Roadmap Based Pursuit-Evasion and Collision Avoidance

Volkan Isler, Dengfeng Sun and Shankar Sastry
CITRIS
University of California Berkeley, CA, 94720
Email: {isler,sundf,sastry}@eecs.berkeley.edu

***Abstract*— We study pursuit-evasion games for mobile robots and their applications to collision avoidance. In the first part of the paper, under the assumption that the pursuer and the evader (possibly subject to physical constraints) share the same roadmap to plan their strategies, we present sound and complete strategies for three different games. In the second part, we utilize the pursuit-evasion results to post-process the workspace and/or configuration space and obtain a collision probability map of the environment. Next, we present a probabilistic method to utilize this map and plan trajectories which minimize the collision probability for independent robots.**

I. Introduction

Motion planning is one of the fundamental problems in robotics. Broadly speaking, it is the problem of selecting a set of actions or controls which, upon execution, would take the robot from a given initial state to a given final state.

Motion planning is a challenging problem whose complexity stems primarily from two sources: environment complexity and system complexity. The former refers to the complexity of planning trajectories for perhaps simple robots but in complex environments. For example, finding shortest paths in 3D is an NP-hard problem [1]. By system complexity, we refer to the complexity of planning trajectories for complex, high degree-of-freedom systems even in the absence of obstacles. Traditionally, the former complexity is addressed by algorithmic/combinatorial techniques and data structures (e.g. Dijkstra's algorithm on visibility graphs) whereas the latter type of complexity is addressed using control-theoretic techniques.

Developing techniques that address both types of complexity has been the focus of significant recent research (see [2], [3] for an overview). Notable success has been achieved by sampling-based roadmap methods [4], [5] which we review in Section II.

In this paper, we present algorithms which solve two dynamic motion planning problems that take place on roadmaps: pursuit-evasion and collision avoidance.

In the first part of the paper, we study pursuit-evasion games that take place on roadmaps. In a pursuit-evasion game, a pursuer tries to capture an evader while the evader is trying to avoid capture. In robotics, solutions to pursuit-evasion games are used to obtain worst-case guarantees to collision-avoidance problems and, in general, to motion planning in dynamic environments. For example, suppose two robots are independently operating in the same workspace. When there is a danger of collision, each robot can execute an evasion strategy obtained as the solution of a pursuit-evasion game where the other robot acts as a pursuer trying to collide. Such a solution would guarantee that each robot will avoid collision regardless of the actions of the other robot. In Section III, we present solutions to pursuit-evasion games on roadmaps. Our model studies interactions between a pursuer and an evader who use the same roadmap to plan their trajectories. Under this assumption, we present algorithms based on the dynamic programming principle to generate *sound and complete* strategies for both players.

The pursuit-evasion models we study in the first part assume that the players have *globally conflicting objectives*. In this model, the pursuer is truly adversarial and its objective is to collide with the evader. In most motion-planning settings, such a model can be too strict for modeling collision-avoidance. Typically, robots plan their trajectories independently. However, when they get close to each other, they may switch to a reactive collision-avoidance mode and become unpredictable. At this point, it is desirable to have the worst-case guarantees given by the game theoretic formulation. In the second part of the paper (Section IV) we propose a model for such scenarios. In this formulation, independent (neither collaborating nor conflicting) agents operate in the workspace while avoiding collisions locally. We utilize the results of the first part to define *local pursuit-evasion games* where the players react to each other only when they are within a given interaction zone. Under this model we show how

- the workspace and/or configuration space can be post-processed to obtain a collision probability map of the environment,
- the players can compute worst-case collision avoidance strategies after they are enter the interaction zone, and
- compute optimal trajectories that minimize the expected probability of a collision.

We start with an overview of related literature.

A. Related work

Due to their many applications, literature on pursuit-evasion games is vast. To model the adversarial nature of the game, pursuit-evasion games are usually studied in a game theoretic framework [6], [7]. The conditions under which the pursuer can capture the evader are obtained by studying a Hamilton-Jacobi-Isaacs equation which brings together the system equa-

tions of the pursuer and the evader. This approach has the advantage of yielding a closed-form solution of the game. Unfortunately, as the environments get complicated, solving Hamilton-Jacobi-Isaacs equations become intractable.

Recently, there has been increasing interest in developing pursuit strategies (which incorporate sensing limitations) to capture intelligent evaders contaminating a complex environment [8], [9], [10].

In robotics, complex environments are modeled either topologically (usually with a graph-based representation) or geometrically (usually a polygonal/polyhedral representation). Classical work on pursuit-evasion games on graphs includes [11], [12], [13], [14]. See [15], [16] for recent results. Pursuit-evasion games in polygonal environments are also well studied. See [17], [18], [19] and recently [20], [21], [22].

As mentioned earlier, solving pursuit-evasion games between robots subject to physical constraints (such as turning radius) which takes place in a complex environment is very challenging. Note that such problems inherit all difficulties of traditional motion-planning. To tackle these two difficulties, our approach in this paper is to reduce a game between two robots subject to physical constraints to a pursuit-evasion game that takes place on a graph, a.k.a. the roadmap. We defer the details to Section II.

