A Real-Time Game Theoretic Planner for Autonomous Two-Player Drone Racing

Riccardo Spica, *Member, IEEE*, Eric Cristofalo, *Graduate Student Member, IEEE*, Zijian Wang, Eduardo Montijano, *Member, IEEE*, and Mac Schwager, *Member, IEEE*

***Abstract*—In this article, we propose an online 3-D planning algorithm for a drone to race competitively against a single adversary drone. The algorithm computes an approximation of the Nash equilibrium in the joint space of trajectories of the two drones at each time step, and proceeds in a receding horizon fashion. The algorithm uses a novel sensitivity term, within an iterative best response computational scheme, to approximate the amount by which the adversary will yield to the ego drone to avoid a collision. This leads to racing trajectories that are more competitive than without the sensitivity term. We prove that the fixed point of this sensitivity enhanced iterative best response satisfies the first-order optimality conditions of a Nash equilibrium. We present results of a simulation study of races with 2-D and 3-D race courses, showing that our game theoretic planner significantly outperforms a model predictive control (MPC) racing algorithm. We also present results of multiple drone racing experiments on a 3-D track in which drones sense each others' relative position with onboard vision. The proposed game theoretic planner again outperforms the MPC opponent in these experiments where drones reach speeds up to $1.25\,\mathrm{m/s}$.**

***Index Terms*—Aerial robotics, drone racing, game theory, motion and path planning, path planning for multiple mobile robot systems, vision-based pose estimation.**

I. Introduction

In THIS article, we consider two-player autonomous drone racing as a practical scenario to investigate strategies for robots to engage in noncooperative tasks with other agents. When a robot is interacting with other agents, the challenge is not only to avoid collisions, but to do so while optimizing its own objective, and while accounting for the feedback between the robot's plans and those of the other agents. These issues are inherently game theoretic, in that one must consider the objectives and constraints of other agents, while attempting to optimize one's own objective subject to one's own constraints.

Specifically, we propose a real-time receding horizon planning algorithm for an autonomous drone (called the "ego drone") to plan a racing trajectory in competition with another drone. The algorithm attempts to optimize the ego drone's progress along a race course subject to 1) not colliding with its opponent, and 2) staying on the track. In order to do so, the ego drone must also plan a racing trajectory for its opponent, anticipating that the opponent is itself trying to win the race. The goal is then to find a Nash equilibrium in these two planned trajectories, that is, a point in joint trajectory space in which neither the ego drone nor the opponent can improve upon its trajectory by itself.

Our key contribution is an algorithm that we call sensitivity enhanced iterative best response (SE-IBR), which uses a sensitivity term within a sequence of iterated optimization problems. The sensitivity term seeks to bias the resulting Nash equilibrium to be more favorable to the ego drone than to the opponent, by predicting at each iteration the amount to which the opponent will yield to the ego drone due to its collision avoidance constraint. We prove that the fixed point condition for this iterative algorithm is equivalent to the first-order conditions for a Nash equilibrium in the space of joint trajectories. We verify the performance of this algorithm in simulation studies in both 2-D and 3-D race courses, as well as in hardware experiments with drones in a 3-D race course. The simulations and experiments show that the ego drone using our SE-IBR algorithm significantly outperforms an opponent using a model predictive control (MPC) racing algorithm. Fig. 1 shows a frame from a typical drone race.

Furthermore, in the experiments, each drone has access to its own pose from a motion capture system. However, each drone has to sense the relative position of its opponent using only an onboard monocular camera. We propose a novel active vision system that uses the predicted trajectory of the opponent (from the game theoretic planner) as an input to a Kalman filter to track the opponent's relative 3-D position over time. This estimated opponent position is, in turn, used in the game theoretic planner to plan the racing trajectory. This closed-loop planner estimator helps to ensure that the opponent remains in the field of view of the ego drone's camera throughout the race.

Although we specifically consider drone racing, we propose this as a prototype scenario for noncooperative drone autonomy

Manuscript received October 15, 2019; accepted April 28, 2020. This work was supported in part by the Toyota Research Institute (TRI), in part by Office of Naval Research (ONR) under Grant N00014-16-1-2787, in part by the National Defense Science and Engineering Graduate Fellowship, in part by the ONR Global Grant N62909-19-1-2027, and in part by the Spanish Project PGC2018-098719-B-I00 (MCIU/AEI/FEDER, UE). This paper was recommended for publication by Associate Editor K. Mombaur and Editor P. Robuffo Giordano upon evaluation of the reviewers' comments. *(Corresponding author: Mac Schwager.)*

Riccardo Spica, Eric Cristofalo, Zijian Wang, and Mac Schwager are with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305 USA (e-mail: riccardo.spica@gmail.com; ecristof@stanford.edu; zjwang@alumni.stanford.edu; schwager@stanford.edu).

Eduardo Montijano is with the Instituto de Investigación en Ingeniería de Aragón, Universidad de Zaragoza, 50018 Zaragoza, Spain (e-mail: emonti@unizar.es).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2020.2994881

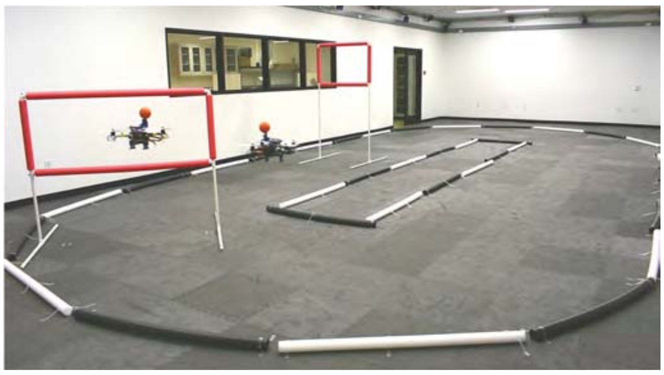

Fig. 1. Drone racing experiment. Two quadrotors follow the 3-D race course while competing against the opponent quadrotor. The relative position of the opponent is detected onboard using a camera.

more generally. Drone racing has already attracted significant interest from the research community as a benchmark for *single drone* autonomy. The first autonomous drone racing competition was held during the 2016 International Conference on Intelligent Robots and Systems (IROS) [1]. Most of the past research on autonomous racing (both for drones and other vehicles) has focused on a time trial style of racing: a single robot must complete a racing course in the shortest amount of time. This scenario poses a number of challenges in terms of dynamic modeling, onboard perception, localization and mapping, trajectory generation, and optimal control. Impressive results have been obtained in this context not only for autonomous unmanned aerial vehicles (UAVs) [2], but also for a variety of different platforms, such as cars [3]–[5] motorcycles [6], and even sailboats [7]. However, much less attention has been devoted to the multiplayer style of racing that we address in this article, sometimes called rotocross among drone racing enthusiasts. In addition to the aforementioned challenges, this kind of race also requires direct competition with other agents, incorporating strategic blocking, faking, and opportunistic passing while avoiding collisions. The algorithm we propose here exhibits all of these competitive behaviors.

A preliminary version of some of the results in this article appeared in the conference paper [8]. This journal version includes the following advancements beyond the conference version. 1) We test the algorithm in multiple hardware experiments, in which the drones sense one another's relative pose in real time using onboard monocular vision. Our perception pipeline uses the predicted trajectory of the opponent to yaw the camera into position to track the opponent, thereby introducing an element of active perception. 2) The article compares the SE-IBR to both a standard MPC, and a game theoretic iterative best response (IBR) planner (without our sensitivity enhancement) to verify that our SE-IBR significantly outperforms both traditional (MPC), as well as game theoretic (IBR), online planning algorithms. Beyond these main advancements we have also updated the text and notation throughout, and included several new figures to give more insight into the performance of the algorithm.

The rest of this article is organized as follows. In Section II, we review the existing literature. In Section III, we model the drone racing problem and introduce the associated sensing and control constraints. In Section IV, we formulate the problem as a Nash equilibrium search, describe our SE-IBR algorithm, and state the main mathematical result: the fixed point is equivalent to the first-order conditions for a Nash equilibrium. In Section V, we describe the onboard active-vision algorithm the drone use to estimate its opponent's positions. In Section VI, we report simulation and experimental results. Finally, Section VII concludes this article.

II. Related Work

In this section, we describe the related literature both for single-robot and multirobot motion planning. In particular, we focus our attention on approaches exploiting results from game theory using either a Stackelberg or a Nash information pattern, which are more closely related to our work.

A. Single-Robot Planning

A number of effective solutions for motion planning in presence of both *static* and *dynamic* obstacles have been proposed in the past. Some classical works use artificial potential fields [9], [10], geometric approaches [11] or sampling-based methods [12], [13]. More recently, also thanks to the availability of efficient numerical optimization schemes, a number of MPC and reinforcement learning approaches have also been proposed [4], [5], [14]. More specifically to our application, Jung *et al.* [2] ranked first in the IROS 2016 drone racing competition exploiting an optical flow sensor and a direct visual servoing control scheme.

Most of these works rely on simple "open-loop" models to predict the obstacle motion. In many situations, however, obstacles behave in a *reactive* way. A human, for example, will in turn actively avoid collision with the controlled robot and, thus, her/his motion will be strongly affected by that of the robot. This reactiveness creates a "loop closure" that, if not properly managed, can induce oscillatory effects sometimes referred to as *reciprocal dances* [15].

B. Cooperative Multirobot Control

Impressive results have been obtained by relying on *communication* to coordinate multiple robots in a navigation context or, more in general, to realize a common task [16], [17]. In other cases, communication is achieved more implicitly by an exchange of forces [18]. Some works remove the communication layer but rely on a common (or at least known) set of motion policies to achieve cooperation among multiple agents [11], [19]–[22]. In general, communication or coordination is not realistic in competitive scenarios such as drone racing or mixed human-robot scenarios like autonomous driving.

C. Game Theoretic Control Using a Stackelberg Information Pattern

While broadly used in economics and social science, game theory has not yet attracted, in our opinion, a sufficient interest from the robotics community, mostly due to the computational complexity typically associated with these methods.

Some interesting results have been obtained applying game theoretic concepts to robust H_∞ optimal control design (see [23] for a recent review on the topic). The disturbance acting on a system can be modeled as an antagonistic agent that explicitly aims at minimizing the system performance thus giving rise to a zero-sum differential game [24]. Similar models have also been employed to calculate the so-called *reachable set* of a system: the set of states from which there exists at least one control strategy that brings the system to a desired set in spite of adversarial disturbances [25]. Finding the reachable sets usually requires the integration of the so-called Hamilton–Jacobi–Isaacs partial differential equations, which are typically limited to offline solutions with a small number of state dimensions (e.g., less than 6). These offline solutions typically yield online policies that are fast, but not reactive to changing conditions. Some recent work has focused on scaling these offline methods to higher dimensional systems through suitable approximations [26]–[28].

Another approach to deal with games involving dynamic agents is the classical differential games approach introduced by Isaacs [29]. This approach typically uses geometrical analyses to find Stacklberg solutions for agents modeled in continuous time. For example, Walrand *et al.* [30] propose a harbor attack/defense game that bears some resemblance to the pass/blocking nature of our drone racing game. This work primarily provides geometric analysis of hand-designed strategies for a Stackelberg information pattern, and also considers Nash equilibria for mixed (i.e., stochastic) strategies. In contrast, our work considers an optimization-based approach that generates deterministic strategies online to suit the complex racing circumstances as they arise.

More recently, similar approaches have also been employed in the context of autonomous driving. In [31], for example, the interaction between an autonomous car and a human-driven one is modeled as a Stackelberg game: the human is assumed to know in advance what the autonomous car will do and to respond optimally according to an internal cost function. This results in a nested optimization problem that can be exploited to control the human motion [32] or to reconstruct the cost function driving his/her actions [31].

