Reactive Symbolic Planning and Control in Dynamic Adversarial Environments

Laya Shamgah, Tadewos G. Tadewos, Ali Karimoddini, Albert C. Esterline

***Abstract*— Satisfaction of both safety and reachability in dynamic environments is a complex problem due to the competitive interactions of the system with its environments. The presence of adversarial objects, particularly when little information about their dynamics and intentions is available, makes the problem more challenging. This paper addresses path planning and control of autonomous vehicles involved in a dynamic adversarial reach-avoid scenario. In the studied scenario, there are two non-cooperative vehicles with the competitive objectives "reaching a target and avoiding the other vehicle" for one of them, called attacker, and "protecting the target and capturing the opponent vehicle" for the other one, called defender. In the proposed solution, first, a discrete version of the problem is formally captured by Linear Temporal Logic formulas. Using a temporal game structure and μ-calculus formulas, two algorithms are developed to find discrete strategies that guarantee both safety and reachability. Finally, a novel correct-by-design hybrid controller is proposed to generate smooth control signals that preserve the satisfiability of these requirements of interest.**

Keywords: Reach-avoid; Symbolic planning and control; Adversarial environment; Temporal logic; Reactive synthesis

I. INTRODUCTION

Classical motion planning techniques are developed to design trajectories from a starting point to a specific destination. With technological advances in control, computation, and communication tools, it has become possible to consider more complex environments and scenarios. For example, static obstacles and no-fly zones can now be avoided by the vehicles under control. The combination of these two objectives (reaching the desired point and avoiding static obstacles) has led to the emergence of new path planning scenarios called *reach-avoid* problems [1], [2]. For these sophisticated scenarios, more advanced control techniques are required to achieve extra levels of reliability, robustness, and efficiency. An interesting recent trend in Robotics society is to employ and connect *Temporal Logics* [3], [4], and *Hybrid Control Systems* [5]–[8], together with their fast developing tools. *Linear Temporal Logic* (LTL) formulas [9]–[11] makes it possible to formally describe requirements, such as safety and reachability, in different scenarios (e.g., persistent monitoring [12], search and rescue [13], etc.). Combined with hybrid control techniques, symbolic control approaches [14], [15] generate correct-by-design, computationally efficient solutions for robot motion planning and control problems, particularly when the environment does not change [16]–[18]. However, in practice, the environment is dynamic. Addressing a problem where there exist non-cooperative agents in the environment is beyond the scope of the developed techniques for static environments.

This paper, therefore, targets for planning and control of autonomous vehicles being driven in dynamic adversarial environments where there exit moving objects with intercepting and adversary intents, with many practical applications, for example, a flight collision avoidance [19], or an air-to-air combat [20]. Specifically, we consider a dynamic reach-avoid scenario involving two competing vehicles: the attacking vehicle, which aims to reach a target region while avoiding the other vehicle, and the defending vehicle, which aims to capture the attacking vehicle to protect the target. The existing methods either formulate the dynamic reach-avoid problem as a pursuit-evasion game [21]–[24], or use probabilistic formulations [25]–[27] to minimize a cost function that is defined based on the probability distribution of the moves of dynamic obstacles. More recently improved techniques formulate the reach-avoid problem as differential zero-sum games [28], [29], and use Hamilton-Jacobi reachability analysis to obtain winning strategies against the behavior of the defenders. Although these numerical methods are very successful in solving static reach-avoid problems, they suffer from several limitations in dealing with the dynamic version of the problem. These methods provide an open-loop solution according to the initial positions of the vehicles, without incorporating real-time information feedbacked from the opponent vehicle. This will result in conservative solutions in the absence of information about the opponent's actions.

This paper addresses the dynamic reach-avoid problem by developing a correct-by-design controller for the vehicle under control to reactively respond to adversarial actions of the opponent vehicle, while also avoiding static obstacles and no-fly zones. For this purpose, a hybrid symbolic planning technique is developed over a rectangular partitioned environment. We, then, effectively capture the assumptions about the environment and requirements of the vehicle under control using LTL formulas in the form of General Reactivity(1), GR(1), [30]–[32]. This allows us to handle the reactive nature of the reach-avoid problem while synthesizing a discrete supervisor that is guaranteed to win the game. Our focus in this paper is to design a controller for the attacker. However, the solution can be similarly synthesized for the defender in a symmetric way if the intention is to control the defender. Compared to our preliminary results in [33] and [34], in this paper, we have calculated winning initial regions from which the attacker can start the mission and win the reach-avoid game. For this purpose, the dynamic reach-avoid problem is formulated as a two-player game. Appropriate operators are introduced to find the safe and winning solution of the game as a fixed-point of these operators through an iterative process. It is proven that the solution of the formulated game exists. An algorithm is provided to implement the proposed technique that is proved to return the solution within a finite number of iterations. After finding the winning initial regions, an algorithmic approach is provided to extract the winning strategies for an initial position through which the attacker can reach the target before being captured by the defender.

Finally, a novel hybrid controller is designed to generate control signals for executing the winning strategies by driving the attacker to win the reach-avoid game while respecting the dynamics of the system. In this paper, we consider the dynamics of the attacking autonomous vehicle modeled with a class of multi-affine nonlinear systems of the form $\dot{x} = f(x) + Bu$ which describes well-known models like Euler, Volterra, Lotka-Volterra equations, attitude and velocity control systems for autonomous vehicles, such as aircraft and underwater vehicles. Commonly, the continuous signals generated from symbolic controllers over partitioned spaces are discontinuous, as it is observed in [35], which may cause discomfort and negatively impact the actuators and reduce the reliability of the system [36], [37]. However, a salient feature of the proposed hybrid controller is the smoothness of the generated control signals, which is achieved by the proper control of the vector field when the system moves over the edges of partitions while guaranteeing the execution of the high-level discrete commands.

The rest of the paper is organized as follows. Section II formulates the dynamic adversarial reach-avoid problem. Section III provides the formal descriptions of the reach-avoid problem and captures the assumptions about the environment and requirements of the vehicle under control using LTL formulas. In Section IV, the reach-avoid problem is formulated as a two-player game, based on which initial winning regions are calculated. Section V synthesizes the high-level winning strategies that can lead the vehicle under control to win the game for any action of the opponent vehicle. Section VI constructs a hybrid controller, which generates smooth control signals to drive the vehicle under control to execute the winning strategies. Section VII provides the simulation results, and Section VIII concludes the paper.

II. Problem Description

A dynamic reach-avoid scenario consists of two vehicles: an attacking vehicle, which aims to reach a target while avoiding collision with another vehicle, and a defending vehicle, which aims to prevent the attacking vehicle from reaching the target, by capturing it. A general schematic for this game is illustrated in Fig. 1.

In this paper, we address the problem of path planning and control for the attacking vehicle in a dynamic reach-avoid scenario. However, the results can be similarly extended for the defending vehicle in a symmetric way.

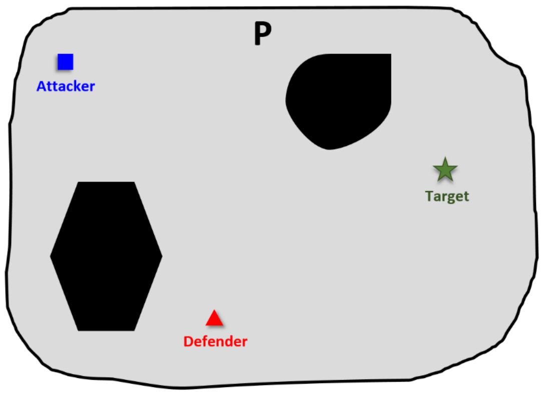


Fig. 1. A general schematic of a dynamic reach-avoid problem.

A. Operation Region

We assume that the vehicles operate within a 2-D bounded region $\Omega \subset \mathbb{R}^2$ and never leave this region. In the dynamic adversarial reach-avoid problem, the attacker tries to reach the target while keeping a safe distance from the defender.

Definition 1: The *minimum safe distance* between the attacker's position, x_a, and the defender's position, x_d, is δ_D. If $\|x_d - x_a\| < \delta_D$, it is assumed that the attacker is captured by the defender.

Definition 2: The *minimum desired distance* between the attacker's position, x_a, and the target's position, x_t, is δ_T. If the attacker approaches the target such that $\|x_a - x_t\| < \delta_T$, it is assumed that the attacker has captured the target.

Consider the rectangle $P = \{x \in \mathbb{R}^2 | a_1 \le x_1 \le b_1, a_2 \le x_2 \le b_2\}$ be the minimum rectangle which embraces the region Ω. To manage the complexity of the reach-avoid problem, we employ symbolic motion planning techniques by rectangular partitioning of P. We set the length of the partitions as:

$$\delta = \min\{\frac{\delta_D}{\sqrt{2}}, \frac{\delta_T}{\sqrt{2}}\} \tag{1}$$

where δ_D and δ_T are defined in Definitions 1 and 2.

This will partition P into nm disjoint squares P_{ij}, where $P_{ij} = \{x \in P | a_1 + (j-1)\delta \le x_1 \le a_1 + j\delta \text{ and } b_2 - i\delta \le x_2 \le b_2 - (i-1)\delta\}$, where (a_1, b_2) are the coordinates of the top-left corner of P.

B. Discrete Information over the Partitioned Space

We assume that the positions of both vehicles are known during the game. We define the following sets of Boolean propositions:

$$\begin{aligned} \mathcal{A} &= \{a_{ij}\} \quad , \quad i \in \{1, \dots, n\}; j \in \{1, \dots, m\} \\ \mathcal{D} &= \{d_{ij}\} \quad , \quad i \in \{1, \dots, n\}; j \in \{1, \dots, m\} \end{aligned} \tag{2}$$

The sets $\mathcal{A}$ and $\mathcal{D}$ include propositions about the positions of the attacker and the defender, respectively. At each time, only one of the elements of these sets is $True$ and the rest are $False$. The indices of these true elements are equivalent to the partitions where the vehicles are located. For example, $a_{12} = True$ means that the attacker is in partition P_{12}.

We also define a proposition e_{ad} which is $True$ whenever the distance of the attacker and the defender is less than δ_D, i.e., $\|x_a - x_d\| < \delta_D$. Similarly, we define e_{at} as a proposition, which is $True$ if the distance of the attacker and the target is less than δ_T, i.e., $\|x_a - x_t\| < \delta_T$.

C. Attacker's Continuous Dynamics

Here, we assume that the attacking vehicle's model is represented by a class of nonlinear systems, called *multi-affine*, which can be written as:

$$\dot{x}(t) = f(x(t)) + Bu(t) \tag{3}$$

where $x(t) \in \mathbb{R}^n$ describes the position of the system, $u(t) \in U \subset \mathbb{R}^q$ is the control input, $B \in \mathbb{R}^{n\times q}$ is a constant matrix, and the function $f(x)$ is affine in each of elements of x as presented in the following:

$$f(x) = f(x_1, ..., x_n) = \sum_i c_i \prod_{j=1,\cdots,n} x_j^{b_{ij}} \tag{4}$$

where $b_{ij} \in \{0, 1\}$ and $c_i \in \mathbb{R}^n$.

We also assume that the velocities of both vehicles are bounded by U_m.

Planning for the attacker in a dynamic reach-avoid scenario, the aim is to reach the target before being captured by the defender. The first question is whether this goal is feasible, or under what conditions the attacker will win this adversarial game. This is described in Problem 1:

Problem 1: Consider a reach-avoid scenario with the partitioned region P, a defending vehicle with the initial position $x_d(0) \in P_{d_0}$, and an attacking vehicle with the continuous dynamics described in (3). The attacker aims to reach a target at position x_t before being captured by the defender. Obtain the set of guaranteed winning initial positions W^a_{init} from which the attacker can eventually win, i.e. $\|x_a - x_t\| < \delta_T$, before being captured by the defender, i.e., always $\|x_a - x_d\| \geq \delta_D$.

If a solution for Problem 1 exists, the next step is to control the attacker and drive it to win the reach-avoid game:

Problem 2: Assume the initial position of the attacker is $x_a(0) \in W^a_{init}$. Design a controller to generate a trajectory $x_a(t) \in P$ that satisfies the attacker's objective which is to reach the target, i.e., $\exists t_w \geq 0$ such that $||x_a(t_w) - x_t(t_w)|| < \delta_T$, while avoiding the defender with initial position $x_d(0)$, i.e $\forall 0 \leq t \leq t_w, ||x_a - x_d|| \geq \delta_D$.

III. Formal Representation of the Dynamic Adversarial Reach-avoid using via temporal logic

In this section, we use Linear Temporal Logic (LTL) to formally describe the dynamic adversarial reach-avoid problem. This allows us later to check the feasibility of a winning strategy for the attacker, as stated in Problem 1, and to synthesize a hybrid controller to address Problem 2.

A. Linear Temporal Logic

a) LTL Syntax: Linear temporal logic (LTL) formulas [10] are constructed over a finite set of atomic propositions, Σ, using the standard Boolean operators, *negation* ($\neg$), *disjunction* ($\bigvee$), *conjunction* ($\bigwedge$), and the temporal operators, *next* ($\bigcirc$), *until* ($\mathcal{U}$), *eventually* ($\Diamond$), and *always* ($\Box$).

b) LTL Semantics: Consider an ordered sequence $\sigma = \sigma_0\sigma_1 \dots$ over propositions in Σ, where σ_i is the set of propositions at position i. We say $\sigma, i \vDash \varphi$ if σ satisfies a formula φ at position i.