Our work is also directly related to the work in multi-robot planning and collision avoidance. For basic results in multiple robot planning see [23]. For recent work in collision avoidance and multi-robot planning see [24], [25], [26], [27]. Recent research on extending the probabilistic roadmap framework to multi-robot and dynamically changing environments can be found in [28], [29], [30], [31].

II. Notation and Preliminaries

In this section, we present the main concepts and the notation used throughout the paper. Let $\mathcal{C}$ be the configuration space of a robot. We assume that we are given a discrete set of points $\tilde{\mathcal{C}}$ that represent the configuration space. Such a set can be obtained, for example, by randomly sampling the configuration space as in the case of probabilistic roadmaps or by putting a grid on the configuration space which is practical for low dimensional configuration spaces. When the robot is in a configuration $c \in C$, we use the notation $W(c)$ to denote the subset of the workspace occupied by the robot.

We are also given $\mathcal{U} = \{u_1, \ldots, u_K\}$, a set of deterministic, 'simple' controllers for the robot. Given two arbitrary configurations $c_i, c_j \in \mathcal{C}$, the notation $c_i \rightarrow_u c_j$ is interpreted as: *when in configuration c_i, if the robot executes controller $u \in \mathcal{U}$, it reaches configuration c_j in a single time unit.* The significance of the simplicity of the controllers in $\mathcal{U}$ lies in easy verification of $c_i \rightarrow_u c_j$ for arbitrary configurations.

The main data structure that will be utilized throughout the paper is the notion of a *timed roadmap*. A timed roadmap is a graph $G = (V, E)$. The vertex set V is given by the discrete sampling of the configuration space, $V = \tilde{\mathcal{C}}$. Let $c : V \rightarrow \tilde{\mathcal{C}}$ be a bijection such that $c(v)$ returns the configuration associated with the vertex v. The edge set E is obtained as follows: For any $v_i, v_j \in V$, the edge $(v_i, v_j) \in E$ if and only if there exists a simple controller $u \in \mathcal{U}$ such that $c(v_i) \rightarrow_u c(v_j)$. The function $u : E \rightarrow \mathcal{U}$ returns the controller associated with an edge [1]. For a given vertex v, $N(v) = \{v' : (v, v') \in E\}$ denotes the neighborhood of v.

For we are interested in pairwise interactions between the robots, we define the *static interaction space* $I_s = \{(v_i, v_j) : v_i, v_j \in V\}$. Similarly, the *dynamic interaction space* $I_d = \{(e_i, e_j) : e_i, e_j \in E\}$. For pursuit-evasion games, we assume the availability of a capture function, $capture : I_d \rightarrow \{true, false\}$ which, given $(e_i, e_j) \in I_d$ with $e_i = (v_i, v_i')$ and $e_j = (v_j, v_j')$, $capture((e_i, e_j))$ returns true if and only if capture occurs while the first robot is executing $v_i \rightarrow_{u(e_i)} v_i'$ and the second robot is executing $v_j \rightarrow_{u(e_j)} v_j'$. Again, the simplicity of the controllers become a crucial factor in computing the capture function. In addition, note that for simplicity of notation we assume that the robots use the same roadmap for planning their trajectories[2].

We demonstrate these concepts with the well-known Dubins car, which will be used as an ongoing example for the pursuit-evasion results.

A. Dubins Vehicle

We model each agent as a Dubins car by assuming that an agent moves at a constant forward speed u_s and its maximum steering angle is $\phi_{\max}$, which results in a minimum turning radius $\rho_{\min}$. As the agent travels, its movement is governed by the following dynamics:

$$\begin{aligned} \dot{x}(t) &= u_s \cos\theta(t) \\ \dot{y}(t) &= u_s \sin\theta(t) \\ \dot{\theta}(t) &= u \end{aligned} \tag{1}$$

where $(x(t), y(t)) \in \mathbb{R}^2$ is the position of the agent at time t and $\theta(t)$ is the orientation; u is chosen from the interval $\mathcal{U} = [-\tan\phi_{\max}, \tan\phi_{\max}]$.

Let us define three motion primitives such that in each motion primitive a constant steering u is applied:

Symbol	S	L	R
Steering u	0	$-u_{\max}$	$u_{\max}$

S primitive means that an agent moves straight ahead, while L and R primitives mean that an agent turns as sharply as possible to the left and right, respectively. The primitive C refers to either L or R.

Dubins showed in [32] that given any two configurations c_1 and c_2, the shortest path can be expressed as a combination of no more than three of these motion primitives. Dubins also showed that only six "words" (combinations of the three motion primitives) are possible:

$$\{LRL, RLR, LSL, RSL, LSR, RSR\}. \tag{2}$$

[1] We assume that there is a unique controller associated with each edge and ambiguities are resolved arbitrarily.

[2] If there are two different roadmaps, we simply define the interaction space as the product of these two graphs.

For the Dubin's car, we start with a discretization of the configuration space $W \times SO(1)$ where W denotes the workspace. We restrict ourselves to three types of basic controllers corresponding to each motion primitive: "turn left α degrees for 1 time-unit", "turn right α degrees for 1 time unit" and "move straight" for 1 time unit. We connect two configurations c_i, c_j if there c_j is reachable from c_i by the execution of a simple controller.

