Reach-Avoid Games Via Mixed-Integer Second-Order Cone Programming

Joseph Lorenzetti, Mo Chen, Benoit Landry, Marco Pavone

***Abstract*— Reach-avoid games are excellent proxies for studying many problems in robotics and related fields, with applications including multi-robot systems, human-robot interactions, and safety-critical systems. However, solving reach-avoid games is difficult due to the conflicting and asymmetric goals of agents, and trade-offs between optimality, computational complexity, and solution generality are commonly required. This paper seeks to find attacker strategies in reach-avoid games that reduce computational complexity while retaining solution quality by using a receding horizon strategy. To solve for the open-loop strategy fast enough to enable a receding horizon approach, the problem is formulated as a mixed-integer second-order cone program. This formulation leverages the use of sums-of-squares optimization to provide guarantees that the strategy is robust to all possible defender policies. The method is demonstrated through numerical and hardware experiments.**

I. Introduction

Many important problems in robotics involve interactions with dynamic environments, such as robot-robot interactions and human-robot interactions. Game theoretic methods are often useful for modeling these types of interactions; some examples include two-player drone racing [1], autonomous driving [2], and control of robotic swarms [3]. In particular, differential games are of specific interest to applications involving dynamical systems evolving in continuous state spaces and time, and has been effective in analyzing practical scenarios such as aircraft collision avoidance [4], multi-vehicle routing [5], and robust trajectory planning [6].

Related work: A challenging class of differential games is known as reach-avoid games, in which Player 1 aims to reach a desired target set of states while staying away from an avoid set, typically induced by Player 2. Reach-avoid games are especially difficult to analyze, since unlike other differential games such as pursuit-evasion games, both a target set and avoid set must be taken into account [8]. Despite this difficulty, reach-avoid games have been applied to robotics competitions such as the Robocup [9], as well as practical scenarios such as unmanned airspace management [10] and rogue vehicle interception [11]. Due to the intricacies of the problem, solution methods tend to involve trade-offs in several axes of consideration: computational complexity, solution optimality, generality of system dynamics, and generality of the problem setup. Trading off computational complexity for solution optimality involves making various assumptions on the information patterns between the two players, with the two ends of the spectrum being open- and closed-loop formulations.

One crucial benefit of closed-loop strategies is that the players are able to react to new information in real-time; this is especially important in many practical scenarios such as autonomous driving, in which a goal may be considered infeasible until more information is collected over time. Dynamic programming methods based on Hamilton-Jacobi (HJ) reachability analyses sacrifice computational complexity to produce optimal closed-loop strategies for players operating in a large variety of problem setups [12]. However, HJ methods are only computationally tractable when the two players have single integrator dynamics, although higher level logic such as maximum matching can be used to obtain approximations to multiplayer reach-avoid games [13].

Since finding optimal closed-loop strategies is very computationally expensive, some work has looked at using open-loop control. One approach considers a pair of simpler differential games where, in each, one player selects their control strategy in an open-loop fashion [14]. With the other player having access to that strategy, the information pattern provides a conservative solution. There are many other methods for analyzing reach-avoid games that make different trade-offs [11], [15]–[22]. These methods rely on geometrical arguments and analytic solutions for special problem setups and system dynamics [11], [15], [19]. This produces computationally tractable solutions, but only for particular application domains.

Open-loop reach-avoid strategies are closely related to obstacle avoidance in dynamic environments, although the notion of an adversarial opponent is not considered. In this area, velocity obstacles [23] and reciprocal velocity obstacles [24] have been used to reactively plan trajectories for avoidance, but cannot guarantee being able to find a collision-free trajectory to the goal. The authors in [25] propose a planning method that guarantee collision avoidance for a finite horizon, assuming *a priori* knowledge of the object's motion. More recently, with advances in computation speed, several model-predictive control and reinforcement learning methods have also been used [26]–[28].

Statement of Contributions: In this paper, we present a receding horizon strategy for reach-avoid games that takes advantage of both the computational simplicity of open-loop formulations and the ability to react to new information as in closed-loop formulations. The key theoretical insight of our work is to parametrize Player 1 trajectories as piece-

The authors are with the Department of Aeronautics and Astronautics, Stanford University. Emails: {jlorenze, mochen72, blandry, pavone}@stanford.edu.

This work was supported by the Toyota Research Institute ("TRI"). This article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. Joseph Lorenzetti is supported by the Department of Defense (DoD) through the National Defense Science and Engineering Fellowship (NDSEG) Program.

wise low-order polynomials, which allows us to obtain an MISOCP that can be solved in real-time. The advantages of our real-time solution is shown in hardware experiments: an autonomous robot using our proposed strategy playing against a human-controlled robot is able to win despite the scenario being initially not winnable.

In addition, our method considers Player 1 models that belong to the class of differentially flat systems, which provides a rich set of interesting nonlinear dynamics. Therefore, our approach offers more problem generality than methods restricted to simple dynamics by computational challenges, and open-loop methods that assume special dynamics required to obtain analytical solutions. Our formulation also allows generalized problem setups, and could be applied to multi-agent reach-avoid games or reach-avoid games in environments with static and dynamic obstacles, unlike other computationally efficient methods which rely on special geometries to obtain analytical solutions.

Organization: The rest of this paper is organized as follows. In Section II-A, we formulate the reach-avoid game and state our assumptions. In Section II-B, we provide background on sums-of-squares (SOS) programming. In Section III, we show how the reach-avoid game can rewritten as an optimization problem with SOS constraints. In Section IV, we show how the problem from Section III can be reduced to a MISOCP. In Section V, we present results from numerical experiments. In Section VI, we present results from hardware experiments. In Section VII, we draw conclusions and discuss directions for future work.