Table I presents syntax and semantics of LTL formulas.

c) Interpretation of Boolean propositions: By interpretation we mean assigning a truth value to a Boolean proposition. The interpretation of a boolean proposition p is either $True$ (or equivalently 1), or $False$ (or equivalently 0). Accordingly, we define Π_Σ as the set of all possible interpretations of the variables in Σ.

B. LTL Specification for the Dynamic Reach-avoid Scenario

To capture the interactions of the attacker with environment (including the defender actions), we use a special class of LTL formulas called General Reactivity(1), or simply GR(1), $\varphi = \varphi_e \rightarrow \varphi_s$, where φ_e contains all assumptions about the environment and φ_s represents the assumptions on the system and its desired behaviors. The formulas φ_e and φ_s are formed by the conjunction of some sub-formulas in all three forms of B, $\Box B$, and $\Box\Diamond B$, where B could be a Boolean or temporal formula.

Formulating the reach-avoid problem within GR(1) formalism, we consider the attacker, which we are going to control, as the system and the defender as environment. This can be captured by a GR(1) formula:

$$\varphi = \varphi_d \rightarrow \varphi_a \tag{5}$$

where φ_d and φ_a can be represented as:

$$\varphi_\gamma = \varphi^\gamma_{init} \wedge \varphi^\gamma_{sing} \wedge \varphi^\gamma_{term} \wedge \varphi^\gamma_{rul} \wedge \varphi^\gamma_{obs} \wedge \varphi^\gamma_{obj} \tag{6}$$

where $\gamma \in \{a, d\}$ includes superscripts a and d for the attacker and the defender, respectively. The sub-formulas φ^γ_{init}, φ^γ_{sing}, φ^γ_{term}, φ^γ_{rul}, φ^γ_{obs}, and φ^γ_{obj} capture the *initial* condition, the *singularity* condition, the *termination* condition, the *transition rules*, the *obstacle avoidance* requirements, and the *objective*s of the vehicles, respectively. These sub-formulas are described in the following:

1) Initial condition: Initially (based on Problem 1), the defender is in P_{d_0}, with $d_0 = (i_{d_0}, j_{d_0})$. Therefore, φ^d_{init} can be described as:

$$\varphi^d_{init} = d_{i_{d_0} j_{d_0}} \wedge \neg e_{ad} \tag{7}$$

The attacker, however, could start from any of the partitions as described by φ^a_{init}:

$$\varphi^a_{init} = \bigvee_{(i,j)\in M} a_{ij} \tag{8}$$

where $M = \{(i, j)|i \in \{1, \dots, m\}, j \in \{1, \dots, n\}\}$ is the set of all indices of partitions in P. Later, we will find the initial positions from which the attacker can win the game.

2) Singularity requirement: Each vehicle can physically be only in one partition at each step. Hence, only one of the

TABLE I
LINEAR TEMPORAL LOGIC (LTL) SYNTAX AND SEMANTICS

Syntaxes	**Description**	**Semantics**
p	Boolean proposition	$\sigma, i \vDash p \iff p \in \sigma(i)$, for $p \in \Sigma$,
$\neg\varphi$	Boolean operator (Negation)	$\sigma, i \vDash \neg\varphi \iff \sigma, i \nvDash \varphi$
$\varphi \vee \psi$	Boolean operator (Disjunction)	$\sigma, i \vDash \varphi \vee \psi \iff \sigma, i \vDash \varphi$ or $\sigma, i \vDash \psi$
$\varphi \rightarrow \psi$	Boolean operator (Implication)	$\sigma, i \vDash \varphi \rightarrow \psi \iff \sigma, i \vDash \neg\varphi$ or $\sigma, i \vDash \psi$
$\varphi \wedge \psi$	Boolean operator (Conjunction)	$\sigma, i \vDash \varphi \wedge \psi \iff \sigma, i \vDash \varphi$ and $\sigma, i \vDash \psi$
$\bigcirc\varphi$	Temporal operator (Next)	$\sigma, i \vDash \bigcirc\varphi \iff \sigma, i+1 \vDash \varphi$
$\varphi\, \mathcal{U}\, \psi$	Temporal operator (Until)	$\sigma, i \vDash \varphi\, \mathcal{U}\, \psi \iff$ there exists $k \geq i$ s.t. $\sigma, k \vDash \psi$ and $\sigma, j \vDash \varphi$ for all $i \leq j < k$
$\Diamond\varphi$	Temporal operator (Eventually)	$\sigma, i \vDash \Diamond\varphi \iff$ there exists $k \geq i$ s.t. $\sigma, k \vDash \varphi$
$\Box\varphi$	Temporal operator (Always)	$\sigma, i \vDash \Box\varphi \iff$ for all $k \geq i$, $\sigma, k \vDash \varphi$

propositions a_{ij} can be $True$ at each time. The same is true for d_{ij}. We call this property the *Singularity Requirement* which can be described as:

$$\varphi^{\gamma}_{sing} = \Box[\bigvee_{(i,j)\in M} (\gamma_{ij} \wedge (\bigwedge_{(\ell,k)\in M-\{(i,j)\}} \neg\gamma_{\ell,k}))] \qquad (9)$$

where $\gamma \in \{a, d\}$.

3) Termination requirement: Once either of the vehicles achieves its goal (the attacker reaches the target and e_{at} becomes $True$, or the defender captures the attacker and e_{ad} becomes $True$) the game terminates. In this case, both players are required to stay at their last position.

The termination requirement for the attacker and the defender $\gamma \in \{a, d\}$, can be described as:

$$\varphi^{\gamma}_{term} = \Box[(e_{at} \vee e_{ad}) \wedge (\bigwedge_{(i,j)\in M} \gamma_{ij} \rightarrow \bigcirc\gamma_{ij})] \qquad (10)$$

4) Transition rules:

a) Defender's transition rules: First, we define the *next decision* set for the defender $POST_D(d_{ij})$ for any d_{ij}, $i = 1, \cdots, m, j = 1, \cdots, n$, as:

$$POST_D(d_{ij}) = \{d_{\ell k} | (\ell, k) \in (N'_d(P_{ij}) - Obs)\} \qquad (11)$$

where $N'_d(P_{ij})$ contains all indices of the diagonal, vertical, and horizontal neighbor partitions of P_{ij} including itself, and Obs includes all indices of obstacle regions.

Since from the perspective of the attacker, there is no control on the transitions of the defender, we only assume that the defender transits to one of its (diagonal, vertical, or horizontal) adjacent partitions, which is not an obstacle or target partition. The defender may also stay in its current partition. These assumptions capture all possible behaviors of the defender and can be described as:

$$\varphi^{d}_{rul} = \Box[\bigwedge_{(i,j)\in M} (d_{ij} \rightarrow \bigvee_{(\ell,k)\in POST_D(d_{ij})} \bigcirc d_{\ell k}) \wedge \neg d_{i_t j_t}] \qquad (12)$$

where (i_t, j_t) is the index of the region where the target is located.

The LTL formula in (12) indicates the strategies for the evolution of defender propositions and basically states that if the defender is in partition P_{ij}, then for the next step, it will either move to one of the adjacent regions which is not obstacle or target, or stay in its current partition. Therefore we consider all possible moves of the defender, as our intention is not to assume any limitation on the transitions of the defender.

b) Attacker's transition rules: Here, we propose an algorithm to derive the attacker's transition rules to optimally make decision and move over the partitioned space. For this purpose, first, similar to the defender, we define the *next decision* set for the attacker $POST_A(a_{ij})$ for any a_{ij}, $i = 1, \cdots, m, j = 1, \cdots, n$, as:

$$POST_A(a_{ij}) = \{a_{i'j'} | (i', j') \in (N'_a(P_{ij}) - Obs - (i, j))\} \qquad (13)$$

where $N'_a(P_{ij})$ contains all indices of the vertical and horizontal neighbor regions of P_{ij}, excluding the obstacle partitions.

Then, we construct a finite two-player zero-sum game in a matrix form based on the current position of the attacker, a_{ij}, and the defender, $d_{\ell k}$. Let $a_{i'j'}$ and $d_{\ell' k'}$ to be two possible next decisions of the attacker and the defender, respectively. The vehicles share the same objective function, L, defined as:

$$L(a_{i'j'}, d_{\ell'k'}) = \begin{cases} \infty, \text{if } (i', j') = (\ell', k') \\ 0, \text{if } (i', j') = (i_t, j_t) \\ \alpha\|(i', j') - (i_t, j_t)\| + \dfrac{\beta}{\|(i', j') - (\ell', k')\|}, \text{o.w.} \end{cases} \qquad (14)$$

where (i_t, j_t) is the index of the region where the target is located. In this game, the attacker is the minimizer player while the defender is the maximizer. By using this game configuration, the attacker tries to minimize the cost function in (14) by maximizing the distance between two vehicles, $\|(i', j') - (\ell', k')\|$, and by minimizing its distance from the target $\|(i', j') - (i_t, j_t)\|$. If the attacker and the defender are in the same region, we will have $L = \infty$, meaning that the attacker looses the game, and if the attacker reaches the target, we will have $L = 0$, meaning that the attacker wins the game. Either way, the game is over and the players should remain in their last position. The parameters $\alpha, \beta \in \mathbb{R}$ are weight factors.

Lemma 1: [38] For any finite two-player zero-sum game in a matrix form, there is at least one security decision.

Lemma 1 assures that a zero-sum game for different configurations (positions of the vehicles) can provide at least one security decision for the attacker to make. For example, consider the current status of the game as shown in Fig. 2, where the attacker is in P_{ij}, and the defender is in P_{lk}. Since the attacker is under our control, we drive it to move constantly and leave its current region by making a different decision, until it reaches the target. Also, we assume that the attacker exits its current region through the edges and not the vertices. Therefore, at each step, the attacker has at most four options to go next, excluding obstacle regions. In contrast, the defender is out of our control and we must consider all its possible behaviors. Accordingly, we assume that the defender may go to either of the eight adjacent partitions of its current partition excluding the obstacles or stay at its current partition. For the specific case shown in Fig. 2, the attacker has three options as one of the adjacent partitions, $P_{i-1,j}$, is an obstacle. and the defender has six options as three of its neighbor partitions are out of P. The matrix-game for this situation is shown in Table II.

The rational strategy for the attacker is to minimize its loss based on the most adversarial decision of the defender. According to the matrix game in Table II, the best decision for the attacker can be obtained as:

$$\min_{a' \in POST_A(a_{ij})} \{ \max_{d' \in POST_D(d_{lk})} \{L(a', d')\}\} \tag{15}$$

For all possible configurations, this game should be solved, and the resulting strategies should be included in φ^d_{rul} with the following general structure:

$$\varphi^a_{rul} = \Box \bigwedge_{(i,j)\in M, (l,k)\in(M-\{(i,j)\})} [(a_{ij} \wedge d_{lk}) \rightarrow \bigvee_{(f,g)\in Q_{ij,lk}} \bigcirc a_{fg}] \tag{16}$$

where $Q_{ij,lk}$ contains the solutions of (15) for the case that the attacker is in a_{ij} and the defender is in d_{lk}.

Proposition 1: For a dynamic reach-avoid scenario over a region with $n \times m$ partitions, totally $(nm(nm-2))$ zero-sum games need to be solved to obtain the transition rules of the attacker.

Proof: For a $n \times m$ partitioned region, there are nm possible discrete positions for the attacker. Also, the defender can be in nm possible discrete positions, but for two of them the game is not valid: One case is when the defender is in target region (we excluded this case and assumed that the defender do not stay in the target region to avoid confusion about the winning conditions), and the other case is when the defender is in the same region of the attacker (when the defender captures the attacker and wins). Hence, totally for $nm(nm-2)$ permutations, zero-sum games should be solved for which, based on Lemma 1, there exists at least one decision. ■

5) Obstacle avoidance requirement: The set Obs contains the indices of all obstacle partitions. The requirement that the vehicles cannot enter these partitions is expressed in the following temporal formula:

$$\varphi^\gamma_{obs} = \Box[\bigwedge_{(s,t)\in Obs} \neg\gamma_{st}] \tag{17}$$

where $\gamma \in \{a, d\}$.

6) Players' objectives: The objective of the attacker is to reach the target, which can be expressed as follows:

$$\varphi^a_{obj} = \Box\Diamond a_t \tag{18}$$

where t is the index of the target region.

The objective of the defender can be written as $\Box\Diamond True$ since the defender is not under our control, therefore, we cannot make any assumptions about the defender's objective.