Figure 1 shows a simple workspace and the shortest trajectory of an agent moving from configuration $c_1 = [2, 2, \frac{\pi}{2}]$ to $c_2 = [14, 8, \frac{\pi}{2}]$. In this scenario, the orientation angles in $\tilde{C}$ are coarsely discretized by $\pi/2$.

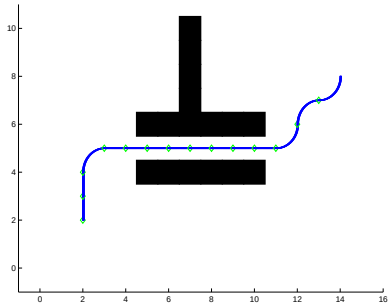

Fig. 1. An agent moves from c_1 to c_2. Diamonds mark the intermediate configurations the agent passes in $\tilde{C}$. Obstacles exist in the areas marked black.

Collision detection: We define a *safety distance* r_s such that whenever the distance between agents s_1 and s_2 is less than r_s, collision occurs. Notice that in practice r_s is time varying depending on both positions and orientations of the agents. In this paper we assume that the agents are in the form of discs and assume that the radius of an agent is r. In this case, $r_s = 2r$.

Since every trajectory of an agent is a combination of the motion primitives, for collision detection between agents, it is enough to check whether there is collision between any $C-C$, $C-S$ and $S-S$ primitives along the two trajectories of the agents.

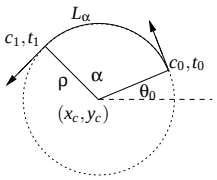

Fig. 2. L primitive

Consider the curve of an L primitive centered at (x_c, y_c) in Figure 2. The curve L_α starts from configuration $c_0 = (x_0, y_0, \frac{\pi}{2} + \theta_0)$ at time t_0 and ends at configuration $c_1 = (x_1, y_1, \frac{\pi}{2} + \theta_0 + \alpha)$ at time t_1. Without loss of generality, assume $t_0 = 0$, $t_1 = T_s$. The coordinates $(x(t), y(t))$ at time $t \in [0, T_s]$ can be derived as:

$$\begin{aligned} x(t) &= x_c + \rho\cos(u_{\max}t + \theta_0) \\ y(t) &= y_c + \rho\sin(u_{\max}t + \theta_0) \end{aligned} \tag{3}$$

Similarly for an R primitive, we have

$$\begin{aligned} x(t) &= x_c + \rho\cos(-u_{\max}t + \theta_0) \\ y(t) &= y_c + \rho\sin(-u_{\max}t + \theta_0) \end{aligned} \tag{4}$$

In the following, we use L primitive as a representative for C type primitives. A similar result can be derived for R primitives.

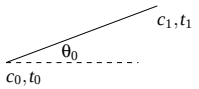

Fig. 3. S primitive

An S primitive is shown in Figure 3, which starts at $c_0 = (x_0, y_0, \theta_0)$ and ends at $c_1 = (x_1, y_1, \theta_0)$. The coordinates at t are (again, we assume $t_0 = 0$, $t_1 = T_s$)

$$\begin{aligned} x(t) &= x_0 + u_s t\cos\theta_0 \\ y(t) &= y_0 + u_s t\sin\theta_0 \end{aligned} \tag{5}$$

Suppose the positions of s_1 and s_2 are $(x_1(t), y_1(t))$ and $(x_2(t), y_2(t))$ at time t. The distance between s_1 and s_2 is

$$d(t) = \sqrt{[x_1(t) - x_2(t)]^2 + [y_1(t) - y_2(t)]^2}. \tag{6}$$

where $x_i(t), y_i(t)$, $i = 1, 2$, are defined in Equations (3, 4, 5). Clearly $d(t)$ is a continuous function on $[0, T_s]$. To detect if collision exists, it is enough to check if

$$\min_{t \in [0, T_s]} d(t) < r_s. \tag{7}$$

III. Solutions to typical pursuit-evasion problems using the timed-roadmap

In this section, we show how typical pursuit-evasion problems can be solved when the motion-plans of the players are restricted to the timed-roadmap.

We use the following formulation for all the games we consider. The games take place between a pursuer $\mathcal{P}$ and evader $\mathcal{E}$. We assume that the players are identical and they plan their trajectories using the same timed-roadmap (Hence, they use the same controller set $\mathcal{U}$). In this section, we consider games with full state information where the players can observe each other's configuration throughout the game. We begin with the following basic version:

GAME 1: Given initial configurations p_0 and e_0, can $\mathcal{P}$ eventually collide with $\mathcal{E}$?

The algorithm in Table I allows us to preprocess the static interaction space and answer this query for all initial conditions.

After the execution of the algorithm, the variable $(p, e).timestamp$ is set to the number of time-steps to a collision after the pursuer and the evader reach the configurations p and e respectively. The following two lemmas show that Algorithm Collision-Detection can used to solve GAME 1 in a sound and complete fashion. We assume, without loss of generality, that the evader wants to maximize the time to collision and the pursuer wants to minimize it.