II. Preliminaries

A. Problem formulation

In a two player reach-avoid game, the dynamics of the players are

$$\dot{x}_1 = f_1(x_1, u_1),\; u_1 \in U_1,\; x_1 \in \mathcal{X}_1, \tag{1}$$

$$\dot{x}_2 = f_2(x_2, u_2),\; u_2 \in U_2,\; x_2 \in \mathcal{X}_2, \tag{2}$$

where x_1 and x_2 are the states of the players, and u_1 and u_2 are their control inputs.

Player 1 tries to reach the target set $\mathcal{S}_T$ in minimum time while avoiding capture by Player 2. The capture region is denoted by $\mathcal{S}_A$. Therefore the optimization problem becomes

$$\begin{aligned} \underset{u_1}{\text{minimize}} \quad & t_f \\ \text{subject to} \quad & x_1(t_f) \in \mathcal{S}_T, \\ & x_1(t) \notin \mathcal{S}_A,\; \forall t \in [0, t_f],\; \forall u_2(t) \in U_2, \\ & u_1(t) \in U_1,\; \forall t \in [0, t_f], \\ & \dot{x}_1 = f_1(x_1, u_1), \\ & \dot{x}_2 = f_2(x_2, u_2), \\ & x_1(0) = x_{1,0}, \\ & x_2(0) = x_{2,0}. \end{aligned} \tag{3}$$

We assume the dynamics f_1 for Player 1 are differentially flat. A system has differentially flat dynamics with flat output z when there exists a function α that is a function of the state x, control u, and the first q derivatives of the control

$$z = \alpha(x, u, \dot{u}, \ldots, u^{(q)}), \tag{4}$$

such that the solutions of the system can be written in terms of z and its first r derivatives

$$x = \beta(z, \dot{z}, \ldots, z^{(r)}), \quad u = \gamma(z, \dot{z}, \ldots, z^{(r)}). \tag{5}$$

Explicit trajectory generation for differentially flat systems is comparatively simple because the trajectory can be planned in output space and then algebraically mapped to the appropriate control inputs. This enables the trajectories of the flat outputs z to be parametrized using any set of smooth basis functions, such as polynomials. For a concrete example see Section V-A, and for further details see [29]. Examples of differentially flat systems include integrator systems, quadrotors, and kinematic cars. We also assume that the control bounds for Player 1 can be written as constraints on the flat outputs in the form of a convex polytope.

For clarity, we simplify the dynamics f_2 for Player 2 to be integrators, which allows for an analytical representation of the boundary of Player 2's forward reachable set (FRS) as a polynomial in time. However, our method extends to general nonlinear dynamics for Player 2 using any tool that produces polytopic representations of FRSs such as [30]–[32].

B. Background on Sums-of-Squares Programming

We will rewrite the optimization (3) using concepts from sums-of-squares (SOS) programming. SOS programming is a technique to check whether polynomials are globally non-negative by demonstrating that they can be written as sums of squared polynomials. SOS programs can also be solved as semidefinite programs (SDPs), and in special cases as second-order cone programs, and have been used in many applications [33]. Reference [34] provides a more detailed introduction to SOS programming.

A polynomial $\phi(t)$ is SOS if

$$\phi(t) = \sum_{i=1}^{l} p_i^2(t), \tag{6}$$

where $p_i(t)$ are also polynomials. The condition (6) can also be written as $\phi(t) = \rho^T(t) Q \rho(t), Q \succeq 0$, with $\rho(t)$ being a vector of polynomial basis functions (such as monomials) up to half the order of $\phi(t)$. Expanding the right hand side and matching coefficients with the polynomial $\phi(t)$ results in affine constraints on the elements of Q. Therefore, when considering polynomials of SOS form, we can convert SOS programs to semidefinite programs (SDPs):

$$\begin{aligned} \underset{X \in S^n}{\text{minimize}} \quad & \text{Tr}(C^T X) \\ \text{subject to} \quad & A_i^T X = b_i,\; i = 1, 2, \ldots, m, \\ & X \succeq 0, \end{aligned} \tag{7}$$

where C, X, and A_i are symmetric matrices.

SOS programming is a useful tool in optimization because in general checking non-negativity of a polynomial $\phi(t)$ is

NP-hard, but checking whether $\phi(t)$ is SOS, a sufficient condition for non-negativity, can be done efficiently. This paper makes extensive use of constraints of the form "$\phi(t)$ is SOS" to certify state and input constraints to model, for example, Player 1's avoidance of Player 2.

Another practical use of SOS constraints is in guaranteeing non-negativity over a set. Given sets that are defined by zero superlevel sets of polynomials,

$$A = \{x(t) : \phi_A(x(t)) \geq 0\}, \quad B = \{x(t) : \phi_B(x(t)) \geq 0\}, \tag{8}$$

the constraint $x(t) \in B \Rightarrow x(t) \in A$ can be written as

$$\phi_A(x(t)) - L(x(t))\phi_B(x(t)) \text{ is SOS}, \quad L(x(t)) \text{ is SOS}. \tag{9}$$

This leads to an SDP with bilinear constraints due to the $L(x(t))\phi_B(x(t))$ terms. These types of SDPs are solved by alternating between fixing the coefficients of the polynomials ($L(x(t))$ or $\phi_B(x(t))$) and solving the resulting problem until convergence.

III. Formulation with SOS Constraints

Problem (3) contains several set constraints on the state and control. Using zero superlevel sets of polynomials to represent sets as in, (8), we are able to naturally rewrite these set constraints as SOS constraints. This conversion is discussed in Sections III-B and IV-A.

