

# Shadow Information Spaces: Combinatorial Filters for Tracking Targets

Jingjin Yu

jyu18@uiuc.edu

Department of Electrical and Computer Engineering  
University of Illinois  
Urbana, IL 61801 USA

Steven M. LaValle

lavalle@uiuc.edu

Department of Computer Science  
University of Illinois  
Urbana, IL 61801 USA

**Abstract**—This paper introduces and solves a problem of maintaining the distribution of hidden targets that move outside the field of view while a sensor sweep is being performed, resulting in a generalization of the sensing aspect of visibility-based pursuit-evasion games. Our solution first applies information space concepts to significantly reduce the general complexity so that information is processed only when the shadow region (all points invisible to the sensors) changes combinatorially or targets pass in and out of the field of view. The cases of distinguishable, partially distinguishable, and completely indistinguishable targets are handled. Depending on whether the targets move nondeterministically or probabilistically, more specific classes of problems are formulated. For each case, efficient filtering algorithms are introduced, implemented, and demonstrated that provide critical information for tasks such as counting, herding, pursuit-evasion, and situational awareness.

## I. INTRODUCTION

Imagine a game of *hide-and-seek* is being played. After the hiders conceal themselves (subsequent relocations are allowed), the seekers, familiar with the environment, start to search for the hiders. Most people who played the game as schoolchildren know that an effective search begins with the seekers checking places having high probabilities of containing a hider, from previous experience: A closet, an attic, a thick bush, and so on. After the most likely locations are exhausted, the next step is to carry out a systematic search of the environment, possibly with some seekers guarding certain escape routes. Occasionally, during the game, hiders may attempt to relocate themselves to avoid being found. While the hiders succeed sometimes, they may end up being spotted by the seekers and are instead getting found earlier.

Although a child’s play, this game captures the two key interacting ingredients of pursuit evasion (PE) games: Passively estimating the distribution of hidden targets and actively planing to reduce the uncertainty of this distribution. The general goal in PE research is to algorithmically clear evading targets from a workspace. As pursuers try to ensure that the workspace is evader-free, they always need to maintain the pursuit status, remembering whether a region outside of the pursuers’ field-of-view (FOV) is contaminated or cleared (one bit of information per region).

This paper, expanding upon [51], [52], studies exactly this passive ingredient of PE games, reasoning about the information residing in unobservable regions of the environment. In

particular, we introduce the notion of *filters* over *shadow information spaces* for tracking moving targets in unobservable regions, as a generalization of this aspect of PE. To achieve this, we first process sensor observation history and compress it in a lossless fashion for our task classes, for storage and effective computation. Next, depending on whether the targets of interest are moving nondeterministically or probabilistically, concrete problems are formulated and solved by carefully manipulating and fusing observation and data. At a higher level, at any time, our algorithm can estimate the number of targets hidden in regions not directly observable. We note that, although the active problem of planning a pursuit path is not addressed in this work, heuristic search strategies can be readily implemented on the space of filter outputs.

The mathematical study of PE games dates back to at least four decades ago, with its roots in differential games [15]–[17]. Although optimal strategies for differential PE games are still actively pursued [2], [25], [29], [46], a variant of differential PE games, visibility based PE, has received much attention recently. Development of visibility based PE games can be traced back to [32], in which a PE game on discrete graph is introduced with the goal of sweeping evaders residing on continuous edges of a graph. The evaders can move arbitrarily fast, but must move continuously. The Watchman Route problem [8], [9], formulated twelve years later as a variant of the Art Gallery problems [27], [38], involves finding shortest route to clear static intruders. An intruder is considered cleared if a line of sight exists between the intruder and a point of the watchman route.

Influenced by these two threads of research, [42] defined what we know today as visibility based PE games in which the discrete graph domain is replaced by a path connected interior of a 2D polygon and a continuously moving evader is considered to be cleared if it falls into the visible region of a pursuer (in this case the pursuer is equipped with two flashlights and is called a 2-searcher). Thinking along the same lines as the Art Gallery problems, it was soon established that for a pursuer with an omnidirectional infinite range sensor (an  $\infty$ -searcher), it is NP-hard to decide the minimum number of pursuers needed for the class of simply connected polygons [14], [24]. The insight that bitangents and inflections fully capture the critical changes leads to a generalization from polygons to curved environments [23] and form the basis of

some critical events used in our work.

Various sensing and motion capabilities are explored in visibility based PE. Interestingly, it turns out that a 2-searcher is as capable as an  $\infty$ -searcher in simple polygons [31]. Pursuers with a single flashlight/beam (a 1-searcher) are investigated in detail in [39] and [20], with the latter limiting the pursuer’s motion to the boundary of the environment. Variations along this line include limited FOV [11], unknown environments [35], and bounded speed [44]. Another theme in PE games is to discretize time and put speed bounds on both pursuers and evaders. In this setting, sufficient conditions and strategies for a single pursuer to capture an evader are given to the classical lion-and-man problem in the first quadrant of the open plane [37]. This problem is then extended to  $\mathbb{R}^n$  and multiple pursuers in [21], and multiple pursuers with limited range in [5]. Finally, PE is also studied in the probabilistic context [18], [47] and abstract metric spaces [1].

Since we provide algorithms for tracking moving targets, our work is also closely related to target tracking and enumeration. The problem of accurately counting the number of targets with overlapping footprints is solved with a novel approach of integrating over Euler characteristics in [3]. With a virtual sensor that reports visible features of polygonal environment as well as indistinguishable targets, static targets are counted under various setups in [12]. A filtering algorithm is provided in [40] to count moving targets with a network of binary proximity sensors. In [48], Simultaneous Localization and Mapping (SLAM) and Detection and Tracking of Moving Objects (DTMO) are combined to attack both problems at the same time. A specialization of our problem is investigated in [43] in which the sensor FOV becomes one dimensional beams. Real-time people counting with a network of image sensors is studied in [50].

