Rapid Recovery from Robot Failures in Multi-Robot Visibility-Based Pursuit-Evasion

Trevor Olsen, Nicholas M. Stiffler, and Jason M. O'Kane

Abstract— **This paper addresses the visibility-based pursuit-evasion problem where a team of pursuer robots operating in a two-dimensional polygonal space seek to establish visibility of an arbitrarily fast evader. This is a computationally challenging task for which the best known complete algorithm takes time doubly exponential in the number of robots. However, recent advances that utilize sampling-based methods have shown progress in generating feasible solutions. An aspect of this problem that has yet to be explored concerns how to ensure that the robots can recover from catastrophic failures which leave one or more robots unexpectedly incapable of continuing to contribute to the pursuit of the evader. To address this issue, we propose an algorithm that can rapidly recover from catastrophic failures. When such failures occur, a replanning occurs, leveraging both the information retained from the previous iteration and the partial progress of the search completed before the failure to generate a new motion strategy for the reduced team of pursuers. We describe an implementation of this algorithm and provide quantitative results that show that the proposed method is able to recover from robot failures more rapidly than a baseline approach that plans from scratch.**

I. Introduction

For a number of important applications of mobile robots, including environmental monitoring [6], [14], [32], disaster recovery [12], and surveillance [2], the central problem is to plan motions for a team robots called *pursuers* to locate unpredictably moving agents called *evaders*. Though progress has been made on several forms of this problem, a key limitation within much of the existing work is an inability to adapt in cases of unrecoverable failures of individual robots. Such failures are particularly likely to occur in the very application domains for which these methods are well-suited.

Figure 1 provides a simple illustration of this problem, in a context wherein each pursuer is equipped with an omnidirectional sensor that can detect the evaders within its line of sight. The objective is to move the pursuers along paths that are guaranteed to locate the evaders, even if the evaders may move arbitrarily quickly. An initial plan to search this environment using three robots appears on the left of Figure 1. Now suppose the robot stationed at the top left of the environment fails during the plan's execution, as shown on the right of Figure 1. In this case, any evader hiding in the center of the environment at that time can now escape to the top left corner without being detected. Thus, the initial plan is no longer guaranteed to locate the evader.

T. Olsen and J. M. O'Kane are with the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208, USA. N. M. Stiffler is with the Department of Computer Science, University of Dayton, Dayton, OH 45469, USA. `tvolsen@email.sc.edu`, `jokane@cse.sc.edu` This material is based upon work supported by the National Science Foundation under Grant Nos. 1659514 and 1849291.

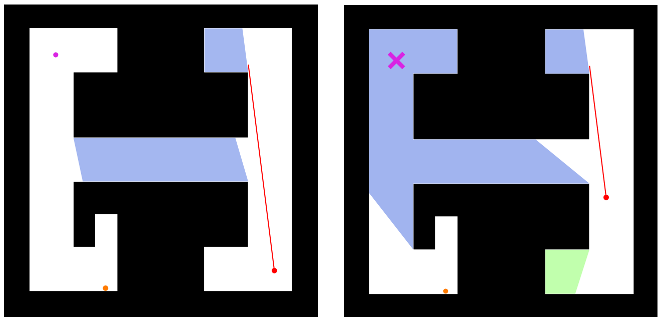


Fig. 1: [left] An initial joint plan for 3 robots to search a simple environment for an evader. Two robots on the left remain stationary and monitor that side of the environment while a third moves to search the right. [right] The robot in the top left corner fails unexpectedly. As a result, the initial plan is no longer correct, because it cannot locate an evader that moves into the top left corner of the environment.

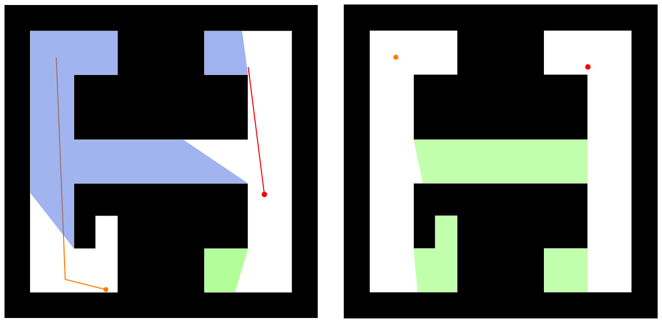


Fig. 2: [left] The robots from Figure 1 replan, using the proposed algorithm, to form a correct plan for the remaining 2 robots. [right] The plan completes successfully.

In response to this limitation, this paper proposes a new approach for this form of visibility-based pursuit-evasion problem, suitable for contexts in which failures of robots are likely. Because generating pursuit plans for this domain is computationally challenging [27], we introduce a method for replanning in cases of robot failure that leverages information derived from the previous plan to accelerate the construction of new plans for the unexpectedly-smaller teams. Figure 2 shows an example of recovery from the failure depicted in Figure 1.

The central idea is to adapt known methods for solving these problems in the failure-free case, which are based on a roadmap-like data structure called the sample-generated pursuit-evasion graph (SG-PEG) [28]. For planning the motions of n robots in this pursuit-evasion problem, the SG-

PEG graph represents the connectivity of the joint configurations space with vertices that represent joint configurations and edges that represent collision free movements, much like a traditional probabilistic roadmap. In addition, each vertex is labeled with information that encodes which hidden portions of the environment can be cleared of unseen evaders along some walk in the graph ending at that vertex. The new replanning approach modifies this data structure when a robot fails to reflect the removal of that robot from the search. The remaining graph then provides a valuable starting point for the process of planning to solve the problem with the remaining $n-1$ robots. The algorithm then expands this graph by adding additional vertices, attempting to recover the contributions made by the failed robot.

