] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, *et al.*, "Deepracer: Autonomous racing platform for experimentation with sim2real reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2746–2754.
- [4] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1205–1212.
- [5] J. Herman, J. Francis, S. Ganju, B. Chen, A. Koul, A. Gupta, A. Skabelkin, I. Zhukov, M. Kumskey, and E. Nyberg, "Learn-to-race: A multimodal control environment for autonomous racing," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9793–9802.
- [6] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for autonomous cars that leverage effects on human actions," in *Robotics: Science and Systems*, vol. 2. Ann Arbor, MI, USA, 2016, pp. 1–9.
- [7] A. Dreves and M. Gerdt, "A generalized nash equilibrium approach for optimal control problems of autonomous cars," *Optimal Control Applications and Methods*, vol. 39, no. 1, pp. 326–342, 2018.
- [8] D. Fridovich-Keil, E. Ratner, L. Peters, A. D. Dragan, and C. J. Tomlin, "Efficient iterative linear-quadratic approximations for non-linear multi-player general-sum differential games," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1475–1481.
- [9] A. Zanardi, E. Mion, M. Bruschetta, S. Bolognani, A. Censi, and E. Frazzoli, "Urban driving games with lexicographic preferences and socially efficient nash equilibria," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4978–4985, 2021.
- [10] M. Wang, N. Mehr, A. Gaidon, and M. Schwager, "Game-theoretic planning for risk-aware interactive agents," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6998–7005.
- [11] N. Mehr, M. Wang, M. Bhatt, and M. Schwager, "Maximum-entropy multi-agent dynamic games: Forward and inverse solutions," *IEEE Transactions on Robotics*, 2023.
- [12] A. Liniger and J. Lygeros, "A noncooperative game approach to autonomous racing," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 3, pp. 884–897, 2020.
- [13] R. Spica, E. Cristofalo, Z. Wang, E. Montijano, and M. Schwager, "A real-time game theoretic planner for autonomous two-player drone racing," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1389–1403, 2020.
- [14] M. Wang, Z. Wang, J. Talbot, J. C. Gerdes, and M. Schwager, "Game-theoretic planning for self-driving cars in multivehicle competitive scenarios," *IEEE Transactions on Robotics*, vol. 37, no. 4, pp. 1313–1325, 2021.
- [15] C. Jung, S. Lee, H. Seong, A. Finazzi, and D. H. Shim, "Game-theoretic model predictive control with data-driven identification of vehicle model for head-to-head autonomous racing," *arXiv preprint arXiv:2106.04094*, 2021.
- [16] S. Zazo, S. Valcarcel Macua, M. Sánchez-Fernández, and J. Zazo, "Dynamic potential games with constraints: Fundamentals and applications in communications," *IEEE Transactions on Signal Processing*, vol. 64, no. 14, pp. 3806–3821, 2016.
- [17] D. González-Sánchez and O. Hernández-Lerma, *Discrete-time stochastic control and dynamic potential games: the Euler–Equation approach*. Springer Science & Business Media, 2013.
- [18] T. Kavuncu, A. Yaraneri, and N. Mehr, "Potential ilqr: A potential-minimizing controller for planning multi-agent interactive trajectories," *arXiv preprint arXiv:2107.04926*, 2021.
- [19] Z. Williams, J. Chen, and N. Mehr, "Distributed potential ilqr: Scalable game-theoretic trajectory planning for multi-agent interactions," *arXiv preprint arXiv:2303.04842*, 2023.
- [20] M. Bhatt, A. Yaraneri, and N. Mehr, "Efficient constrained multi-agent interactive planning using constrained dynamic potential games," *arXiv preprint arXiv:2206.08963*, 2022.
- [21] S. Lucia, A. Tăulea-Codrean, C. Schoppmeyer, and S. Engell, "Rapid development of modular and sustainable nonlinear model predictive control solutions," *Control Engineering Practice*, vol. 60, pp. 51–62, 2017.
- [22] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADI – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.
- [23] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.
- [24] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.
- [25] J. A. Preiss*, W. Hönig*, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3299–3304, software available at <https://github.com/USC-ACTLab/crazyswarm>. [Online]. Available: <https://doi.org/10.1109/ICRA.2017.7989376>