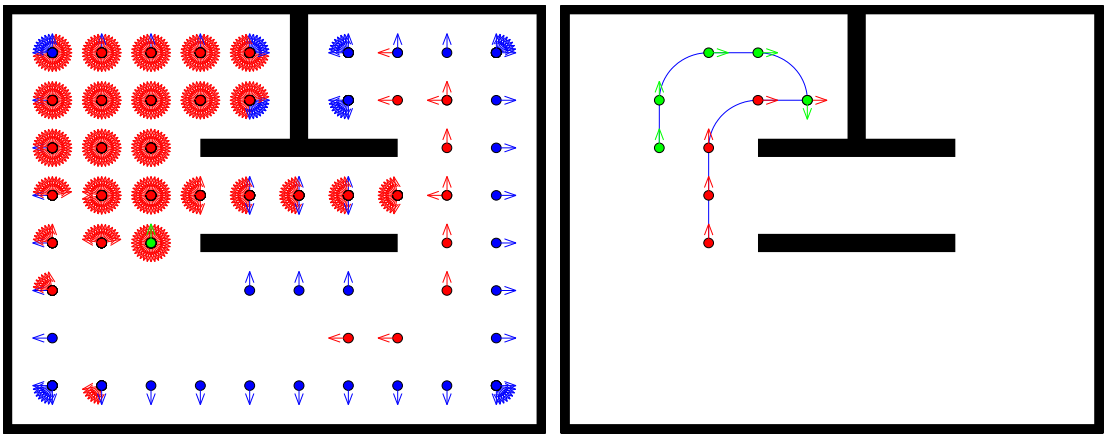

Fig. 4. An instance of GAME 1. **Left:** When the initial configuration of the pursuer is given by the green arrow, it can cause a collision with all the red evader configurations. The evader configurations marked blue yield collisions with the walls. **Right:** A sample collision (green: evader, red: pursuer).

Algorithm Collision-Detection

For each pair $(p,e) \in I_s$
  $(p,e).timestamp \leftarrow \infty$
For t = 1 **to** $|I_s|$,
  For each pair $(p,e) \in I_s$ with $(p,e).timestamp = \infty$
    If $\forall e' \in N(e), \exists p' \in N(p)$ with
      timestamp rule :
        $(p',e').timestamp < t$ **or**
      collision rule :
        $capture((p,p'),(e,e'))$
    then
      $(p,e).timestamp \leftarrow t$

TABLE I
SOLUTION OF GAME 1

Lemma 1: Let p_0 and e_0 be the initial configurations of the players. If at the end of Algorithm Collision-Detection $(p_0,e_0).timestamp$ is finite, then for every evader trajectory, there exists a pursuer trajectory that results in a collision.

Proof: By induction on the timestamp of (p_0,e_0). For the basis, consider initial configurations with $(p_0,e_0).timestamp = 1$. When $(p_0,e_0).timestamp$ is updated, only the collision rule is applicable. Therefore, for every possible action of the evader, the pursuer can cause a collision. For the inductive step, assume all initial conditions with timestamp less than k lead to collision. Suppose $(p_0,e_0).timestamp = k$. Then, by the timestamp rule, no matter where the evader moves, the pursuer can either reach a state whose time-stamp is less than k and cause a collision from then on (by the inductive hypothesis) or directly cause a collision during the transition. ∎

Lemma 2: Let p_0 and e_0 be the initial configurations of the players. If at the end of Algorithm Collision-Detection $(p_0,e_0).timestamp$ is infinite, then for every pursuer trajectory, there exists an evader trajectory that avoids a collision.

Proof:: Let p and e be the current configurations of the pursuer and the evader respectively. We observe that the evader can always postpone collision until the players reach configurations p_f and e_f with $(p_f,e_f).timestamp = 1$. This is because, these are the only configurations where the pursuer can cause a collision for all $e' \in N(e)$. For all other configuration pairs (p,e), there must be a next configuration e' for which the timestamp rule applies. Hence, the evader can move there and postpone collision to the next time step.

Now consider (p_0,e_0). Suppose, for contradiction, $(p_0,e_0).timestamp$ is infinite but the evader can not avoid collision. This means that the current configuration pair $(p_t,e_t).timestamp$ must become one at some time t. However, since $(p_0,e_0).timestamp$ is infinite, there must be a configuration $e' \in N(e)$ such that for every possible $p' \in N(p_0)$, the transition $e \rightarrow e'$, $p \rightarrow p'$ is collision free and the resulting pair (p',e') has timestamp infinite – otherwise the timestamp of (p_0,e_0) would be updated. This argument shows that the evader can guarantee that at all times $(p_t,e_t).timestamp$ remains infinite. This contradicts with an eventual collision because the timestamp would never become 1. ∎

Game 1 is illustrated in Figure 4.

Next, we present a variant of GAME 1, where the evader's goal is to arrive at a configuration e_f and simultaneously avoid a collision.

GAME 2: Given initial configurations p_0 and e_0, and a destination e_f for $\mathcal{E}$, can $\mathcal{P}$ collide with $\mathcal{E}$ before it arrives at e_f?