Another research area of relevance to this paper, especially the probabilistic formulations we give in Section VI, is the study of *optimal search* [41], which proposes a Bayesian approach for maintaining a target distribution and use that information for guiding the planning of optimal search paths. The essential idea from optimal search is to plan a path to eliminate regions with highest probability of containing the targets. In doing so, optimal search algorithms allow the prediction of the *no-detection likelihood* [4], [7], [26], [49], which is the probability that the targets remain undiscovered at given stages of a search effort, even before the actual search is carried out. Although our work also seeks to maintain a target distribution along a given path, we focus on the computational problem of how *topological* changes of non observable components, which are *combinatorial* in nature, can be correctly and efficiently processed as the target distribution evolves. This topological/combinatorial element of target tracking exists whether the problem formulation is probabilistic or not. In this aspect, the problems we address here are mostly orthogonal to classical optimal search problems, which cover environments (support surfaces of the distribution) that are mainly two dimensional, obstacle-less planes such as these appearing in typical maritime applications. As such, the results presented in this paper should benefit the extension of optimal search results to covering more diverse workspaces such as urban

areas and hilly terrains, where topological changes of non observable components are frequent.

The main contributions of this work are twofold. First, as explained above, we generalize visibility based PE by introducing a richer class of problems and providing a framework as a submodule for systematically attacking these problems. Second, the capability of effectively tracking hidden, moving targets, a general type of *situation awareness*, applies to a large class of time critical tasks in both civilian and national security applications. For example, in a fire evacuation scenario, knowing the the possible/expected number of people trapped in various parts of a building, firefighters can better decide which part of the building should be given priority when they coordinate the search and rescue effort.

The rest of the paper is organized as follows. Section II provides a mathematical definition of what we mean by “shadows” and “component events”, which can be best captured using a chronological sequence. Section III suggests the general problem of tracking hidden targets after bringing in moving targets and FOV events, . Section IV formulates the problem of estimating the number of targets hidden in shadows for nondeterministically moving targets and establishes its polynomial time solvability using results from integer linear programming theory. Section V shows how information spaces [22] can guide the design of efficient algorithms for solving the nondeterministic formulation in a more intuitive fashion. Section VI extends the problem formulation and solutions to probabilistically moving targets and imperfect sensors. We provide implementation, simulation results, and algorithmic analysis in Section VII, and conclude with Section VIII.

## II. COMPONENT EVENTS, SHADOWS, AND SHADOW SEQUENCES

Intuitively, in the hide-and-seek game, the part of the world that is not observable is comprised of many components, each of which has a life span. To study the information flow in them, a formal definition of components is first in order; the temporal relationship among them then naturally comes up.

### A. Component events and shadows

Let a nonempty set of robots move along continuous trajectories in a *workspace*,  $W = \mathbb{R}^2$  or  $W = \mathbb{R}^3$ . Let the configuration space of the robots be  $\mathcal{C}$ . At some time  $t$ , there may be configuration space obstacles  $\mathcal{C}_{obs}$  which may vary over time, leaving  $\mathcal{C}_{free} := \mathcal{C} \setminus \mathcal{C}_{obs}$  as the free configuration space. Let  $q \in \mathcal{C}_{free}$  be the configuration of the robots at time  $t$ . Returning to the workspace, there is a closed *obstacle region*  $O(t) \subset W$ , leaving  $F(t) := W \setminus O(t)$  as the free space. The robots are equipped with sensors that allow them to make shared observations in a joint *FOV* or *visible region*  $V(q, t) \subset F(t)$ . For convenience, we take the closure of  $V(q, t)$  and assume that the visible region is always closed. Let  $S(q, t) := F(t) \setminus V(q, t)$  be the *shadow region*, which may contain zero or more nonempty path connected components (path components for short). A path component is assumed to be nonempty unless otherwise specified. At any instant,  $O(t)$ ,  $V(q, t)$ , and  $S(q, t)$  have disjoint interiors by definition

and  $W = O(t) \cup V(q, t) \cup S(q, t)$ . Fig. 1 shows  $V(q), S(q)$  for a point robot holding a flashlight with  $F \subset W = \mathbb{R}^2$ ,  $\mathcal{C}_{free} \subset SE(2)$ , which is the set of two dimensional translations and rotations (here we omit the parameter  $t$  from  $F, V$ , and  $S$  since the obstacle region does not vary over time).

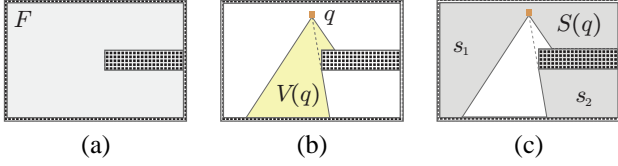


Fig. 1. (a) The environment and the free space,  $F$ . Note that in this example, the obstacle region is fixed; therefore,  $F$  is constant. (b) The visible region,  $V(q)$ . (c) The shadow region,  $S(q)$ , with two path components  $s_1, s_2$ .