This work is, to the authors' best knowledge, the first to address the problem of recovery from pursuer failures for this form of pursuit-evasion problem. In the remainder of the paper, we review some related work (Section II), formally define the problem (Section III). Then we describe and evaluate the algorithm (Sections IV and V respectively) before concluding with discussion and a preview of future work (Section VI).

II. Related work

The visibility-based pursuit-evasion problem posed in this paper can be thought of as a specialization of the broader problem of *search and target tracking*. The common theme across this work is the pursuit of an agent (or agents) by one (or more) pursuers to either establish or maintain visibility of the target [25], [26], [37].

The literature on these problems can by understood by organizing according to underlying models, including differential game theory, graph variants, and geometric variants. Though this work considers the geometric formulation, we present a brief synopsis of the contributions in the domains of differential game theory and graph theory.

The seminal work of Isaacs [13] and Ho et. al. [11] was the first to adapt the pursuit-evasion problem to a dynamic game-theoretic framework. This remains an active research area [23], [34]. Recent results include continued progress by utilizing techniques such as reinforcement learning [35] and the exploitation of rich representations such as Voronoi partitions to aid in the search [18], [36].

A different formulation in which the domain is modeled as a discrete graph was initially proposed by Parsons [21] and is referred to as the edge-searching problem. Petrov later independently rediscovered some of Parsons' results in the context of differential game theory [22]. Golovach later showed that both problems considered an equivalent discrete game on graphs [8]. A number of survey papers [1], [3], [5] provide overviews of the many problem variants that can be realized within the graph model, such as specifying the rules of movement for the pursuers and for the evaders [24], the kind of graph [16], etc.

This paper specifically focuses on a variant of the problem where the pursuers and evaders operate in a geometric environment [10], [20], [31]. There are a number of results for the single pursuer variant of the problem that range from providing theoretical properties such as completeness [10] and optimality [29], to more restricted scenarios where there are limits on the actuation and sensing capabilities of the pursuers [4], [17], [30], [33]. Due to the broad range of practical applications, the multi-pursuer variant of the problem has drawn continued interest [7], [9], [15], [28] in recent years. The multi-pursuer scenario poses additional challenges owing to the problem complexity [27]. A common thread through much of the existing work is an assumption that the pursuer(s) can reliably execute the trajectories generated for them by the planner. This work seeks to address this limitation.

III. Problem statement

We first define the basic problem in the absence of pursuer failures (Section III-A) and describe how to cast that problem in terms of a discrete representation of which areas of the environment are 'clear' or 'contaminated' (Section III-B), before introducing the possibility of pursuer failures (Section III-C).

A. Environment, pursuers, and evaders

The *environment* F is a closed, bounded, and connected polygonal region in $\mathbb{R}^2$. A team of n *pursuers*, who can travel throughout the environment at bounded speed, are equipped with omnidirectional sensors whose range is only bounded by line of sight within the environment. That is, a pursuer at point $q \in F$ can detect anything within its *visibility polygon* $V(q) = \{r \in F \mid \overline{qr} \subset F\}$. We denote the location of the i^{th} pursuer as a function of the time t by the continuous function $f_i(t) : [0, T] \to F$, in which T is some termination time which the pursuers may choose. The n-robot *joint pursuer configuration* (JPC) at time t is the vector $\langle f_1(t), f_2(t), \dots, f_n(t)\rangle \in F^n$.

A single *evader* seeks to avoid detection by the pursuers by moving continuously within the environment, without any bound on its speed. We denote its location, as a function of time, by the continuous function $e(t) : [0, \infty) \to F$, unknown to the pursuers. Observe that, because we plan for the worst case, any strategy for the pursuers that guarantees detection of a single evader can also guarantee detection for each of potentially many evaders.

The pursuers' objective is to establish visibility with the evader. Thus, for a given environment F, the goal is to choose the termination time T and the functions $f_1, f_2, \dots, f_n$ to ensure that for any evader trajectory e, there exists a time $t_0 \in [0, T]$, such that $e(t_0) \in \bigcup_{i \le n} V(f_i(t_0))$. Figure 3 illustrates the notation.

B. Shadows

The primary difficulty in this type of visibility-based pursuit-evasion concerns reasoning about the regions of the environment that are not currently visible to the pursuers at the present time. To resolve that difficulty, Guibas, Latombe, LaValle, Lin, and Motwani [10] introduced a reformulation of the problem, based upon tracking which, if any, of the

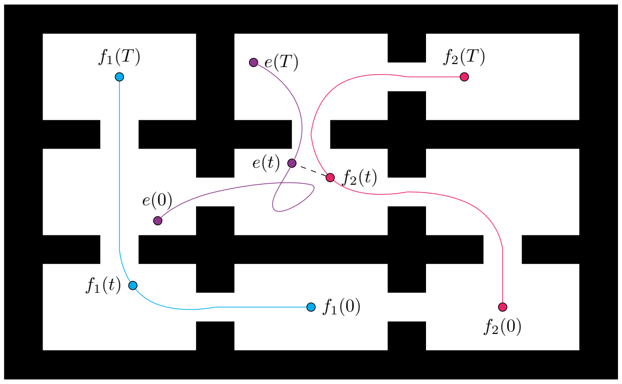


Fig. 3: Two pursuers execute a search. Note that at time t, the pursuer on path f_2 is capable of detecting the evader e.

regions of the environment not currently perceptible by the pursuers might contain an as-yet-undetected evader.