D. Game Theoretic Control Using a Nash Information Pattern

Giving the other players some information advantage can, in general, improve the robustness of the system. However, in many applications such as drone racing and autonomous driving, no agent would have any information advantage with respect to the others. For this reason, we believe that Stackelberg information models could result in overly conservative actions. A more realistic model is that of Nash equilibria that, instead, assume a fully symmetric information pattern.

Williams *et al.* [33] propose an information theoretic MPC approach for a stochastic racing game. The approach uses an IBR algorithm, although the objective of the game is for the two players to collaborate to be robust to the random effects of noise. In a racing scenario with scale ground vehicles moving at high speed, they show the algorithm is effective when both vehicles collaborate over a communication network to maintain a desired distance relative to one another despite various sources of noise. In contrast, we consider a deterministic set up in which the agents compete against one another to win a race, and they do not communicate over a communication network.

A recent paper [34] proposes a control algorithm for coordinating the motion of multiple cars through an intersection exploiting generalized Nash equilibria. The numerical resolution is in the order of several seconds which is close to real time but still not sufficient for the approach to be used for online control.

In the context of car racing, Liniger and Lygeros [35] investigate both Stackelberg and Nash equilibria. Computational performance close to real time, however, can only be obtained in a simplified scenario in which only one of the two players avoids collisions. In addition to this, the authors also discuss the importance of exploiting blocking behaviors. However, while in our work these behaviors naturally emerge from the use of sensitivity analysis, in [35] these are hardcoded in the cost function optimized by the players.

The main limiting factor for applying game theory more widely to robotic control problems seems to lie in the associated computational complexity. We believe, however, that game theory can still be used as an *inspiration* for guiding the design of effective and computationally efficient heuristics. The algorithm we present here represents a step in this direction.

III. Preliminaries

Consider two quadrotor UAVs competing against each other in a drone racing scenario. To simplify the computational demands of the solution, we assume a low-level trajectory following controller is in place, removing the need to include roll and pitch in the dynamics. Therefore, we seek to plan the trajectory using simplified holonomic dynamics given by

$$\begin{bmatrix} \dot{\boldsymbol{p}}_i \\ \dot{\psi}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_i & 0 \\ \boldsymbol{0}_3 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_i \\ \omega_i \end{bmatrix} \tag{1}$$

where $\boldsymbol{p}_i \in \mathbb{R}^3$ is the robot position in the world frame, $\boldsymbol{R}_i = \boldsymbol{R}_z(\psi_i) \in \mathbb{R}^{3\times3}$ represents the rotation matrix associated to the robot yaw $\psi_i \in \mathbb{S}^1$, and $\boldsymbol{v}_i \in \mathbb{R}^3$ and $\omega_i \in \mathbb{R}$ are the *body-frame* linear velocity and angular rates, which serve as the control inputs. Given (1), the robot state is $\boldsymbol{x}_i = (\boldsymbol{p}_i, \psi_i) \in \mathbb{R}^3 \times \mathbb{S}^1$ and it is assumed locally available, e.g., using onboard GPS and an inertial measurement unit (IMU).

Due to limitations of onboard actuators, the robots linear velocities are limited, i.e.,

$$\|\boldsymbol{v}_i\| = \|\dot{\boldsymbol{p}}_i\| \le \overline{v}_i \in \mathbb{R}^+.$$

The race track center line is defined by a twice continuously differentiable immersed closed curve $\boldsymbol{\tau}$ (see Fig. 2). For such a curve, there exists an arc-length parameterization

$$\boldsymbol{\tau} : [0, l_{\boldsymbol{\tau}}] \mapsto \mathbb{R}^3, \text{ with } \boldsymbol{\tau}(0) = \boldsymbol{\tau}(l_{\boldsymbol{\tau}})$$

where $l_{\boldsymbol{\tau}}$ is the total length of the track. Moreover, one can also define a local signed curvature κ and unit tangent, normal, and binormal vectors ($\boldsymbol{t}$, $\boldsymbol{n}$, $\boldsymbol{b}$, respectively) as follows:

$$\boldsymbol{t} = \boldsymbol{\tau}' \tag{2}$$

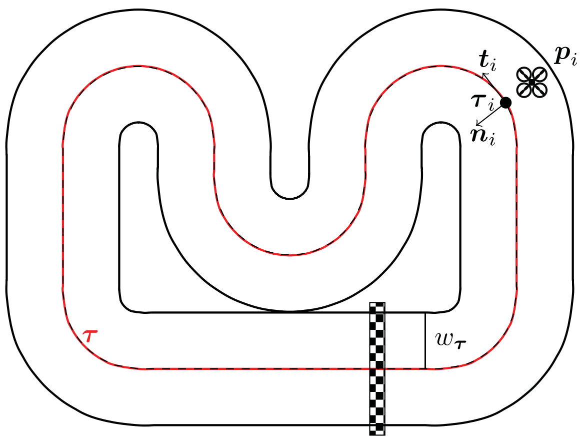

Fig. 2. Representation of the race track used for the simulations. The track is parameterized by its center line τ and its half-width w_{τ}. Given the current robot position p_i, we can define a local track frame with origin τ_i as the closest point to p_i and with t and n being the local tangent and normal vectors to the track in τ_i.

$$n\kappa = \tau'' \tag{3}$$

$$b = t \times n. \tag{4}$$

To remain within the boundaries of the track, the robot's distance from the track center line must be smaller than the track width $w_{\tau} \in \mathbb{R}^+$, i.e.,

$$\left| n(s_i)^T [p_i - \tau(s_i)] \right| \le w_{\tau}$$

where $s_i \in [0, l_{\tau}]$ is the robot position along the track, i.e., the arc length of the point on the track that is closest to p_i

$$s_i(p_i) = \arg\min_s \tfrac{1}{2} \left\| \tau(s) - p_i \right\|^2. \tag{5}$$

In order to avoid potential collisions, each robot always maintains a minimum distance $\bar{d}_i \in \mathbb{R}^+$ with respect to its opponent, i.e.,

$$\left\| p_i - p_j \right\| \ge \bar{d}_i. \tag{6}$$

Note that here, as well as in the rest of this article, we always use i $(= 1$ or $2)$ to refer to a generic robot and j $(= 2$ or 1, respectively) to refer to its opponent.

Each UAV uses onboard sensing to estimate the relative position of the opponent expressed in their local body frame, i.e.,

$$p_{ij} = R_i^T \left(p_j - p_i \right). \tag{7}$$

Fusing (7) with their ego state estimate, each robot can then estimate the world frame position of its opponent.

Since the robot cameras have a limited field of view, we need to impose a visibility constraint for measurement (7) to be available. We assume that robots are equipped with a spherical marker for detection with a size such that, even when the two robots are at a minimum distance $\bar{d}_i$ from each other, a robot marker is entirely visible by its opponent camera, provided that the following condition is satisfied:

$$\frac{p_{ij}^T}{\left\| p_{ij} \right\|} e_1 := \beta_{ij}^T e_1 \ge \cos(\alpha) \tag{8}$$

where β_{ij} is a relative bearing vector, α is the camera field of view, and $e_1 = (1, 0, 0)$ is the optical axis that is assumed, without loss of generality, to be aligned with the x-axis of the robot body frame.

Since we exploit a receding horizon control approach, the objective for each robot is to have a more advanced position along the track, with respect to the opponent, at the end of the planning horizon T. The final position is given by

$$d_i = N_i l_{\tau} + s_i(p_i(t+T))$$

where N_i is the number of completed track loops and s_i is computed as in (5). Neglecting the constant terms, the objective function of player i is then to maximize the difference

$$f_i = s_i(p_i(t+T)) - s_j(p_j(t+T)). \tag{9}$$

Because of the collision avoidance constraints (6), in order to calculate its optimal trajectory, each robot needs access to its opponent's strategy. However, since the robots are competing against each other, we do not expect them to share/communicate their plans. Instead, each robot needs to model the opponent and predict its actions. We believe that game theory [24] is the correct framework to describe this noncooperative scenario. In particular, drone racing can be seen as a *zero-sum* differential game because clearly from (9) one has $f_1 + f_2 = 0$.

Since the cost function (9) only depends on the robots' positions and the constraints on the robot positions and yaw angles can be separated, we perform the planning for the robots' position and yaw angles separately: first we apply a game theoretic approach to calculate optimal control inputs for the translational part of the robot dynamics; then, we calculate the yaw angle control in such a way that the visibility constraints (8) remain satisfied at all times given the planned/expected translational motions.

IV. Game Theoretic Formulation

In this section, we address the planning problem for the translational component of the robot state, i.e., the first row of (1). Since the robots know their relative positions and their state with respect to the world frame, we can rewrite the optimization problem in world frame coordinates. By doing this, the robots dynamics further simplify to $\dot{p}_i = u_i$.

To make the problem tractable, we discretize the planning horizon and we assume piecewise constant control inputs for both players, i.e., $u_i(t) = u_i^k / \delta t = \text{const}\ \forall t \in [t_0, t_0 + k\delta t)$ where δt is a constant sampling interval. Defining $\theta_i = (p_i^1, \dots, p_i^N, u_i^1, \dots, u_i^N)$ and $\bar{u}_i = \bar{v}_i \delta t$, the problem can then be rewritten as

$$\max_{\theta_i} \ s_i(p_i^N) - s_j(p_j^N) \tag{10a}$$

$$\text{s.t.} \quad p_i^k = p_i^{k-1} + u_i^k \tag{10b}$$

$$\left\| p_j^k - p_i^k \right\| \ge \bar{d}_i \tag{10c}$$

$$\left| n(p_i^k)^T [p_i^k - \tau(p_i^k)] \right| \le w_{\tau} \tag{10d}$$

$$\left\| u_i^k \right\| \le \bar{u}_i. \tag{10e}$$

For simplicity of notation, let us rewrite problem (10) in a more compact and general form

$$\max_{\boldsymbol{\theta}_i} \; s_i(\boldsymbol{\theta}_i) - s_j(\boldsymbol{\theta}_j) \tag{11a}$$

$$\text{s.t. } \boldsymbol{h}_i(\boldsymbol{\theta}_i) = 0 \tag{11b}$$

$$\boldsymbol{g}_i(\boldsymbol{\theta}_i) \leq 0 \tag{11c}$$

$$\boldsymbol{\gamma}_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) \leq 0 \tag{11d}$$

where the following conditions hold:

1) $\boldsymbol{h}_i$ represents the equality constraints (10b) involving a single player;
2) $\boldsymbol{g}_i$ represents the inequality constraints (10d) and (10e) involving a single player;
3) γ_i represents the inequality constraints (10c) involving both players.

There are some important details regarding this general formulation. On one hand, it should be noted that $s_j(\boldsymbol{\theta}_j)$ does not depend on player i's actions, which means that $\arg\max_{\boldsymbol{\theta}_i} s_i(\boldsymbol{\theta}_i) - s_j(\boldsymbol{\theta}_j) = \arg\max_{\boldsymbol{\theta}_i} s_i(\boldsymbol{\theta}_i)$. On the other hand, even if we drop $s_j(\boldsymbol{\theta}_j)$ from the problem, it is important to highlight that (11) should not be seen as a two separate optimization problems for player i and j but as a full differential game. A peculiarity of (11) is that the two problems are only coupled through the constraints [namely, the collision avoidance constraint (10c), (11d)] and not through the cost functions, as is usually the case in the differential games literature. Define $\Theta_i \subseteq \mathbb{R}^{6N}$ as the space of admissible strategies for player i, i.e., strategies that satisfy (11b) and (11d). Note that, due to (10c), one has $\Theta_i = \Theta_i(\boldsymbol{\theta}_j)$, i.e., the strategy of one player determines the set of admissible strategies of its opponent and, as a consequence, can influence the latter's behavior.

The information required to solve it is reasonable and most likely available to each robot: the constraints are imposed by the shape of the race track, which is known, and by the shape and dynamics of the opponent, which can be guessed with reasonable accuracy in the context of drone racing; the objective of the players in a race is obvious: to win the race, which reduces to (9).