IV. Winning Initial Positions

Here, we address Problem 1 by finding the set of winning initial positions, W^a_{init}, from which it is guaranteed that there is at least one winning strategy for the attacker against every possible behavior of the defender. For this purpose, based on the captured specifications in (5), we model the dynamic reach-avoid problem as a temporal game described as the following Reach-Avoid Game Structure (RAGS):

$$RAGS =< \mathcal{S}, \mathcal{A}, \mathcal{D}, \Theta_a, \Theta_d, \rho_a, \rho_d, \varphi^a_{obj} > \tag{19}$$

where:

- $\mathcal{S} = \mathcal{A} \cup \mathcal{D}$ is the set of *Boolean propositions* for this game, where $\mathcal{A}$ and $\mathcal{D}$ are defined in (2). We define Π_A and Π_D as the set of all interpretations of sets $\mathcal{A}$ and $\mathcal{D}$, respectively. For example, $\pi_a = (a_{11} = True, a_{12} = False, \ldots, a_{nm} = False) \in \Pi_A$ is an interpretation of $\mathcal{A}$. For any $\pi_a \in \Pi_A$ and $\pi_d \in \Pi_D$, the *state* $\pi = (\pi_a, \pi_d) \in \Pi$ describes the positions of the vehicles at a particular instance of the game, where $\Pi = \Pi_A \times \Pi_D$ is the *state space* of $RAGS$.
- $\Theta_a = \{\pi_a \in \Pi_A | \pi_a \models \varphi^a_{sing}\}$ captures the set of all possible initial positions of the attacker,
- $\Theta_d = \{\pi_d \in \Pi_D | \pi_d \models \varphi^d_{init} \wedge \varphi^d_{sing}\}$ captures the initial position of the defender, which is initially located at $P_{i_{d_0}, j_{d_0}}$,
- $\rho_d = \{((\pi_a, \pi_d), \pi'_d) \in \Pi_A \times \Pi_D \times \Pi_D \mid \sigma = \pi\pi' \models \varphi^d_{sing} \wedge \varphi^d_{term} \wedge \varphi^d_{rul} \wedge \varphi^d_{obs}$, where $\pi = (\pi_a, \pi_d)$, $\pi' = (\pi'_a, \pi'_d)$, and $\pi'_a \in \Pi_A\}$ is the defender's transition relation, and the sequence $\sigma = \pi\pi'$ captures two successive states of the game,
- $\rho_a = \{((\pi_a, \pi_d), (\pi'_a, \pi'_d)) \in \Pi_A \times \Pi_D \times \Pi_A \times \Pi_D | (\pi_a, \pi_d, \pi'_d) \in \rho_d$ and $\sigma = \pi\pi' \models \varphi^a_{sing} \wedge \varphi^a_{term} \wedge \varphi^a_{rul} \wedge \varphi^a_{obs}$, where $\pi = (\pi_a, \pi_d)$, $\pi' = (\pi'_a, \pi'_d)\}$ is the attacker's transition relation,
- φ^a_{obj} is the winning condition of the game.

Each state of RAGS, $\pi \in \Pi$, represents one position of the attacker and one position of the defender over the partitioned

	$a_{i',j'-1}$	$a_{i'+1,j'}$	$a_{i',j'+1}$
$d_{\ell',k'-1}$	$L(a_{i',j'-1}, d_{\ell',k'-1})$	$L(a_{i'+1,j'}, d_{\ell',k'-1})$	$L(a_{i',j'+1}, d_{\ell',k'-1})$
$d_{\ell',k'}$	$L(a_{i',j'-1}, d_{\ell',k'})$	$L(a_{i'+1,j'}, d_{\ell',k'})$	$L(a_{i',j'+1}, d_{\ell',k'})$
$d_{\ell',k'+1}$	$L(a_{i',j'-1}, d_{\ell',k'+1})$	$L(a_{i'+1,j'}, d_{\ell',k'+1})$	$L(a_{i',j'+1}, d_{\ell',k'+1})$
$d_{\ell'+1,k'-1}$	$L(a_{i',j'-1}, d_{\ell'+1,k'-1})$	$L(a_{i'+1,j'}, d_{\ell'+1,k'-1})$	$L(a_{i',j'+1}, d_{\ell'+1,k'-1})$
$d_{\ell'+1,k'}$	$L(a_{i',j'-1}, d_{\ell'+1,k'})$	$L(a_{i'+1,j'}, d_{\ell'+1,k'})$	$L(a_{i',j'+1}, d_{\ell'+1,k'})$
$d_{\ell'+1,k'+1}$	$L(a_{i',j'-1}, d_{\ell'+1,k'+1})$	$L(a_{i'+1,j'}, d_{\ell'+1,k'+1})$	$L(a_{i',j'+1}, d_{\ell'+1,k'+1})$

TABLE II

ZERO-SUM GAME MATRIX FOR DECISION-MAKING FOR THE EXAMPLE IN FIG.2

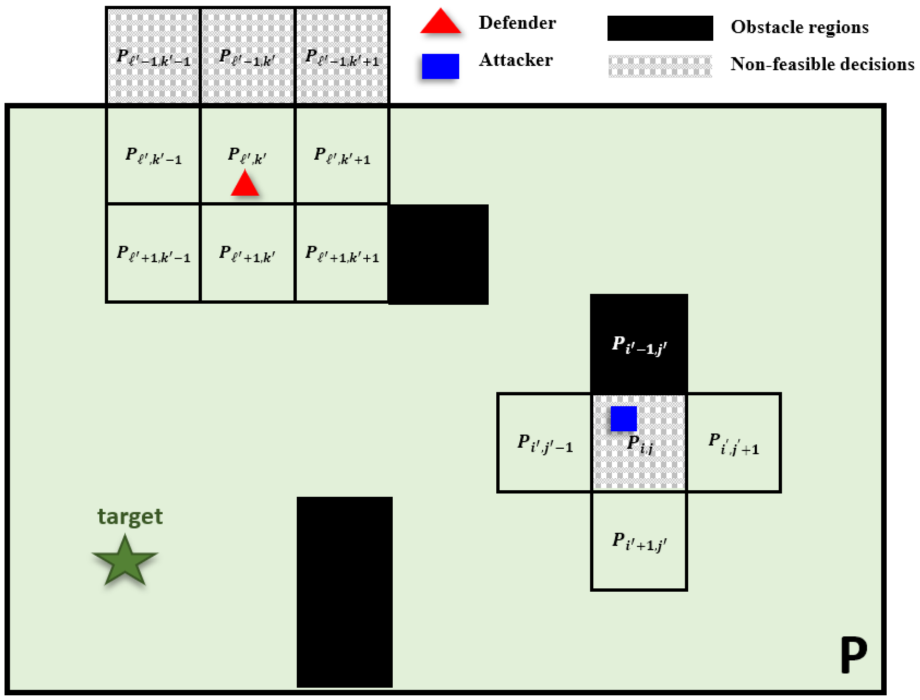


Fig. 2. A possible configuration of a reach-avoid game. The defender has six options ($P_{\ell',k'-1}$, $P_{\ell',k'}$, $P_{\ell',k'+1}$, $P_{\ell'+1,k'-1}$, $P_{\ell'+1,k'}$,$P_{\ell'+1,k'+1}$) as three of its adjacent partitions ($P_{\ell'-1,k'-1}$, $P_{\ell'-1,k'}$, $P_{\ell'-1,k'+1}$) are out of region P. The attacker, on the other hand, must leave its current region through edges, and has only three options ($P_{i',j'-1}$, $P_{i',j'+1}$, $P_{i'+1,j'}$) as one of its neighbor partitions, $P_{i'+1,j'}$, is obstacle.

space. The objective is to explore and obtain a safe subset of Π as the winning set of states, W, from which we can guarantee that for any sequence of actions of defender, there exists a sequence of attacker actions, which keeps the state of the game in W and ends at Π_t. In this way, we satisfy both safety (by staying in W) and reachability (by reaching Π_t).

A. The Solution of RAGS

To find the solution of RAGS, we formulate it as a μ-calculus fixed-point problem with a solution W, which contains all winning states of the RAGS. The solution to Problem 1 is then a subset of W, in which the defender's initial position is fixed to $d_{i_{d_0},j_{d_0}}$. For this purpose, we define $Var = \{X, Y\}$ a set of rational variables. Each rational variable is assigned to a specific subset of Π through the function $\xi : Var \to 2^{\Pi}$.

In [31], it is shown that the solution of a GR(1) game can be found using μ-calculus. Compared to [31], we use lattice theory to formally prove the existence and correctness of the solution. The μ-calculus formulas are defined over $\mathcal{S}$ and Var as follows:

$$\phi =: s \mid \neg s \mid X \mid \xi(X) \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \qquad (20)$$
$$\mid \mu X.\xi(X) \mid \nu X.\xi(X) \mid Pre(\phi)$$

where,

- $X \in Var$ and $s \in \mathcal{S}$,
- μ and ν are the least and greatest fixed-point operators, respectively,

- and for a positive monotone function $\xi : Var \rightarrow 2^{\Pi}$, we have[1]:
  - $\mu X.\xi(X) = \cup_i X_i$, where $X_0 = \emptyset$ and $X_{i+1} = \xi(X_i)$
  - $\nu X.\xi(X) = \cap_i X_i$, where $X_0 = \Pi$ and $X_{i+1} = \xi(X_i)$
- $Pre(\phi) = \{\pi = (\pi_a, \pi_d) \in \Pi \mid \forall \pi'_d \in \Pi_D, \exists \pi'_a \in \Pi_A$ such that $((\pi_a, \pi_d), \pi'_d) \in \rho_d$, $((\pi_a, \pi_d), (\pi'_d, \pi'_a)) \in \rho_a$, and $\pi' \in [[\phi]]\}$, where $[[\phi]] = \{(\pi'_a, \pi'_d) \mid (\pi'_a, \pi'_d) \models \phi\}$ and $\pi' = (\pi'_a, \pi'_d)$ contains the information about the next position of the attacker and the defender.

Here, $Pre(\phi)$ computes the set of all the states from which the attacker can make a single-step safe transition, independent of the defender moves, where the new position of the attacker, π'_a, and the new position of defender, π'_d, form a state $\pi' = (\pi'_a, \pi'_d)$ consistent with ϕ. The transition rules ρ_a and ρ_d impose the GR(1) formula in (5). We introduce the function $\xi(X, Y) : 2^{\Pi} \times 2^{\Pi} \rightarrow 2^{\Pi}$ as:

$$\xi(X, Y) = (\Pi_t \wedge Pre(Y)) \vee Pre(X) \tag{21}$$

where $\Pi_t = \{\pi \in \Pi \mid \pi \models a_t \wedge \varphi^a_{sing} \wedge \varphi^d_{sing}\}$ contains all the states that capture the situations when the attacker is in the target partition whereas the defender can be in any partitions.

The solution of RAGS is the set of winning states W which can be represented as the following μ-calculus formula,

$$W = \nu Y.\mu X.\xi(X, Y) \tag{22}$$

In (22), the greatest fixed-point operator, νY, calculates the states that are safe for the attacker (not being captured by the defender), while the least fixed-point operator, μX, starts from $\Pi_t \wedge Pre(Y)$ with the most recent updated Y, then, recursively and in a backward way computes the set of states from which the attacker can reach one of the states in $\Pi_t \wedge Pre(Y)$. In this way, the solution of the game satisfies both safety and reachability.

Algorithm 1 implements the recursive procedure in $\nu Y.\mu X.\xi(X, Y)$ through two **while** loops by initially setting $X = \emptyset$ and $Y = \Pi$. Then, the inner-loop finds the least fixed-point of $\xi(X, Y)$ for a given Y, and the outer loop calculates the greatest fixed-point of the function $\xi'(Y) = \mu X.\xi(X, Y)$.

The procedure starts with $Y = \Pi$ assuming that all the states of RAGS are safe. The inner-loop (lines 8-12) starts from $mX[0] = \Pi_t \wedge Y$ and finds $mX[i+1] = mX[i] \vee Pre(X)$ until $mX[i] = mX[i-1]$ producing the sequence $mX[0], mX[1], \cdots, mX[maxi]$. The final set $mX[maxi]$ contains all the reachable states to the target. However, the safety requirement for these states (not being captured by the defender) is not guaranteed yet.

[1]Intuitively the least fixed-point operator starts by under-approximation and converges to the solution by consecutively expanding the initially under-approximated solution. In contrary, the greatest fixed-point operator starts by over-approximation and converges to a solution by consecutively reducing the initially over-approximated solution.

Algorithm 1: Winning Regions Calculation

```
   Input  : $RAGS =< \mathcal{S}, \mathcal{A}, \mathcal{D}, \Theta_a, \Theta_d, \rho_a, \rho_d, \Box\Diamond a_t >$
   Output: $W$ = winning states, $W^a_{init}$ = winning initial
           positions
1  Let $i, j := 0$
2  Let $X := \emptyset$; $Y := \Pi$
3  do
4      $mY[j] := Y$
5      $j := j + 1$
6      $X := \Pi_t \wedge Pre(Y)$
7      $i := 0$
8      do
9          $mX[i] := X$
10         $X = X \vee Pre(X)$
11         $i := i + 1$
12     while $(mX[i-1] \neq X)$;
13     $maxi = i - 1$
14     $Y := mX[maxi]$
15 while $(Y \neq mY[j-1])$;
16 $maxj = j - 1$
17 Let $W := mY[maxj]$
18 Let $W^a_{init} = \{a_{ij} | \exists(\pi_a, \pi_d) \in W,\ \pi_a \models a_{ij}$ and
       $\pi_d \models \varphi^d_{init}\}$
19 Return $W^a_{init}$, $W$, $mX$
```

The outer loop checks if $mY[j]$ is the greatest fixed-point of the function $\mu X.\xi(X, Y)$, i.e., if $mY[j] = mY[j-1]$. This is equivalent to finding the safe reachable states. For this purpose, the unsafe regions are removed from the initial approximation of the least fixed-point $mX[0]$ (line 6). Then, the least fixed-point $mX[maxi]$ is recomputed in the inner-loop (lines 8-12), which is an approximation of the greatest fixed-point $mY[j]$. This will be used to update the initial approximation of the least fixed-point $mX[0]$, from which the unsafe states should be removed again. This recursive process will result in a sequence of approximations of the greatest fixed-point $mY[0], mY[1], mY[2], \cdots$ in the outer-loop until $mY[j] = mY[j-1]$.