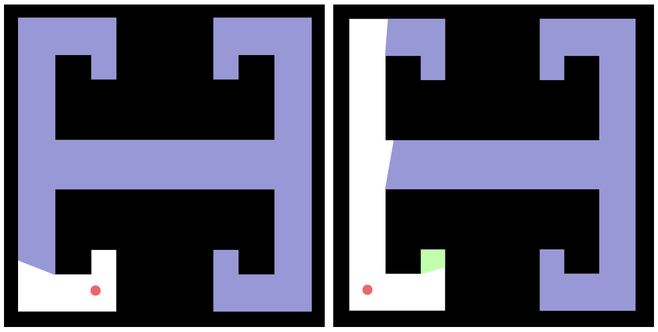
To formalize this idea, define the *shadow region*, $S(t) = F \setminus \bigcup_{i \leq n} V(f_i(t))$ as the portion of the environment unseen by any pursuer at time t. The maximal connected components of $S(t)$ are called *shadows*. The terms *cleared* and *contaminated* can be applied to a shadow to reflect the relative *status* of the shadow at that point in the pursuers' search. A cleared shadow is an unseen area of the environment that, based upon the pursuers' motions up to the current time t, is guaranteed to not contain an unseen evader. Any shadow that is not cleared is called contaminated.

To compactly describe the status for all of the shadows, we utilize a *shadow label*, which is a binary string comprised of one bit for each shadow, in which the i^{th} bit is 1 if the i^{th} shadow (in an arbitrary but fixed ordering) is contaminated, and 0 otherwise.

Though shadows change continuously as the pursuers move within F, the cardinality of the shadows and their labels can change only when a shadow event occurs, i.e. a shadow appears, disappears, splits, or multiple shadows merge into a single shadow.

- *Appear*: A shadow appears when the pursuers move in such a way that makes a previously observed part of the environment no longer visible. In this case, the new shadow is assigned a cleared status.
- *Disappear*: If a pursuer gains vision of a shadow, we say the shadow disappears. Here, the shadow bit is deleted.
- *Split*: If a pursuers' vision disconnects a shadow, we say that the shadow has split. Each new component is given the label of the original pre-split shadow.
- *Merge*: If two or more shadows become a single connected component, we say the shadows have merged. The newly merged shadow takes on the cleared status if and only if each merging component was cleared prior to the merge. Otherwise, the merged shadow is considered to be contaminated.

This formulation of the problem in terms of clear and contaminated shadows is valuable because it enables a planner to reason over those shadows and labels, rather than directly over the space of all possible evader paths. That is, the overall visibility-based pursuit-evasion problem can be restated as a search for pursuer motions that lead to a system state in which the binary string of shadow labels contains all zeroes, indicating that every shadow is clear.



(a) The initial location of a single pursuer. Here, the shadow label is 1.

(b) Movement of a pursuer that would cause one shadow to appear and another to split. The result has one cleared and two contaminated shadows. In our implementation, this scenario is assigned shadow label 110.

Fig. 4: An example of shadow events and labels.

C. Pursuer Failures

The basic problem described so far has been addressed in prior work in a number of different ways. The new contribution in this paper is to consider this problem in an online setting, in which the pursuer robots may fail during the execution of a plan. Such failures are assumed to be both unpredictable and permanent, but known to all of the pursuers when they occur.

More precisely, we consider the case in which the n pursuers are executing the paths n$f_i : [0, T] \rightarrow F$ for $i \in \{1, \ldots, n\}$, with termination time T. Suppose that at some time $0 \leq \hat{t} < T$, the k^{th} pursuer fails. The *replanning problem* addressed in this paper is to generate new pursuer paths $f'_i : [\hat{t}, T'] \rightarrow F$ for $i \in \{1, \ldots, n\} \setminus \{k\}$. These revised paths must begin at the pursuers' locations at the time of failure, so that for each surviving pursuer, we have $f'_i(\hat{t}) = f_i(\hat{t})$. The revised paths end at a new termination time T'.

Generalizing the objective from the original failureless model, we say that the pursuers' execution of the prefix of $f_1, \ldots, f_n$ up to time $\hat{t}$, followed by $f'_1, \ldots, f'_n$ from time $\hat{t}$ to T' is a *solution* if, for any evader trajectory e, either (i) there exists a time $t_0 \in [0, \hat{t}]$, such that $e(t_0) \in \bigcup_{i \leq n} V(f_i(t_0))$, or (ii) there exists a time $t_0 \in [\hat{t}, T']$, such that $e(t_0) \in \bigcup_{i \leq n; i \neq k} V(f'_i(t_0))$. That is, we seek to guarantee that the evader is seen by any of the robots at some time before or after the pursuer failure, or by one of the non-failing robots at some time after the failure. Similar but notationally tedious generalizations can be made for multiple failures within a single execution.

Fortunately, we can also generalize the notions of shadow label updates to account for the abrupt change in shadows that occur at time $\hat{t}$. Specifically, a shadow extant immediately after a robot failure is contaminated if and only if it intersects with a contaminated shadow from immediately before the failure occurred. Notice, for example, in the right portion of Figure 1, that the large shadow encompassing the center and upper left portion of the environment is marked contaminated because it overlaps the central shadow which was contaminated before the failure. In contrast, the smaller shadow in the lower right has a clear label after the failure, because the only pre-failure shadow with which it intersects (namely, itself) had a clear label. This feature of the definition of success, which allows shadows to remain clear even across a failure of one of the pursuers, is crucial because it allows the pursuers the possibility of retaining some of their progress (i.e. cleared shadows) toward completing the task, rather than starting from scratch each time.

IV. Algorithm Overview

This section provides a detailed description of our algorithm. Because no efficient algorithm for solving even the failure-free case is known [27], we take a sampling-based approach. The basic idea is to construct a roadmap within the pursuers' joint configuration space, using an existing data structure called the sample-generated pursuit-evasion graph (SG-PEG), which a subset of the present authors originally introduced for the failure-free case [28]. We leverage this data structure in a new way by introducing new sampling strategies designed to rapidly re-acquire a solution in cases where a pursuer must be removed.