Algorithm Navigation

For each pair $(p,e) \in I_s$
  $(p,e).timestamp \leftarrow \infty$
For each pair $(p,e) \in I_s$ with $e_f \in N(e)$
  If $\forall p' \in N(p)$ $capture((p,p'),(e,e_f)) = false$ **then**
    $(p,e).timestamp \leftarrow 0$
For t = 1 **to** $|I_s|$,
  For each pair $(p,e) \in I_s$ with $(p,e).timestamp = \infty$
    If $\exists e' \in N(e)$ such that $\forall p' \in N(p)$, $(p',e').timestamp < t$ **then**
      $(p,e).timestamp \leftarrow t$

TABLE II
SOLUTION OF GAME 2

The following lemmas are analogous to Lemmata 1 and 2 and show the correctness of the algorithm. The proofs are

similar and hence omitted.

Lemma 3: Let p_0 and e_0 be the initial configurations of the players. If at the end of Algorithm Navigation, $(p_0, e_0).timestamp = k < \infty$, then the evader can reach e_f in k steps while avoiding a collision.

Lemma 4: Let p_0 and e_0 be the initial configurations of the players. If at the end of Algorithm Navigation $(p_0, e_0).timestamp$ is infinite, then the pursuer can collide with the evader before it reaches e_f.

Finally, we present the solution of a dog-fight game. First we define the capture condition. $capture((p, p'), (e, e'))$ is true if and only if:

(i) during the transition from p to p', the angle between the heading of the pursuer and the ray from the pursuer to the evader becomes less than a threshold

(ii) the distance between the players is less than a threshold

(iii) there are no obstacles between the players

We are now ready to define the dog-fight game:

GAME 3: Given initial configurations p_0 and e_0, can $\mathcal{E}$ capture $\mathcal{P}$ before $\mathcal{P}$ captures $\mathcal{E}$?

Note that during a dog-fight, the roles of the pursuer and the evader are not uniquely defined. The players must both avoid a capture and capture simultaneously. However, Algorithm Collision-Detection can be modified to solve GAME 3 as follows. We run the algorithm with the modified capture function.

Algorithm DogFight

For each pair $(p,e) \in I_s$
  $(p,e).timestamp \leftarrow \infty$
For t = 1 **to** $|I_s|$,
  For each pair $(p,e) \in I_s$ with $(p,e).timestamp = \infty$
    If $\forall e' \in N(e), \exists p' \in N(p)$ with
      timestamp rule :
        $(p',e').timestamp < (e',p').timestamp$ **or**
      collision rule :
        $capture((p,p'),(e,e'))$
    then
      $(p,e).timestamp \leftarrow t$

TABLE III
SOLUTION OF GAME 3

Lemma 5: Let p_0 and e_0 be the initial configurations of $\mathcal{P}$ and $\mathcal{E}$ respectively.

(i) If at the end of Algorithm Collision-Detection $(p_0, e_0).timestamp \leq (e_0, p_0).timestamp$, then $\mathcal{P}$ wins the dog-fight game.

(ii) If, $(p_0, e_0).timestamp \geq (e_0, p_0).timestamp$, then $\mathcal{E}$ wins the dog-fight game.

The proof of Lemma 5 is similar to the proof of Lemma 1.

Note that, it is possible to have $(p_0, e_0).timestamp = (e_0, p_0).timestamp < \infty$ and both players win the game simultaneously as well as $(p_0, e_0).timestamp = (e_0, p_0).timestamp = \infty$ with no winner. These cases are illustrated in Figure 5.

IV. A PROBABILISTIC APPROACH FOR COLLISION AVOIDANCE

In the previous section we modeled the collision-avoidance problem as a pursuit-evasion game where one of the robots tries to cause collision whereas the other tries to prevent it. The advantage of this formulation is that it requires no coordination between the robots and if a solution exists, we obtain a worst-case guarantee for avoiding collisions.

However, for most applications such an adversarial formulation may be too strict. On the other hand of the spectrum, we can have truly cooperative robots, which broadcast their destinations to a central location where a multi-robot collision-free plan is computed. Even though this approach has the advantage of producing optimal, cooperative motion plans, it has two main drawbacks. First, it is centralized and requires that the destinations of all robots are known apriori which may not be feasible for some applications. Second, computation of multi-robot motion plans is provably computationally hard (c.f. [23] for details).

In this section, we propose an intermediate solution. We start with the case of two robots, operating independently in the same workspace. Given the configuration space $\mathcal{C}$, we preprocess it by running the Collision-Detection Algorithm given in Table I. Recall that the output of the algorithm is a timestamp for each pair of configurations which is equal to the maximum length of the path leading to a collision.

Next, for each configuration $c \in \mathcal{C}$, we define *a reaction zone*, $\mathcal{R}(c)$ with the interpretation that a robot in configuration c reacts to another robot in configuration c' only if $c' \in R(c)$. Typically $\mathcal{R}(c)$ is given by configurations that are visible from c whose distances are within a certain threshold.

We say a configuration $c' \in R(c)$ is a *dangerous configuration* for c if $(c', c).timestamp$ is finite. This is analogous to a pursuit-evasion game that starts only if the pursuer enters the reaction zone of the evader. As the robots have no information about their trajectories, configurations which may lead to a possible collision are marked as dangerous. For illustration purposes, in Figure 6 we present a 2D configuration space where two configurations, their reaction zones and corresponding dangerous states are marked.