Finally, we let $W = mY[maxj]$, where W contains all winning permutations so that for any finite sequence of actions of the defender $\pi^1_d, \pi^2_d, ..., \pi^n_d$, there exists a safe (non-colliding) sequence of attacker actions $\pi^1_a, \pi^2_a, ..., \pi^n_a$, ending in any target states $\pi^n_a \models a_t\}$, while $\pi^k = (\pi^k_a, \pi^k_d)$ remains in W, $k = 1, \ldots, n$.

Once W is calculated, for a particular initial position of the defender $d_{i_{d_0} j_{d_0}}$, we can calculate the set $W^a_{init} = \{a_{ij} \mid \pi_a \models a_{ij}$ and $\exists(\pi_a, \pi_d) \in W, \pi_d \models \varphi^d_{init}\}$.

B. Discussion on the correctness and termination of the algorithm

Next, we show that Algorithm 1 returns the solution of RAGS after a finite number of iterations, but first, we need the following definitions and lemmas.

Definition 3: A pair $\langle \Sigma, \preceq \rangle$ is a partially ordered set (poset), where Σ is a set and $\preceq$ is a partial order relation.

Definition 4: [39] A poset $\langle \Sigma, \preceq \rangle$ is a *lattice* if for any finite $Z \subseteq \Sigma$ we have $supZ, infZ \in \Sigma$. If this property holds for any arbitrary $Z \subseteq \Sigma$, then $\langle \Sigma, \preceq \rangle$ is a *complete lattice*. Clearly, any lattice $\langle \Sigma, \preceq \rangle$ over a finite set, Σ, is complete.

Lemma 2: The poset $\langle 2^\Pi, \subseteq \rangle$ is a complete lattice, where Π is the state space of $RAGS$ in (19).

Proof: The set Π is finite, so is 2^Π. Therefore, to prove that $\langle 2^\Pi, \subseteq \rangle$ is a complete lattice, it suffices to show that it is a lattice. For any power set 2^Σ, we have $sup(2^\Sigma) = \Sigma$ and $inf(2^\Sigma) = \emptyset$. We also know that for any $Z \subseteq 2^\Sigma$, we have $supZ \subseteq sup2^\Sigma$. This means that $supZ \subseteq \Sigma$, concluding that $supZ \in 2^\Sigma$ as all subsets of Σ are a member of 2^Σ. Similarly, since we have $infZ \subseteq \Sigma$, then $infZ \in 2^\Sigma$. Therefore, $\langle 2^\Pi, \subseteq \rangle$ is a lattice, and since 2^Π is finite, it is a complete lattice. ■

Lemma 3: Functions $\xi(X,Y) = (\Pi_t \wedge Pre(Y)) \vee Pre(X)$ and $\xi'(Y) = \mu X.\xi(X,Y)$ are positive monotone.

Proof: First, we prove the positive monotonicity of $\xi(X,Y)$ with respect to both X and Y. All Boolean operators, including conjunction ($\wedge$) and disjunction ($\vee$), are increasing, except for negation ($\neg$). The function $\xi(X,Y)$ does not have any negation, and hence to prove positive monotonicity of $\xi(X,Y)$, we only have to show the operator $Pre(*)$ is increasing. Let, $Z_1 \subseteq Z_2$. Then $pre(Z_2) = pre(Z_1) \cup pre(Z_2 \backslash Z_1)$, implying that $pre(Z_1) \subseteq pre(Z_2)$. Therefore, for any $X_1 \subseteq X_2$ and $Y_1 \subseteq Y_2$, we have $\xi(X_1,Y_1) \subseteq \xi(X_2,Y_2)$.

Now, we show that $\xi'(Y)$ is positive monotone. According to (20), $\xi'(Y) = \mu X.\xi(X,Y) = \cup_{i=0}\xi^i(X,Y)$, where $\xi^0(X,Y) = X$, and $\xi^{i+1}(X,Y) = \xi(\xi^i(X,Y),Y)$. For any X and any $Y_1 \subseteq Y_2$, since $\xi(X,Y)$ is positive monotone, we have:

$$\begin{aligned} Y_1 \subseteq Y_2 &\iff \xi(X,Y_1) \subseteq \xi(X,Y_2) \\ &\iff \xi(\xi^i(X,Y_1),Y_1) \subseteq \xi(\xi^i(X,Y_2),Y_2) \\ &\iff \cup_{i=0}\xi^i(X,Y_1) \subseteq \cup_{i=0}\xi^i(X,Y_2) \\ &\iff \mu X.\xi(X,Y_1) \subseteq \mu X.\xi(X,Y_2) \\ &\iff \xi'(Y_1) \subseteq \xi'(Y_2) \end{aligned} \tag{23}$$

■

Theorem 1: [Existence of solution] The least fixed-point of $\xi(X,Y) = (\Pi_t \wedge Pre(Y)) \vee Pre(X)$ and the greatest fixed-point of $\mu X.\xi(X,Y)$ exist.

Proof: a) First we show that over the complete lattice $\langle 2^\Pi, \subseteq \rangle$, the least fixed-point of the function $\xi(X,Y)$ with respect to X exists.

Let $prefix = \{X \mid \xi(X,Y) \subseteq X\}$ with the greatest lower bound p. We show that p exists and is the least fixed-point of ξ with respect to X.

Since $\langle 2^\Pi, \subseteq \rangle$ is a complete lattice, $p = inf(prefix)$ exists. Then, for any $X \in prefix$, we have $p \subseteq X$. Since, ξ is positive monotone and $X \in prefix$, we have $\xi(p,Y) \subseteq \xi(X,Y) \subseteq X$, meaning that $\xi(p,Y)$ is a lower bound of $prefix$. As p is the greatest lower bound of $prefix$, $\xi(p,Y) \subseteq p$, which means $p \in prefix$.

In addition, since $\xi(X,Y)$ is positive monotone, from $\xi(p,Y) \subseteq p$ we have $\xi(\xi(p,Y),Y) \subseteq \xi(p,Y)$. As a result, $\xi(p,Y) \in prefix$ and given that p is the greatest lower bound, $p \subseteq \xi(p,Y)$.

From the above derivations, we have $\xi(p,Y) \subseteq p$ and $p \subseteq \xi(p,Y)$, which implies that $p = \xi(p,Y)$. Therefore, provided that any fixed-point is a member of $prefix$ and p is the greatest lower bound of this set, we can conclude that p is the least fixed-point of ξ.

b) Now, we show that the greatest fixed-point of $\xi'(Y) = \mu X.(\Pi_t \wedge Pre(Y)) \vee Pre(X)$ exists. Let $post = \{Y \mid Y \subseteq \xi'(Y)\}$ with the least upper bound q. We show that q is the greatest fixed-point of $\xi'(Y)$.

Since $\langle 2^\Pi, \subseteq \rangle$ is a complete lattice, $q = sup(post)$ exists. Then, for any $Y \in post$, we have $Y \subseteq q$. Since, ξ' is positive monotone and $Y \in post$, we have $Y \subseteq \xi'(Y) \subseteq \xi'(q)$, meaning that $\xi'(q)$ is an upper bound of $post$. As q is the least upper bound of $post$, $q \subseteq \xi'(q)$, which means $q \in post$.

As $\xi'(Y)$ is positive monotone and $q \subseteq \xi'(q)$, then $\xi'(q) \subseteq \xi'(\xi'(q))$. As a result, $\xi'(q) \in post$ and given that q is the least upper bound, $\xi'(q) \subseteq q$.

From the above derivations, we have $\xi'(q) \subseteq q$ and $q \subseteq \xi'(q)$, concluding that $q = \xi'(q)$. Therefore, provided that any fixed-point is a member of $post$ and q is the least upper bound of this set, we can conclude that q is the greatest fixed-point of ξ', which also serves as the solution of the game as $q = \nu Y.\xi'(Y) = \nu Y.\mu X.\xi(X,Y)$.

■

Theorem 2: The solution of RAGS can be implemented through a finite search.

Proof: As proved in Theorem 1, the least and greatest fixed-points of ξ and ξ' exist. Also, based on Lemma 3, both functions ξ and ξ' are positive monotone, which allows us to use (20) to calculate these fixed-points through an iterative process. Next, we show that Algorithm 1 returns the solution of RAGS after a finite number of iterations.

According to (20), we can calculate the least fixed-point of $\xi(X,Y)$ as $\mu X.\xi(X,Y) = \cup_i X_i$ where $X_{i+1} = \xi(X_i,Y) = (\Pi_t \wedge Pre(Y)) \vee Pre(X_i) = X_1 \vee Pre(X_i)$ and $X_0 = \emptyset$. This recursive procedure is implemented within the inner-loop of Algorithm 1 (lines 8-12). The variable X is being updated in line 10 as $X = X \vee Pre(X_i)$ to form $X = \cup_i X_i$ until $X_i = X_{i-1}$, which can happen when $Pre(X_i) = Pre(X_{i-1})$, i.e., no other state can be added to X. Since, $X = X \vee Pre(X_i)$ is monotonically increasing over the finite set Π, this inner-loop either ends at $X \subset \Pi$, or continues until $X = \Pi$.

Similarly, according to (20), we can calculate the greatest fixed-point of ξ' as $\nu Y.\xi'(Y) = \cap_j Y_j$ where $Y_{j+1} = \xi'(Y_j) = \mu X.\xi(X,Y_j)$ and $Y_0 = \Pi$, as implemented in the outer-loop of Algorithm 1 (lines 3-15). Since ξ' is positive monotone, for any two successive approximation of the greatest fixed-point $Y_{j+1} \subseteq Y_j$, we have $\xi'(Y_{j+1}) \subseteq \xi'(Y_j)$. Since, $Y_0 = \Pi$ and $Y_1 = \mu X.\xi(X,Y_0) \subseteq \Pi$, we have $Y_1 \subseteq Y_0$. Therefore, starting from Y_0, we will have a decreasing sequence $Y_0, Y_1, \cdots$ over the finite set Π until $Y_j = Y_{j-1}$. Therefore, the outer-loop will terminate after a finite number of iterations. ■

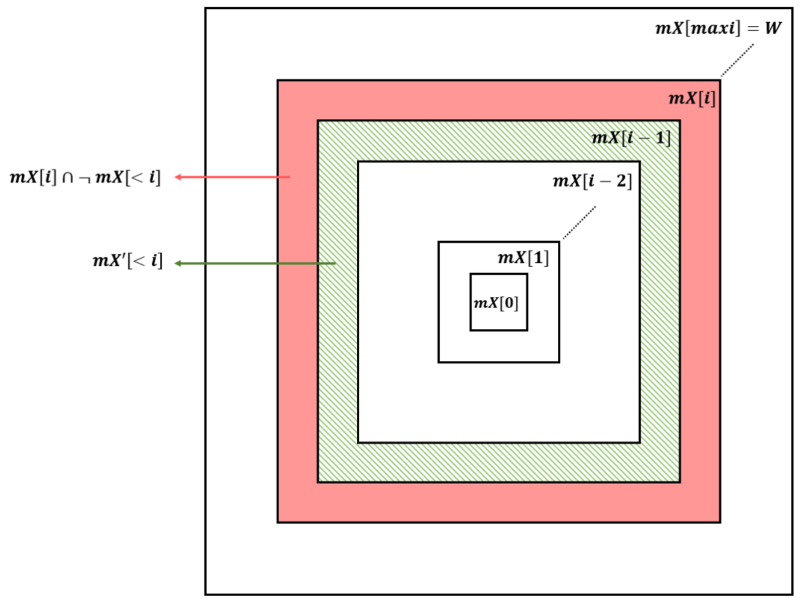


Fig. 3. A schematic of the sets $mX[i] \in mX$ for $i = 1, \cdots, maxi$.

V. High-level planning

Assume that the attacker is initially located in $P_{i_0 j_0}$. If $a_{i_0 j_0} \in W_a^{init}$, then there exists a winning strategy for the attacker against any behavior of the defender. Therefore, we can control the attacker to win the game to address Problem 2. We solve this problem in two steps: first, in this section, we use the winning set, W, to extract the winning strategies over the partitioned environment. Then, in the next section, using the resulting winning strategies, we construct a hybrid controller to smoothly drive the attacker over the operation region, P, to win the game.