To observe how path components of the shadow region evolve over time, let the robots follow some path  $\tau : [t_0, t_f] \rightarrow \mathcal{C}$  in which  $[t_0, t_f] \subset T \subset \mathbb{R}$  is a time interval. Let  $Z = W \times T$ . We may let  $O : T \rightarrow Pow(Z)$  be the map that yields the obstacle region and define  $V, S$  as:

$$V, S : \mathcal{C} \times T \rightarrow Pow(Z).$$

Since a path  $\tau$ , parameterized over  $t \in T$ , is always assumed in the paper, we abusively write  $V(t), S(t)$  in place of  $V(\tau(t), t), S(\tau(t), t)$ , respectively. In particular, we are interested in  $S(t)$  and call it a *slice*. For any  $(t_a, t_b) \subset [t_0, t_f]$ , let the union of all slices over the interval,

$$S(t_a, t_b) := \bigcup_{t \in (t_a, t_b)} S(t),$$

be called a *slab*, which is an open subset of  $Z$ . For any subset  $z$  of  $Z$ , define its projection onto the time axis as

$$\begin{aligned} \pi_t : Pow(Z) &\rightarrow Pow(T), \\ z &\mapsto \{t \mid (p, t) \in z \text{ for some } p \in W\}. \end{aligned}$$

Let  $s_{t,i} \subset S(t)$  denote the  $i$ -th path component of  $S(t)$  (assuming some arbitrary ordering). Let  $s_{i'}$  denote the  $i'$ -th path component of a slab  $S(t_a, t_b)$  (again, assuming some arbitrary ordering).  $S(t_a, t_b)$  is *homogeneous* if for all  $t \in (t_a, t_b)$  and all  $i$ , there exists  $i'$  such that

$$s_{t,i} = S(t) \cap s_{i'},$$

and separately,

$$\pi_t(s_{i'}) = (t_a, t_b) \text{ for all } i'.$$

A homogeneous slab is called *maximal* if it is not a proper subset of another homogeneous slab. The definition then partitions  $S(t_0, t_f)$  into some  $m$  disjoint maximally homogeneous slabs plus some slices

$$S(t_0, t_f) = S(t_0, t_1) \cup S(t_1) \cup \dots \cup S(t_{m-1}) \cup S(t_{m-1}, t_f).$$

That is, homogeneity of  $S(t_0, t_f)$  is broken at  $t_1, \dots, t_{m-1}$ . What exactly happens at  $t_1, \dots, t_{m-1}$ ? Let there be two homogeneous slabs  $S(t_a, t_b)$  and  $S(t_b, t_c)$  such that for some  $k \in \{1, \dots, m-1\}$ ,  $t_{k-1} \leq t_a < t_b = t_k < t_c \leq t_{k+1}$ . Let  $s_i$  be an arbitrary path component of  $S(t_a, t_b)$ , at  $t = t_b$ ,  $s_i$  may ( $\bar{s}$  denotes the closure of  $s$ )

- 1) *live on*, if there exists a path component  $s_j \subset S(t_b, t_c)$  such that  $\bar{s}_i \cap S(t_b) = \bar{s}_j \cap S(t_b) \neq \emptyset$ .
- 2) *disappear*, if  $\bar{s}_i \cap S(t_b) = \emptyset$ .

Similarly, a path component  $s_j \subset S(t_b, t_c)$  may

- 3) *appear*, if  $\bar{s}_j \cap S(t_b) = \emptyset$ .

Finally, a nonempty set of path components  $\{s_i\}$  of  $S(t_a, t_b)$  may

- 4) *evolve*, if there is a nonempty set of path components  $\{s_j\}$  of  $S(t_b, t_c)$  such that  $|\{s_i\}| + |\{s_j\}| \geq 3$  and  $\bigcup_i \bar{s}_i \cap S(t_b) = \bigcup_j \bar{s}_j \cap S(t_b) \neq \emptyset$  is a single path component of  $S(t_b)$ .

By definition, appear, disappear, and evolve are critical changes that only (and at least one of which must) happen between two adjacent maximally homogeneous slabs. We call these changes *component events*. With component events, homogeneity and maximality readily extend to path components of slabs. A path component  $s_i \subset S(t_a, t_b)$  is called *homogeneous* if no component events happen to a subset of  $s_i$  in  $(t_a, t_b)$ ;  $s_i$  is called *maximal* if it is not a proper subset of another homogeneous path component.

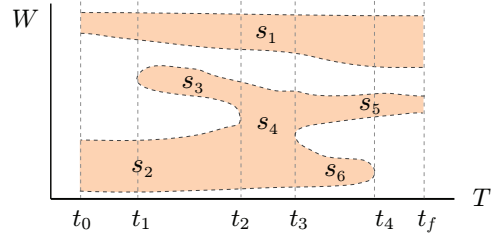


Fig. 2. Evolution of shadows:  $t_1$ :  $s_3$  appears,  $t_2$ :  $s_2, s_3$  merge into  $s_4$ ,  $t_3$ :  $s_4$  splits into  $s_5, s_6$ ,  $t_4$ :  $s_6$  disappears. The “sizes” of the shadows have no effect on the critical events.

At this point, a type of general position is assumed to avoid two tedious cases: 1) Four or more path components cannot be involved in an evolve event, and 2) Two or more component events cannot occur at the same time. In practice, non general position scenarios form a measure zero set and can be dealt with via small perturbations to the input if required. With such an assumption, exactly one component event happens between two maximally homogeneous slabs. Moreover, the evolve event can be divided into two sub events: *split* if  $|\{s_i\}| = 1, |\{s_j\}| = 2$  and *merge* if  $|\{s_i\}| = 2, |\{s_j\}| = 1$ .