The core of the algorithm is a method called DropRobot which, given a solution path for k robots (for some k), uses an SG-PEG to attempt to rapidly generate a solution for $k-1$ robots, using the original k-robot solution as a guide. Our algorithm relies upon DropRobot both to generate an initial solution for the full set of n robots —by iteratively reducing from a rapidly-generated trivial solution— and for replanning when a pursuer fails.

The remainder of this section presents details of the method. After a brief review of the SG-PEG (Section IV-A), we describe the DropRobot method (Section IV-B) and how that method is used to generate the initial solution (Section IV-C.1) and for replanning (Section IV-C.2).

A. SG-PEG

The SG-PEG is a data structure the represents a roadmap of valid joint paths for a team of pursuers in a known environment F, augmented with information about the shadow labels that can be achieved by executing those paths. We present here a concise overview; additional detail may be found in the original paper [28].

An SG-PEG is a directed graph $G = (V_G, E_G)$, in which one vertex v_0 is designated as the *root vertex*. Each SG-PEG is constructed for a specific number n of pursuers. Each vertex $v \in V_G$ corresponds to a specific JPC $\langle p_1, \dots, p_n \rangle \in F^n$. Each directed edge $e \in E_G$ connects two vertices $v, u \in V_G$ for which it is possible for every pursuer to make a collision-free straight line motion between the representative configurations. That is, the existence of an edge from v to u means that, for each $1 \le i \le n$, $\overline{v_i u_i} \subset F$.

In addition to this graph structure, each vertex v maintains a set of reachable shadow labels. Specifically, a shadow label ℓ will be recorded at a particular vertex v as a reachable shadow label if there exists a walk from v_0 to v that results in the shadow marked clear within ℓ indeed being clear.

The primary operation that can be performed on a SG-PEG is AddSample$(\langle p_1, \dots, p_n \rangle)$, which accepts a collision-free JPC as input and performs the following steps:

(i) It inserts a new vertex v at the given JPC.
(ii) For every existing vertex u for which the segment $\overline{uv}$ is collision free in F^n, it adds the edges $\overrightarrow{uv}$ and $\overrightarrow{vu}$. The operation then computes a mapping that describes how the shadows at vertex u evolve as the pursuers move from the JPC at vertex u to the JPC at vertex v. (The inverse mapping is applied to $\overrightarrow{vu}$).
(iii) Finally, the reachable shadow label information across the graph is updated by propagating the reachable shadow labels, using the mappings attached to each edge, recursively across the graph, to determine what new reachable shadow labels, if any, arise due to the inclusion of the new sample v.

The SG-PEG data structure is useful for our problem because, starting from a root vertex at the pursuers' initial positions, executing a sequence of AddSample operations can eventually lead to a vertex being marked with an all-zero reachable shadow label. From there, a sequence of JPCs solving the problem can readily be extracted by walking backward along through the graph.

B. Dropping a robot

Suppose k pursuers are at some JPC q with shadow label ℓ, and have computed a sequence of future JPCs to visit that will solve the problem from that point, eventually reaching JPC with an all-clear shadow label. How can we use this information to construct a new solution that can be executed from this point by only $k-1$ of these pursuers, removing one particular pursuer from the solution? Notice that this scenario applies both to the case of a failed pursuer (in which case q and ℓ can be derived from the current state when the failure occurred, and ℓ may mark some shadows as clear) and to a complete solution starting from the pursuers' starting position and all-contaminated shadow label. To simplify the notation below, we assume without loss of generality that n^{th} pursuer is the one removed.

The DropRobot method, shown in Algorithm 1, solves this problem. The algorithm constructs an SG-PEG G_{k-1}, starting with a root vertex at which the n^{th} pursuer has been removed and the shadow label has been updated accordingly. From there, it adds a collection of *junction samples*, designed to recover information lost due to the removal of the n^{th} pursuer at each step of the existing solution. If G_{k-1} does not contain a solution after that step, DropRobot continues by inserting additional samples called *web samples* designed to provide good coverage, in the sense of visibility, of the environment. The process continues until a solution is found, or until some arbitrary timeout expires. Details about junction sampling and web sampling appear below.

Algorithm 1 DropRobot($F, k, q_1, \ldots, q_m, \ell$)

```
Input: An environment $F$; a positive integer $k$; a sequence $q_1, \ldots, q_m$ of $k$-pursuer JPCs; a shadow label $\ell$ for $q_1$.
Output: A sequence $q'_1, \ldots, q'_{m'}$ of $(k-1)$-pursuer JPCs leading to an all clear shadow label at $q'_{m'}$ or FAILED.
 1: $G_{k-1} \leftarrow$ new SG-PEG for $k-1$ pursuers
 2: $\langle p_1, \ldots, p_k \rangle \leftarrow q_1$
 3: $r \leftarrow G_{k-1}$.AddRoot($\langle p_1, \ldots, p_{k-1} \rangle$)
 4: $\ell' \leftarrow \ell$ updated for the removal of $p_n$
 5: $r$.AddReachable($\ell'$)
 6: for $i \leftarrow 1, \ldots, m$ do
 7:     AddJunctionSamples($k$, $G_{k-1}$, $q_i$)
 8: while $G_{k-1}$ has no solution and time remains do
 9:     $q \leftarrow$ WebSample($G_{k-1}$)
10:     $G$.addSample($q$)
11: if $G_{k-1}$ has a solution then
12:     return $G_{k-1}$.ExtractSolution()
13: else
14:     return Failed
```