To extract the winning strategies, having the result of Algorithm 1 including the winning states W of the game structure $RAGS$, Algorithm 2 obtains the winning strategies for the attacker by constructing the transition relation ρ' over the winning set W. The transition relation ρ' can be constructed by refining ρ as the following:

$$\rho' = \bigcup_{i=1}^{maxi} \big((mX[i] \cap \neg mX[< i]), mX'[< i]\big) \qquad (24)$$

where:

- $\big((mX[i] \cap \neg mX[< i]), mX'[< i]\big) \in \rho$,
- $\rho = \{\big((\pi_a, \pi_d), (\pi_a', \pi_d')\big) \mid ((\pi_a, \pi_d), \pi_d') \in \rho_d,\ ((\pi_a, \pi_d), (\pi_d', \pi_a')) \in \rho_a\}$,
- $mX[i]$ is the set of intermediate winning states obtained from Algorithm 1 ($mX[0] \subseteq \Pi_t$ and $mX[maxi] = W$),
- $mX[< i] = \cup_{l \in [1, \cdots, i-1]} mX[l]$,
- $mX'[< i] = mX[< i] - mX[< i-1]$,
- $\neg mX[< i] = W - mX[< i]$.

To construct the winning strategies, ρ', we start in a backward way and find the transitions from the states in $mX[1] \cap \neg mX[< 1] = mX[1] - mX[0]$ to the states in $mX'[< 1] = mX[0] \subseteq \Pi_t$. Similarly, at the ith step, we find the possible transitions from the states in $mX[i] \cap \neg mX[< i]$ (the red area in Fig. 3) to the states in $mX'[< i] = mX[< i] - mX[< i-1]$ (the green area in Fig. 3). At the end of this process, we will have the set of the winning strategies, ρ', which is achieved by revising the original ρ so that by following ρ', the attacker wins the game against all possible moves of the defender.

Algorithm 2: Winning Strategies

```
Input  : $\rho$, $maxi$, $mX$, $W$
Output: $\rho'$: Transition Relation
1 $\rho' = \{\big((\pi_a, \pi_d), (\pi_a', \pi_d') \in \rho\big) \mid (\pi_a, \pi_d) \in mX[0]$, and $((\pi_a', \pi_d')) \in W\}$
2 Let $mXp = mXp' = mX[0]$
3 forall $r \in \{1, \cdots, maxi\}$ do
4     $\neg mXp = W - mXp$
5     $\rho' = \rho' \cup \{\big((\pi_a, \pi_d), (\pi_a', \pi_d') \in \rho\big) \mid (\pi_a, \pi_d) \in mX[r] \cap \neg mXp, (\pi_a', \pi_d') \in mXp'\}$
6     $mXp' = mX[r] - mX[r-1]$
7     $mXp = mX[r] \cup mXp$
8 end
9 Return $\rho'$
```

The relation ρ' might not be deterministic, as, for some defender decisions, there may exist different winning options for the attacker. Therefore, in Algorithm 3, we make the transition relations of the attacker deterministic by selecting the best action of the attacker that is the move to a safe region which is the closest to the target.

VI. Hybrid Controller Synthesis

In order to drive the attacker to achieve its objective, the high-level plans achieved in the previous section, should be

[2] $dist(\pi_a', \pi_t) = ||(i_a, j_a) - (i_t, i_t)||$, where $\pi_a' \models a_{i_a j_a}, \pi_t' \models a_{i_t j_t}$

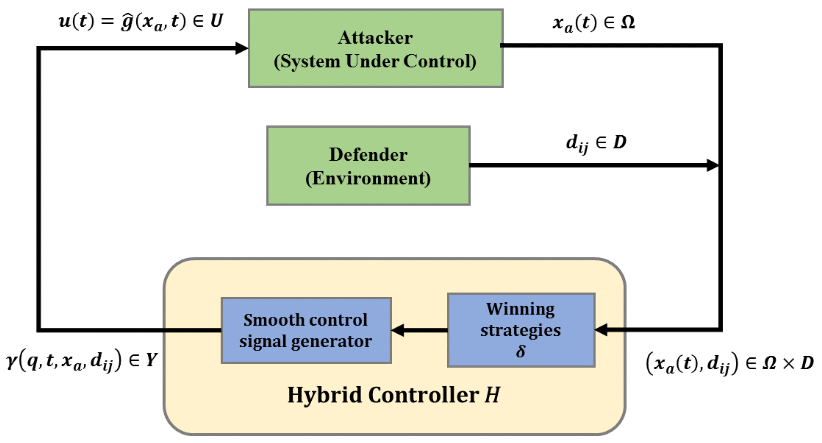

$u(t) = \hat{g}(x_a, t) \in U$
Attacker
(System Under Control)
$x_a(t) \in \Omega$
Defender
(Environment)
$d_{ij} \in D$
Smooth control signal generator
Winning strategies δ
$\gamma(q, t, x_a, d_{ij}) \in Y$
$(x_a(t), d_{ij}) \in \Omega \times D$
Hybrid Controller H

Fig. 4. The proposed hybrid controller structure.

Algorithm 3: Deterministic Winning Strategies

```
Input  : $\rho'$, $W$
Output: $\delta_a$: Deterministic Transition Relation
1 $\delta_a = \emptyset$
2 forall $(\pi_a, \pi_d) \in W$ do
3     $\Gamma = \{\pi'_a | \exists \pi'_d, ((\pi_a, \pi_d), (\pi'_a, \pi'_d)) \in \rho'\}$
4     Pick $\pi'_{min}$ from $\underset{\pi'_a \in \Gamma}{argmin}\ dist^2(\pi'_a, \pi_t)$
5     $\delta_a = \delta_a \cup \{((a_{ij}, d_{lk}), a_{i'j'}) |\ \pi_a \models a_{ij}\ and\ \pi_d \models d_{lk}\ and\ \pi_{min} \models a_{i'j'}\}$
6 end
7 Return $\delta_a$
```

converted to continuous control signals. For this purpose, we propose a hybrid controller, shown in Fig. 4 as follows:

$$H = (Q, Z, W, Y, f_H, Init, Inv, \delta, \gamma) \tag{25}$$

where:

- $Q = \mathcal{A} \times S$ is the set of discrete states of the hybrid controller, where $\mathcal{A}$ is defined in (2) and $s \in S = \{Rest, Stay, Left, Right, Up, Down\}$ is the previous high-level command of the controller (with the assumption that the controller is initially at the $Rest$ situation).
- $Z = t$ is the continuous state of the controller, which captures the time.
- $W = \Omega \times \mathcal{D}$ is the input set of the controller. For any input $(x, d_{ij}) \in W$, $x \in \Omega$ is the state (position) of the (attacker) system, and $d_{ij} \in \mathcal{D}$ is the position of the defender.
- Y is the output of the control system (the input to the attacker).
- $f_H = \dot{t} = 1$ is the vector field.
- $Init = (a_0, s_0, t_0)$ is the initial state of the hybrid control system, where $a_0 \in W^a_{init}$, $s_0 = Rest$, and $z_0 = t_0 = 0$.
- $Inv : Q \to Z \times \Omega$ assigns to each state of the hybrid controller an invariant set, where $Inv(q = (a_{ij}, s)) = \{(t, x) | x \in P_{ij}\}$.
- $\delta : Q \times W \to Q$ is the transition relation. $\delta(q = (a_{ij}, s), w = (x, d_{\ell k})) = (a_{i'j'}, s') = q'$ if there exists a wining transition $\delta_a(\pi_a, \pi_d) = \pi'_a$ (defined in (19)), where $\pi_a \models a_{ij}$, $\pi_d \models d_{\ell k}$, $\pi'_a \models a_{i'j'}$, and $s' \in S$ is the proper discrete output (high-level command) to guide the attacker from P_{ij} to $P_{i'j'}$, where $s' = \varsigma(a_{ij}, a_{i'j'})$ can be found from:

$$\varsigma(a_{ij}, a_{i'j'}) = \begin{cases} \textit{Stay}, & \text{if } i' = i, j' = j \\ \textit{Down}, & \text{if } i' > i, j' = j \\ \textit{Right}, & \text{if } i' = i, j' > j \\ \textit{Up}, & \text{if } i' < i, j' = j \\ \textit{Left}, & \text{if } i' = i, j' < j \end{cases} \tag{26}$$

- $\gamma : Q \times Z \times W \to Y$ is the output map function, where $\gamma(q, t, x, d) = \hat{g}(x, t)$. For any q and d, we design $\hat{g}(x, t)$ in Section VI-A.

Assume that the attacker is currently in region $P_{ij} \in P$ and the discrete state of the hybrid controller H is $q = (a_{ij}, s)$. The proposed hybrid controller observes the behavior of the defender and based on the defender's position, $d_{\ell k} \in \mathcal{D}$, triggers a discrete transition from P_{ij} to $P_{i'j'}$, denoted by $\delta(q, d_{\ell k}) = q'$. To move to the new state q', when the attacker and the defender are in the partitions P_{ij} and $P_{\ell k}$, respectively, we design a smooth control signal $u = \hat{g}(x, t) = \gamma(a_{ij}, s, t, x, d_{\ell k})$. Since we have considered all possible transitions (decisions) of the defender in Algorithms 1 and 2, all actions of the defender will be reacted by an action of the attacker with a smooth control signal, leading the attacker to win the game. The mechanism for the generation of a smooth control signal is discussed next.

A. Smooth Continuous Control Signal Generation

Having the discrete transitions (high-level plan), we should generate a continuous path for the attacker. Knowing that the dynamics of the attacker is multi-affine given in (3), we can take advantage of an important property that the value of a multi-affine function $g(x)$, inside a polygon can be described as a unique convex combination of its values at the vertices of that polygon [35]. To mathematically describe this property, consider a two-dimensional rectangular partition $P_{ij} = \{x \in P | (a_1 + (j-1)\delta \leq x_1 \leq a_1 + j\delta)$ and $(b_2 - i\delta \leq x_2 \leq b_2 - (i-1)\delta)\}$, where (a_1, b_2) are the coordinates of the top left corner of P. For the rectangle P_{ij}, we define the set of *vertices*, V_{ij}:

$$V_{ij} = \{v_{ij}^1, v_{ij}^2, v_{ij}^3, v_{ij}^4\} \tag{27}$$

where

$$v_{ij}^1 = \begin{pmatrix} a_1 + (j-1)\delta \\ b_2 - i\delta \end{pmatrix}, v_{ij}^2 = \begin{pmatrix} a_1 + j\delta \\ b_2 - i\delta \end{pmatrix} \tag{28}$$
$$v_{ij}^3 = \begin{pmatrix} a_1 + j\delta \\ b_2 - (i-1)\delta \end{pmatrix}, v_{ij}^4 = \begin{pmatrix} a_1 + (j-1)\delta \\ b_2 - (i-1)\delta \end{pmatrix}$$

We also define the set of *facets*, E_{ij}, as:

$$E_{ij} = \{e_{ij}^1, e_{ij}^2, e_{ij}^3, e_{ij}^4\} \tag{29}$$

where the facets e_{ij}^ℓ and their corresponding outward unit normal vector n_ℓ^T, $\ell = 1, \cdots, 4$ are defined in the following:

$$\begin{aligned} e_{ij}^1 &= \{x \in P_{ij} | x_2 = b_2 - i\delta\}, \quad n_1^T = (0, -1) \\ e_{ij}^2 &= \{x \in P_{ij} | x_1 = a_1 + j\delta\}, \quad n_2^T = (1, 0) \\ e_{ij}^3 &= \{x \in P_{ij} | x_2 = b_2 - (i-1)\delta\}, \quad n_3^T = (0, 1) \\ e_{ij}^4 &= \{x \in P_{ij} | x_1 = a_1 + (j-1)\delta\}, \quad n_4^T = (-1, 0) \end{aligned} \tag{30}$$

The graphical visualization of the vertices and facets is provided in Fig. 5. Now, it can be verified that at any point x inside P_{ij}, the value of $g(x)$ can be written as follows:

$$g(x) = \sum_{\ell=1,\cdots,4} \lambda_\ell(x, v_{ij}^\ell) g(v_{ij}^\ell) \tag{31}$$

where $g(v_{ij}^\ell)$ is the control value at the vertex v_{ij}^ℓ, and:

$$\begin{aligned} \lambda_\ell(x, v_{ij}^1) &= \frac{1}{\delta^2}(a_1 + j\delta - x_1)(b_2 - (i-1)\delta - x_2) \\ \lambda_\ell(x, v_{ij}^2) &= \frac{1}{\delta^2}(x_1 - a_1 - (j-1)\delta)(b_2 - (i-1)\delta - x_2) \\ \lambda_\ell(x, v_{ij}^3) &= \frac{1}{\delta^2}(x_1 - a_1 - (j-1)\delta)(x_2 - b_2 - i\delta) \\ \lambda_\ell(x, v_{ij}^4) &= \frac{1}{\delta^2}(a_1 + j\delta - x_1)(x_2 - b_2 - i\delta) \end{aligned} \tag{32}$$

Remark 1: It can be simply verified that $\sum_{\ell=1,\cdots,4} \lambda_\ell(x, v_{ij}^\ell) = 1$.

By using the aforementioned properties and designing the feedback control signal $u = Bg(x)$, the vector field of the system in (3) will be $h(x) = f(x) + Bg(x)$. Since both functions f and g are multi-affine, h is a multi-affine function as well.