A. Configuration space post-processing for collision avoidance

The notion of a dangerous configuration allows us to post-process the configuration space to obtain a collision-probability map. Let $p : \mathcal{C} \rightarrow [0,1]$ be our prior belief that the pursuer is present at a given configuration. If we have no information, p will be uniform. When our robot (evader) visits a configuration c, we define the probability of collision at c as:

$$\mathcal{P}[c] = \frac{\sum_{\{c' : c' \text{is dangerous for c}\}} p(c')}{\sum_{\{c' : c' \in \mathcal{R}(c)\}} p(c')} \tag{8}$$

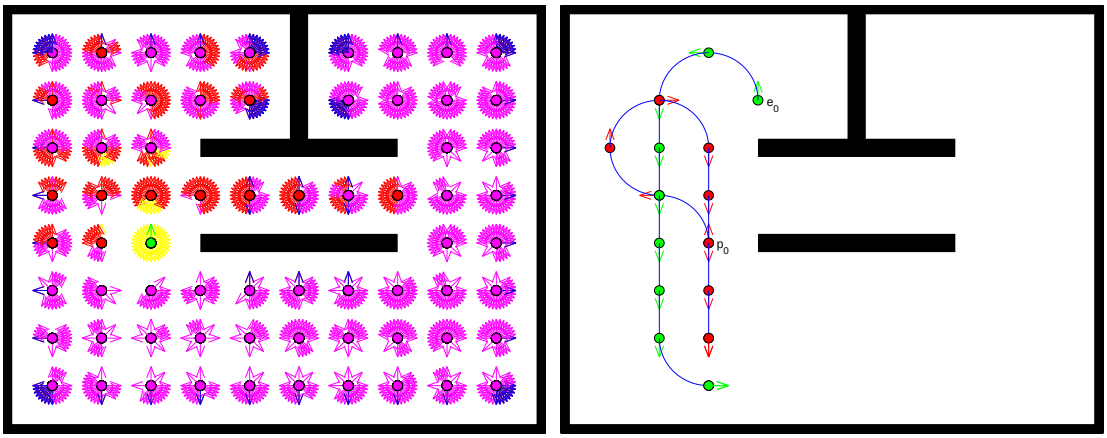

Fig. 5. An instance of GAME 3. **Left:** For the green pursuer initial configuration: If the evader starts with initial configurations in red, it will lose the dog-fight game; in this case, $(p,e).timestamp < (e,p).timestamp$. If the evader starts with initial configurations in yellow, both players win the game simultaneously; in this case $(p,e).timestamp = (e,p).timestamp < \infty$. If the evader starts with initial configurations in blue, it will lose the game due to a collision with a wall. For all other evader initial configurations, both players chase each other forever and can not win the game. **Right:** A sample collision (green: evader, red: pursuer).

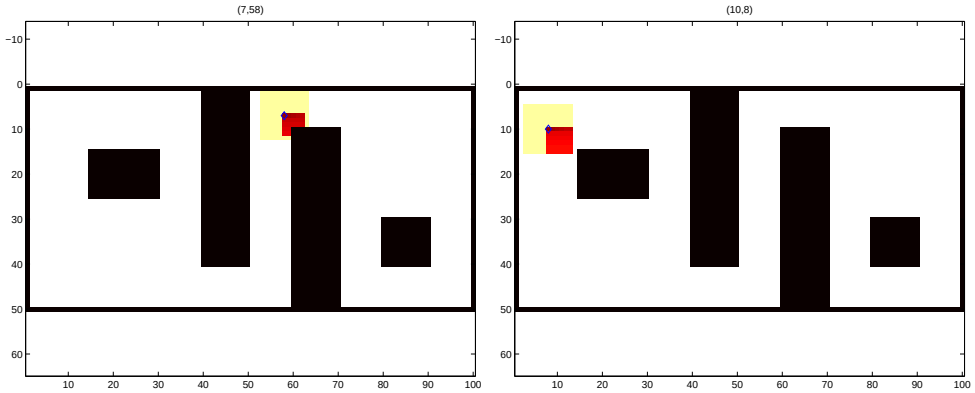

Fig. 6. Reaction zones (yellow) and dangerous configurations (red) for two configurations.

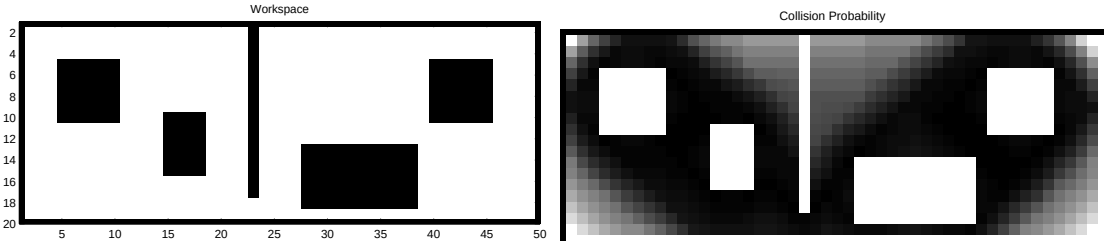

Fig. 7. A sample workspace and collision probabilities. Brighter colors indicate higher collision probability.

The collision map for a 2D configuration space is presented in Figure 7.