We now piece together the above definitions using an example illustrated in Fig. 2. Restricting to the time interval  $(t_0, t_f)$ , there are five maximally homogeneous slabs,  $S(t_0, t_1), \dots, S(t_4, t_f)$ . Certain proper subsets of one of these, such as  $S(t_2^-, t_3^-)$  with  $t_2 < t_2^+ < t_3^- < t_3$ , are again homogeneous but no longer maximal; on the other hand, supersets such as  $S(t_2^-, t_3^+)$  with  $t_2^- < t_2 < t_3 < t_3^+$ , are no longer homogeneous. The slab  $S(t_0, t_f) \subset Z$  has two path components ( $s_1$  and  $\text{Int}(\bar{s}_2 \cup \bar{s}_3 \cup \bar{s}_4 \cup \bar{s}_5 \cup \bar{s}_6)$ , in which  $\text{Int}$  denotes the interior of a set) and six maximally homogeneous path components  $s_1, \dots, s_6$ . The intersection of a vertical line at  $t \in (t_0, t_f)$  with  $S(t_0, t_f)$  corresponds to the slice  $S(t)$ . For convenience, it is assumed that  $t = t_0$  is not a critical time in the sense that for each path component  $s_{t_0,i} \subset S(t_0)$ ,

$s_{t_0,i} = \overline{s_i} \cap S(t_0)$  for some path component  $s_i \subset S(t_0, t_1)$ . A similar assumption is made for  $t = t_f$ . Under this setup, there are four component events: 1.  $s_3$  appears at  $t = t_1$ ; 2.  $s_2$  and  $s_3$  merge to form  $s_4$  at  $t = t_2$ ; 3.  $s_4$  splits into  $s_5, s_6$  at  $t = t_3$ ; and 4.  $s_6$  disappears at  $t = t_4$ . In contrast, the path component  $s_1 \cap S(t_0, t_1)$  lives on through  $t_1, \dots, t_4$ .

Finally in this subsection, we define the main concept of the paper: *shadow*. It is easy to see that maximally homogeneous path components are pairwise disjoint. Let such a path component be called a *shadow* and let  $\{s_i\}$  be the set of shadows of  $S(t_0, t_f)$ ; note that  $S(t_0, t_f)$  is contained in the closure of  $\cup_i s_i$ . In the above example,  $\{s_i\} = \{s_1, \dots, s_6\}$ . For some  $t \in (t_0, t_f)$ , let a path component of  $S(t)$  be labeled as  $s_{t,i}$  if it is a slice of a shadow  $s_i$ . More precisely,  $s_{t,i} = s_i \cap \{(p, t) \mid p \in W\}$ . For  $t = t_0$ ,  $s_{t_0,i}$  is labeled such that  $s_{t_0,i} = \overline{s_i} \cap S(t_0)$  for some path component  $s_i \subset S(t_0, t_1)$ . The same applies to the labeling of  $s_{t_f,i}$ . A path component of  $S(t)$  has no label exactly when it is the border of two or more shadows of a slab. Since such labeling is unique, we drop time subscript of  $s_{t,i}$  if  $t$  is fixed. In the rest of the paper, we use the set  $\{s_i\}$  to denote both shadows and slices of shadows; we simply call both types of path components *shadows* when no confusion arises from the context. When we do need to distinguish, the former will be called *workspace-time shadows* and the later *workspace shadows*.

### B. Shadows are everywhere

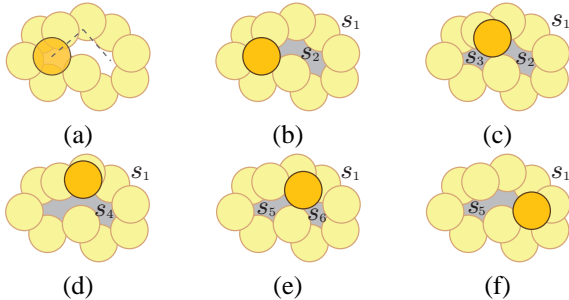


Fig. 3. An example of shadows and their indexing/labeling. (a) The set of spotlights and the path to be followed by the darker (orange) colored spotlight. (b) Initially, only shadow  $s_2$  and unbounded shadow  $s_1$  exist. (c) A new shadow  $s_3$  appears. (d)  $s_2, s_3$  merge into a single shadow  $s_4$ . (e)  $s_4$  splits into new shadows  $s_5, s_6$ . (f)  $s_6$  disappears.

To promote the intuition behind the mathematical definitions, let us look at a realistic example shown in Fig. 3(a). With the intention of guarding a planar region, spotlights are cast on the ground, creating a set of illuminated discs as shown. Assume that only the darker (orange) colored disc of light moves and follows the dashed line. For any position of the moving spotlight, the combined, illuminated set can be thought of as the FOV. Its complement in the plane is the shadow region, in which targets cannot be directly observed. Initially, there are two connected components, labeled  $s_1$  (unbounded) and  $s_2$ , in the shadow region. As the spotlight moves along the dashed line, we observe that shadows may appear, disappear, merge, and split, as illustrated in Fig. 3(b) to (f). We constructed this example so that the events and

evolution of shadows match exactly these of the example from Fig. 2.

The naive example suggests that shadows and component events arise from very simple setups. Indeed, shadows and component events are ubiquitous, showing up whenever moving sensors are placed inside environments. We provide three additional examples to corroborate this point; many others could be presented. In Fig. 4(a), omni-directional, infinite range sensors partition the 2D environment into polygonal shadows. The component events happen exactly when the sensors make *inflection* and *bitangent* crossings (see aspect graphs [33]), which gives rise to the concept of *gaps* and *gap navigation trees* as discussed in [45]. If the sensors have limited viewing angle [11] or limited range (Fig. 4(b)), alternate models governing visible and shadow regions are obtained. In Fig. 4(c), fixed infrared beams and surveillance cameras are placed inside a building, creating a set of three fixed shadows  $s_1, s_2, s_3$ . Such a setting is common in offices, museums, and shopping malls. As a last example, Fig. 4(d) shows a simplified mobile sensor network with coverage holes. In this case, the joint sensing range of the sensor nodes is the FOV and the coverage holes are the shadows, which fluctuate continuously even if the sensor nodes remain stationary (consider cellphone signals).