In order to construct the control signal $u = Bg(x)$ to implement a discrete command $s \in S$, we first, determine $g(v)$ at the vertices $v \in V_{ij}$. These can be selected in a way that the vector field at the vertices, $h(v) = f(v) + Bg(v)$, has the values given in Table III. In this table, $\beta_u > 0$ is a constant determined by the upper-bound of the control signal u. Therefore, having the values of the vector field at the vertices, $h(v)$, and the system function $f(v)$, the control values at the vertices will be:

$$Bg(v) = h(v) - f(v) \tag{33}$$

TABLE III
VECTOR FIELD VALUES $h(v)$ OVER THE VERTICES FOR DISCRETE COMMANDS $s \in S$

s	$h(v_1)$	$h(v_2)$	$h(v_3)$	$h(v_4)$
Stay	$\beta_u \begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\beta_u \begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\beta_u \begin{pmatrix} -1 \\ -1 \end{pmatrix}$	$\beta_u \begin{pmatrix} 1 \\ -1 \end{pmatrix}$
Down	$\beta_u \begin{pmatrix} 1 \\ -1 \end{pmatrix}$	$\beta_u \begin{pmatrix} -1 \\ -1 \end{pmatrix}$	$\beta_u \begin{pmatrix} -1 \\ -1 \end{pmatrix}$	$\beta_u \begin{pmatrix} 1 \\ -1 \end{pmatrix}$
Right	$\beta_u \begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\beta_u \begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\beta_u \begin{pmatrix} 1 \\ -1 \end{pmatrix}$	$\beta_u \begin{pmatrix} 1 \\ -1 \end{pmatrix}$
Up	$\beta_u \begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\beta_u \begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\beta_u \begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\beta_u \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
Left	$\beta_u \begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\beta_u \begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\beta_u \begin{pmatrix} -1 \\ -1 \end{pmatrix}$	$\beta_u \begin{pmatrix} -1 \\ -1 \end{pmatrix}$

Once we found the control values at the vertices, at any point x, the control signal $u = Bg(x)$ can be designed by substituting the control values at the vertices into (31). Using this controller, the resulted vector field, $h(x) = f(x) + Bg(x)$, for the system in (3) will drive it to transit from a rectangle to a desired adjacent rectangle. The proof is similar to [8] and [35]. The problem is when the hybrid controller requires the system to move from P_{ij} to $P_{i'j'}$, the implementation of this transition may cause a discontinuity in the value and direction of the control signal, from $g_{old}(x, \theta^-)$ to $g_{new}(x, \theta^+)$, where g_{old} and g_{new} are generated control signals when the hybrid controller is at $q = (a_{ij}, s)$ and $q' = (a_{i'j'}, s')$, respectively, and θ is the time when the discrete command resets upon entering a new region $P_{i'j'}$. This may accordingly cause a discontinuity in the vector field,

$$h(x) = f(x) + Bg(x) \tag{34}$$

In order to avoid such discontinuity in the vector field of the system and the control signals during the transitions from P_{ij} to $P_{i'j'}$, we propose to use the following smooth control signal for $x \in P_{ij}$:

$$\begin{aligned} &\hat{g}(x,t) \\ &= \sum_{v_\ell \in V_J} \lambda_\ell(x, v_\ell)[(g_{new}(v_\ell) - g_{old}(v_\ell))erf(\gamma(t-\theta)) \\ &\quad + g_{old}(v_\ell)] + \sum_{v_\ell \in V_{ij}/V_J} \lambda_\ell(x, v_\ell) g_{new}(v_\ell) \end{aligned} \tag{35}$$

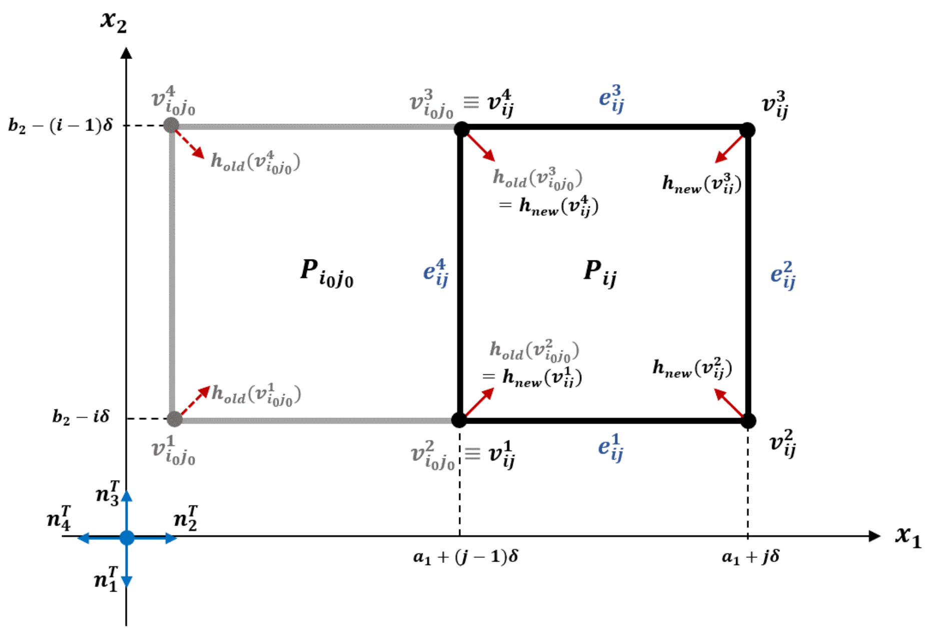


Fig. 5. The implementation of the discrete command $s' = Stay$, after the command $s = Right$: The system moves to the right and enters P_{ij} and stays in this region. The values of the vector field $h_{old}(v)$ and $h_{new}(v)$ are shown with red arrows at the vertices of the corresponding regions. Note that in this case, no change is observed at the vector field at the vertices during this transition to P_{ij} to execute the command $Stay$.

where $V_J \subseteq V_{ij} \cap V_{i'j'}$ is the set of common vertices at which there is a jump in the control value, g_{old} is the previous value of the control signals at vertices before entering the region P_{ij}, and g_{new} is the value of the control signals after entering P_{ij} to drive the system to $P_{i'j'}$. The values of control signals g are calculated using (33). Here, θ is the time instance when the system enters $P_{i,j}$, γ is the transition rate, and the function $erf(t) = 2/\sqrt{\pi}\int_0^t e^{-\lambda^2} d\lambda$ is a special function of sigmoid shape with $erf(t=0)=0$ and $erf(t \geq 3) = 1$. Using the *erf* function, the change in the value of the vector field at the common facet will be smooth. Accordingly, the smooth control values will result in a smooth vector field:

$$\hat{h}(x,t) = f(x) + B\hat{g}(x,t) \tag{36}$$

Theorem 3: The proposed hybrid controller in (25) can drive the system to any adjacent rectangle in a finite time, or keep the system in any rectangle, by generating smooth control signals in the form of (35).

Proof: By construction, from (35), the value of the control signal $\hat{g}$ is smooth despite the fact that the values of g at the vertices on the common facet may change when the system transits from one region to another region. Since $\hat{g}$ has no jump, so does $\hat{h} = f + B\hat{g}$.

Now, we need to show that using the proposed smooth controller, the closed-loop system can implement any generated discrete command (stay in a rectangle or transit to an adjacent rectangle) as it is discussed through the following cases:

Case 1: The discrete command *Stay*: The discrete command *Stay* requires the system to remain inside its current partition. Without loss of generality, assume that the system just moved from the region $P_{i_0j_0}$ to P_{ij} following the discrete command $s = Right$. Now, consider the new discrete command is $s' = Stay$. This has been shown in Fig. 5.

To implement $s' = Stay$, according to (33) and Table III, we chose the value of g at the vertices so that:

$$h_{new}(v_1) = \beta_u \begin{pmatrix}1\\1\end{pmatrix}, h_{new}(v_2) = \beta_u \begin{pmatrix}-1\\1\end{pmatrix}$$
$$h_{new}(v_3) = \beta_u \begin{pmatrix}-1\\-1\end{pmatrix}, h_{new}(v_4) = \beta_u \begin{pmatrix}1\\-1\end{pmatrix} \tag{37}$$

The previous values of h at the common vertices for the command *Right* are shown in Fig. 5. As it can be seen in this figure, at two common vertices of these regions ($v^2_{i_0j_0} \equiv v^1_{ij}$ and $v^3_{i_0j_0} \equiv v^4_{ij}$) the values of $h(v)$ do not change, and hence, $h_{new}(v^4_{ij}) = h_{old}(v^4_{ij})$ and $h_{new}(v^1_{ij}) = h_{old}(v^1_{ij})$. Therefore, $\hat{h}$, calculated by (36), and h, calculated by (34), will be the same.

We show that using (36) to construct $\hat{h}$, the system cannot leave the region P_{ij} from any of its facets, when the discrete command is $Stay$.

The common facet between $P_{i_0j_0}$ and P_{ij}, is $e^4_{ij} = \{x \in P_{ij} | x_1 = a_1 + (j-1)\delta\}$ with the outer normal vector $n_4^T = (-1, 0)$. With the values of $h_{new}(v)$ defined in (37), the vector field $\hat{h}(x,t)$ at all the points over the facet e^4_{ij} will be:

$$\hat{h}(x,t) = h(x) = \sum_{v_\ell \in \{v^1_{ij}, v^4_{ij}\}} \lambda(x, v_\ell) h_{new}(v_\ell) \tag{38}$$

Therefore, we have:

$$\begin{aligned} n_4^T\hat{h}(x,t) &= n_4^T h(x) \\ &= (-1,0)h(x) \\ &= (-1,0)\sum_{v_\ell\in\{v_{ij}^1,v_{ij}^4\}}\lambda(x,v)h(v) \\ &= -\beta_u\sum_{v_\ell\in\{v_{ij}^1,v_{ij}^4\}}\lambda(x,v) \\ &= -\beta_u<0 \end{aligned} \tag{39}$$

as $\beta_u>0$.

This means that the vector field $\hat{h}(x,t)=h(x)$ at all points on the facet e_{ij}^4 is always toward the inside of the rectangle, and hence, the system cannot exit the rectangle P_{ij} from this facet. Similar reasoning is valid for all other three facets of this rectangle, meaning that the trajectories of the system using the generated control signal, $\hat{g}(x,t)$, never exit P_{ij} from any of its facets, and always remain inside this partition.

Case 2: The discrete command Up:

Without loss of generality, assume that the system just moved from the region $P_{i_0j_0}$ to P_{ij} following the discrete command $s=Right$. Now, consider the new discrete command is $s'=Up$, which rerquires the system to transit to $P_{i'j'}$. This has been shown in Fig. 6. We show that for this case, with the generated control law $\hat{g}$ and the resulting vector field $\hat{h}$, the system exits P_{ij} from the desired facet $e_{ij}^3=\{x\in P_{ij}|x_2=b_2-(i-1)\delta\}$, with the outer normal $n_3^T=(0,1)$. For this purpose, based on (33) and Table III, the values of the control signal at the vertices are selected so that the values of the vector field h at the vertices are as follows:

$$\begin{aligned} h_{new}(v_1)&=\beta_u\begin{pmatrix}1\\1\end{pmatrix}, h_{new}(v_2)=\beta_u\begin{pmatrix}-1\\1\end{pmatrix} \\ h_{new}(v_3)&=\beta_u\begin{pmatrix}-1\\1\end{pmatrix}, h_{new}(v_4)=\beta_u\begin{pmatrix}1\\1\end{pmatrix} \end{aligned} \tag{40}$$

In general, if $s=s'$, no change in the value of $h(v\in V_{ij})$ will occur. However, if $s\neq s'$, a change in the direction of the control value at the common vertices among the three partitions $P_{i_0j_0}$, P_{ij}, and $P_{i'j'}$, i.e., $V_c=V_{i_0j_0}\cap V_{ij}\cap V_{i'j'}$, is inevitable. For example, here, since $s=Right\neq s'=Up$, it can be easily verified that the value of the vector field changes at the common vertex $V_c=V_{i_0j_0}\cap V_{ij}\cap V_{i'j'}=\{v_{i_0j_0}^3\equiv v_{ij}^4\}$ as shown in Fig. 6. At this vertex, we have:

$$\begin{cases} h_{old}(v_{i_0j_0}^3)=\beta_u(1,-1)^T;\ \text{when } x\in P_{i_0j_0} \\ h_{new}(v_{ij}^4)=\beta_u(1,1)^T;\ \text{when } x\in P_{ij} \end{cases} \tag{41}$$

Hence, at this vertex, we have $h_{new}(v)-h_{old}(v)=(0,2\beta_u)^T$. For all other vertices, there is no jump in the values of the vector field as shown in Fig. 6.