B. Trajectory planning

Once we obtain the collision probabilities, we can obtain optimal collision avoiding paths using a Markov Decision Process (MDP) formulation. A finite state Markov Decision Process is given by a finite set of states S, a finite set of actions A, transition probabilities $P(r|s,a)$ of arriving at state r when action a is taken from state s, and rewards $R(r|s,a)$ from arriving at state r from state s via action a. A policy π is a function that takes a state-action pair (s,a) and returns a real number in [0,1], indicating the probability of taking action a when in state s. An optimal policy is a policy which maximizes expected return at each state. Given a finite MDP it is possible to find an optimal policy using dynamic programming or its variants such as Policy Iteration. A comprehensive introduction to MDPs can be found in [33], [34].

To compute trajectories from all initial configurations to a given final configuration $c_f \in \mathcal{C}$, we build an MDP as follows:

The state space of the MDP is given by $S = \mathcal{C} \cup \{COL\}$ where COL is a special state to denote collision. The set of actions is equal to the set of controls $A = \mathcal{U}$. For each $c, c' \in C$ with $c \rightarrow_u c'$, we have

$$\begin{aligned} P(c'|c,u) &= 1 - \mathcal{P}[c'] \\ P(COL|c,u) &= \mathcal{P}[c'] \end{aligned}$$

and the corresponding rewards:

$$\begin{aligned} R(c'|c,u) &= -1 \\ R(COL|c,u) &= -\infty \end{aligned}$$

We add the special cases

$$\begin{aligned} P(c_f|c_f,u) &= 1 \\ P(COL|COL,u) &= 1 \end{aligned}$$

for all u with

$$\begin{aligned} R(c_f|c_f,u) &= 0 \\ R(COL|COL,u) &= -\infty \end{aligned}$$

All other rewards and probabilities are zero. Once we build the MDP, we compute the optimal policy. In Figure 8, the optimal collision avoiding policy for an arbitrary final state is displayed in contrast to the shortest-path policy.

C. Multi-robot settings

To explore the utility of the collision probabilities we performed a simulation where our robot tries to reach a final configuration amidst $k = \{1, \ldots, 10\}$ robots performing a random walk. For each k, we performed 1000 experiments and compared the success rate of the MDP optimal policy with the shortest path policy. The results are summarized in Figure 9.

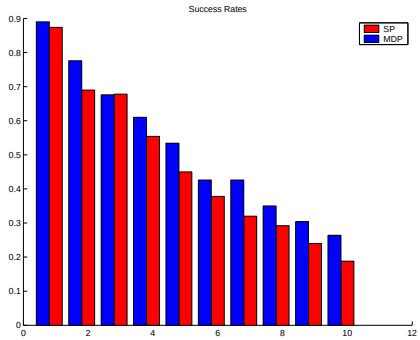

Fig. 9. Success rates in the presence of k (horizontal axis) robots performing a random walk for the shortest path and optimal MDP policies.

V. Conclusion

In the recent years, roadmap based methods have been extremely successful for motion-planning in complex configuration spaces. In this paper, we address two important robotics problems, pursuit-evasion and collision avoidance, and present solutions based on roadmaps.

In the first part of the paper, by using time aware controllers to build the roadmaps, we show how to obtain solutions to various pursuit-evasion games. The algorithms are based on the dynamic programming (DP) principle and inherit both advantages and disadvantages of DP based approaches. The primary advantage is in the soundness and completeness of the algorithms. However, their running times increase with the dimension of the configuration space; making them practical for only low-dimensional configuration spaces. In our future work, we plan to investigate this issue further.

In the second part of the paper, we presented a probabilistic framework for collision avoidance. We study a scenario where the robots react to each other only when they enter a certain reaction-zone. We assume that the robots are independent, however once they are within the reaction zone, they become unpredictable. To capture this unpredictability, we model them as adversarial pursuers and utilize the results from the first part to obtain worst-case guarantees for collision avoidance. This formulation allows us to process the workspace and build a collision-probability map. Afterwards, optimal collision avoiding trajectories are computed using standard MDP algorithms. Preliminary simulations demonstrate the utility of our approach. In our future work, we plan to investigate the effect of the diameter of the collision reaction zone and implement the algorithm on real robots to further study their feasibility.

Acknowledgements

This work has been supported in part by NSF grants IIS-0438125 and EIA-0122599. We gratefully acknowledge the industrial support through CITRIS organization.

References

[1] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *28th Annual IEEE Symposium on Foundations of Computer Science*, 1987, pp. 49–60.

[2] S. M. LaValle, *Planning Algorithms*. [Online], 2004, available at http://msl.cs.uiuc.edu/planning/.

[3] H. Choset and et. al., *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, 2004.

[4] L. E. Kavraki, J.-C. Latombe, and R. Motwani, "Randomized query processing in robot path planning," *Journal of Computer and System Sciences*, vol. 57, no. 1, pp. 50–60, August 1998.

[5] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *International Journal of Robotics Research*, 2003.

[6] R. Isaacs, *Differential Games*. Dover, 1965.

[7] T. Basar and G. J. Olsder, *Dynamic Noncooperative Game Theory*. SIAM, 1998.

[8] J. Hespanha, H. J. Kim, and S. Sastry, "Multiple-agent probabilistic pursuit-evasion games," in *In Proc. of the 38th Conf. on Decision and Contr*, 1999, pp. 2432–2437.