In a game, the concept of an *optimal solution* loses meaning because, in general, and especially in a zero-sum game, it is not possible to find a pair of strategies $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ that maximize the cost function of both agents simultaneously. On the other hand, various types of *equilibria* can be defined depending on the degree of cooperation between the agents and the information pattern of the game (see [24] for a complete description of the possible alternatives). In this work, in particular, we exploit the concept of *Nash equilibria*, which, by modeling a perfectly symmetric information pattern, do not induce overly optimistic or conservative behaviors.

A Nash equilibrium is a strategy profile $(\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*) \in \Theta_1 \times \Theta_2$ such that no player can improve its own outcome by unilaterally changing its own strategy, i.e.,

$$\boldsymbol{\theta}_i^* = \arg \max_{\boldsymbol{\theta}_i \in \Theta_i(\boldsymbol{\theta}_j^*)} s_i(\boldsymbol{\theta}_i). \tag{12}$$

An alternative definition of Nash equilibria can be given by defining a *best reply map*

$$\mathcal{R}_i(\boldsymbol{\theta}_j) = \{\boldsymbol{\theta}_i \in \Theta_i(\boldsymbol{\theta}_j) : s_i(\boldsymbol{\theta}_i) = s_i^*(\boldsymbol{\theta}_j)\}$$

where

$$s_i^*(\boldsymbol{\theta}_j) = \max_{\boldsymbol{\theta}_i \in \Theta_i(\boldsymbol{\theta}_j)} s_i(\boldsymbol{\theta}_i) \tag{13}$$

is player i's *best-response return* to player j's strategy $\boldsymbol{\theta}_j$. One can show that a Nash equilibrium is a fixed point of the best reply map, i.e., such that $\boldsymbol{\theta}_i^* \in \mathcal{R}_i(\boldsymbol{\theta}_j^*)$.

Unfortunately, since problem (10) is not convex due to (10c), in general multiple Nash equilibria may exist (e.g., left versus right side overtaking). There exist few algorithms in the literature for finding Nash equilibria for problems such as ours, with continuous strategy spaces, nonconvex objectives, and state constraints. Those that do exist for continuous strategy spaces and state constraints, e.g. [36], require convexity and regularity conditions not met by our problem, and are too computationally intensive to be suitable for online implementation. Furthermore, due to the intractability of computing Nash equilibria in discrete games (see the celebrated work [37] and the related [38]), approximating our problem with a discrete set of strategies also seems ill-suited to online implementation. Therefore, the following section describes our main algorithmic contribution, an iterative algorithm that allows to *compute an approximation* of the Nash equilibria in real time.

A. Sensitivity Enhanced Iterative Best Response

In order to approximate Nash equilibria in real time, we propose a variant of an IBR algorithm. Starting from an initial guess of the Nash equilibrium strategy profile, the ego drone sets its own strategy as the best response to its opponent's strategy, then updates its opponent's strategy to the best response to its own, alternating these updates until a convergence condition is achieved, or a time limit is reached. This is done by solving a standard optimization problem in which one player strategy is allowed to change, whereas the opponent's one is kept constant. Intuitively, if the resulting sequence of strategy profiles converges, it follows that each player is best responding to its opponent. If this is the case, then no profitable unilateral change of strategy exists as required by the Nash equilibrium definition (12).

Unfortunately, a direct application of IBR to (10) does not allow to fully capture the implications of the collision avoidance constraints (10c). As already mentioned, in fact, since player i has no direct influence over the final position of player j (i.e., s_j), the second term in (9) can be neglected in (10). However, since player j is calculating its strategy by solving an optimization problem similar to (10), due to the presence of the joint constraints (10c), player i does have an effect on $s_j^*(\boldsymbol{\theta}_i^*)$ (see the counterpart of (13) for player j). In other words, while player i does not affect player j's final position *in general*, it does affect it at the Nash equilibrium. To capture these effects, we would intuitively want to substitute (11a) with the following cost function:

$$s_i(\boldsymbol{\theta}_i) - \alpha s_j^*(\boldsymbol{\theta}_i)$$

where $\alpha \geq 0$ is a free parameter. However, a closed-form expression for $s_j^*(\boldsymbol{\theta}_i)$ is difficult to obtain. Instead, inspired by [39], we can exploit sensitivity analysis to calculate a linear approximation around the current guess for the Nash equilibrium strategy profile. As we show in the following, although this modifies the cost function in each iteration of IBR, the properties of the fixed point the IBR are preserved. Namely, the fixed point is still equivalent to the first-order conditions of a Nash equilibrium for the original game.

Let us assume that, at the lth iteration, a guess $\boldsymbol{\theta}_i^{l-1}$ for player i's strategy is available to player j. Given this strategy for its opponent, player j can solve the optimal control problem (10) with $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i^{l-1}$ (fixed). This step will result in a new best-responding strategy for player j, $\boldsymbol{\theta}_j^l$, with the associated payoff $s_j^*(\boldsymbol{\theta}_i^{l-1})$. Assuming player i is now given the opportunity to modify its own strategy, we are interested in characterizing the variations of $s_j^*(\boldsymbol{\theta}_i)$ for $\boldsymbol{\theta}_i$ in the vicinity of $\boldsymbol{\theta}_i^{l-1}$ using a first-order Taylor approximation

$$s_j^*(\boldsymbol{\theta}_i) \approx s_j^*(\boldsymbol{\theta}_i^{l-1}) + \left.\frac{\mathrm{d}s_j^*}{\mathrm{d}\boldsymbol{\theta}_i}\right|_{\boldsymbol{\theta}_i^{l-1}} (\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^{l-1}). \tag{14}$$

Exploiting the Karush–Kuhn–Tucker (KKT) necessary optimality conditions associated to player j's optimal control problem (11), one can prove the following result.

Lemma 1: If s_j^* is the optimal value of an optimization problem obtained from (11) by exchanging subscripts i and j, then

$$\left.\frac{\mathrm{d}s_j^*}{\mathrm{d}\boldsymbol{\theta}_i}\right|_{\boldsymbol{\theta}_i^{l-1}} = -\boldsymbol{\mu}_j^l \left.\frac{\partial \gamma_j}{\partial \boldsymbol{\theta}_i}\right|_{(\boldsymbol{\theta}_i^{l-1}, \boldsymbol{\theta}_j^l)} \tag{15}$$

where $\boldsymbol{\theta}_j^l \in \mathcal{R}_j(\boldsymbol{\theta}_i^{l-1})$ is the best response of player j to $\boldsymbol{\theta}_i^{l-1}$ and $\boldsymbol{\mu}_j^l$ is the row vector of Lagrange multipliers associated to the joint inequality constraints (11d).

Proof: A full discussion on sensitivity analysis can be found in [40]. A brief proof, specific to the case at hand, is reported in Section A. ■

Neglecting any term that is constant with respect to $\boldsymbol{\theta}_i$, we then propose that the ego vehicle solves the following optimization problem alternatively for itself and its opponent:

$$\max_{\boldsymbol{\theta}_i \in \Theta_i^l} s_i(\boldsymbol{\theta}_i) + \alpha \boldsymbol{\mu}_j^l \left.\frac{\partial \gamma_j}{\partial \boldsymbol{\theta}_i}\right|_{(\boldsymbol{\theta}_i^{l-1}, \boldsymbol{\theta}_j^l)} \boldsymbol{\theta}_i \tag{16}$$

where Θ_i^l represents the space of strategies $\boldsymbol{\theta}_i$ that satisfy (11b) and (11d) with $\boldsymbol{\theta}_j = \boldsymbol{\theta}_j^l$.

Theorem 1: If $\gamma_1(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) = \gamma_2(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ and the iterations converge to a solution $(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l)$, then the strategy tuple $(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l)$ satisfies the first-order conditions for a Nash equilibrium.

Proof: See Section B. ■

We stress two important points about Theorem 1 and what it implies about the performance of our SE-IBR algorithm. 1) The first-order conditions for a Nash equilibrium are *necessary* conditions (not necessary and sufficient), analogously to how the KKT conditions are necessary optimality conditions in nonlinear optimization. 2) Furthermore, we have not proved that our SE-IBR algorithm always converges; only that if it does, the fixed point is equivalent to the first-order Nash conditions. Although empirically we find that it does converge, and does so quickly enough to compute online in a receding horizon loop.

In the drone racing scenario, in particular, using (5) and (6) after some straightforward calculation, (16) reduces to

$$\max_{\boldsymbol{\theta}_i \in \Theta_i^l} \left[\arg\min_s \frac{1}{2} \left\| \boldsymbol{\tau}(s) - \boldsymbol{p}_i^N \right\|^2 + \alpha \sum_{k=1}^{N} \mu_j^{k,l} \boldsymbol{\beta}_{ij}^{k,l^T} \boldsymbol{p}_i^k \right] \tag{17}$$

where

$$\boldsymbol{\beta}_{ij}^{k,l} = \frac{\boldsymbol{p}_j^{k,l} - \boldsymbol{p}_i^{k,l-1}}{\left\| \boldsymbol{p}_j^{k,l} - \boldsymbol{p}_i^{k,l-1} \right\|}.$$

To obtain a more intuitive interpretation of this result, let us assume that the track is linear and aligned to a unit vector $\boldsymbol{t}$ so that the first term in (17) can be rewritten as $\boldsymbol{t}^T \boldsymbol{p}_i^N$ (see Section IV-B for details). Since player i cannot modify the strategy of player j, the following problem has the same solutions as (17)

$$\max_{\boldsymbol{\theta}_i \in \Theta_i^l} \boldsymbol{t}^T \boldsymbol{p}_i^N - \alpha \sum_{k=1}^{N} \mu_j^{k,l} \boldsymbol{\beta}_{ij}^{k,l^T} (\boldsymbol{p}_j^{k,l} - \boldsymbol{p}_i^k). \tag{18}$$

We can then notice the following insightful facts. First of all, if none of the collision avoidance constraints (10c) were active in the lth instance of problem (10), i.e., if $\mu_j^{k,l} = 0$, then (18) reduces to (10). This has an intuitive explanation: if the collision avoidance constraints are not active, the optimal control problems for the two players are independent of each other and the original dynamic game reduces to a pair of classical optimal control problems. Interestingly, in this case, the only sensible strategy for a player is to advance as much as possible along the track.

The problem becomes much more interesting when the collision constraints are active ($\mu_j^{k,l} > 0$). In this case, indeed, the cost function optimized in (18) contains additional terms with respect to (10c). By inspecting these terms, one can notice that they have a positive effect on player i's reward if robot i reduces its distance from player j's predicted position ($\boldsymbol{p}_j^{k,l}$) along the direction of $\boldsymbol{\beta}_{ij}^{k,l}$. The intuition behind this is that, when the collision avoidance constraints are active, player i can win the race by either going faster along the track or by getting in the way of player j, thus obstructing its motion along the path.