Applying the control law in (35), the smoothened vector field inside the region P_{ij} will be:

$$\begin{aligned} \hat{h}(x,t)=&\sum_{v\in V_{ij}/V_C}\lambda(x,v)h_{new}(v) \\ &+\sum_{v\in V_C}\lambda(x,v)[(h_{new}(v)-h_{old}(v))erf(\gamma(t-\theta)) \\ &+h_{old}(v)] \end{aligned} \tag{42}$$

Now we show that starting from any $x\in P_{ij}$, the system's trajectory cannot leave P_{ij} from any of its facets other than e_{ij}^3. Consider, for example, the facet $e_{ij}^1=\{x\in P_{ij}|x_2=b_2-i\delta\}$ with the outer normal vector $n_1^T=(0,-1)$. At any points x on the facet e_{ij}^1, we have:

$$\begin{aligned} n_1^T\hat{h}(x,t)&=(0,-1)\ \hat{h}(x,t) \\ &=(0,-1)\sum_{v\in\{v_{ij}^1,v_{ij}^2\}}\lambda(x,v)h_{new}(v) \\ &=-\beta_u\sum_{v\in\{v_{ij}^1,v_{ij}^2\}}\lambda(x,v) \\ &=-\beta_u<0 \end{aligned} \tag{43}$$

This means that the system trajectories cannot leave the rectangle P_{ij} from the facet e_{ij}^1. Similar reasoning is valid for all other facets except the desired exit facet e_{ij}^3.

Now, consider the facet $e_{ij}^3=\{x\in P_{ij}|x_2=b_2-(i-1)\delta\}$, with the normal outer vector $n_3^T=(0,1)$. Next, we show that the vector field at all points inside the region P_{ij} will have a positive value along with n_3^T, guaranteeing that the system trajectories will leave the partition from e_{ij}^3 in a finite time.

Knowing that $V_C=\{v_{ij}^4\}$, for any $x\in P_{ij}$ from (42), the vector field will be:

$$\begin{aligned} \hat{h}(x,t)=&\sum_{v\in\{v_{ij}^1,v_{ij}^2,v_{ij}^3\}}\lambda(x,v)h_{new}(v) \\ &+\lambda(x,v_{ij}^4)[(h_{new}(v_{ij}^4)-h_{old}(v_{ij}^4))erf(\gamma(t-\theta)) \\ &+h_{old}(v_{ij}^4)] \end{aligned} \tag{44}$$

According to (41), $h_{new}(v_{ij}^4)-h_{old}(v_{ij}^4)=(0,2\beta_u)^T$, and $h_{old}(v_{ij}^4)=\beta_u(1,-1)^T$. Also, $erf(\gamma(t-\theta))=1$ for $(t-\theta)\geq 3/\gamma$. Therefore, for $t\geq\theta+3/\gamma$, we have:

$$\begin{aligned} \hat{h}(x,t)=&\sum_{v\in\{v_{ij}^1,v_{ij}^2,v_{ij}^3\}}\lambda(x,v)h_{new}(v) \\ &+\beta_u\lambda(x,v_{ij}^4)(1,1)^T \end{aligned} \tag{45}$$

According to (40), for $v\in\{v_{ij}^1,v_{ij}^2,v_{ij}^3\}$, we have $n_3^T h_{new}(v)=(0,1)h_{new}(v)=\beta_u$. Therefore,

$$\begin{aligned} n_3^T\hat{h}(x,t)&=\beta_u\sum_{v\in\{v_{ij}^1,v_{ij}^2,v_{ij}^3\}}\lambda(x,v)+\beta_u\lambda(x,v_{ij}^4) \\ &=\beta_u\sum_{v\in V_{ij}}\lambda(x,v)=\beta_u>0 \end{aligned} \tag{46}$$

We showed that the system cannot leave P_{ij} from any of the facets other than e_{ij}^3. We also showed that for all the points inside the region P_{ij}, after the transition time,

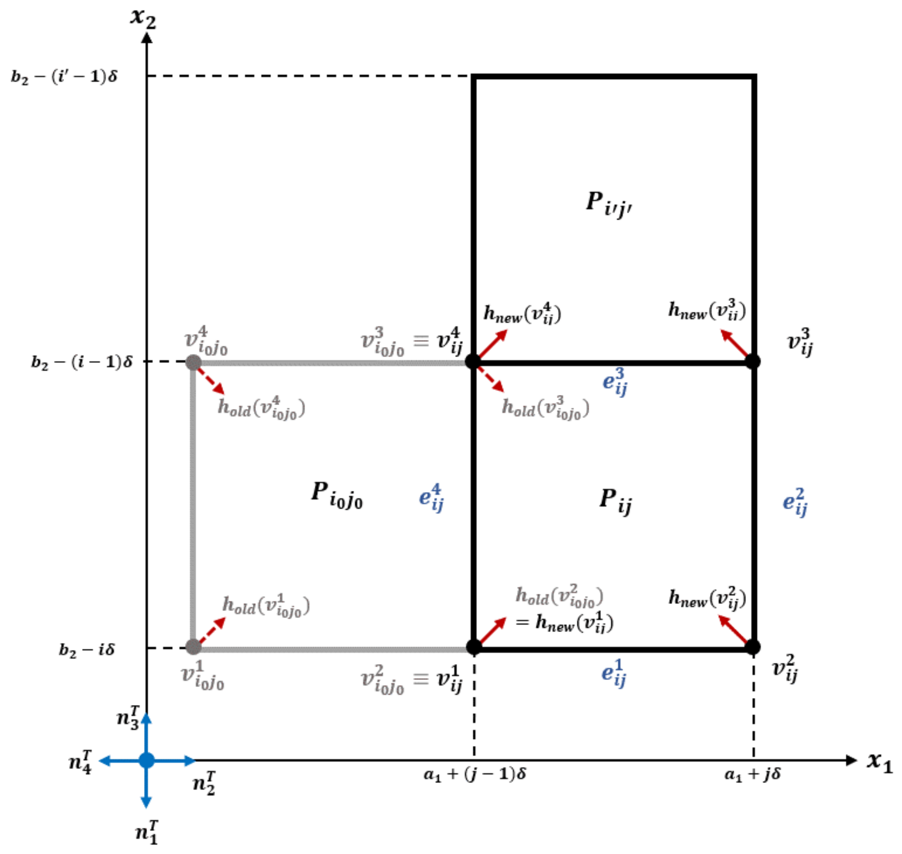


Fig. 6. The implementation of the discrete command $s' = Up$, issued after the command $s = Right$. By the discrete command $s = Right$, the system leaves $P_{i_0j_0}$ and moves to the right and enters P_{ij}. Then, after the discrete command $s' = Up$, the system moves up toward the region $P_{i'j'}$. The values of vector field $h_{old}(v)$ (when the system was at $P_{i_0j_0}$) and $h_{new}(v)$ (when the system is at P_{ij} and moves toward $P_{i'j'}$) are shown with red arrows at the vertices of the corresponding regions. A change can be observed in h at the common vertex among the three regions, v_{ij}^4, as $h_{old}(v_{ij}^4 \equiv v_{i_0j_0}^3) \neq h_{new}(v_{ij}^4)$.

$t \geq \theta + 3/\gamma$, we have $n_3^T \hat{h}(x,t) > 0$. We then can conclude that with the command $s' = Up$, the system will leave P_{ij} from the facet e_{ij}^3, when $s = Right$. For all other sequences of discrete commands, a similar argument can be made, concluding that the control law $\hat{g}$ in (35) generates a smooth vector field $\hat{h}$ which smoothly drives the system to exit from the desired exit facets based on the discrete command $s' \in \{Left, Right, Down, Up\}$. ■

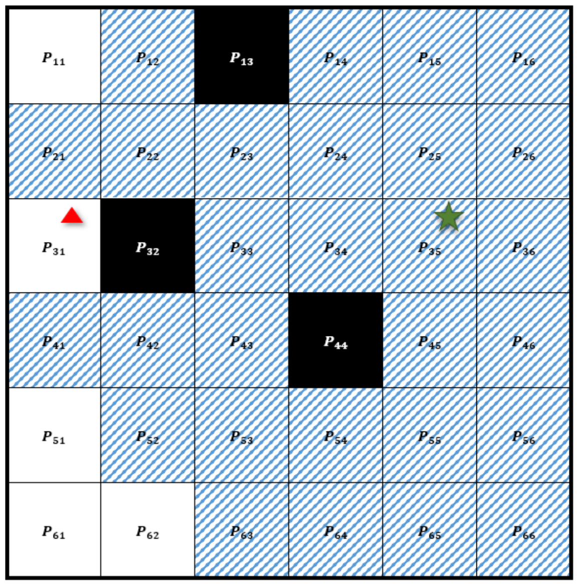
VII. Simulation Results

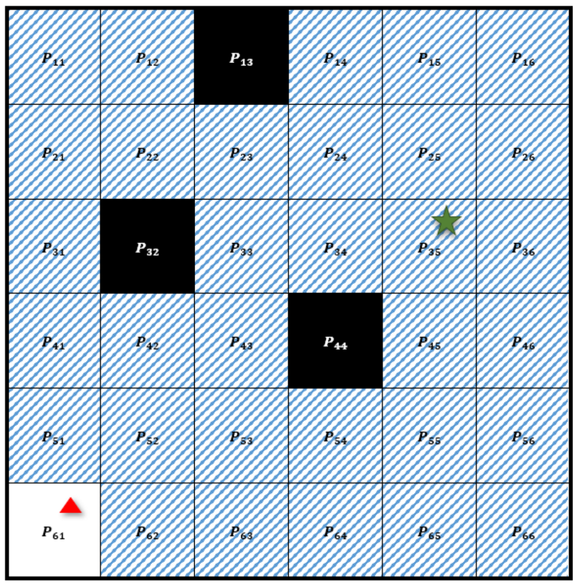
Consider a dynamic reach-avoid scenario with an attacking vehicle with the continuous dynamics $\dot{x} = u$, where $x \in P = [0,12] \times [0,12]$ and $u \in U = [-2,2] \times [-2,2]$. For the simplicity, assume that $\delta_T = \delta_D = 2\sqrt{2}$, which results in the partitioning length $\delta = 2$ according to (1). The target is located at $x_t = (5,9)^T \in P_{35}$. Also, there are three static obstacles in partitions P_{13}, P_{23}, and P_{44}, which the vehicles have to avoid. Here, we consider three different cases where the defender is either in region P_{31}, P_{61}, or P_{14}.

We present the solutions of Problems 1 and 2 using the proposed framework. The first problem is to find the winning initial regions for the attacker to win the game using the proposed algorithms. For this purpose, a RAGS game structure is formulated according to (19). Given this game structure as input, Algorithm 1 is executed and has resulted in winning initial states W_{init}^a for the three scenarios with different initial positions of the defender, as shown in Fig. 7.

Remark 2: The obtained winning set W_{init}^a for each scenario shows the set of initial positions from which the attacker can win the game for any action of the defender. However, it does not necessarily mean that the attacker could not win starting from other partitions, as it depends on the behavior of the defender. In other words, for bad actions of the defender, the attacker still can win the game starting from outside W_{init}^a but it is not guaranteed, and it depends on the actions of the defender.

Next, we solve Problem 2 to construct a hybrid controller, H, in the form of (25) to generate control signals for the control of the attacker in order to win the game. Such a controller is realizable only if the initial position of the attacker is in the set of winning initial partitions W_{init}^a. For this purpose, first, we execute Algorithm 2 to obtain the winning discrete strategies to make a decision on the transitions over the partitions. Then, the hybrid controller H in (25) is constructed. This controller is used for three arbitrary behaviors of the defender for which the resulting


(a) $x_d(0) \in P_{31}$


(b) $x_d(0) \in P_{61}$

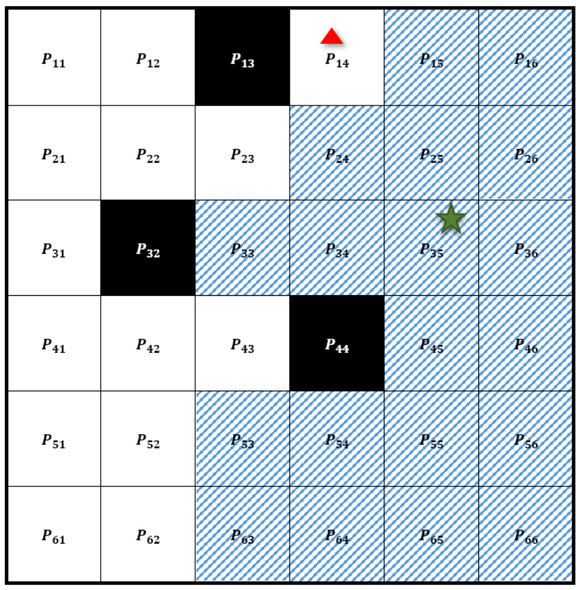

(c) $x_d(0) \in P_{14}$

Fig. 7. The winning initial regions, W^a_{init}, for three different scenarios with different initial positions of the defender. If the attacker starts from any point in W^a_{init}, by applying the proposed controller, it is guaranteed that the attacker will reach the target before being captured by the defender. The red triangles show the initial positions of the defender, the green stars show the positions of the target, and the blue dashed regions show the winning initial set for the attacker.

trajectories are shown in Fig. 8.

As it can be seen in Fig. 8, the proposed hybrid controller H is able to generate a winning continuous path for the attacker, in reaction to adversarial behaviors of the defender. Further, for the three discussed examples in this section, the smooth control signals are shown in Fig. 9.