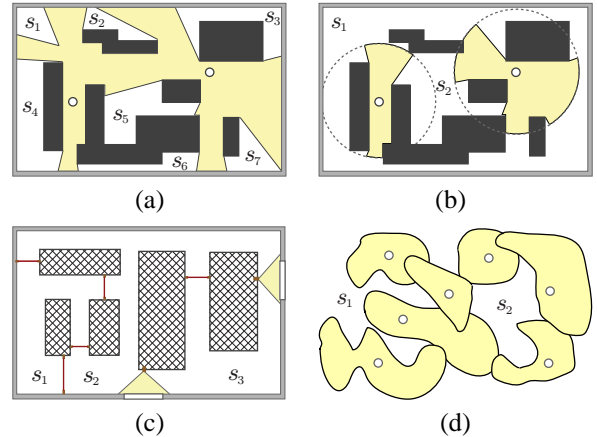


Fig. 4. a) Two robots (white discs) carrying omni-directional, infinite range sensors. The free space is partitioned into seven shadows. b) When sensing range is limited, the topology of shadows changes; only two shadows are left. c) An indoor environment guarded by fixed beam sensors (red line segments) and cameras (yellow cones). There are three connected shadows. d) A simple mobile sensor network in which the white discs are mobile sensing nodes, with shaded regions being their sensing range at the moment. There are two shadows with  $s_1$  being unbounded.

For some environments, shadows are readily available or can be effectively computed with high accuracy, such as visibility sensors placed in 2D polygonal environments. In some other cases, shadows are not always easy to extract. As one example, estimating coverage holes in a wireless sensor network is rather hard since it is virtually impossible to know whether a point  $p$  is covered unless a probe is dispatched to  $p$  to check. It is also well known that 3D visibility structure is difficult to compute [28], [34]. Even though we do not claim to overcome such inherent difficulties in acquiring visibility region and/or shadows, the method presented here applies as

long as a reasonably accurate characterization of the shadows is available.

### C. Shadow sequences

We conclude this section with the introduction of a *shadow sequence*, of which the importance will become more apparent in coming sections. When the shadows of  $S(t_0, t_f)$  for a fixed path  $\tau$  are put together, a sequential structure comes up. This structure, which we call a shadow sequence, captures the combinatorial changes of the labeled shadows through component events. A graphical illustration of the shadow sequence for the example in Fig. 3 is given in Fig. 5.

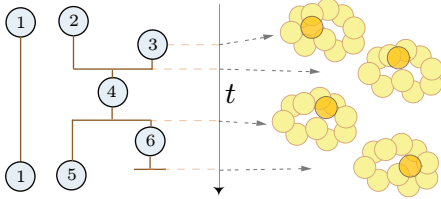


Fig. 5. A graphical illustration of the shadow sequence for the example from Fig. 3. The numbers in the circles represent the labels of the shadows. The four events marked on the time line, from top to bottom, are appear, merge, split, and disappear. As one expects, this figure closely resembles Fig. 2.

## III. TARGETS, FOV EVENTS, AND IMPERFECT SENSORS

### A. Targets

Our interest in shadows lies with maintaining information that is not directly observable by sensors. To effectively investigate how to track such information, we briefly characterize what we mean by information. We assume that there is a non-negative integer number of *targets* in  $F$ , which are point entities that move arbitrarily fast but follow some continuous, unpredictable trajectories. The robots' sensors can detect certain *attributes* of these targets. We are interested in two types of attributes: location and identity.

**Location.** When the targets move in/out the sensors' FOV, their appearance/disappearance may be detected. Depending on the sensors' capabilities, at least two levels of precision are possible: 1. The sensors can tell whether the FOV contains no target or at least one. In other words, each sensor's output is *binary* (motion detector is a sensor of this type). 2. Each target inside the FOV can be precisely located and counted.

**Identity.** When multiple targets are present, it may be possible to tell them apart. That is, the sensors may be able to *distinguish* the targets in the FOV. Roughly speaking, the targets may be: 1. *Fully distinguishable*. When targets possess unique IDs recognizable by the sensors, they are fully distinguishable. 2. *Indistinguishable*. Although it appears that full distinguishability is the most powerful, it is not always available due to sensor cost constraint or even desirable due to concerns such as privacy. It is not hard to make targets indistinguishable: In the sensor output, erase any attributes that can be used to distinguish among the targets. 3. *Partially distinguishable*. Everything between the previous two notions

of distinguishability belongs to this class. For instance, targets may form *teams* that are distinguishable by color.

Location and identity are related – full distinguishability implies that the sensors should be able to locate targets in the FOV. On the other hand, tracking locations over time can be used to distinguish targets. However, these two attributes are not identical and it benefits to treat them orthogonally. For example, when colored teams of targets are present, a low resolution overhead camera can easily tell whether a team is present in the FOV via a color scan, acting as a combination of binary location sensor and identity sensor. Given sensors that can detect some subsets of the above mentioned attributes of targets, each labeled shadow can be assigned one or more variables that describe these attributes of the targets residing in the shadow. Note that although we deal mostly with binary and integer variables in this paper, variables of other forms, such as real numbers, can also be incorporated over the structure of shadows and component events introduced here. When we consider targets in the shadows, a type of invariance arises:

**Observation 1** *In an environment with only component events, the number of targets hidden in a workspace-time shadow is invariant along its span over time; furthermore, a workspace-time shadow is a maximal set in which such invariance holds.*

By the assumption that a hidden target moves continuously, its trajectory is contained in the same workspace-time shadow when no component events happen. Two workspace shadows, as different time slices of the same workspace-time shadow, must intersect the same number of such trajectories since no target enters or exits the component in the time being. This yields the invariance. The second claim follows the definition of workspace-time shadow as a maximal union of all such workspace shadows.