Isolating the last term in the summation in (17), and assuming once again the track is linear and aligned to a unit vector $\boldsymbol{t}$, one can also rewrite (17)

$$\max_{\boldsymbol{\theta}_i \in \Theta_i^l} \left(\boldsymbol{t} + \alpha \mu_j^{k,N} \boldsymbol{\beta}_{ij}^{k,N} \right)^T \boldsymbol{p}_i^N + \alpha \sum_{k=1}^{N-1} \mu_j^{k,l} \boldsymbol{\beta}_{ij}^{k,l^T} \boldsymbol{p}_i^k. \tag{19}$$

From this alternative expression, it is clear that, depending on the value of $\alpha \mu_j^{k,N}$, player i might actually find it more convenient to move its last position in the direction of player j ($\boldsymbol{\beta}_{ij}^{k,N}$) rather than along the track ($\boldsymbol{t}$). One can then also interpret the free scalar gain α as an *aggressiveness* factor. Using (10b) one can also substitute $\boldsymbol{p}_i^N = \boldsymbol{p}_i^n + \sum_{k=n+1}^{N} \boldsymbol{u}_i^k$ and draw similar

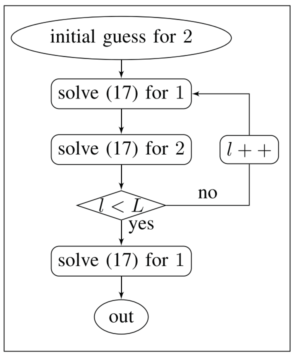

Fig. 3. Flowchart describing the proposed SE-IBR algorithm used by the ego drone to find an approximation of the Nash equilibrium. Notice that, as part of SE-IBR algorithm, the ego drone plans its own trajectory, and also plans an anticipated trajectory for its opponent.

conclusions for any intermediate position $\boldsymbol{p}_i^n$. Note that player i can exploit this effect only so long as it does not cause a violation of its own collision avoidance constraint (10c).

Before concluding this section, we want to stress the fact that, since the players do not communicate with each other, the ego drone must independently run the iterative algorithm described above and alternatively solve the optimization problem (17) for itself and for its opponent. In order to generate control inputs in real time, in our implementation we continue the iterations until a convergence condition is reached, or until a maximum number of iterations L has been reached. By exploiting the similarity between the solutions at consecutive time steps, we bootstrap the algorithm feeding the previous result as an initial guess for the current computation. This way we are able to obtain near-optimal solutions while keeping the value of L at a sufficiently low value to achieve real time performance. Additionally, since updating the opponent's strategy is only useful if this is exploited for recomputing a player's own strategy, we conclude the ego drone's iterations with an extra resolution of its own optimization problem. This also ensures that the resulting strategy profile satisfies the player's own constraints.

SE-IBR still requires the ego drone to solve a sequence of challenging optimization problems (17) at each time step (see the flowchart for SE-IBR in Fig. 3). In the following, we describe our method for solving these optimization problems.

B. Solution of Each Optimization Iteration

The solution strategy described in Section IV-A relies on the assumption that, at each iteration l, an optimal solution to problem (17) can be found given the current guess for the opponent strategy at the equilibrium.

Note that (17) is a well-posed problem. Indeed, due to the system dynamic constraints (10b), the input boundedness imposed by constraints (10e), and assuming that the sampling time is finite, the set Θ_i^l is bounded. On the other hand Θ_i^l is also never empty because the solution $\boldsymbol{\theta}_i = (\boldsymbol{p}_i^0, \ldots, \boldsymbol{p}_i^0, \mathbf{0}_2, \ldots, \mathbf{0}_2)$ is always feasible, assuming that the robots do not start from a position that violates the collision (10c) or track (10d) constraints.

Unfortunately, problem (17) is also nonlinear and nonconvex and thus we cannot guarantee the uniqueness of an optimal solution. A source of nonconvexity, in particular, is the reciprocal collision avoidance constraint (10c). For example, player i can potentially overtake player j by passing on the left or right side and, in some situations, these two solutions might even result in an equivalent payoff.

Because of the aforementioned nonconvexity, local optimization strategies will, in general, result in suboptimal solutions. In this work, however, we must calculate solutions to (17) in a very limited amount of time for online control. We then opt for a local optimization strategy thus potentially sacrificing optimality for the sake of increasing speed.

Assume that player i is at the lth iteration of the Nash equilibrium search. The predicted strategy for player j is then $\boldsymbol{\theta}_j^l$ and it remains fixed while player i is solving problem (17), again, iteratively. In order to simplify the notation, in this section we drop the superscript l that indicates the Nash equilibrium search iteration and, instead, we use the superscript to indicate the *internal* iterations used to solve (17). Moreover, to clarify the notation even further, we use a $\bar{\cdot}$ accent to indicate all quantities that remain constant across all inner iterations used to solve a single instance of (17). Therefore, assume that player i's current guess of its optimal strategy is $\bar{\boldsymbol{\theta}}_i^m$. We use $\bar{\boldsymbol{\theta}}_i^m$ to compute a convex quadratically constrained linear program (QCLP) approximation of problem (17).

Constraints (10b) and (10e) can be used as they are because they are either linear or quadratic and convex. The linear approximation of (10c) and (10d) is also straightforward and results in the following constraints:

$$\bar{\boldsymbol{\beta}}_{ij}^{k,m\,T}(\bar{\boldsymbol{p}}_j^k - \boldsymbol{p}_i^k) \geq \bar{d}_i$$

$$\left| \bar{\boldsymbol{n}}_i^{k,m\,T}(\boldsymbol{p}_i^k - \bar{\boldsymbol{\tau}}_i^{k,m}) \right| \leq w_{\boldsymbol{\tau}}$$

with $\bar{\boldsymbol{\beta}}_{ij}^{k,m} = \frac{\bar{\boldsymbol{p}}_j^k - \bar{\boldsymbol{p}}_i^{k,m}}{\|\bar{\boldsymbol{p}}_j^k - \bar{\boldsymbol{p}}_i^{k,m}\|}$, $\bar{\boldsymbol{n}}_i^{k,m} = \boldsymbol{n}(\bar{\boldsymbol{p}}_i^{k,m})$, and $\bar{\boldsymbol{\tau}}_i^{k,m} = \boldsymbol{\tau}(\bar{\boldsymbol{p}}_i^{k,m})$.

The only term that requires some attention is the linear approximation of the cost function in (16) and, in particular, of its first term because we do not have a closed-form expression for s_i as a function of $\boldsymbol{p}_i^N$. However, since $\boldsymbol{p}_i^N$ is a constant parameter in the optimization problem that defines s_i, we can exploit sensitivity analysis again to compute the derivative of s_i with respect to $\boldsymbol{p}_i^N$. To this end, let us rewrite

$$s_i = \arg\min_s d(s, \boldsymbol{p}_i^N), \text{with } d(s, \boldsymbol{p}_i^N) = \tfrac{1}{2}\left\|\boldsymbol{\tau}(s) - \boldsymbol{p}_i^N\right\|^2.$$

Then, as shown in [40] (and summarized in Section C for the case at hand), the derivative of s_i with respect to $\boldsymbol{p}_i^N$ can be calculated as

$$\frac{\mathrm{d}s_i}{\mathrm{d}\boldsymbol{p}_i^N} = -\left(\frac{\partial^2 d}{\partial s^2}\right)^{-1} \frac{\partial^2 d}{\partial s \partial \boldsymbol{p}_i^N} = \frac{\boldsymbol{\tau}'}{\|\boldsymbol{\tau}'\|^2 - \left(\boldsymbol{p}_i^N - \boldsymbol{\tau}\right)^T \boldsymbol{\tau}''}. \tag{20}$$

Exploiting the arc-length parameterization and the relations (2) and (3), we conclude

$$\frac{\mathrm{d}s_i}{\mathrm{d}\boldsymbol{p}_i^N} = \frac{\boldsymbol{t}^T}{1-\kappa\left(\boldsymbol{p}_i^N-\boldsymbol{\tau}\right)^T\boldsymbol{n}} := \boldsymbol{\sigma}(\boldsymbol{p}_i^N)$$

where $\boldsymbol{t}, \boldsymbol{n}$, and $\boldsymbol{\tau}$ must be computed for $s = s_i(\boldsymbol{p}_i^N)$. Neglecting any term that does not depend on $\boldsymbol{\theta}_i$, the cost function can then be approximated around $\boldsymbol{\theta}_i^m$ as

$$\boldsymbol{\sigma}_i^m \boldsymbol{p}_i^N + \alpha \sum_{k=1}^{N} \overline{\mu}_j^k \overline{\boldsymbol{\beta}}_{ij}^{k\,T} \boldsymbol{p}_i^k$$

with $\boldsymbol{\sigma}_i^m = \boldsymbol{\sigma}(\boldsymbol{p}_i^{N,m})$.

The solution $\boldsymbol{\theta}_i^{m+1}$ to the approximate QCLP problem can then used to build a new approximation of problem (17). The sequential QCLP optimization terminates when either a maximum number of iterations has been reached or the difference between two consecutive solutions, $r = \|\boldsymbol{\theta}_i^{m+1} - \boldsymbol{\theta}_i^m\|$, is smaller than a given threshold.

V. Opponent Position Estimation

The proposed planning strategy requires the ego drone to know both players positions $(\boldsymbol{p}_1^0, \boldsymbol{p}_2^0)$ at the beginning of each planning phase. As already mentioned, we assume that each robot knows its own position from, e.g., GPS and IMU measurements. On the other hand, because of the lack of communication between the players, the position of the opponent must be estimated by fusing the visual and inertial measurements from onboard camera and IMU sensors. In our implementation, we first exploit the onboard camera and gyro, to estimate the opponent position expressed in the local body frame of robot i, i.e., $\boldsymbol{p}_{ij}$. Then, we transform the final estimate into the world reference frame using the available ego state estimates.

The belief over the opponent's relative state is maintained via a Kalman filter and the expected value of this belief is used as the opponent's state estimate in the final solution to Problem (17). In order to be robust with respect to altitude control errors and robot roll and pitch rotations, for estimation purposes, we consider a 3-D dynamical model. We approximate the relative dynamics of opponent j with respect to i as a second-order kinematic model. Assuming constant world-frame linear velocities for both robots (i.e., $\dot{\boldsymbol{v}}_i = \dot{\boldsymbol{v}}_j = 0$), differentiating (7) we obtain

$$\begin{bmatrix}\dot{\boldsymbol{p}}_{ij}\\ \dot{\boldsymbol{v}}_{ij}\end{bmatrix} = \begin{bmatrix}-\boldsymbol{S}(\boldsymbol{\omega}_i) & \boldsymbol{I}_3\\ \boldsymbol{0}_{3\times3} & -\boldsymbol{S}(\boldsymbol{\omega}_i)\end{bmatrix}\begin{bmatrix}\boldsymbol{p}_{ij}\\ \boldsymbol{v}_{ij}\end{bmatrix} + \boldsymbol{w}\,. \tag{21}$$

In these dynamics, $\boldsymbol{v}_{ij} = \boldsymbol{R}_i^T(\boldsymbol{v}_j - \boldsymbol{v}_i)$, $\boldsymbol{S}(\boldsymbol{\omega}_i)$ is the skew symmetric matrix built with the components of robot i's body-frame rotation rates $\boldsymbol{\omega}_i$—measured via gyroscope—and $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}_6, \mathbf{Q})$ is additive, zero-mean Gaussian white noise with covariance matrix $\mathbf{Q} \in \mathbb{R}^{6\times6}$.

As discussed in the following, robot i can measure the opponent's relative position using an onboard camera, i.e.,

$$\boldsymbol{y}_i = \boldsymbol{p}_{ij} + \boldsymbol{v} \tag{22}$$

where $\boldsymbol{y}_i \in \mathbb{R}^3, \boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}_3, \mathbf{R})$ is additive, zero-mean Gaussian process noise with covariance matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$. Equations (21) and (22) form a time-varying linear system with additive Gaussian input and measurement noise. Standard Kalman filtering techniques can then be applied to design an estimator.

In practice, we achieve the measurement equation above by using a blob tracker, and tracking a colored ball mounted atop each of the drones. Since the size of the ball and its location on the drone is known before hand, the blob in a camera image can be used to infer the full relative position of the opponent drone, including depth.

Specifically, we use the technique detailed in [41] and [42] to obtain a relative positions estimate by extracting an ellipse from the image with thresholding. We then use the second-order moments of the ellipse to estimate the relative position of the ball, and therefore the opponent drone. We also assume that the extrinsic and intrinsic camera parameters are calibrated for each robot, using the pinhole camera model [43].