In this paper we represent trajectories as piecewise low-order polynomials, although in general using a single polynomial of any order is sufficient to formulate SOS constraints. Piecewise polynomials are introduced here for consistency of presentation, since in Section IV we will show that using low-order piecewise polynomials allow the problem to reduce to an MISOCP.

A. Parameterizing Player 1's Policy

As mentioned in Section II-A, when considering the class of differentially flat systems for Player 1 we can construct feasible trajectories from flat outputs that are parameterized by polynomial basis functions. More specifically, we choose to represent Player 1's trajectory in flat output space as a *piecewise* polynomial in time. This provides two benefits: first it helps enable the mixed-integer formulation discussed later, and second (and more importantly) each piecewise component can be a low-order polynomial but the resulting trajectory can still have rich features.

The full trajectory is defined with N segments for each of n state space dimensions. Each polynomial segment j of state space dimension i is of degree d, and thus is defined by $d+1$ coefficients $c_j^i = [c_{j,0}, \ldots, c_{j,d}]$ and a set of polynomial basis functions $\rho = [1, t, t^2, \ldots, t^d]$. Thus the j^{th} segment of the i^{th} dimension is defined by

$$p_j^i(t) = \sum_{k=0}^{d} c_{j,k}^i \rho_k, \;\; t \in [t_{j-1}, t_j].$$

B. Control Bounds

We assume the control bounds can also be written as constraints that are linear functions of the polynomials $p_j^i(t)$ or their derivatives:

$$A_U \left[p_j^i(t), \dot{p}_j^i(t), \ldots, p_j^{(d),i}(t)\right]^T - b_U \text{ is SOS}. \tag{10}$$

For example, a 1D double integrator system with polynomial $p_j^i(t) = x(t)$, dynamics $\ddot{x} = u$, and control bounds $\underline{u}$, $\overline{u}$ the constraint would simply be

$$\ddot{x}(t) - \underline{u} \text{ is SOS}, \quad -\ddot{x}(t) + \overline{u} \text{ is SOS}.$$

C. Target Region

The target region $\mathcal{S}_T \subset \mathcal{X}_1$ is defined in the state space as a convex polytope with P sides that inner approximates the true target region. The polytope is defined by a set of linear constraints such that a point is in the target region ($p_1 \in \mathcal{S}_T$) if $A_T p_1 \geq b_T$, where $A_T \in \mathbb{R}^{P \times N}$ and $b_T \in \mathbb{R}^P$.

With Player 1's trajectory parameterized as a piecewise polynomial this constraint can be rewritten by evaluating the N^{th} segment at the final time t_f, that is

$$A_T \left[p_N^1(t_f), \ldots, p_N^n(t_f)\right]^T - b_T \geq 0. \tag{11}$$

Since the objective is to minimize final time, t_f, this constraint is nonlinear. To remove the nonlinearity we fix the final time[1] and use an alternative objective function. As a surrogate for minimizing final time and assuring that there is always a feasible solution to the optimization problem, slack variables γ are introduced and their sum is minimized

$$\min \sum_{k=1}^{P} \gamma_k. \tag{12}$$

Then the reach constraints are rewritten

$$\begin{aligned} A_T \left[p_N^1(t_f), \ldots, p_N^n(t_f)\right]^T - b_T \\ + [\gamma_1, \ldots, \gamma_P]^T \geq 0, \\ \gamma_k \geq 0 \;\forall k \in [1, \ldots, P], \end{aligned} \tag{13}$$

such that they become simple linear constraints.

D. Avoid Region

Avoiding capture requires staying outside of the avoid set $\mathcal{S}_A$ for all time and for all possible Player 2 actions. This is accomplished by ensuring Player 1 is outside of the FRS of Player 2, with $\mathcal{S}_A$ as the initial set. We consider a polytopic *outer approximation* for the FRS, which could be computed via methods such as [30]–[32]. As our focus is not on any specific approximation method, we assume integrator dynamics for Player 2 and an avoid set $\mathcal{S}_A$ defined as

$$\mathcal{S}_A = \{x : ||x - x_2||_\infty < w\}, \tag{14}$$

where x_2 is the position of Player 2 and w is based on physical parameters such as capture range or robot size.

[1]Fixing the final time is also needed to split the trajectory into time segments so that the mixed integer formulation can be used, which is discussed later.

Using this avoid set and integrator dynamics yields a polynomial representation of the FRS. For a concrete example see Section V-B.

Let the polynomials defining the boundaries of the FRS along state space dimension i be given by $\overline{\theta}^i(t)$ and $\underline{\theta}^i(t)$, and define polynomials $\phi_j^i(t)$ for each trajectory segment j in the following manner:

$$\phi_{j,1}^i(t) \geq 0 \Leftrightarrow p_j^i < \underline{\theta}^i(t),$$
$$\phi_{j,2}^i(t) \geq 0 \Leftrightarrow p_j^i > \overline{\theta}^i(t).$$

Collecting all polynomials $\phi_{j,1}^i$ and $\phi_{j,2}^i$ for a given state space dimension i yields $2N$ polynomials. Together, they define the capture condition for a particular trajectory segment j as follows:

$$\phi_{j,1/2}^i(t) < 0, \quad \forall i, \tag{15}$$

which means that polynomials $\phi_{j,1}^i(t)$ and $\phi_{j,2}^i(t)$ for all $i \in [1,\ldots,n]$ have to be negative at a point in time in order to have Player 1 captured by Player 2. Therefore, a sufficient condition to avoid capture is that at least one of the polynomials $\phi_{j,1/2}^i(t) \geq 0$ for a given segment j

$$\phi_{j,1}^1(t) \geq 0 \vee \phi_{j,2}^1(t) \geq 0 \vee \cdots \vee \phi_{j,1}^n(t) \geq 0 \vee \phi_{j,2}^n(t) \geq 0. \tag{16}$$

IV. Reduction to MISOCP

Even with the simplifications presented in Section III, the resulting optimization is still complicated due to the avoid constraints (16) being non-convex *or*-constraints.