Algorithm 2 AddJunctionSamples(k, G_{k-1}, q)

```
Input: A positive integer $k$; an SG-PEG $G_{k-1}$ for $k-1$ pursuers; a $k$-pursuer JPC $q$
Output: No return value, but samples are added to $G_{k-1}$.
 1: $\langle p_1, \ldots, p_k \rangle \leftarrow q$
 2: $G_{k-1}$.addSample($\langle p_1, \ldots, p_{n-1} \rangle$)
 3: for $i \leftarrow 1, \ldots, n-1$ do
 4:     if $V(p_i) \cap V(p_n) \neq \emptyset$ then
 5:         $z \leftarrow$ random point in $V(p_i) \cap V(p_n)$
 6:         $G$.addSample($\langle p_1, p_2, \ldots, z, \ldots, p_{n-1} \rangle$)
 7:         $G$.addSample($\langle p_1, p_2, \ldots, p_n, \ldots, p_{n-1} \rangle$)
```

1) Junction sampling: The objective in junction sampling is, informally, to add vertices and edges to the SG-PEG that allow remaining pursuers to 'fill in' for the removed robot, wherever possible. Figure 5 shows a simple example of a pursuer removed from a JPC during DropRobot. In this example, the lower pursuer is removed, leaving the bottom portion of the environment unobserved. Junction sampling adds new samples that provide a path within the SG-PEG for the rightmost robot to visit the site of this lower portion.

This process, called AddJunctionSamples, is formalized in Algorithm 2. In the general case, the algorithm identifies a remaining pursuer at a position p_i for which the visibility polygon intersects the visibility polygon of the position p_n of the removed pursuer. When this relationship is detected, we add a sample that places the i^{th} pursuer in the intersection of the visibility polygons (see Figure 5c) and another that places the i^{th} pursuer at the former location of the n^{th} pursuer (Figure 5d). This process is repeated for each i and, via repeated calls to AddJunctionSamples, each step of the previous k-pursuer solution.

2) Web sampling: Though the structures introduced by junction sampling may be sufficient to build a SG-PEG that can generate a solution with $k-1$ pursuers, such success cannot be guaranteed. Therefore, after exhausting the junction samples, Algorithm 1 continues with a broader sampling strategy called web sampling. Web sampling was originally proposed for the failure-free version of the problem [19].

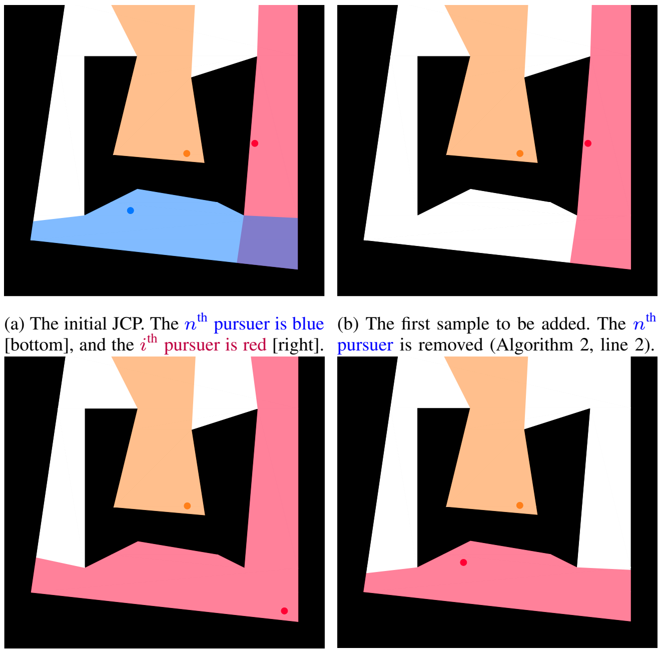

(a) The initial JCP. The n^{th} pursuer is blue [bottom], and the i^{th} pursuer is red [right]. (b) The first sample to be added. The n^{th} pursuer is removed (Algorithm 2, line 2). (c) The second sample. The n^{th} pursuer is removed and the i^{th} pursuer moves to a random point in $V(p_i) \cap V(p_n)$. (Algorithm 2, line 6). (d) The third sample. The n^{th} pursuer is removed and the i^{th} pursuer takes its place. (Algorithm 2, line 7).

Fig. 5: An example of junction sampling.

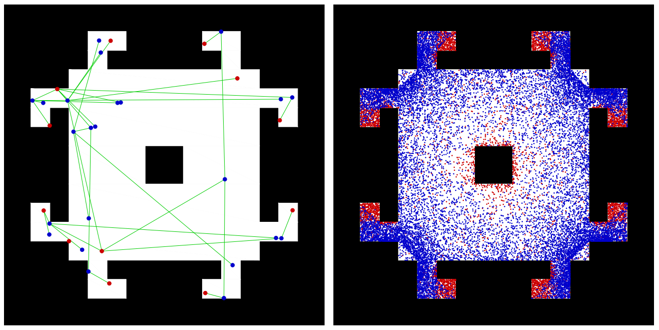

(a) A set of 35 samples that form one complete web. (b) 25000 samples drawn using web sampling. Notice how the points from A (red) are biased towards the outer hooks, while the points from B (blue) favor the regions connecting adjacent hooks.

Fig. 6: An illustration of web sampling.

The sampling approach is based on an underlying notion of a *web*. The intuition is select a collection of positions that can see the entire environment while also forming a connected graph via straight-line connections within F. Generating a web occurs in two stages. First, we draw a collection of points $A = \{a_1, a_2, \ldots, a_n\} \subset F$ which provide full visibility of the environment, i.e. $\bigcup_{1 \leq i \leq n} V(a_i) = F$. This is done incrementally, by drawing samples from the unseen portion if F until all of F is seen by some point in A. The second stage generates an intersection set B as follows. For each pair of distinct points $a_i, a_j \in A$, if $V(a_i) \cap V(a_j) \neq \emptyset$, we add a point $b \in V(a_i) \cap V(a_j)$ to B. The combination $A \cup B$ forms one complete web; those points are utilized in a randomly shuffled order. See Figure 6.