A key novelty in our perception system is that it uses the opponent's planned trajectory computed as part of the SE-IBR as an input to the Kalman filter. Notice in (21) the relative velocity $\boldsymbol{v}_{ij}$ between the drones must be known as an input to the relative dynamics. We use the first two waypoints from the game theoretic planner to estimate this input as $\boldsymbol{v}_{ij} = (\boldsymbol{p}_j^1 - \boldsymbol{p}_i^1)/\delta t$. This closes the loop between the planner and the perception system.

Furthermore, we also control the drone's yaw degree-of-freedom to keep the opponent within its camera's field of view (recall the field of view constraint (8)). A simple, yet effective, strategy to maintain visibility is to always align the camera axis to the relative bearing vector $\boldsymbol{\beta}_{ij}$ thus maintaining the opponent in the center of the image. Given the planned (respectively predicted) trajectory for player i and its opponent, we calculate the desired yaw angle for player i as

$$\psi_i^k = \operatorname{atan2}(\boldsymbol{e}_2^T\boldsymbol{\beta}_{ij}^k, \boldsymbol{e}_1^T\boldsymbol{\beta}_{ij}^k), \text{with } \boldsymbol{\beta}_{ij}^k = \frac{\boldsymbol{p}_j^k - \boldsymbol{p}_i^k}{\|\boldsymbol{p}_j^k - \boldsymbol{p}_i^k\|}.$$

A desired angular velocity is also be computed by differentiating consecutive samples as $\omega_i^k = \frac{\psi_i^k - \psi_i^{k-1}}{\delta t}$.

VI. Simulations and Hardware Experiments

We compare the proposed game theoretic planner from Section IV in both simulations and hardware experiments with real-time quadrotor perception and control. We denote the game theoretic planner (or SE-IBR) as GTP in the remainder of this article. We compare GTP to two baseline control algorithms in order to assess its effectiveness when racing: 1) an MPC-based approach and 2) the IBR algorithm (Section IV) that does not include the novel sensitivity term. The MPC strategy is based on the realistic, but naive, assumption that player i's opponent will follow a straight line trajectory at (constant) maximum linear velocity along the local direction of the track, i.e., $\overline{v}_j \boldsymbol{t}(s(\boldsymbol{p}_j^0))$. Based on this assumption, player i can predict player j's strategy and solve (10) as a single classical optimal control problem. The numerical optimization scheme described in Section IV-B can be used also in this case to efficiently compute a locally optimal solution. We stress that while this may seem naive compared to our GTP, nonlinear MPC for online planning is a state-of-the-art method, and it is common in the literature to assume that other agents in the environment are unreactive, and will continue with their current velocity [21].

A. Simulations

We perform extensive simulations in both 2-D and 3-D. Our planning algorithm is implemented in C++ and interfaces with the simulator using the robot operating system. Each receding horizon trajectory optimization step is solved using Gurobi [44] for all control approaches. We use a simulation time step of 10 ms, however the planners run at 20 Hz. In simulations, we do not use the vision-based tracking and estimation method described in Section V and instead only test the interaction of the agents under the assumption that they can access their opponent's position.

In the 2-D simulations, we compare GTP to the MPC benchmark while simulating the full quadrotor dynamics (see Fig. 4) via the open-source RotorS package [45]. We also use state-of-the-art nonlinear controllers to drive our quadrotors along the trajectory resulting from the solution of (17). We refer the reader to [46] and [47] for further information about the control pipeline. We set the maximum linear speeds of the two robots to 0.5 m/s and 0.6 m/s to enforce some interaction between the robots and always ensure the faster robot starts behind the slower one. The initial starting positions of the faster robot (starting in second place) are sampled from a uniform distribution with rectangle support defined by $[-0.1, 1.5] \times [-0.7, 0.7]$. Similarly, the initial starting position of the slower robot (starting in first place) is sampled from the rectangle $[1.6, 1.7] \times [-0.7, 0.7]$. We discarded any pair of sampled initial positions that would violate the collision avoidance constraints (6). The two simulated robots have a radius of 0.3 m and maintain a minimum relative distance $\overline{d}$ of 0.8 m from their opponent. The simulated track is represented in Fig. 2. The track fits into a 15 × 11 m rectangle and its half-width w_{τ} is 1.5 m. We terminated each simulation as soon as one robot completed an entire track loop and reached the finish line positioned at $x = 2.32$ m.

We additionally test on a 3-D racecourse using a similar planning and simulation setup as in the 2-D case. Along with MPC, we also compare GTP to IBR. In the 3-D simulations, all robots plan and execute in 3-D. We also make use of a single integrator model for simulating the robot motion, which is consistent with our modeling in (10). These simulations include faster speeds (1.8 and 2.0 m/s) that match state-of-the-art single drone racing results in the literature [48], [49]. The 3-D racecourse has richer elements than the 2-D, such as varying track width, varying altitude, and space in vertical direction, which allows the robots to move up and down. The robots are randomly initialized in a similar manner to the 2-D simulations such that the initial conditions among different simulation cases are consistent. The 3-D track has varying width ranging from 0.6 to 1.1 and a constant height of 0.7 (see Fig. 6). The rest of the setup is the same as in the 2-D case.

In total, we compare the performance of GTP in the following simulation cases.

Case I: 2-D fast GTP versus slow MPC.
Case II: 2-D fast MPC versus slow GTP.

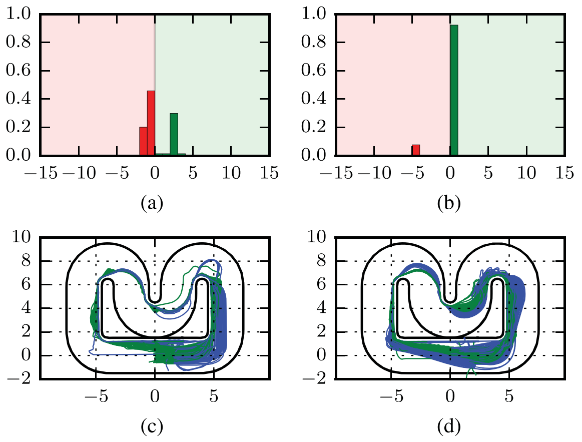
Fig. 4. Results from the 2-D simulation studies. (a) and (b) *Race result histograms* of final arc-length differences [as in (9)] in 150 runs of the simulation. Green indicates a win for the proposed GTP, whereas red is a win for MPC. In (c) and (d), the trajectories of the two competing robots in all of the simulations are plotted, with GTP in green and MPC in blue. (a) Case I: fast GTP versus slow MPC. (b) Case II: fast MPC versus slow GTP. (c) Case I: fast GTP versus slow MPC. (d) Case II: fast MPC versus slow GTP.

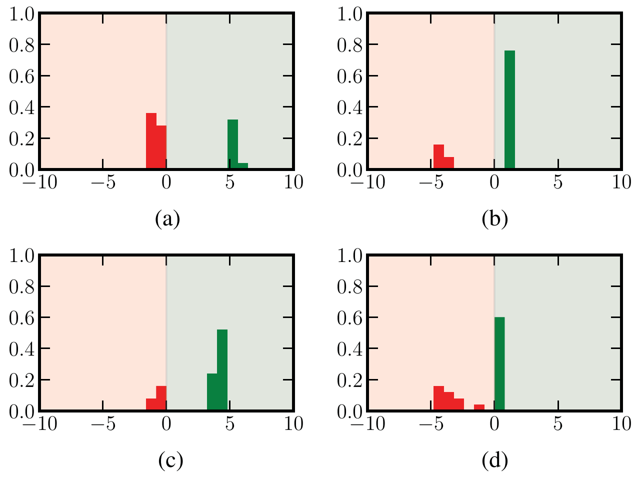
Fig. 5. Race result histograms for the 3-D simulation studies. These figures are plotted in the same way as in Fig. 4(a) and (b). (a) and (c) GTP successfully overtaking MPC and IBR when starting behind. (b) and (d) GTP successfully blocking MPC and IBR when starting in front. GTP often passes MPC and IBR early, leading to a win by a large margin [(a) and (c)]. In comparison, IBR sometimes takes a long time to pass despite the speed advantage (d). (a) Case III: fast GTP versus slow MPC. (b) Case IV: fast MPC versus slow GTP. (c) Case V: fast GTP versus slow IBR. (d) Case VI: fast IBR versus slow GTP.

Case III: 3-D fast GTP versus slow MPC.
Case IV: 3-D fast MPC versus slow GTP.
Case V: 3-D fast GTP versus slow IBR.
Case VI: 3-D fast IBR versus slow GTP.

We report a *race result histogram* representation of the final distance along the track [i.e., the arc-length difference (9)] between the two robots in Figs. 4(a) and (b) and 5. The distance is calculated in GTP-focused manner such that it is positive (green) when the robot controlled using GTP wins the race and negative (red) when its opponent wins the race. We also report the position traces of the robots for all of the simulations in Figs. 4(c) and (d) and 6. The green traces indicate GTP, blue are for MPC, and red are IBR.

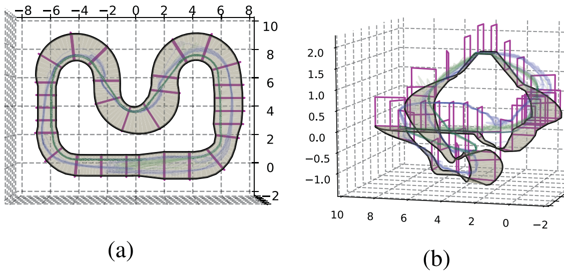

Fig. 6. Simulation trajectory traces for the GTP and MPC 3-D trials. An aerial view is presented in (a) and an isometric view is presented in (b). Note that the 3-D track has varying width, height, and elevation. Purple boxes are used to visualize the swept 3-D volume of the track. Similar *overestimation* characteristics are visible in the GTP trajectories as in the 2-D results. (a) Case IV: fast MPC versus slow GTP. (b) Case IV: fast MPC versus slow GTP.

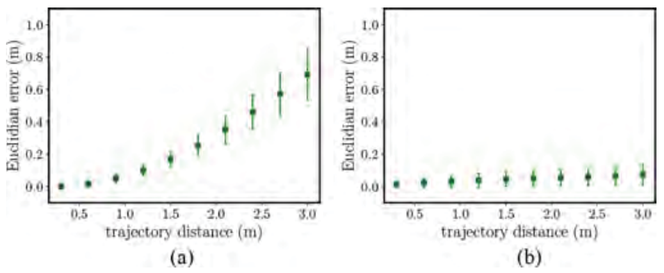

Fig. 7. Trajectory estimation error results for 3-D simulation trials. The *x*-axis denotes the trajectory horizon distance (10 total trajectory points). The *y*-axis shows the mean trajectory point Euclidean error. The error bars depict one standard deviation. GTP predicts IBR's trajectory more accurately than MPC's because it assumes its opponent is also using an IBR-like planner. (a) GTP versus MPC. (b) GTP versus IBR.

In the GTP versus MPC cases, both planners are effective in following the track and differ in the way they interact with the opponent (in fact, the algorithms are identical when the two robots do not interact) (see Figs. 4, 5(a) and (b), and 6). In the cases where GTP is the faster robot [see Figs. 4(a) and 5(a)], GTP overtakes MPC and wins approximately 30% of the races. When GTP does win, it wins by a large margin despite starting behind the MPC robot. A closer look at the simulation trajectories [see Fig. 4(c)] reveals that GTP often tends to *overestimate* its opponent (as it assumes the opponent is using GTP as well). The consequence of this is that, when attempting to overtake, it expects the opponent to block its motion and ends up following an overly cautious trajectory moving sideways along the track more than necessary. This is evident in oscillatory motion in Fig. 4(c) immediately after the upper right corner of the track.