A. Proof of Theorem 1

Proof: Since agents' dynamics are separable, the state and control input of each agent can be written as a function of their strategies. We will explicitly denote this dependence in this proof. Consider two sets of open-loop strategies $\gamma := (\gamma^{i^\top}, \gamma^{-i^\top})^\top$ and $\gamma' := (\nu^{i^\top}, \gamma^{-i^\top})^\top$. Let the states and control inputs generated by the set of strategies γ be $\{x_t\}_{t \in \{0,1,\dots,T\}}$ and $\{u_t\}_{t \in \{0,1,\dots,T-1\}}$ respectively. Likewise, let the states and control inputs generated by the set of open-loop strategies γ' be $\{x'_t\}_{t \in \{0,1,\dots,T\}}$ and $\{u'_t\}_{t \in \{0,1,\dots,T-1\}}$ respectively. Note that since between γ and γ' , only the strategy of agent i is being changed, all the other agents' states and actions will be the same when using strategies γ and γ' . Therefore, we have the following relation:

$$x'^j_t = x^j_t, u'^j_t = u^j_t \quad \forall j \neq i, j \in [N]. \quad (16)$$

Now, we can compute the difference of the cost for agent i as

$$\begin{aligned} J^i(x_0, \gamma) - J^i(x_0, \gamma') &= -r(x_T^i) + r(x_T'^i) \\ &+ \alpha \sum_{t=0}^{T-1} \sum_{\substack{j \in [N] \\ j \neq i}} \left(d(x_t^i, x_t^j)^2 - d(x_t^i, x_t'^j)^2 \right). \end{aligned} \quad (17)$$

Using results from (16), since we have that $x'^j_t = x^j_t, u'^j_t = u^j_t$, for all $j \neq i$, we obtain

$$\begin{aligned} J^i(x_0, \gamma) - J^i(x_0, \gamma') &= -r(x_T^i) + r(x_T'^i) \\ &+ \alpha \sum_{t=0}^{T-1} \sum_{\substack{j \in [N] \\ j \neq i}} \left(d(x_t^i, x_t^j)^2 - d(x_t^i, x_t^j)^2 \right). \end{aligned} \quad (18)$$

Now, let's consider the change in the potential function (10) when the strategy of agent i changes from γ^i to ν^i :

$$\begin{aligned} P(x_0, \gamma) - P(x_0, \gamma') &= - \sum_{k=1}^N \left(r(x_T^k) - r(x_T'^k) \right) \\ &+ \alpha \sum_{t=0}^{T-1} \sum_{k=1}^N \sum_{l=k+1}^N \left(d(x_t^k, x_t^l)^2 - d(x_t'^k, x_t'^l)^2 \right). \end{aligned} \quad (19)$$

Using (16), we have:

$$\sum_{k=1}^N \left(r(x_T^k) - r(x_T'^k) \right) = r(x_T^i) - r(x_T'^i), \quad (20)$$

and similarly, using (16),

$$\begin{aligned} &\sum_{k=1}^N \sum_{l=k+1}^N \left(d(x_t^k, x_t^l)^2 - d(x_t'^k, x_t'^l)^2 \right) \\ &= \sum_{k=1}^i \sum_{l=k+1}^N \left(d(x_t^k, x_t^l)^2 - d(x_t'^k, x_t'^l)^2 \right) \\ &+ \sum_{k=i+1}^N \sum_{l=k+1}^N \left(d(x_t^k, x_t^l)^2 - d(x_t'^k, x_t'^l)^2 \right) \end{aligned} \quad (21)$$

As we can see, second term in (21) is zero. In the first term, for each value of $0 < k < i$, all the terms corresponding to all the values of l except $l = i$ will be zero due to (16). Therefore,

$$\begin{aligned} &\sum_{k=1}^N \sum_{l=k+1}^N \left(d(x_t^k, x_t^l)^2 - d(x_t'^k, x_t'^l)^2 \right) \\ &= \sum_{k=1}^{i-1} \left(d(x_t^k, x_t^i)^2 - d(x_t'^k, x_t'^i)^2 \right) \\ &+ \sum_{l=i+1}^N \left(d(x_t^i, x_t^l)^2 - d(x_t'^i, x_t'^l)^2 \right) \\ &= \sum_{\substack{j \in [N] \\ j \neq i}} \left(d(x_t^i, x_t^j)^2 - d(x_t'^i, x_t'^j)^2 \right) \end{aligned} \quad (22)$$

Putting together (20) and (22) in (19), we obtain

$$\begin{aligned} P(x_0, \gamma) - P(x_0, \gamma') &= -r(x_T^i) + r(x_T'^i) \\ &+ \alpha \sum_{t=0}^{T-1} \sum_{\substack{j \in [N] \\ j \neq i}} \left(d(x_t^i, x_t^j)^2 - d(x_t'^i, x_t'^j)^2 \right). \\ \Rightarrow P(x_0, \gamma) - P(x_0, \gamma') &= J^i(x_0, \gamma) - J^i(x_0, \gamma'), \forall i \in [N]. \end{aligned} \quad (23)$$

Therefore, P is indeed the potential function for the underlying dynamic game using Definition 2. Therefore using results from Theorem 1 in [20], we can compute generalized open-loop Nash equilibria of the original game by solving the multivariate optimal control problem given in (11). ■