A. Formulating Mixed Integer Avoid Constraints

Reach-avoid games from the perspective of the Player 1 are in some sense similar to path planning with collision avoidance requirements, which has already been studied using mixed integer linear programming. The authors of [35] and [36] use this technique by discretizing the path and guaranteeing collision avoidance at points using mixed-integer linear constraints. In [37], the authors utilize SOS and mixed integer programming to require polynomial trajectory segments to lie in convex safe regions in the environment.

Similar to [37], we propose to introduce binary slack variables $z_{j,k}$ to handle the non-convexity in the *or*-constraints given by (16) and convert them into a set of mixed-integer *and*-constraints using the big-M method,

$$\begin{aligned}
\forall j \in [1,\ldots,N]: & \\
& \phi_{j,1}^1(t) + M(1 - z_{j,1}) \text{ is SOS},\\
& \phi_{j,2}^1(t) + M(1 - z_{j,2}) \text{ is SOS},\\
& \quad \ldots\\
& \phi_{j,1}^n(t) + M(1 - z_{j,2n-1}) \text{ is SOS},\\
& \phi_{j,2}^n(t) + M(1 - z_{j,2n}) \text{ is SOS},\\
& \sum_{k=1}^{2n} z_{j,k} = 1,\\
& \quad z_{j,k} \in \{0,1\}.
\end{aligned} \tag{17}$$

The value of M can be determined by considering reasonable limits of $\phi_j^i(t)$ for a given problem. Note that there are $2n$ mixed integer SOS constraints for each trajectory segment, and with N segments we end up with $2nN$ total binary variables.

B. Mixed Integer Semidefinite Program

We are now ready to write the full optimization problem, which takes on a form that can be converted to a mixed integer semidefinite program (MISDP).

$$\begin{aligned}
&\text{minimize } \sum_{k=1}^{P} \gamma_k\\
&\text{subject to}\\
&\quad A_U[p_j^i(t), \dot{p}_j^i(t), \ldots, p_j^{(d),i}(t)]^T - b_U \text{ is SOS}, \quad \forall\, i,j,\\
&\quad A_T p_N(t_f) - b_T + \gamma \geq 0,\\
&\quad \gamma_k \geq 0, \quad \forall k \in [1,\ldots,P]\\
&\forall j \in [1,\ldots,N]:\\
&\quad \phi_{j,1}^1(t) + M(1 - z_{j,1}) \text{ is SOS},\\
&\quad \phi_{j,2}^1(t) + M(1 - z_{j,2}) \text{ is SOS},\\
&\quad \ldots\\
&\quad \phi_{j,1}^n(t) + M(1 - z_{j,2n-1}) \text{ is SOS},\\
&\quad \phi_{j,2}^n(t) + M(1 - z_{j,2n}) \text{ is SOS},\\
&\quad \sum_{k=1}^{2n} z_{j,k} = 1,\\
&\quad z_{j,k} \in \{0,1\},\\
&\quad x_1(0) = x_{1,0},\\
&\quad x_2(0) = x_{2,0}.
\end{aligned} \tag{18}$$

This optimization problem can be written as a MISDP, which can be globally solved using a branch and bound algorithm. Additionally, when trajectory polynomials $p_j^i(t)$ are of order two or three ($d = 2$ or $d = 3$) the constraints reduce to rotated second-order cone constraints and the problem can be solved as a MISOCP.

C. Reduction to MISOCP

Consider writing a constraint for a general low-order polynomial $\phi(t)$ defined over the interval $[t_{j-1}, t_j]$ as

$$\phi(t) = \begin{cases} (t - t_{j-1})\sigma_1(t) + (t_j - t)\sigma_2(t) & d = 3,\\ \sigma_1(t) + (t_j - t)(t - t_{j-1})\sigma_2(t) & d = 2, \end{cases} \tag{19}$$
$$\sigma_1(t), \sigma_2(t) \text{ are SOS},$$

where if $d = 3$ both $\sigma_1(t)$ and $\sigma_2(t)$ are second-order, and if $d = 2$, $\sigma_1(t)$ is second-order and $\sigma_2(t)$ is a constant. Now the decision variables would be the coefficients of $\sigma_1(t)$ and $\sigma_2(t)$, which identify the original polynomial.

Note that we have considered a polynomial over a specific time interval $[t_{j-1}, t_j]$, which in our problem corresponds to a specific trajectory segment $p_j^i(t)$. Therefore writing a polynomial SOS constraint using (19) relaxes it from being globally non-negative to being non-negative over the appropriate time interval, which is less restrictive.

Since both $\sigma_1(t)$ and $\sigma_2(t)$ are at most second-order polynomials the constraint that they must be SOS can be written as rotated second-order cone constraints.

$$\sigma(t) = c_0 + c_1 t + c_2 t^2 = \begin{bmatrix} 1 & t \end{bmatrix} \begin{bmatrix} c_0 & \frac{c_1}{2} \\ \frac{c_1}{2} & c_2 \end{bmatrix} \begin{bmatrix} 1 \\ t \end{bmatrix},$$

so $\sigma(t)$ is SOS $\Leftrightarrow \begin{bmatrix} c_0 & \frac{c_1}{2} \\ \frac{c_1}{2} & c_2 \end{bmatrix} \succeq 0$, which is also equivalent to $c_1^2 - 4c_0c_2 \leq 0,\ c_0, c_2 \geq 0$.