To use these webs within WEBSAMPLE (recall line 9 in Algorithm 1), we generate one web for each of the $k-1$ pursuers. Then select a random vertex v from G_{k-1} and, for two of the robots in that JPC, form a new sample by replacing the existing positions with positions drawn (without replacement) from those pursuers' respective webs. If any web ever has no more points to choose from, we generate new webs for each pursuer and continue the process.

C. Planning, execution, and replanning

Armed with the DROPROBOT method, we can consider how to use that algorithm for the overall problem.

1) Generating the initial solution: To begin, we must generate an initial solution that the full complement of n robots can begin to execute. First, we generate a trivial solution, namely a strategy where no movement is required by the pursuers because their visibility polygons fully cover the environment. We do so by iteratively adding pursuers at random unseen locations until no shadows remain. This single JPC becomes our trivial solution.

Note, however —recalling that only n robots are available at the start—, that it is rather likely that the trivial solution will require more than n robots. If so, we repeatedly apply DROPROBOT, selecting the pursuer to remove at random, until a solution requiring only n pursuers has been formed. The pursuer team then begins to execute this strategy.[1]

2) Replanning after pursuer failures: If a pursuer fails during the execution of the search, a replanning operation is required. In that case, we pause the pursuers' movement until a new solution with one fewer pursuer is generated. Should two or more pursuers fail simultaneously, our algorithm sequentially resolves each failure individually. This new solution is generated directly by DROPROBOT. Notice that the inputs to that algorithm include the current state of the search (including the current JPC and the current shadow label), which are leveraged to replan more rapidly than planning from scratch each time. Once a new solution is computed, the pursuers resume their search.

V. EVALUATION

We implemented our algorithm in C++ and executed the code on an Ubuntu 20.04 laptop equipped with an Intel i7-10510U CPU and 16GB of RAM.

An example execution is illustrated in Figure 7. First, an initial solution is generated (Figure 7a). Next, Figures 7b,c represent the input and output of Algorithm 1 when the green pursuer malfunctions. Similarly, Figures 7d,e show the state before and after the failure of the orange pursuer.

We simulated teams initially consisting of $n = 5$ pursuers[2] in three different environments, depicted in Figures 3, 4, and 6. These environments were selected because they highlight several interesting attributes, such as hard to reach corners, narrow corridors, and evenly spaced obstacles. Additionally, these environments allow us to more directly compare against existing results. In particular, we compare the algorithm presented in Section IV ('this paper') against our previous algorithm [19] ('OTSO21'), which was designed for the failure-free setting, as a baseline. During each execution, we simulated m pursuer failures. For each failure, a randomly-selected pursuer was removed when the pursuers had completed a percentage β of their planned paths. For OTSO21, the algorithm was executed from scratch for the initial solution and at each robot failure. Runs were conducted for all four combinations of $m \in \{2, 3\}$ and $\beta \in \{30\%, 70\%\}$. Because each of the selected environments requires at least two pursuers to solve, choosing $m \geq 4$ would guarantee that no solution exists.

Each trial was limited to at most 10 minutes of run time, including both planning time and (simulated) execution time. If, after that time, the robots had not yet successfully cleared all shadows, the simulation would have been considered a failure. In the results presented here, none of the trials failed.

For each combination of environment, algorithm, team size n, number of failures m, and failure time β, we conducted 25 trials. The success or failure of the run and total computation time spent planning and replanning were recorded. Planning time is summarized by the mean (μ) and the standard deviation (σ) over all trials. Tables I and II report the results, from which a few conclusions may be drawn.

Replanning is beneficial Recall from Section IV-B.1 that junction sampling was developed to "recover" information in the event of a pursuer failure. The notable improvements for the proposed algorithm compared to OTSO21 in the environments of Figure 3 and Figure 4 can be attributed to efficiencies gained by re-planning rather than starting from scratch. In the environment of Figure 6, the proposed algorithm had a similar execution time average, with slightly higher variance, when compared to OTSO21. This is likely due to the complexity of the environment resulting in a high number of pursuers in its trivial solutions and subsequently more calls to DROPROBOT to reach the initial solution.

Later failures are easier to recover For the trials with $\beta = 70\%$, the total planning time was less than when $\beta = 30\%$. This is likely due to the fact that allowing more time to traverse the solution path will, in many cases, provide the next planning stage with an improved shadow label (i.e. more cleared shadows), reducing the difficulty of the replanning problem.

[1]It is possible in principle that the trivial solution may require n robots or less. In that case, we can ignore any additional robots beyond the m that are required for the trivial solution and simply 'execute' the trivial solution.

[2]Increasing n has a positive effect on the planning time of the proposed algorithm, since, by construction, we need to generate solutions for each number of pursuers between the number of pursuer in the trivial solution and n. In contrast, OTSO21 can struggle with larger values of n due to the increased complexity of the joint configuration space, making it more difficult to connect pursuer configurations. Thus, we hold n fixed at 5 to enable a fair comparison.