### B. FOV events

If a location sensor also has *memory*, it will be able to detect changes to the number of targets in the FOV during a short time interval. We call such a change a *field-of-view event* (FOV event for short), which is a second type of critical events of our interest. Furthermore, if the sensors know where a FOV event happens, these events can be associated with corresponding shadows. For a shadow  $s_i$ , three FOV events are possible: 1) A target *enters*  $s_i$  from the FOV, 2) A target *exits*  $s_i$  into the FOV, and 3) Nothing happens at the boundaries between  $s_i$  and the FOV (for a period of time), or *null* event. Denoting these events  $e_e, e_x, e_n$ , respectively, the collection of possible FOV events for a shadow  $s_i$  is the set

$$E_{FOV} = \{e_e, e_x, e_n\}.$$

Some sensors may only detect the enter and exit events explicitly, such as a sensing node in a sensor network that only senses targets passing through the boundary of its sensing range. For detection beams, the FOV is a line segment, which causes two FOV events to happen consecutively (see Figure 6). Certain systems may not have FOV events at all; an instance is a pursuit evasion game in which the evader always avoids



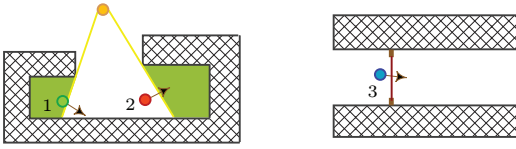


Fig. 6. Illustration of FOV events for an environment with obstructed visibility (left) and for an environment with detection beams (right). 1) A target is about to exit a shadow into the FOV of the sensor (yellow disc). 2) A target is about to enter a shadow from the sensor's FOV. 3) A target is about to enter and exit the FOV of a beam sensor.

appearing in the pursuer's FOV. The game ends when an evader is found or when it is confirmed that no evader is in the environment.

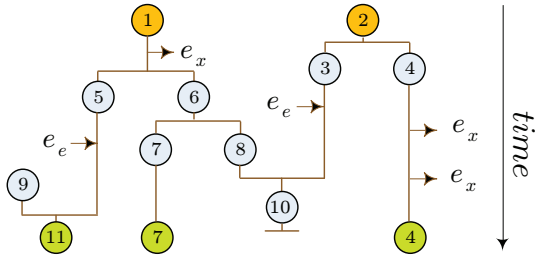


Fig. 7. A typical sequence of critical events. The circles with numbers represent the shadows; the labeled arrows associate FOV events to shadows.

Since component events and FOV events both happen as robots move along some path  $\tau$  in the free space  $F$ , it makes sense to treat them as a whole. It does not take much to represent them together: We can simply augment the shadow sequence to include the FOV events. A typical combined sequence of critical events is shown in Fig. 7. To incorporate FOV events, the invariance from Observation 1 needs to be updated.

**Observation 2** *In an environment with component and FOV events, the number of targets hidden in a workspace-time shadow is invariant between FOV events (excluding null events) associated with the shadow; the time span of such invariance is again maximal.*

To see why the above statement is true, note that an enter FOV event can be viewed as an appear component event immediately followed by a merge component event. Same breakdown holds for exit FOV events. The case is then reduced to Observation 1. Observation 2 establishes that for the task of tracking hidden targets that move continuously, any sensor data unrelated to critical events can be safely discarded without adverse effects.

### C. The problem of tracking hidden targets

With the introduction of component and FOV events, we can formally define the general problem of tracking hidden targets that move continuously. The following inputs are assumed:

- 1) An initial distribution of targets (in shadows) whose total number remains invariant,

- 2) An ordered sequence of component and FOV events for the time interval  $[t_0, t_f]$ , and
- 3) Any target motion dynamics (for example, nondeterministic) that may provide additional information about critical events.

From these inputs, the task is to track the evolution of the target distribution and in particular, to estimate at  $t = t_f$  the possible number of targets in a given set of shadows. For the rest of this paper, we focus on two flavors of this problem: 1) A nondeterministic setting in which targets move nondeterministically but critical events are observed without error, and 2) A probabilistic setting in which the targets' movement has a probabilistic model and there are imperfect sensors.

## IV. TRACKING NONDETERMINISTICALLY MOVING TARGETS: THE FORMULATION AND AN ILP PERSPECTIVE

### A. Problem formulation

In the *nondeterministic* setting, we assume that the targets move nondeterministically. In particular, when a shadow  $s_i$  splits into shadows  $s_j, s_k$ , the targets inside  $s_i$  can split in any possible way as long as the numbers of targets in  $s_j, s_k$  are both nonnegative. The component events and FOV events are assumed to be observed without error. Given such assumptions, the observation history can be partitioned into two inputs to our filter algorithm:

- 1) A sequence of shadow and FOV events, and
- 2) The initial conditions of targets in the shadows at time  $t = t_0$ .

A typical initial condition for a shadow takes the form

$$\{(a_1, l_1, u_1), (a_2, l_2, u_2), \dots, (a_k, l_k, u_k)\}, \quad (1)$$

in which  $a_i$  denotes a subset of target attributes (such as having red color). We assume that elements of the set  $\{a_i\}$  for a shadow are pairwise disjoint: If  $a_i$  has red color, then no  $a_j, j \neq i$  can include targets with the attribute of having red color. The corresponding  $l_i$  and  $u_i$  denote the lower and upper bounds on the number of targets in the shadow with attribute  $a_i$ . For example, we may know that at the beginning, a shadow have 6 to 9 green targets and 5 targets that may be blue or red. In this case, the initial condition can be written as

$$\{(c = \text{green}, 6, 9), (c = \text{blue or red}, 5, 5)\}.$$

With these inputs, the main task is to determine the lower and upper bounds on the number of targets in any given set of shadows at  $t = t_f$  for any combinations of attributes. These obtained bounds are always tight in the sense that any target distribution falling in these bounds is a possible outcome given the initial condition and the observation history.