Under these conditions the optimization problem (18) can be solved as a MISOCP instead of as a general MISDP. As mentioned before, piecewise low-order polynomials can still effectively represent complex trajectories, as long as there is a sufficient number of segments. Solving an MISOCP instead of an MISDP also allows us to take advantage of MOSEK's [38] mixed integer optimization methods, enabling real-time computations needed for our receding horizon approach.

V. Numerical Examples

We now consider an example reach-avoid game and demonstrate how to formulate and solve it as an MISOCP. Consider a kinematic car that is trying to reach a safety box around the origin while avoiding capture by a single integrator opponent.

A. Kinematic Car (Player 1)

The dynamics of the kinematic car are given by

$$\dot{x}_1 = v\cos(\theta_1), \quad \dot{y}_1 = v\sin(\theta_1), \quad \dot{\theta}_1 = \omega,$$

where the control variables are the speed v and turn rate ω. We use a control constraint $|v| \leq v_{1,\max}$ and a friction constraint that is translated into a constraint on $\ddot{x}$ and $\ddot{y}$. The full non-linear constraints are outer-approximated as follows:

$$\begin{aligned} -\frac{v_{1,\max}}{\sqrt{2}} &\leq \dot{x}_1 \leq \frac{v_{1,\max}}{\sqrt{2}}, & -\frac{\mu_s g}{\sqrt{2}} &\leq \ddot{x}_1 \leq \frac{\mu_s g}{\sqrt{2}}, \\ -\frac{v_{1,\max}}{\sqrt{2}} &\leq \dot{y}_1 \leq \frac{v_{1,\max}}{\sqrt{2}}, & -\frac{\mu_s g}{\sqrt{2}} &\leq \ddot{y}_1 \leq \frac{\mu_s g}{\sqrt{2}}, \end{aligned}$$

where μ_s is the coefficient of static friction.

This system is differentially flat with flat outputs $z(t) = (x(t), y(t))$; a trajectory in the output space is mapped to the corresponding control using $z(t)$ and its first two derivatives:

$$\begin{aligned} \theta_1(t) &= \tan^{-1}\left(\frac{\dot{y}_1}{\dot{x}_1}\right) \\ v(t) &= \sqrt{\dot{x}_1^2 + \dot{y}_1^2} \\ \omega(t) &= -\frac{1}{v}\sin(\theta_1)\ddot{x}_1 + \frac{1}{v}\cos(\theta_1)\ddot{y}_1 \end{aligned} \tag{20}$$

B. Single Integrator (Player 2)

We constrain control input of the single integrator opponent as follows:

$$-v_{2,\max} \leq \dot{x}_2 \leq v_{2,\max}, \qquad -v_{2,\max} \leq \dot{y}_2 \leq v_{2,\max}.$$

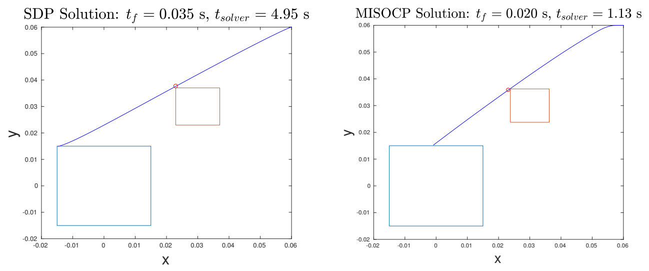

Fig. 1: Comparison between the baseline SDP solution method (left) and the MISOCP solution method (right). The MISOCP method reduces computation time and improves solution quality.

For an avoid set defined by (14) the boundaries of the FRS can be expressed as

$$\begin{aligned} \overline{x}_2 &= x_2(0) + w + v_{2,\max}t, & \underline{x}_2 &= x_2(0) - w - v_{2,\max}t, \\ \overline{y}_2 &= y_2(0) + w + v_{2,\max}t, & \underline{y}_2 &= y_2(0) - w - v_{2,\max}t. \end{aligned} \tag{21}$$

C. Baseline Solution Method (SDP)

As a performance baseline for the MISOCP method we look to solve the problem in its SDP form (before the addition of integer variables). Recall that the integer variables were introduced to handle the non-convex *or*-constraints given by (16). An alternative solution method is to utilize the fact that we can write implications of the form $x(t) \in B \Rightarrow x(t) \in A$ using (9).

In the SDP formulation, we also consider a single higher-order polynomial ($N = 1$, $d = 6$) for the trajectory rather than piecewise low-order polynomials. Therefore, in our problem the avoid constraints indicate that if all $\phi^i_{1/2}(t) < 0$ (from (16)) except for one, which we can call $\phi_0(t)$, then it is implied that $\phi_0(t) \geq 0$. Further, we need to write this implication $2n$ times, considering each $\phi^i_{1/2}(t)$ to be $\phi_0(t)$. Each of these implications are written in the form (9):

$$\phi_0(t) - \sum_r L_r(t)\phi_r(t) \text{ is SOS}, \quad L_r(t) \text{ is SOS}, \quad \forall r, \tag{22}$$

where the index r represents each other $\phi^i_{1/2}(t)$ that is not $\phi_0(t)$. Note that when writing the avoid constraints in this form, bilinear terms are introduced. The problem is then written as an SDP with bilinear terms and solved with an iterative method as mentioned in Section II-B.

D. Performance Comparison

We now compare the computation time and solution quality between the baseline SDP method and the proposed MISOCP method described in Section IV. "Solution quality" refers to the final time t_f determined by the solution. The total MOSEK solver time is given as t_{solver}. In this case the SDP method required about 60 iterations to converge. Note that the red box in the figures presented show the FRS at the time when the car (the red circle) is at the location indicated.