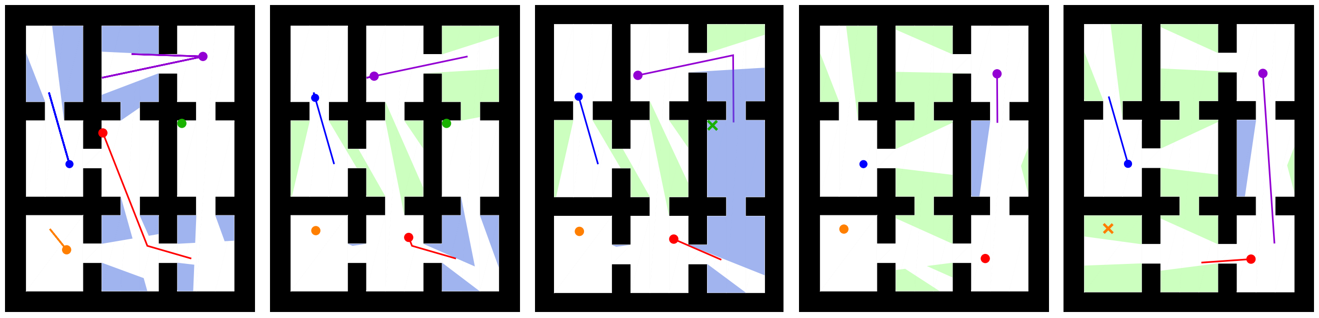

(a) An initial solution with 5 pursuers. (b) The problem state right before the green pursuer fails. (c) The new solution paths generated after the green pursuer fails. (d) The problem state right before the orange pursuer fails. (e) The new solution paths generated after the orange pursuer fails.

Fig. 7: Snapshots of our algorithm through a single successful execution ($n = 5, m = 2$).

Table I: Simulation results for $n = 5$ initial pursuers and $m = 2$ failures.

	success rate	planning time (s) μ	σ
Figure 3			
This paper ($\beta = 30\%$)	100%	46.09	21.50
OTSO21 ($\beta = 30\%$)	100%	99.57	65.03
This paper ($\beta = 70\%$)	100%	32.46	14.84
OTSO21 ($\beta = 70\%$)	100%	89.36	59.62
Figure 4			
This paper ($\beta = 30\%$)	100%	6.47	7.46
OTSO21 ($\beta = 30\%$)	100%	56.13	17.20
This paper ($\beta = 70\%$)	100%	5.07	5.97
OTSO21 ($\beta = 70\%$)	100%	52.87	17.59
Figure 6			
This paper ($\beta = 30\%$)	100%	89.13	39.08
OTSO21 ($\beta = 30\%$)	100%	94.79	27.95
This paper ($\beta = 70\%$)	100%	72.06	34.17
OTSO21 ($\beta = 70\%$)	100%	73.98	20.56

Table II: Simulation results for $n = 5$ initial pursuers and $m = 3$ failures.

	success rate	planning time (s) μ	σ
Figure 3			
This paper ($\beta = 30\%$)	100%	63.92	29.89
OTSO21 ($\beta = 30\%$)	100%	117.90	64.78
This paper ($\beta = 70\%$)	100%	39.88	20.61
OTSO21 ($\beta = 70\%$)	100%	102.57	63.03
Figure 4			
This paper ($\beta = 30\%$)	100%	8.77	8.20
OTSO21 ($\beta = 30\%$)	100%	58.35	16.60
This paper ($\beta = 70\%$)	100%	6.54	6.41
OTSO21 ($\beta = 70\%$)	100%	53.78	16.64
Figure 6			
This paper ($\beta = 30\%$)	100%	112.19	39.02
OTSO21 ($\beta = 30\%$)	100%	108.11	30.13
This paper ($\beta = 70\%$)	100%	74.81	35.71
OTSO21 ($\beta = 70\%$)	100%	77.03	19.77

Impacts of the number of failures Increasing from $m = 2$ to $m = 3$ increased the planning time for both algorithms. We speculate that this can be attributed to the additional pursuer failure for which both the proposed algorithm and OTSO21 are required to recompute strategies.

VI. Conclusion

We presented a method of deconstructing higher dimensional solutions in order to alleviate the issue of potential robotic failures in a visibility-based pursuit-evasion problem. We did this by building a new sampling strategy which allowed us to utilize previously computed information. Our algorithm was able to greatly our-perform existing algorithms in the context of our problem. Future work could include generating solutions that are intentionally robust to failures. That is, a solution that would still be a solution if a limited number of pursuers were to completely malfunction. One possible approach to this problem would be to expand the shadow labels from single clear/contaminated bits, to a richer representation of the sets of pursuer failures under which that shadow would nonetheless be clear.

References

[1] T. V. Abramovskaya and N. N. Petrov, "The theory of guaranteed search on graphs", *Vestnik St. Petersburg University*, vol. 46, no. 2, pp. 49–75, 2013.

[2] J. J. Acevedo, B. C. Arrue, I. Maza, and A. Ollero, "A decentralized algorithm for area surveillance missions using a team of aerial robots with different sensing capabilities", in *Proc. IEEE International Conference on Robotics and Automation*, 2014.

[3] B. Alspach, "Searching and sweeping graphs: A brief survey", *Matematiche*, vol. 59, pp. 5–37, 2004.

[4] D. Bhadauria, K. Klein, V. Isler, and S. Suri, "Capturing an evader in polygonal environments with obstacles: The full visibility case", *International Journal of Robotics Research*, vol. 31, pp. 1176–1189, 10 2012.

[5] R. Borie, S. Koenig, and C. Tovey, "Pursuit-evasion problems", in *Handbook of Graph Theory*, J. Gross, J. Yellen, and P. Zhang, Eds., Chapman and Hall, 2013, ch. 9.5, pp. 1145–1165.

[6] P. Dames, P. Tokekar, and V. Kumar, "Detecting, localizing, and tracking an unknown number of moving targets using a team of mobile robots", *International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1540–1553, 2017.

[7] J. W. Durham, A. Franchi, and F. Bullo, "Distributed pursuit-evasion without mapping or global localization via local frontiers", *Autonomous Robots*, vol. 32, pp. 81–95, 2012.