To make the explanation of the algorithm clear, we first work with a single attribute and ignore FOV events. We also assume for the moment that the initial conditions are tight in the sense that all possible choices of values must be consistent with the later observations (for example, we cannot have a initial condition of 4 to 6 targets in a shadow and later find that it is only possible to have 2 targets in it). We will then show

how FOV events, multiple attributes, and other extensions can be handled incrementally.

### B. An integer linear programming (ILP) perspective

For the simplest case, since there is a single attribute and the FOV events are ignored, we can represent the number of targets in a shadow with a single unknown quantity. Let the set of shadows be  $\{s_i\}$ ; we denote the set of corresponding unknowns as  $\{x_i\}$ . We can write the initial condition for each shadow at  $t = t_0$  as two constraints

$$l_i \leq x_i \leq u_i. \quad (2)$$

For each event in the sequence of component events, we then obtain one extra constraint of the following form:

$$\begin{array}{ll} \text{Appear or disappear:} & x_i = d_i \\ \text{Split or merge:} & x_i = x_j + x_k. \end{array} \quad (3)$$

Here we allow that as an appear event happens,  $d_i$  targets may hide in  $s_i$  at the same time. This is more general than letting  $d_i = 0$ . To unify notation, we write these in the same way as the initial conditions by letting  $l_i = u_i = d_i$ . The same applies to the disappear events. Additionally, we have for each shadow  $s_i$ , the constraint

$$x_i \geq 0. \quad (4)$$

Finally, the task becomes finding the lower and upper bounds of targets for a set of shadows at time  $t = t_f$  indexed by  $\mathcal{I}$ . For the upper bound, we can write the problem as maximizing the sum of the set of unknowns

$$\text{maximize } \sum_{i \in \mathcal{I}} x_i. \quad (5)$$

Finding the lower bound then becomes maximizing the set of unknowns not indexed by  $\mathcal{I}$  because the total number of targets are preserved. We have obtained an integer linear programming (ILP) problem: All critical events can be expressed using constraints of forms from (2),(3) and (4), with the objective function having the form from (5). For example, if we are to express the ILP problem in the *canonical form*, all we need to do is to split each equality constraint (given by (3)) into two inequality constraints (for example,  $x_i = d_i$  becomes  $x_i \leq d_i$  and  $x_i \geq d_i$ ) and multiply all inequality constraints with  $-1$  where necessary ( $x_i \leq d_i \Rightarrow -x_i \geq -d_i$ ). This gives us the ILP problem in canonical form,

$$\text{minimize } \sum_{i \in \mathcal{I}} -x_i, \quad \text{subject to } Ax \geq b, x \geq 0, \quad (6)$$

in which  $A$  is the constraint coefficient matrix accumulated from initial condition and the critical events;  $x$  is the vector of unknowns (one for each shadow). The size of  $A$  is determined by the number of shadows and the number of critical events. For additional discussion on ILP modeling, see [30].

### C. Polynomial time solvability of the ILP problem

It is well known that the class of ILP problems is NP-complete in general. It turns out, however, that our ILP problem is not only feasible, but also efficiently solvable. We point out that an actual target tracking problem may require

solving more than a pair (upper and lower bounds) of ILP problems as formulated in 6. For example, in a fire rescue scenario, it may be necessary to estimate upper and lower bounds on all current shadows individually. Nevertheless, as long as the number of ILP problems are manageable (say, linear with respect to the size of the inputs), the overall problem can also be efficiently solved.

**Proposition 3** *A polynomial time algorithm exists for the system described by (6).*

As a first step in proving the proposition, we establish the following lemma:

**Lemma 4** *The constraint matrix  $A$  in (6) is totally unimodular<sup>1</sup>.*

**PROOF.** We use induction over the size of square submatrices of  $A$  to prove that all such submatrices must have determinant 0 or  $\pm 1$ . As the base case, every element of  $A$  is 0 or  $\pm 1$ . Suppose that all square submatrices of order  $n$  have determinant 0,  $\pm 1$ . Denote these matrices  $\mathcal{M}_n$ . Suppose there is a square submatrix  $M$  of  $A$  of order  $(n+1)$  with determinant not in  $\{0, \pm 1\}$ . Every constraint arising from (2), (3), and (4) except  $x_i = x_j + x_k$  introduces rows in  $A$  with a single  $\pm 1$  in them; the rest of the row contains only 0s. If  $M$  contains a row arising from these types of constraint then  $M$  must have determinant 0,  $\pm 1$  by induction. Suppose not. In this case, all rows of  $M$  are introduced by constraint of type  $x_i = x_j + x_k$ . Each such constraint brings in two rows of  $A$  with opposite signs and therefore cannot both appear in  $M$ . We can assume that  $M$ 's first row have coefficients coming from one of the rows introduced by a split event,  $x_i = x_j + x_k$ . As a first case, let the  $i, j, k$ th columns of  $A$  correspond to  $i', j', k'$ th columns of  $M$ , respectively. To make  $M$ 's determinant not in  $\{0, \pm 1\}$ , there needs to be another row in  $M$  that contains exactly two non zero elements among  $i', j', k'$ th columns. This is only possible if  $s_j$  and  $s_k$  merge again, giving a constraint of the form  $x_j + x_k = x_l$ . We may let this row be the second row in  $M$ . This suggests that  $j', k'$ th column of  $M$  are all zeros after the second row; but this gives us that  $M$  has determinant 0. The second case is that  $M$  includes only two columns of  $A$ 's  $i, j, k$ th columns. It can be checked similarly that  $M$  must have determinant 0. ■

**PROOF OF PROPOSITION 3.** When the constraint matrix  $A$  is totally unimodular and  $b$  is a vector of integers, then the minimal faces of the constraint polytope must assume integer coordinates, making the solution of the relaxed linear programming (LP) problem also the solution to the original ILP problem [36]. It is clear that  $b$  in (6) is integer. Lemma 4 gives us that  $A$  is totally unimodular. Therefore, a polynomial time algorithm such as interior point method can be applied to solve (6). ■

<sup>1</sup>An integer square matrix  $A$  is unimodular if  $\det A = \pm 1$ . A matrix  $B$  is totally unimodular if every non-singular square submatrix of  $B$  is unimodular.