TABLE I: Results Comparison

	SDP	MISOCP
t_f (s)	0.035	0.02
t_{solver} (s)	4.95	1.13

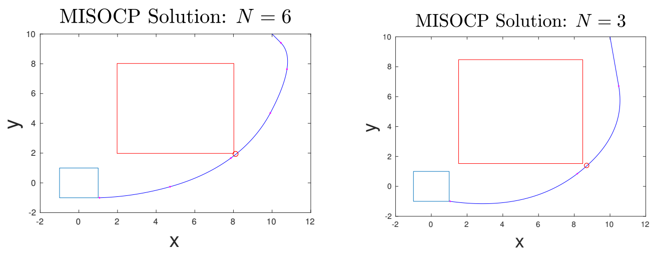

Fig. 2: Comparison showing the effect of number of trajectory segments on solution. The path when $N = 3$ corresponds to a larger t_f than when $N = 6$.

As can be seen in Figure 1, and in Table I the MISOCP solution yields both a faster trajectory as well as a lower computation time. It is also worth noting that the scale of this problem is quite small (final time on the order of milliseconds). We had difficulty finding solutions when using the SDP method, except for small time-scale problems such as shown here. However, the MISOCP method had no difficulties in finding solutions (when they exist). This is likely due to the iterative nature of solving the bilinear SDP problem, as well as additional flexibility afforded through the use of piecewise polynomials in the MISOCP method.

E. MISOCP Performance

Three examples with varying difficulty are explored to demonstrate performance of the MISOCP method. In Case 1, Player 2 starts far enough away from both Player 1 and the goal region that it has no effect on the path of Player 1, and the optimal solution is a straight line. In this simple example the MISOCP does in fact recover this straight line path. In Case 2 and Case 3, Player 2 begins directly between Player 1 and the goal region. In Case 1 and Case 2 the problem parameters are $v_{1,\max} = 40$, $\mu_s g = 100$, $v_{2,\max} = 2$. In Case 3, the max velocity of Player 2 is increased to $v_{2,\max} = 6$. All experiments are done on a Dell XPS laptop with a 2.5 GHz Intel Core i5 processor and 8GB of memory.

Table II shows the results for these problems. We see the computation times vary depending on the difficulty of the problem. In each of these problems the number of trajectory segments was $N = 6$.

TABLE II: MISOCP Performance

	Case 1	Case 2	Case 3
t_f (s)	0.53	0.59	0.775
t_{solver} (s)	0.03	0.37	0.83

The number of trajectory segments used also has an affect on computation time and solution quality. Figure 2 shows the resulting trajectory after solving the MISOCP for Case 3 with $N = 6$ and when the number of trajectory segments was cut in half. We see the trajectory is less direct since the time to get to the goal region increases from $t_f = 0.775$ s to $t_f = 0.915$ s. However with less complexity the solver time decreased from $t_{\text{solver}} = 0.83$ s to $t_{\text{solver}} = 0.34$ s.

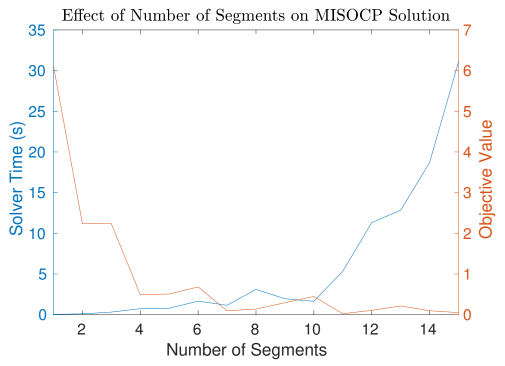

Fig. 3: Solutions to a reach-avoid problem with varying number of trajectory segments. A larger mixed-integer program results in increased computation time and better solution quality.

F. Effect of Segmenting the Trajectory

Intuitively, changing the number of segments should have an impact on the solution quality and computation time. To quantify these relationships, consider a problem similar to Section V-E. However now we choose a final time such that the trajectory cannot easily reach the goal region so that the magnitude of the slack variables γ can indicate the effects of changing N. Figure 3 shows the effect of changing the number of segments on the solver time and objective value. In general we see that adding segments gives better solutions, at the expense of solver time.

VI. Hardware Experiments

A. Receding Horizon Scheme

A receding horizon scheme provides the ability to react to new information. Our receding horizon scheme solves an MISOCP, and then executes the first Δt portion of the solution while solving a new problem. Current position information of Player 2 is updated at each iteration.

Since the optimal final time is not known *a priori* we consider a potentially shorter time horizon. This means that each open-loop plan may not necessarily find a winning strategy, and so $\sum_{k=1}^{P} \gamma_k > 0$. When an open-loop plan is found such that all slack variables are zero, a winning solution has been found and no more planning is required.

One implementation detail is that during each time interval, the MISOCP being solved (the solution is a plan for the next time interval) requires the positions of both Player 1 and 2 at the beginning of the *next* interval. Player 1's position will be the end position of the current interval's plan, but Player 2's current path is unknown. To handle this we add a buffer to the initial set of Player 2's FRS to compensate for all possible motions during the current interval Δt. For the single integrator case from Section V-B, the expressions for the boundary of the buffered FRS is given by (21), with t replaced by $t+\Delta t$, where Δt is the execution interval length.