In the cases where GTP is the slower robot [see Figs. 4(b) and 5(b)], GTP defends its position and wins approximately 80% of the races. In the trajectory images (see Figs. 4(d) and 6), we visualize the strategy adopted by GTP to defend its position, especially toward the end of the race (the bottom straight section of the track). GTP clearly moves sideways along the track to block MPC, thus exploiting the collision avoidance constraint to its own advantage. MPC cannot adopt a similar strategy because it does not properly model the reactions of its opponent. On the contrary, by assuming that the opponent will move straight along the path MPC is often forced to make room for the opponent because of its own collision avoidance constraints [see, for example, the blue traces in the top-right part of Fig. 4(c)].

We compare GTP and IBR because it highlights the advantage of adding the sensitivity term when racing [see Fig. 5(c) and (d)]. In Fig. 5(c), GTP manages to win more than 70% of the races despite starting behind IBR and often wins by a large margin. It does this because IBR does not include the sensitivity term for blocking like GTP does. Conversely, in Fig. 5(d), GTP intelligently blocks IBR approximately 60% of the races using the collision avoidance constraint in the sensitivity term. Additionally, GTP is able to accurately predict how IBR will attempt to pass, and then block accordingly.

We also compare the trajectory estimation results in Fig. 7. Here, we compute the mean Euclidean estimation error of the GTP's prediction of its opponent's executed trajectory [MPC in Fig. 7(a) and IBR in Fig. 7(b)] over all simulation trials. The results indicate the estimation is very accurate at the beginning of the trajectory and decreases further in planning horizon. The estimations are more accurate when GTP is competing against IBR because it assumes the opponent is also using an iterative response strategy. In general, we expect the trajectory estimate error to be worst at the end of the horizon, which is approximately 0.3m per unit trajectory length when competing against MPC.

B. Experiments

We also validate our approach by implementing and testing in hardware experiments with vision-based pose estimation. Our quadrotors utilize the DJI F330 frame and equipped with off-the-shelf and custom-made components [see Fig. 9(b)]. The robots weigh approximately 1.1kg and have a diagonal rotor distance of 33cm. A PixFalcon open-source autopilot board running PX4 software provides, among other sensors, an IMU and a microprocessor for attitude sensing and control. Each quadrotor carries a forward-facing global shutter RGB camera and an Odroid XU4 single-board computer that they use to compute the high-level trajectory control and the vision-based pose tracking. The GTP for both quadrotors is executed offboard on a standard laptop computer with an Intel i7-6700HQ CPU.

At each iteration of the GTP, the quadrotors have access to their own pose from an Optitrack motion capture system (MOCAP) through a WiFi network. This ego pose is fused with onboard IMU using an extended Kalman filter for full state estimation. The opponent pose, in contrast, is estimated using only the onboard vision system's filter described in Section V, by tracking a sphere mounted on the quadrotor [see Fig. 9(b)]. Using both poses, GTP computes the trajectories, defined as a list of waypoints, for the two quadrotors and sends them to the Odroids over WiFi. The planned opponent's trajectory is used as an additional input for the desired yaw of the quadrotor as well as to maintain accuracy when the sphere is out of the ego drone's field of view. This information is used to compute the full state low-level controllers exploiting differential flatness [50] and using Optitrack for onboard feedback. The whole control pipeline runs at 15Hz.

The 3-D oval race course in our experiment is shown in Figs. 8 and 9(a) and is nearly 10 m long by 6 m wide. It additionally

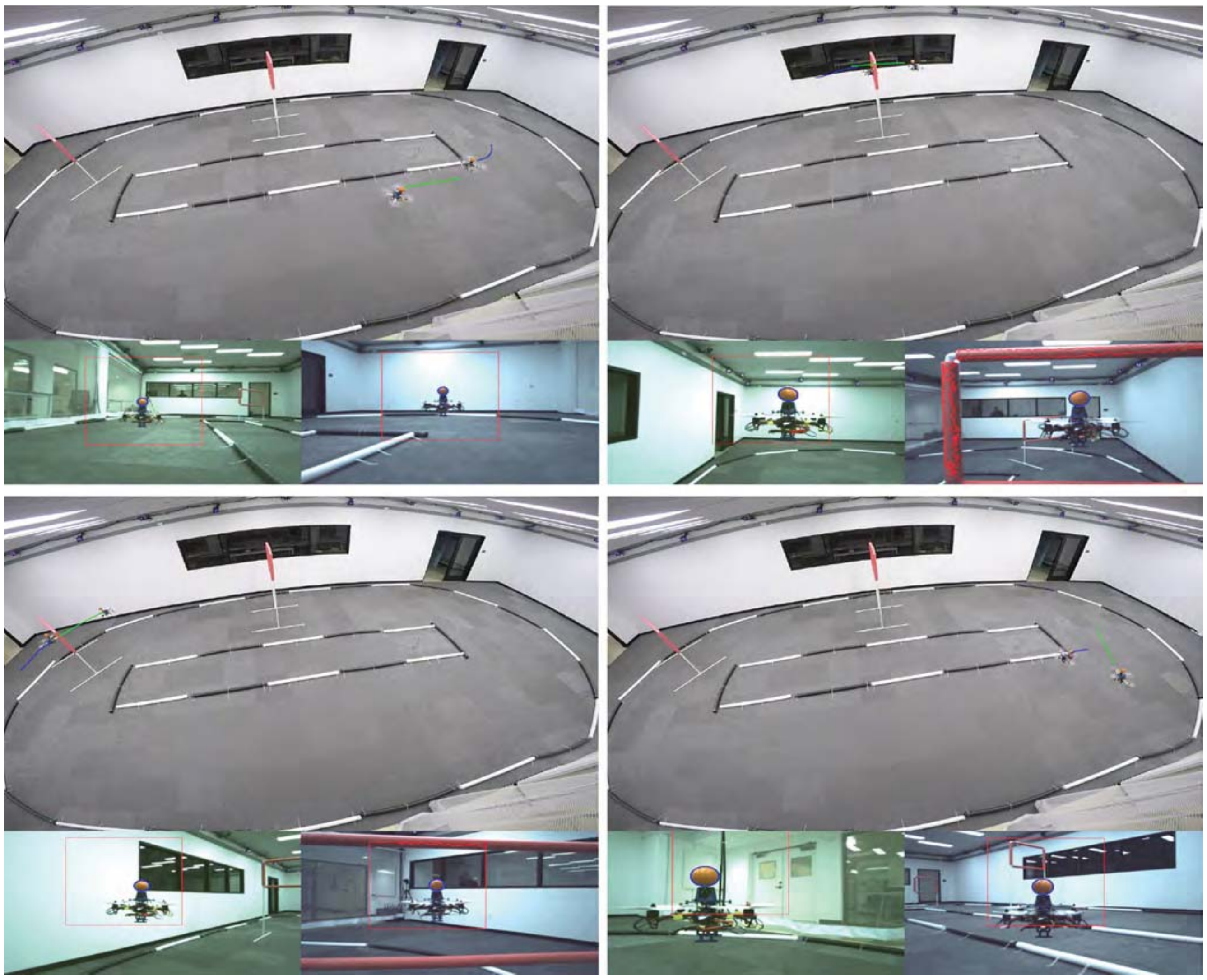
Fig. 8. Snapshots of hardware experiments of our GTP competing against the MPC planner. The large image depicts a birds-eye view of the environment, whereas the two views from the drone's onboard cameras are shown below it. Solid lines indicate the current trajectory planned by each robot for itself. The robots' onboard images include an overlay of the visual tracking results. We encourage the reader to view the video that includes multiple simulation and experimental trials: https://youtube/ayPamTiUzvA.

includes two gates to help visualize the different 3-D boundaries of the track at those locations. The track is about 1 m wide and 0.5m high, though these vary slightly along the track length.

We compare three different cases in our experiments: faster GTP versus slower MPC, faster MPC versus slow GTP, and faster GTP versus slower GTP. In all cases, the maximum speed is 0.6 m/s for the faster robot and 0.5 m/s for the slower robot. The faster robot is always behind the slower robot initially as in the simulations. The initial relative yaw angle also ensures that the two quadrotors can see each other using the front-facing cameras. Snapshots of the experiment are shown in Fig. 8 but we highly encourage the reader to see the video[1] to better appreciate the robots' behavior. In the results, we use green and blue lines to represent the planned trajectory of the GTP and MPC, respectively.

[1][online]. Available: https://youtube/ayPamTiUzvA

TABLE I
STATISTICAL DATA FROM EXPERIMENTS

	Fast GTP vs Slow MPC	Fast MPC vs Slow GTP	Fast GTP vs Slow GTP
Overtake/Total Laps	1/2	0/7	1/19
Overtake Percentage	50%	0%	5.3%

We report the total number of valid laps finished by the two robots, and the number of laps where there is a successful overtaking. The ratio of these two numbers are reported as the overtake percentage.

Statistical results of the experiment are summarized in Table I. Although we are not able to run as many trials as in the simulation due to operational limitations, the trend is similar to what was found in the simulations. The GTP overtakes MPC 50% of the time when it starts behind and very quickly after the takeoff, whereas it is never overtaken by MPC when it starts in front. As a point of comparison, when GTP races against itself, passing is relatively infrequent, with one pass in 19 runs.

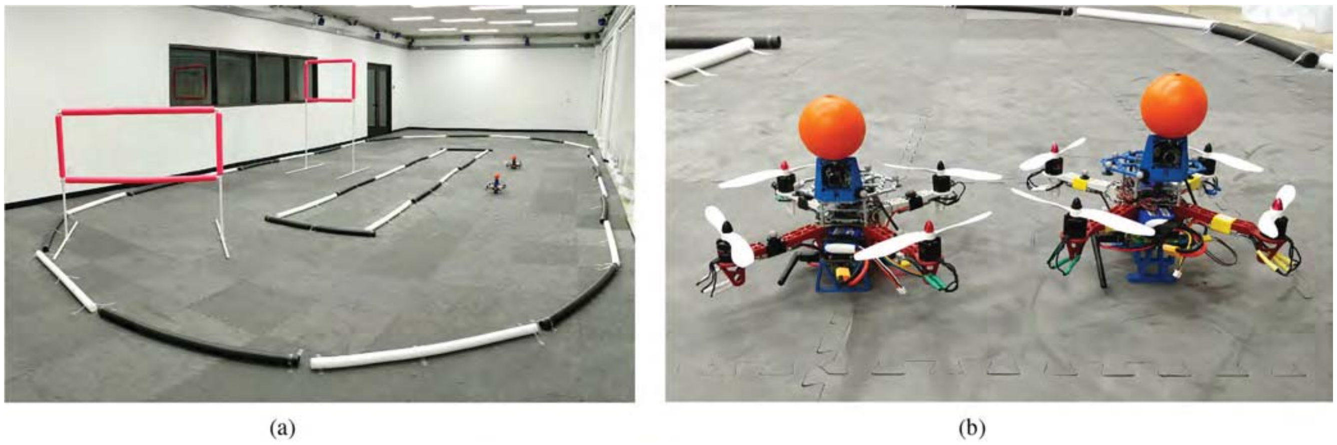

Fig. 9. Two custom-built quadrotors race along an oval-shaped 3-D race course (a). Two gates are placed in the track to induce richer interactions during the game. The quadrotors must climb and descend to successfully navigate the gates. The high gate is only narrow enough to allow only one quadrotor through. The quadrotors are equipped with forward-facing cameras and onboard computers to track the relative pose of the opponent by detecting the blob mounted on the quadrotor (b). (a) Stanford flight room. (b) Autonomous quadrotors.

We finally emphasize that our autonomous drone racing implementation runs at 0.6 m/s due to the real-time game-theoretic planning and vision-based opponent pose estimation. Although this is slower than human-piloted first-person view drones, our hardware is subject to real-world sensing and communication delays. However, we also test GTP on quadrotors without onboard perception to ensure GTP can successfully operate at real-world race speeds. We compare GTP and MPC at 1.25 and 1.0 m/s as 1.25 m/s is within state-of-the-art speeds for single-player drone racing experiments [48]. We report that GTP is successfully able to pass and block MPC (see results in video[1]).