VIII. CONCLUSION

This paper developed a hybrid symbolic control technique to addresses the path planning and control of autonomous vehicles involved in a dynamic adversarial reach-avoid scenario. The temporal logic formula in the form of GR(1) format was used to describe all assumptions and requirements of interacting vehicles in a dynamic form. The reach-avoid problem was then formulated as a two-player. Based on the target's position and the initial position of the defender, the solution of the game was found using μ-calculus operators over a complete lattice, which consists of winning initial regions from which the attacker can win the game for any action of the defender. The winning strategies were

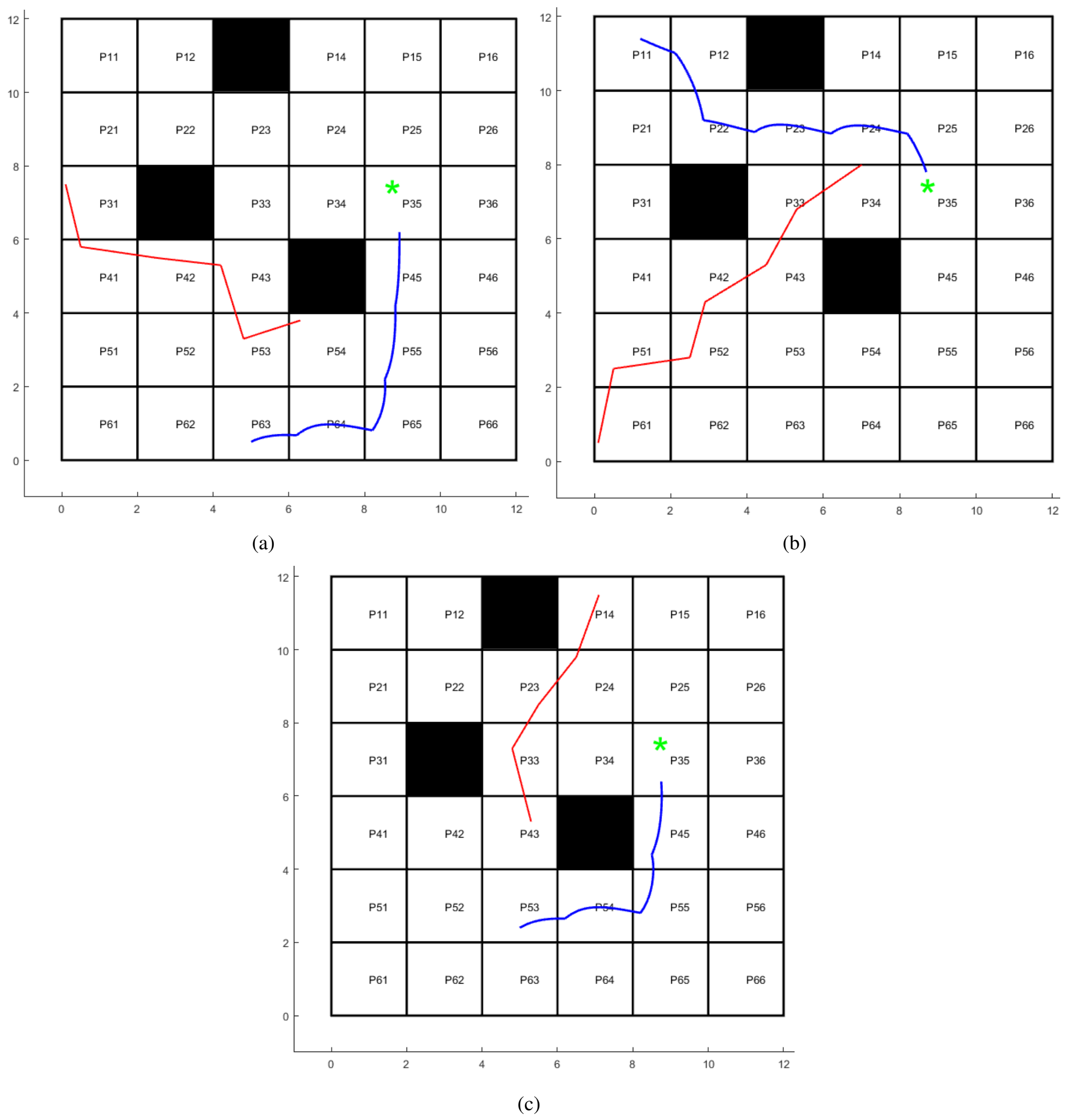
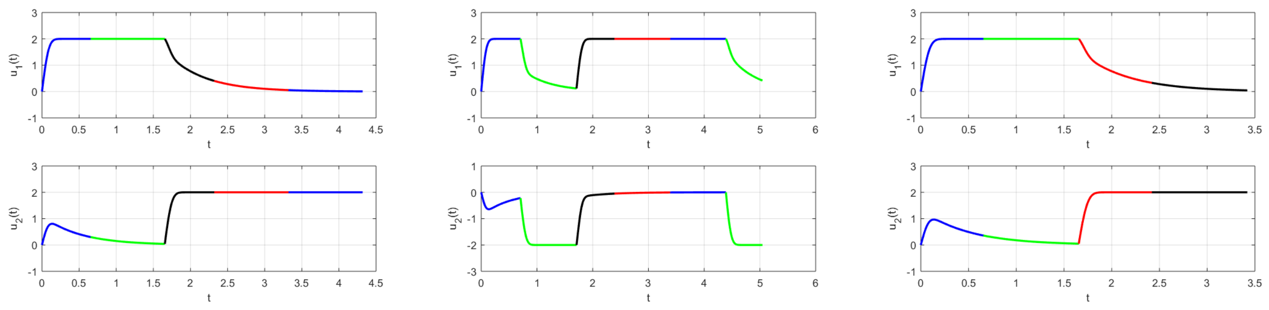


Fig. 8. Simulation results for three different games, in which the attacker has started from one of the regions in winning initial regions shown in Fig. 7, and has won the game (reached the target before being captured by the defender).



(a) $x_d(0) \in P_{31}$ (b) $x_d(0) \in P_{61}$ (c) $x_d(0) \in P_{14}$

Fig. 9. The control signals for the attacker, in scenarios (a), (b), and (c) in Fig. 8, with no discontinuity.

then extracted. A smooth hybrid controller was designed to execute the winning strategies. Illustrative examples and simulation results were provided.

Acknowledgment

The authors would like to acknowledge the support from NSF under the award number 1832110 and Air Force Research Laboratory and OSD for sponsoring this research under agreement number FA8750-15-2-0116. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory, OSD or the U.S. Government.

References

[1] J. Barraquand, B. Langlois, J.-C. Latombe, Numerical potential field techniques for robot path planning, IEEE Transactions on Systems, Man and Cybernetics 22 (2) (1992) 224–241.
[2] C. Torras, Robot motion planning: A survey, in: Teleoperation: Numerical Simulation and Experimental Validation, Springer, 1992, pp. 27–39.
[3] E. M. Clarke, E. A. Emerson, A. P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Transactions on Programming Languages and Systems (TOPLAS) 8 (2) (1986) 244–263.
[4] P. Wolper, Temporal logic can be more expressive, in: 22nd IEEE Annual Symposium on Foundations of Computer Science (SFCS), 1981, pp. 340–348.
[5] X. D. Koutsoukos, P. J. Antsaklis, J. A. Stiver, M. D. Lemmon, Supervisory control of hybrid systems, Proceedings of the IEEE 88 (7) (2000) 1026–1049.
[6] P. J. Antsaklis, A. Nerode, Hybrid control systems: An introductory discussion to the special issue, IEEE Transactions on Automatic Control 43 (4) (1998) 457–460.
[7] R. Alur, T. A. Henzinger, G. Lafferriere, G. J. Pappas, Discrete abstractions of hybrid systems, 2000 Proceedings of the IEEE 88 (7) 971–984.
[8] A. Karimoddini, H. Lin, Hierarchical hybrid symbolic robot motion planning and control, Asian Journal of Control 17 (1) (2015) 23–33.
[9] A. Pnueli, The temporal logic of programs, in: 18th IEEE Annual Symposium on Foundations of Computer Science, 1977, pp. 46–57.
[10] E. A. Emerson, Temporal and modal logic, handbook of theoretical computer science (jan van leeuwen, ed.) (1990).
[11] M. Y. Vardi, An automata-theoretic approach to linear temporal logic, in: Logics for concurrency, Springer, 1996, pp. 238–266.
[12] Y. Chen, K. Deng, C. Belta, Multi-agent persistent monitoring in stochastic environments with temporal logic constraints, in: 2012 IEEE 51st Annual Conference on Decision and Control (CDC), pp. 2801–2806.
[13] C. Wiltsche, J. Lygeros, F. A. Ramponi, Synthesis of an asynchronous communication protocol for search and rescue robots, in: 2013 IEEE European Control Conference (ECC), pp. 1256–1261.
[14] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, G. J. Pappas, Symbolic planning and control of robot motion [grand challenges of robotics], IEEE Robotics & Automation Magazine 14 (1) (2007) 61–70.
[15] C. Belta, V. Isler, G. J. Pappas, Discrete abstractions for robot motion planning and control in polygonal environments, IEEE Transactions on Robotics 21 (5) (2005) 864–874.
[16] H. Kress-Gazit, G. E. Fainekos, G. J. Pappas, Where's waldo? sensor-based temporal logic motion planning, in: 2007 IEEE International Conference on Robotics and Automation, pp. 3116–3121.
[17] M. Kloetzer, C. Belta, Temporal logic planning and control of robotic swarms by hierarchical abstractions, IEEE Transactions on Robotics 23 (2) (2007) 320–330.
[18] G. E. Fainekos, A. Girard, H. Kress-Gazit, G. J. Pappas, Temporal logic motion planning for dynamic robots, Automatica 45 (2) (2009) 343–352.
[19] K. Margellos, J. Lygeros, Hamilton-jacobi formulation for reach-avoid problems with an application to air traffic management, in: 2010 IEEE American Control Conference (ACC), pp. 3045–3050.
[20] J. S. McGrew, J. P. How, B. Williams, N. Roy, Air-combat strategy using approximate dynamic programming, Journal of guidance, control, and dynamics 33 (5) (2010) 1641–1654.
[21] M. Aigner, M. Fromme, A game of cops and robbers, Discrete Applied Mathematics 8 (1) (1984) 1–12.
[22] D. Bhadauria, K. Klein, V. Isler, S. Suri, Capturing an evader in polygonal environments with obstacles: The full visibility case, The International Journal of Robotics Research 31 (10) (2012) 1176–1189.
[23] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, R. Motwani, A visibility-based pursuit-evasion problem, International Journal of Computational Geometry & Applications 9 (04n05) (1999) 471–493.
[24] B. P. Gerkey, S. Thrun, G. Gordon, Visibility-based pursuit-evasion with limited field of view, The International Journal of Robotics Research 25 (4) (2006) 299–315.
[25] S. Russell, P. Norvig, A. Intelligence, A modern approach, Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25 (1995) 27.
[26] S. Thrun, W. Burgard, D. Fox, Probabilistic robotics, MIT press, 2005.
[27] M. P. Vitus, C. J. Tomlin, Closed-loop belief space planning for linear, gaussian systems, in: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 2152–2159.
[28] H. Huang, J. Ding, W. Zhang, C. J. Tomlin, A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag, in: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 1451–1456.
[29] H. Huang, J. Ding, W. Zhang, C. J. Tomlin, Automation-assisted capture-the-flag: A differential game approach, IEEE Transactions on Control Systems Technology 23 (3) (2015) 1014–1028.
[30] N. Piterman, A. Pnueli, Y. Sa'ar, Synthesis of reactive (1) designs, in: Verification, Model Checking, and Abstract Interpretation, Springer, 2006, pp. 364–380.
[31] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, Y. Sa'ar, Synthesis of reactive (1) designs, Journal of Computer and System Sciences 78 (3) (2012) 911–938.
[32] H. Kress-Gazit, G. E. Fainekos, G. J. Pappas, Temporal-logic-based reactive mission and motion planning, IEEE Transactions on Robotics 25 (6) (2009) 1370–1381.
[33] L. Shamgah, T. G. Tadewos, A. Karimoddini, A. Homaifar, Path planning and control of autonomous vehicles in dynamic reach-avoid scenarios, in: 2018 IEEE Conference on Control Technology and Applications (CCTA), pp. 88–93.
[34] L. Shamgah, A. Karimoddini, A. Homaifar, A symbolic motion planning approach for the reach-avoid problem, in: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 3955–3960.
[35] C. Belta, L. C. Habets, Controlling a class of nonlinear systems on rectangles, IEEE Transactions on Automatic Control 51 (11) (2006) 1749–1759.
[36] J. J. Park, B. Kuipers, A smooth control law for graceful motion of differential wheeled mobile robots in 2d environment, in: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 4896–4902.
[37] Y. B. Shtessel, I. A. Shkolnikov, A. Levant, Smooth second-order sliding modes: Missile guidance application, Automatica 43 (8) (2007) 1470–1476.
[38] T. Basar, G. J. Olsder, Dynamic noncooperative game theory, Vol. 23, Siam, 1999.
[39] G. Birkhoff, Lattice theory, Vol. 25, American Mathematical Soc., 1940.