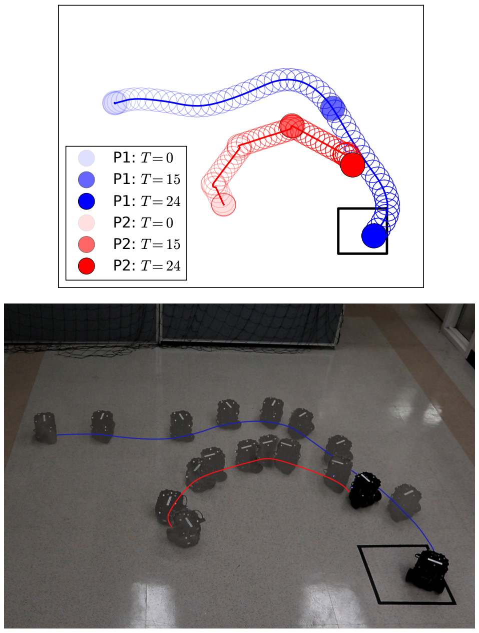

Fig. 4: An example experiment with Player 1 (blue) using the receding horizon method and Player 2 (red) being human-controlled. The planning horizon is $\Delta t = 1$ s, $v_{1,\text{max}} = 0.2$ m/s, and $v_{2,\text{max}} = 0.1$ m/s. Both players had no initial velocity. The trajectory plot includes some time snapshots of the player's locations, denoted with solid color circles.

We now demonstrate the receding horizon scheme with hardware experiments. Our test setup utilizes two Turtlebot 3 platforms with position information from a Vicon system. Player 1's paths are computed off-board, and Player 2 is human-controlled. The person controlling Player 2 was given time to practice manuevering, but was unfamiliar with Player 1's solution methodology. Player 2's maneuvers were limited to constant forward/backward velocity and constant angular velocity motions. Several experiments were performed while varying the capabilities of the two Players by changing their maximum velocities. In all of the experiments the planning horizon was $\Delta t = 1$ s and $N = 4$.

Figure 4 shows an experiment where the autonomous robot was fast enough to be able to get around the human controlled robot relatively easily. Figure 5 shows an experiment where the max velocity of the autonomous robot is decreased such that the open-loop MISOCP method is not able to initially find a winning path, but is able to eventually succeed. The latter experiment demonstrates the potential of our receding horizon strategy in situations in which the game may not be winnable until more observations about other agents are made; such situations frequently occur in practical applications such as autonomous driving. Videos from these experiments can be found at: https://www.youtube.com/playlist?list=PL8-2mtIlFIJpmqQYN2nwMjEoTWwrXpXWS

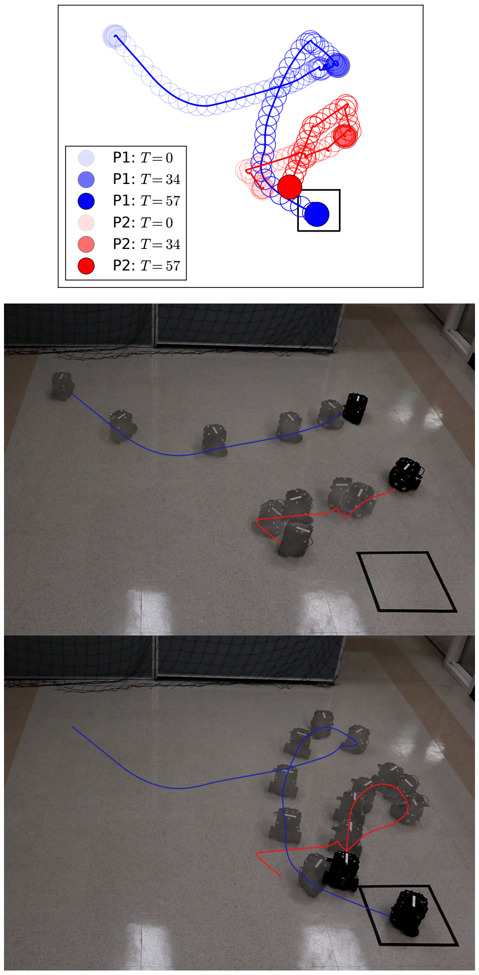

Fig. 5: Player 1 (blue) uses the receding horizon method and Player 2 (red) is human-controlled. The planning horizon is $\Delta t = 1$ s, $v_{1,\text{max}} = 0.18$ m/s, and $v_{2,\text{max}} = 0.1$ m/s. Both players had no initial velocity. Player 2's speed advantage keeps Player 1 from winning, until a mistake is made by the human-controlled robot. This example demonstrates the advantage of using a receding horizon strategy. The time-lapse of the experiment is split into two images for better clarity.

VII. Conclusions and Future Work

This paper presents a novel receding horizon strategy for reach-avoid games. By representing Player 1 trajectories as piecewise low-order polynomials, we formulated an MISOCP that significantly reduces computation time compared to formulations involving sums-of-squares constraints. This allows open-loop trajectories based on the current two-player joint state to be computed for Player 1 in real-time, enabling us to take advantage of both the computational simplicity of open-loop formulations and the ability to react to new information in closed-loop formulations. In addition, our method allows generalized problem setups and is applicable whenever the dynamics of Player 1 are differentially flat. Our numerical simulations show the benefits the MISOCP over a general SDP, and provide some example MISOCP solutions that demonstrate the characteristics of our method. Finally, we demonstrated the ability to use the approach in a receding horizon planner on a robotic platform to win a game against a human-controlled robot.

Immediate future work includes extending our approach to multi-player systems, considering a more general class of dynamics for Player 2, perhaps using the work presented in [30]–[32], and comparing the performance of our receding horizon strategy (using single integrators) to the optimal closed-loop strategies given in [12].

References

[1] R. Spica, D. Falanga, E. Cristofalo, E. Montijano, D. Scaramuzza, and M. Schwager. (2018, Jan.) A Real-Time Game Theoretic Planner for Autonomous Two-Player Drone Racing. Available at http://arxiv.org/abs/1801.02302.