VII. Conclusion

In this article, we described a novel online motion planning algorithm for two-player drone racing. By exploiting sensitivity analysis within an IBR algorithm, our planner can effectively model an opponent's reactions in order to gain an advantage in the race.

From a theoretical point of view, we showed that if the SE-IBR strategy converges to a solution, then the resulting trajectories satisfy necessary conditions for a Nash equilibrium in the joint space of trajectories. Moreover, we demonstrated the effectiveness of our approach through 2-D and 3-D simulations in which our GTP competed against a more traditional MPC planner. Finally, we also presented an active vision-based tracking and estimation algorithm that uses the opponent's planned trajectory to improve the visual tracking performance.

Both our planner and our estimation strategies can run in real time and their performance was demonstrated through experimental tests on real hardware, in which the GTP again outperformed an MPC opponent.

We have several open directions of research. First, we plan to characterize the convergence properties of our SE-IBR algorithm. We hope to prove that it converges and to analyze the speed of that convergence. We also intend to apply this game theoretic planning methodology to other noncooperative problems, for example, freeway driving and freeway merging autonomous cars. In addition, we intend to scale up our two-drone racing algorithm to be suitable for racing against an arbitrary number of drone opponents. Finally, we plan to investigate algorithms to learn or adapt to the strategies of opponents on line during a race.

Appendix

A. Proof of Lemma 1

In order to simplify the notation as much as possible, in this section we consider a streamlined form for the optimization problem of the form

$$\max_x s(x) \quad \text{s.t. } \gamma(x,c)=0 \tag{23}$$

where c is a scalar parameter and s and γ are scalar differentiable functions of their arguments. For each value of c, let us indicate with $x^*(c)$ the solution of (23) and with $s^*(c)=s(x^*(c))$ the associated optimal outcome. We want to study how the optimal cost s^* changes when c changes around a point $\bar{c}$, i.e.,

$$\left.\frac{\mathrm{d}s^*(c)}{\mathrm{d}c}\right|_{\bar{c}} = \left.\frac{\mathrm{d}s(x^*(c))}{\mathrm{d}c}\right|_{\bar{c}} = \left.\frac{\mathrm{d}s(x)}{\mathrm{d}x}\right|_{x^*(\bar{c})} \left.\frac{\mathrm{d}x^*(c)}{\mathrm{d}c}\right|_{\bar{c}}. \tag{24}$$

Since, for all c, $x^*(c)$ is an optimal solution to (23), it must satisfy the KKT necessary optimality conditions associated to (23), i.e.,

$$\left.\frac{\mathrm{d}s(x)}{\mathrm{d}x}\right|_{x^*} - \mu \left.\frac{\partial\gamma(x,c)}{\partial x}\right|_{x^*} = 0 \tag{25}$$

$$\gamma(x^*(c),c)=\gamma^*(c)=0 \tag{26}$$

where μ is the Lagrange multiplier associated to the equality constraint. Isolating the first term in (25) and substituting it in (24), we obtain

$$\frac{\mathrm{d}s^*(c)}{\mathrm{d}c}\bigg|_{\overline{c}} = \mu \frac{\partial \gamma(x,c)}{\partial x}\bigg|_{x^*(\overline{c})} \frac{\mathrm{d}x^*(c)}{\mathrm{d}c}\bigg|_{\overline{c}}. \tag{27}$$

Note that, since (26) must remain true for all c, its total derivative w.r.t. c must also be zero, i.e.,

$$\frac{\mathrm{d}\gamma^*(c)}{\mathrm{d}c} = \frac{\partial \gamma(x,c)}{\partial x}\bigg|_{x^*} \frac{\mathrm{d}x^*(c)}{\mathrm{d}c} + \frac{\partial \gamma(x,c)}{\partial c}\bigg|_{x^*} = 0. \tag{28}$$

Isolating the first term from (28) and substituting it in (27), we finally conclude that

$$\frac{\mathrm{d}s^*(c)}{\mathrm{d}c}\bigg|_{\overline{c}} = -\mu \frac{\partial \gamma(x,c)}{\partial c}\bigg|_{x^*(\overline{c})}$$

which reduces to (15) for $x = \boldsymbol{\theta}_j$, $\overline{c} = \boldsymbol{\theta}_i^{l-1}$, and $x^* = \boldsymbol{\theta}_j^l$.

This proof can trivially be extended to problems with multiple joint constraints or with additional constraints that do not depend on c (their derivatives with respect to c will simply be null). If the problem contains inequality constraints, instead, under the assumption that, in the vicinity of c, the set of active constraints remains the same, the proof can readily be applied by just considering an equivalent problem in which any active inequality constraint is transformed into an equality constraints and any inactive constraint is ignored.

B. Proof of Theorem 1

Applying KKT conditions to (11), one obtains the following set of necessary conditions for a Nash equilibrium $(\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*)$ and the associated Lagrange multipliers:

$$\begin{aligned}
&\frac{\partial s_i}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^*) - \boldsymbol{\mu}_i^* \frac{\partial \gamma_i}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^*, \boldsymbol{\theta}_j^*) \\
&\qquad - \boldsymbol{\lambda}_i^* \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^*) - \boldsymbol{\nu}_i^* \frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^*) = 0 && \text{(29a)}\\
&\boldsymbol{h}_i(\boldsymbol{\theta}_i^*) = 0 && \text{(29b)}\\
&\boldsymbol{g}_i(\boldsymbol{\theta}_i^*) \le 0 && \text{(29c)}\\
&\boldsymbol{\nu}_i^* \boldsymbol{g}_i(\boldsymbol{\theta}_i^*) = 0, \boldsymbol{\nu}_i^* \ge 0 && \text{(29d)}\\
&\gamma_i(\boldsymbol{\theta}_i^*, \boldsymbol{\theta}_j^*) \le 0 && \text{(29e)}\\
&\boldsymbol{\mu}_i^* \gamma_i(\boldsymbol{\theta}_i^*, \boldsymbol{\theta}_j^*) = 0, \boldsymbol{\mu}_i^* \ge 0. && \text{(29f)}
\end{aligned}$$

Now assume that the iterative algorithm described in Section IV converges to a solution $(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l)$, i.e., $\boldsymbol{\theta}_i^{l+1} = \boldsymbol{\theta}_i^l$ for both players. Then, by applying the KKT conditions to problem (16), $(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l)$ must satisfy

$$\begin{aligned}
&\frac{\partial s_i}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^l) + \alpha_i \boldsymbol{\mu}_j^l \frac{\partial \gamma_j}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^l, \boldsymbol{\theta}_j^l) - \boldsymbol{\mu}_i^l \frac{\partial \gamma_i}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^l, \boldsymbol{\theta}_j^l) && \text{(30a)}\\
&\qquad \boldsymbol{\lambda}_i^l \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^l) - \boldsymbol{\nu}_i^l \frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^l) = 0 \\
&\boldsymbol{h}_i(\boldsymbol{\theta}_i^l) = 0 && \text{(30b)}\\
&\boldsymbol{g}_i(\boldsymbol{\theta}_i^l) \le 0 && \text{(30c)}\\
&\boldsymbol{\nu}_i^l \boldsymbol{g}_i(\boldsymbol{\theta}_i^l) = 0, \boldsymbol{\nu}_i^l \ge 0 && \text{(30d)}\\
&\gamma_i(\boldsymbol{\theta}_i^l, \boldsymbol{\theta}_j^l) \le 0 && \text{(30e)}\\
&\boldsymbol{\mu}_i^l \gamma_i(\boldsymbol{\theta}_i^l, \boldsymbol{\theta}_j^l) = 0, \boldsymbol{\mu}_i^l \ge 0. && \text{(30f)}
\end{aligned}$$

If one additionally has $\frac{\partial \gamma_i}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^l, \boldsymbol{\theta}_j^l) = \frac{\partial \gamma_j}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i^l, \boldsymbol{\theta}_j^l)$ (as it is the case for our problem), then one can see that $(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l)$ satisfy (29a) and (29e) with $\boldsymbol{\lambda}_i^* = \boldsymbol{\lambda}_i^l$, $\boldsymbol{\nu}_i^* = \boldsymbol{\nu}_i^l$ and $\boldsymbol{\mu}_i^* = \boldsymbol{\mu}_i^l - \alpha_i \boldsymbol{\mu}_j^l$. In order to satisfy (29f), however, one also needs to impose that

$$\begin{aligned}
&(\boldsymbol{\mu}_1^l - \alpha_1 \boldsymbol{\mu}_2^l)\gamma_1(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l) = 0 && \text{(31a)}\\
&\boldsymbol{\mu}_1^l \ge \alpha_1 \boldsymbol{\mu}_2^l && \text{(31b)}\\
&(\boldsymbol{\mu}_2^l - \alpha_2 \boldsymbol{\mu}_1^l)\gamma_2(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l) = 0 && \text{(31c)}\\
&\boldsymbol{\mu}_2^l \ge \alpha_2 \boldsymbol{\mu}_1^l. && \text{(31d)}
\end{aligned}$$

Using (30f), (31a), and (31c) reduce to

$$\alpha_1 \boldsymbol{\mu}_2^l \gamma_1(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l) = 0$$
$$\alpha_2 \boldsymbol{\mu}_1^l \gamma_2(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l) = 0.$$

Exploiting, again, (30f), this condition is satisfied if $\gamma_1(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l) = \gamma_2(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l)$ for all active constraints and if the sets of active constraints are the same for both players (i.e., $\boldsymbol{\mu}_1^l > 0 \iff \boldsymbol{\mu}_2^l > 0$). Both these conditions are satisfied if, as it is the case for our application, $\gamma_1(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l) = \gamma_2(\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l)$. As for (31b) and (31d), instead, if $\gamma_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \gamma_j(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$, one can enforce it by making α_i arbitrarily small.

C. Proof of (20)

Consider the following optimization problem:

$$\min_s d(s, \boldsymbol{p}_i^N).$$

We can interpret $\boldsymbol{p}_i^N$ as a constant parameter and study how the solution s_i to the above problem changes when $\boldsymbol{p}_i^N$ changes around a point $\overline{\boldsymbol{p}}_i^N$. Under the optimality assumption, for each value $\boldsymbol{p}_i^N$, the corresponding solution $s_i(\boldsymbol{p}_i^N)$ must satisfy the following necessary condition:

$$\frac{\partial d(s, \boldsymbol{p}_i^N)}{\partial s}\bigg|_{s_i(\boldsymbol{p}_i^N)} = 0. \tag{33}$$

Note that the left-hand side of (33) is a function of $\boldsymbol{p}_i^N$ only and it must be zero for all $\boldsymbol{p}_i^N$. Therefore, its derivative with respect to $\boldsymbol{p}_i^N$ must also be zero

$$\begin{aligned}
0 &= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{p}_i^N}\left[\frac{\partial d(s, \boldsymbol{p}_i^N)}{\partial s}\bigg|_{s_i(\boldsymbol{p}_i^N)}\right]\\
&= \frac{\partial^2 d(s, \boldsymbol{p}_i^N)}{\partial s^2}\bigg|_{s_i(\boldsymbol{p}_i^N)} \frac{\mathrm{d}s_i(\boldsymbol{p}_i^N)}{\mathrm{d}\boldsymbol{p}_i^N} + \frac{\partial^2 d(s, \boldsymbol{p}_i^N)}{\partial s \partial \boldsymbol{p}_i^N}.
\end{aligned}$$

We can, then, conclude that

$$\frac{\mathrm{d}s_i(\boldsymbol{p}_i^N)}{\mathrm{d}\boldsymbol{p}_i^N} = -\left[\frac{\partial^2 d(s, \boldsymbol{p}_i^N)}{\partial s^2}\bigg|_{s_i(\boldsymbol{p}_i^N)}\right]^{-1} \frac{\partial^2 d(s, \boldsymbol{p}_i^N)}{\partial s \partial \boldsymbol{p}_i^N}.$$

■

ACKNOWLEDGMENT

This article solely reflects the opinions and conclusions of its authors and not Toyota Research Institute or any

other Toyota entity. A video of the simulations and experiments can be found in the supplementary material and also at https://youtube/ayPamTiUzvA.