[9] R. Vidal, O. Shakernia, J. Kim, D. Shim, and S. Sastry, "Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation," *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 662–669, 2002.

[10] J. P. Hespanha, G. J. Pappas, and M. Prandini, "Greedy control for hybrid pursuit-evasion games," in *Proceedings of the European Control Conference*, Porto, Portugal, September 2001, pp. 2621–2626.

[11] R. Nowakawski and P. Winkler, "Vertex-to-vertex pursuit in a graph," *Discrete Math*, vol. 43, pp. 235–239, 1983.

[12] T. D. Parsons, "Pursuit evasion in a graph," in *Theory and Application of Graphs*, Y. Alavi and D. R. Lick, Eds. Springer Verlag, 1976, pp. 426–441.

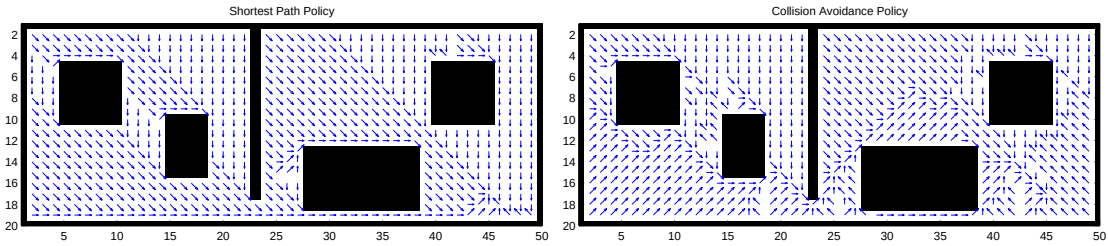

Fig. 8. Vector fields for reaching a final configuration in the lower right. Left figure is the vector field for generating shortest paths and the right figure is the optimal policy corresponding to the MDP.

[13] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, "The complexity of searching a graph," *J. ACM*, 1988.

[14] M. Aigner and M. Fromme, "A game of cops and robbers," *Discrete Applied Math*, vol. 8, pp. 1–12, 1984.

[15] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B. Vöcking, "Randomized pursuit-evasion in graphs," *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2002. [Online]. Available: citeseer.nj.nec.com/510108.html

[16] V. Isler, S. Kannan, and S. Khanna, "Randomized pursuit-evasion with limited visibility," in *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.

[17] I. Suzuki and M. Yamashita, "Searching for a mobile intruder in a polygonal region," *SIAM Journal on Computing*, vol. 21, no. 5, pp. 863–888, 1992.

[18] S.-M. Park, J.-H. Lee, and K.-Y. Chwa, "Visibility-based pursuit-evasion in a polygonal region by a searcher," *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, vol. 2076, pp. 456–468, 2001. [Online]. Available: citeseer.nj.nec.com/park01visibilitybased.html

[19] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, "A visibility-based pursuit-evasion problem," *International Journal of Computational Geometry and Applications*, vol. 9, no. 4/5, pp. 471–, 1999. [Online]. Available: citeseer.nj.nec.com/guibas96visibilitybased.html

[20] J. Sgall, "Solution of David Gale's lion and man problem," *Theoret. Comput. Sci.*, vol. 259, no. 1-2, pp. 663–670, 2001.

[21] V. Isler, S. Kannan, and S. Khanna, "Locating and capturing an evader in a polygonal environment," in *Proc. of 6th Workshop on Algorithmic Problems in Robotics (WAFR'04)*, 2004, pp. 351–367.

[22] B. Gerkey, S. Thrun, and G. Gordon, "Clear the building: Pursuit-evasion with teams of robots," in *Proceedings of the AAAI National Conference on Artificial Intelligence*. San Jose, CA: AAAI, 2004.

[23] J. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[24] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation*, vol. 4, no. 1, 1997.

[25] S. M. LaValle and S. A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Trans. on Robotics and Automation*, vol. 14, no. 6, pp. 912–925, Dec. 1998.

[26] J. M. Phillips, L. E. Kavraki, and N. Bedrossian, "Spacecraft rendezvous and docking with real-time, randomized optimization," in *AIAA Guidance, Navigation, and Control*, 2003.

[27] C. Clark, S. Rock, and J. Latombe, "Dynamic networks for motion planning in multi-robot space systems," in *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2003.

[28] R. Bohlin and L. Kavraki, "Path planning using lazy prm," in *International Conference on Robotics and Automation*, 2000.

[29] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact collision checking ofrobot paths," in *Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2002.

[30] C. M. Clark, T. Bretl, and S. M. Rock, "Applying kinodynamic randomized motion planning with a dynamic priority system to multi-robot space systems," in *IEEE Aerospace Conference*, 2002.

[31] L. Jaillet and T. Simeon, "A prm-based motion planner for dynamically changing environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[32] L. Dubins, "On curves of minimum length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents." *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.

[33] R. S. Sutton and A. G. Barto, *Reinforcement Learning:An Introduction*. The MIT Press, 1998.

[34] D. P. Bertsekas, *Dynamic Programming and Optimal Control: 2nd Edition*. Athena Scientific, 2000.