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

[3] J. R. Marden and J. S. Shamma, "Game Theory and Control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. annurev–control–060 117–105 102, May 2018.

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

[5] G. Chasparis and J. Shamma, "Linear-programming-based multi-vehicle path planning with adversaries," in *American Control Conference*. IEEE, Jun. 2005, pp. 1072–1077.

[6] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "FaSTrack: a modular framework for fast and guaranteed safe motion planning," in *Proc. IEEE Conf. on Decision and Control*, 2017, submitted.

[7] A. Bressan, "Noncooperative differential games," *Milan Journal of Mathematics*, vol. 79, no. 2, pp. 357–427, 2011.

[8] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Hybrid Systems: Computation and Control*, 2015.

[9] H. Huang, J. Ding, W. Zhang, and C. J. Tomlin, "Automation-assisted capture-the-flag: A differential game approach," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 3, pp. 1014–1028, 2015.

[10] M. Chen, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Robust sequential path planning under disturbances and adversarial intruder," *IEEE Transactions on Control Systems Technology*, 2018, in press.

[11] A. Pierson, Z. Wang, and M. Schwager, "Intercepting Rogue Robots: An Algorithm for Capturing Multiple Evaders With Multiple Pursuers," *IEEE Robotics and Automation Letters*, vol. 2, pp. 530–537, Apr. 2017.

[12] H. Huang, J. Ding, W. Zhang, and C. J. Tomlin, "A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag," in *IEEE Conf. on Robotics and Automation*, 2011.

[13] M. Chen, Z. Zhou, and C. J. Tomlin, "Multiplayer Reach-Avoid Games via Pairwise Outcomes," *IEEE Transactions on Automatic Control*, vol. 62, no. 3, pp. 1451–1457, Mar. 2017.

[14] Z. Zhou, R. Takei, H. Huang, and C. J. Tomlin, "A general, open-loop formulation for reach-avoid games," in *IEEE Conf. on Decision and Control*, 2012.

[15] M. Chen, Z. Zhou, and C. J. Tomlin, "A path defense approach to the multiplayer reach-avoid game," in *IEEE Conf. on Decision and Control*, 2014.

[16] Z. Zhou, W. Zhang, J. Ding, H. Huang, D. M. Stipanović, and C. J. Tomlin, "Cooperative pursuit with Voronoi partitions," *Automatica*, vol. 72, pp. 64–72, 2016.

[17] M. Kothari, J. G. Manathara, and I. Postlethwaite, "Cooperative Multiple Pursuers against a Single Evader," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 3-4, pp. 551–567, Jun. 2017.

[18] B. Xue, A. Easwaran, N.-J. Cho, and M. Franzle, "Reach-Avoid Verification for Nonlinear Systems Based on Boundary Analysis," *IEEE Transactions on Automatic Control*, vol. 62, no. 7, pp. 3518–3523, Jul. 2017.

[19] W. Zha, J. Chen, Z. Peng, and D. Gu, "Construction of Barrier in a Fishing Game With Point Capture," *IEEE Transactions on Cybernetics*, vol. 47, no. 6, pp. 1409–1422, Jun. 2017.

[20] R. Yan, Z. Shi, and Y. Zhong, "Escape-avoid games with multiple defenders along a fixed circular orbit," in *IEEE International Conference on Control & Automation*, 2017.

[21] W. Li, "Escape Analysis on the Confinement-Escape Problem of a Defender Against an Evader Escaping From a Circular Region," *IEEE Transactions on Cybernetics*, vol. 46, no. 9, pp. 2166–2172, Sep. 2016.

[22] ——, "A Dynamics Perspective of Pursuit-Evasion: Capturing and Escaping When the Pursuer Runs Faster Than the Agile Evader," *IEEE Transactions on Automatic Control*, vol. 62, no. 1, pp. 451–457, Jan. 2017.

[23] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[24] J. van den Berg, Ming Lin, and D. Manocha, "Reciprocal Velocity Obstacles for real-time multi-agent navigation," in *IEEE Conf. on Robotics and Automation*, 2008.

[25] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[26] 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, sep 2015. [Online]. Available: http://doi.wiley.com/10.1002/oca.2123

[27] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *IEEE Conf. on Robotics and Automation*. IEEE, May 2016, pp. 1433–1440. [Online]. Available: http://ieeexplore.ieee.org/document/7487277/

[28] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," in *IEEE Conf. on Robotics and Automation*. IEEE, May 2017, pp. 1714–1721. [Online]. Available: http://ieeexplore.ieee.org/document/7989202/

[29] R. M. Murray, "Optimization-based control," *California Institute of Technology, CA*, 2009.

[30] M. Althoff and B. H. Krogh, "Reachability analysis of nonlinear differential-algebraic systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 2, pp. 371–383, 2014.

[31] J. H. Gillula, S. Kaynama, and C. J. Tomlin, "Sampling-based approximation of the viability kernel for high-dimensional linear sampled-data systems," in *Hybrid Systems: Computation and Control*, 2014.

[32] T. Dreossi, T. Dang, and C. Piazza, "Parallelotope bundles for polynomial reachability," in *Hybrid Systems: Computation and Control*, 2016.

[33] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *Int. Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[34] P. A. Parrilo, "Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization," Ph.D. dissertation, Massachusetts Institute of Technology, 2000.

[35] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *European Control Conference*, 2001.

[36] H. Ding, G. Reißig, D. Groß, and O. Stursberg, "Mixed-integer programming for optimal path planning of robotic manipulators," in *Int. Conf. on Automation Science and Engineering*, 2011.

[37] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," in *Proc. IEEE Conf. on Robotics and Automation*, 2015.

[38] Mosek APS. (2010) The MOSEK optimization software. Available at http://www.mosek.com.