References

[1] H. Moon, Y. Sun, J. Baltes, and S. J. Kim, "The IROS 2016 competitions [competitions]," *IEEE Robot. Autom. Mag.*, vol. 24, no. 1, pp. 20–29, Mar. 2017.

[2] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, "A direct visual servoing-based framework for the 2016 IROS autonomous drone racing challenge," *J. Field Robot.*, vol. 35, no. 1, pp. 146–166, 2018.

[3] N. R. Kapania, J. Subosits, and J. C. Gerdes, "A sequential two-step algorithm for fast generation of vehicle racing trajectories," *J. Dyn. Syst., Meas., Control*, vol. 138, no. 9, 2016, Art. no. 091005.

[4] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Proc. IEEE Int. Conf. Robot. Autom.*, Stockholm, Sweden, May 2016, pp. 1433–1440.

[5] G. Williams *et al.*, "Information theoretic MPC for model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, Singapore, May 2017, pp. 1714–1721.

[6] Yamaha Motor and SRI International, "Yamaha MOTOBOT 2," 2017. [Online]. Available: https://youtu.be/BjZPvXKewFk

[7] R. Stelzer and T. Pröll, "Autonomous sailboat navigation for short course racing," *Robot. Auton. Syst.*, vol. 56, no. 7, pp. 604–614, 2008.

[8] R. Spica, D. Falanga, E. Cristofalo, E. Montijano, D. Scaramuzza, and M. Schwager, "A real-time game theoretic planner for autonomous two-player drone racing," in *Proc. Robot., Sci. Syst.*, 2018.

[9] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *Int. J. Robot. Res.*, vol. 4, no. 3, pp. 109–117, 1985.

[10] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[11] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.*, vol. 17, no. 7, pp. 760–772, 1998.

[12] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *J. Guid., Control, Dyn.*, vol. 25, no. 1, pp. 116–129, 2002.

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[14] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optimal Control Appl. Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[15] F. Feurtey and T. Chikayama, "Simulating the collision avoidance behavior of pedestrians," Master's thesis, Dept. Electron. Eng., Univ. Tokyo, Tokyo, Japan, 2000.

[16] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, Jan. 2007.

[17] B. D. Anderson, C. Yu, B. Fidan, and J. Hendrickx, "Rigid graph control architectures for autonomous formations," *IEEE Control Syst. Mag.*, vol. 28, no. 6, pp. 48–63, Dec. 2008.

[18] Z. Wang and M. Schwager, "Kinematic multi-robot manipulation with no communication using force feedback," in *Proc. IEEE Int. Conf. Robot. Autom.*, Stockholm, Sweden, May 2016, pp. 427–432.

[19] C. Cai, C. Yang, Q. Zhu, and Y. Liang, "Collision avoidance in multi-robot systems," in *Proc. Int. Conf. Mechatronics Autom.*, Harbin, China, Sep. 2007, pp. 2795–2800.

[20] G. M. Hoffmann and C. J. Tomlin, "Decentralized cooperative collision avoidance for acceleration constrained vehicles," in *Proc. 47th IEEE Conf. Decis. Control*, Cancun, Mexico, Dec. 2008, pp. 4357–4363.

[21] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Proc. 14th Int. Symp. Robot. Res.*, Lucerne, Switzerland, Aug. 2011, pp. 3–19.

[22] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, on-line collision avoidance for dynamic vehicles using buffered Voronoi cells," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 1047–1054, Apr. 2017.

[23] T. Başar and P. Bernhard, *H-Infinity Optimal Control and Related Minimax Design Problems: A Dynamic Game Approach*. Boston, MA, USA: Birkhäuser, 2008.

[24] T. Başar and G. J. Olsder, *Dynamic Noncooperative Game Theory*, 2nd ed. Philadelphia, PA, USA: Soc. Ind. Appl. Math., 1998.

[25] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin, "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games," *IEEE Trans. Autom. Control*, vol. 50, no. 7, pp. 947–957, Jul. 2005.

[26] J. H. Gillula, G. M. Hoffmann, H. Huang, M. P. Vitus, and C. J. Tomlin, "Applications of hybrid reachability analysis to robotic aerial vehicles," *Int. J. Robot. Res.*, vol. 30, no. 3, pp. 335–354, 2011.

[27] B. Landry, M. Chen, S. Hemley, and M. Pavone, "Reach-avoid problems via sum-of-squares optimization and dynamic programming," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Madrid, Spain, 2018, pp. 4325–4332.

[28] M. Chen, S. L. Herbert, M. Vashishtha, S. Bansal, and C. J. Tomlin, "Decomposition of reachable sets and tubes for a class of nonlinear systems," *IEEE Trans. Autom. Control*, vol. 63, no. 11, pp. 3675–3688, Nov. 2018.

[29] R. Isaacs, *Differential Games: A Mathematical Theory With Applications to Warfare and Pursuit, Control and Optimization*. Hoboken, NJ, USA: Wiley, 1965.

[30] J. Walrand, E. Polak, and H. Chung, "Harbor attack: A pursuit-evasion game," in *Proc. 49th Annu. Allerton Conf. Commun., Control, Comput.*, 2011, pp. 1584–1591.

[31] D. Sadigh, S. S. Sastry, S. A. Seshia, and A. Dragan, "Information gathering actions over human internal state," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Daejeon, South Korea, Oct. 2016, pp. 66–73.

[32] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for autonomous cars that leverage effects on human actions," in *Proc. Robot., Sci. Syst.*, Cambridge, MA, USA, Jul. 2016.

[33] G. Williams, B. Goldfain, P. Drews, J. M. Rehg, and E. A. Theodorou, "Best response model predictive control for agile interactions between autonomous ground vehicles," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 2403–2410.

[34] A. Dreves and M. Gerdts, "A generalized Nash equilibrium approach for optimal control problems of autonomous cars," *Optimal Control Appl. Methods*, vol. 39, no. 1, pp. 326–342, 2018.

[35] A. Liniger and J. Lygeros, "A non-cooperative game approach to autonomous racing," *IEEE Trans. Control Syst. Technol.*, vol. 28, no. 3, pp. 884–897, May 2020.

[36] F. Facchinei and C. Kanzow, "Generalized Nash equilibrium problems," *4OR*, vol. 5, no. 3, pp. 173–210, Sep. 2007.

[37] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, "The complexity of computing a Nash equilibrium," *SIAM J. Comput.*, vol. 39, no. 1, pp. 195–259, 2009.

[38] X. Chen and X. Deng, "Settling the complexity of two-player Nash equilibrium," in *Proc. 47th Annu. IEEE Symp. Found. Comput. Sci.*, 2006, pp. 261–272.

[39] T. Raivio and H. Ehtamo, "On the numerical solution of a class of pursuit-evasion games," in *Advances in Dynamic Games and Applications* (Annals of the International Society of Dynamic Games). Boston, MA, USA: Birkhäuser, 2000, pp. 177–192.

[40] A. V. Fiacco, *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*. New York, NY, USA: Academic, 1983.

[41] R. T. Fomena and F. Chaumette, "Improvements on visual servoing from spherical targets using a spherical projection model," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 874–886, Aug. 2009.

[42] R. Spica, P. R. Giordano, and F. Chaumette, "Active structure from motion: Application to point, sphere, and cylinder," *IEEE Trans. Robot.*, vol. 30, no. 6, pp. 1499–1513, Dec. 2014.

[43] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision: From Images to Geometric Models*, vol. 26. Berlin, Germany: Springer, 2012.

[44] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2019. [Online]. Available: http://www.gurobi.com

[45] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)* (Studies in Computational Intelligence), vol. 625. New York, NY, USA: Springer, 2016, ch. 23.

[46] M. Faessler, F. Fontana, C. Forster, and D. Scaramuzza, "Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor," in *Proc. IEEE Int. Conf. Robot. Autom.*, Seattle, WA, USA, May 2015, pp. 1722–1729.

[47] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3D mapping with a quadrotor MAV," *J. Field Robot.*, vol. 33, no. 4, pp. 431–450, 2016.

[48] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," in *Proc. Conf. Robot. Learn.*, 2018, pp. 133–145.

[49] E. Kaufmann *et al.*, "Beauty and the beast: Optimal methods meet learning for drone racing," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 690–696.

[50] D. Zhou and M. Schwager, "Vector field following for quadrotors using differential flatness," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 6567–6572.

Riccardo Spica (Member, IEEE) received the Ph.D. degree in signal processing and telecommunication from the University of Rennes I, Rennes, France, in December 2015.

He carried out his Ph.D. research with the Lagadic Team, Institut de recherche en informatique et systèmes alèatoires, Rennes, where he also spent one additional year as a Postdoc with CNRS. In 2017, he joined the Multi-Robot Systems Lab, University of Stanford, Stanford, CA, USA, where he was at the time of this publication. Since 2018, he has been a Guidance Control and Navigation Engineer with MBDA. His research deals with vision-based estimation and control strategies for manipulators and mobile robots. In particular, he is interested in active perception, visual servoing, and decentralized multirobot control.

Eric Cristofalo (Graduate Student Member, IEEE) received the B.S. degree in mechanical engineering from Drexel University, Philadelphia, PA, USA, in 2013 and the M.S. degree in mechanical engineering from Boston University, Boston, MA, USA, in 2016. He is currently working toward the Ph.D. degree in aeronautics and astronautics with the Multi-Robot Systems Lab, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, USA.

He is also a National Defense Science and Engineering Graduate Fellow with the Multi-Robot Systems Lab, Department of Aeronautics and Astronautics, Stanford University. His research interests include vision-based control, active perception, distributed pose estimation, and 3-D reconstruction with multirobot systems.

Zijian Wang received the B.S. degree in automation and mechatronic engineering from Beihang University, Beijing, China, in 2013, the M.S. degree in mechanical engineering from Boston University, Boston, MA, USA, in 2016, and the Ph.D. degree in aeronautics and astronautics from Stanford University, Stanford, CA, USA, in 2019.

His current research focuses on multirobot systems, particularly on designing distributed planning and control algorithms that enable a group of intelligent robots to either collaborate on a common task, or compete among adversarial agents.

Eduardo Montijano (Member, IEEE) received the M.Sc. and Ph.D. degrees in ingeniería de sistemas e informatica from the Universidad de Zaragoza, Zaragoza, Spain, in 2008 and 2012, respectively.

He is currently an Assistant Professor with the Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza. Between 2012 and 2016, he was a Faculty Member with the Centro Universitario de la Defensa, Zaragoza. His research interests include distributed algorithms, cooperative control, and computer vision.

Dr. Montijano's Ph.D. obtained the extraordinary award of the Universidad de Zaragoza in the 2012–2013 academic year.

Mac Schwager (Member, IEEE) received the B.S. degree from Stanford University, Stanford, CA, USA, in 2000, and the M.S. and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2005 and 2009, respectively, all in mechanical engineering.

He is currently an Assistant Professor with the Aeronautics and Astronautics Department, Stanford University. He was a Postdoctoral Researcher working jointly with the GRASP Lab, University of Pennsylvania, Philadelphia, PA, USA and Computer Science and Artificial Intelligence Laboratory, MIT from 2010 to 2012, and was an Assistant Professor with Boston University, Boston, MA, USA from 2012 to 2015. His research interests include distributed algorithms for control, perception, and learning in groups of robots, and models of cooperation and competition in groups of engineered and natural agents.

Dr. Schwager was the recipient of the NSF CAREER Award in 2014, the DARPA YFA in 2018, the Google Faculty Research Award in 2018, and the IROS Toshio Fukuda Young Professional Award in 2019.