[8] P. Golovach, “A topological invariant in pursuit problems”, *Differentsial’nye Uraveniya (Differential Equations)*, vol. 25, pp. 923–929, 1989.

[9] L. Gregorin, S. Givigi, E. Freire, E. Carvalho, and L. Molina, “Heuristics for the multi-robot worst-case pursuit-evasion problem”, *IEEE Access*, vol. 5, pp. 17 552–17 566, Aug. 2017.

[10] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, “Visibility-based pursuit-evasion in a polygonal environment”, *International Journal on Computational Geometry and Applications*, vol. 9, no. 5, pp. 471–494, 1999.

[11] Y. C. Ho, A. Bryson, and S. Baron, “Differential games and optimal pursuit-evasion strategies”, *IEEE Trans. Automatic Control*, vol. 10, pp. 385–389, 1965.

[12] G. Hollinger, A. Kehagias, and S. Singh, “Probabilistic strategies for pursuit in cluttered environments with multiple robots”, in *Proc. IEEE International Conference on Robotics and Automation*, 2007.

[13] R. Isaacs, *Differential Games*. New York: Wiley, 1965.

[14] V. Isler, N. Noori, P. Plonski, A. Renzaglia, P. Tokekar, and J. V. Hook, “Finding and tracking targets in the wild: Algorithms and field deployments”, in *Proc. IEEE International Symposium on Safety, Security, and Rescue Robotics*, 2015.

[15] A. Kolling and S. Carpin, “Multi-robot pursuit-evasion without maps”, in *Proc. IEEE International Conference on Robotics and Automation*, 2010.

[16] A. Kolling and S. Carpin, “Pursuit-evasion on trees by robot teams”, *IEEE Trans. Robotics*, vol. 26, pp. 32–47, Mar. 2010.

[17] A. Q. Li, F. Amigoni, R. Fioratto, and V. Isler, “A search-based approach to solve pursuit-evasion games with limited visibility in polygonal environments”, in *Proc. International Conference on Autonomous Agents and Multiagent Systems*, 2018, pp. 1693–1701.

[18] A. V. Moll, D. Casbeer, E. Garcia, and D. Milutinovic, “Pursuit-evasion of an evader by multiple pursuers”, in *Proc. International Conference on Unmanned Aircraft Systems*, Jun. 2018, pp. 133–142.

[19] T. Olsen, A. M. Tumlin, N. M. Stiffler, and J. M. O’Kane, “A visibility roadmap sampling approach for a multi-robot visibility-based pursuit-evasion problem”, in *Proc. IEEE International Conference on Robotics and Automation*, 2021.

[20] S. Park, J. Lee, and K. Chwa, “Visibility-based pursuit-evasion in a polygonal region by a searcher”, in *Proc. International Colloquium on Automata, Languages and Programming*, 2001, pp. 281–290.

[21] T. D. Parsons, “Pursuit-evasion in a graph”, in *Theory and Application of Graphs*, Y. Alavi and D. R. Lick, Eds., Berlin: Springer-Verlag, 1976, pp. 426–441.

[22] N. N. Petrov, “The cossack-robber differential game”, *Differentsial’nye Uraveniya (Differential Equations)*, vol. 19, pp. 1366–1374, 1983.

[23] U. Ruiz and R. Murrieta-Cid, “Time-optimal motion strategies for capturing an omnidirectional evader using a differential drive robot”, *IEEE Trans. Robotics*, vol. 21, no. 3, Jun. 2013.

[24] F. Shkurti and G. Dudek, “On the complexity of searching for an evader with a faster pursuer”, in *Proc. IEEE International Conference on Robotics and Automation*, 2013, pp. 4062–4067.

[25] F. Shkurti and G. Dudek, “Topologically distinct trajectory predictions for probabilistic pursuit”, in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 5653–5660.

[26] F. Shkurti, N. Kakodkar, and G. Dudek, “Model-based probabilistic pursuit via inverse reinforcement learning”, in *Proc. IEEE International Conference on Robotics and Automation*, 2018, pp. 7804–7811.

[27] N. M. Stiffler and J. M. O’Kane, “A complete algorithm for visibility-based pursuit-evasion with multiple pursuers”, in *Proc. IEEE International Conference on Robotics and Automation*, 2014.

[28] N. M. Stiffler and J. M. O’Kane, “A sampling based algorithm for multi-robot visibility-based pursuit-evasion”, in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[29] N. M. Stiffler and J. M. O’Kane, “Complete and optimal visibility-based pursuit-evasion”, *International Journal of Robotics Research*, vol. 36, pp. 923–946, Jul. 2017.

[30] N. M. Stiffler and J. M. O’Kane, “Planning for robust visibility-based pursuit-evasion”, in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.

[31] 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, Oct. 1992.

[32] P. Tokekar, D. Bhadauria, A. Studenski, and V. Isler, “A robotic system for monitoring carp in Minnesota lakes”, *Journal of Field Robotics*, vol. 27, no. 3, pp. 681–685, 2010.

[33] B. Tovar and S. M. LaValle, “Visibility-based pursuit-evasion with bounded speed”, *International Journal of Robotics Research*, vol. 27, pp. 1350–1360, 12 2008.

[34] J. Vander Hook and V. Isler, “Pursuit and evasion with uncertain bearing measurements”, in *Proc. Candadian Conference on Computational Geometry*, 2014.

[35] Y. Wang, L. Dong, and C. Sun, “Cooperative control for multi-player pursuit-evasion games with reinforcement learning”, *Neurocomputing*, vol. 412, pp. 101–114, 2020.

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

[37] R. Zou and S. Bhattacharya, “On optimal pursuit trajectories for visibility-based target tracking game”, *IEEE Trans. Robotics*, vol. 35(2), pp. 449–465, 2019.