## V. TRACKING NONDETERMINISTICALLY MOVING TARGETS: AN INFORMATION SPACE PERSPECTIVE AND EFFICIENT ALGORITHMS VIA COMBINATORIAL FILTERS

Although Proposition 3 tells us that the nondeterministic formulation can be solved in polynomial time using generic LP algorithms, it is not clear that these algorithms fully explore the intrinsic structure of the problem at hand. In this section, we briefly review the *information space* (I-space for short, see Chapter 11 of [22] for an introduction) and show how the I-space framework can help with the systematic exploration of the structure of filtering problems that are combinatorial in nature. For our particular problem, we show that additional information can be discarded from the shadow sequence to yield a further condensed *information state* (I-state). Algorithmic solutions based on max-flow are then introduced, followed by various extensions.

### A. Information space as a guiding principle for task based data filtering

As a shadow sequence is extracted from an observation history, a much condensed combinatorial structure is left. This choice is not arbitrary: The general task of tracking unpredictable targets outside the sensor range induces an equivalence relation over the workspace-time space that yields the space of shadows; the evolution of these shadows then gives rise to a space of shadow sequences. In this subsection, we review I-space/I-state concepts and explain how shadow sequences can be viewed as *derived* I-states and the space formed by them a *derived* I-space. We also characterize how I-spaces/I-states, tasks, and filters are closely related.

For any problem, I-space analysis begins with the *history I-space*,  $\mathcal{I}_{hist}$ , which is essentially the set of all data that robots may ever obtain. Formally, for a time period  $[t_0, t_f] \subset T$ , a perfect description of everything that occurred would be a *state trajectory*  $\tilde{x}_t : [t_0, t_f] \rightarrow X$ , in which  $X$  is the combined state space of robots and targets. It is impossible to obtain this because not all target positions are known. What is available is the robots' trajectory  $\tilde{q}_t = \tau$  and the sensor *observation history*  $\tilde{y}_t : [t_0, t_f] \rightarrow Y$ , produced by a sensor mapping  $h : X \rightarrow Y$ , in which  $Y$  is the observation space of the sensors. Let the robots also have access to some initial information  $\eta_0$  at  $t = t_0$ . The *history I-state* at time  $t$ ,  $\eta_t = (\eta_0, \tilde{q}_t, \tilde{y}_t)$ , represents all information available to the robots. The *history I-space*  $\mathcal{I}_{hist}$  is the set of all possible history I-states.  $\mathcal{I}_{hist}$  is an unwieldy space; it must be greatly reduced if we expect to solve interesting problems. Imagine a robot equipped with a GPS and a video camera moves along some path  $\tau$ . Without a specific task, the robot will not be able to decide what information it gathers is useful; therefore, it has to store all of  $\tilde{q}_t, \tilde{y}_t$ . Even at a relatively low spatial resolution and a frequency of 30 Hz, just keeping the robot's locations and the camera's images in compressed form requires a large amount of storage space, which presently is not generally possible over a long time period.

Once a task is fixed, however, it may become possible to reduce  $\mathcal{I}_{hist}$  dramatically. For our specific task of tracking

hidden targets in shadows, as we have established in Observation 2, all we need to know is the initial distribution of targets, the component events, and the FOV events. Since targets move unpredictably, other information contained in  $\eta_t$  does not help: the robots' exact location, the shape of the workspace shadows, and what the targets in the FOV are doing are not relevant. Thus, Observation 2 allows us to construct a *derived I-space*  $\mathcal{I}_{ss}$ , called the *shadow sequence I-space* that discards the irrelevant information. Consider the information contained in  $\eta_t = (\eta_0, \tilde{q}_t, \tilde{y}_t)$ . To derive  $\mathcal{I}_{ss}$ , the following reductions are made over  $\eta_0, \tilde{q}_t, \tilde{y}_t$ :

- 1) The initial distribution of targets is extracted from  $\eta_0$ .
- 2) The shadow sequence is extracted via processing  $\tilde{q}_t$  and  $\tilde{y}_t$ .
- 3) The observation history  $\tilde{y}_t$  is compressed so that only critical events and temporal order between these events need to be recorded.

The result from this reduction is the *shadow sequence I-state*  $\eta'_t$  (Fig. 7 gives an example) that lives in  $\mathcal{I}_{ss}$ .  $\mathcal{I}_{ss}$ , as a complete yet more compact representation, immediately reveals much more structure that is intrinsic to our task than  $\mathcal{I}_{hist}$  does. From this we observe a general pattern that we exploit: Given  $\mathcal{I}_{hist}$  and a task, we try to find one or more sufficient derived I-spaces, and work exclusively in these derived I-spaces. In signal processing, a filter is defined as a device or process that removes from a signal unwanted features [19]. In this sense, the process of extracting shadows from  $\tilde{q}_t$  and  $\tilde{y}_t$  is exactly a filter. Moreover,  $\mathcal{I}_{hist}$  and  $\mathcal{I}_{ss}$  are connected through this filter. From this perspective, solving a task becomes finding the correct I-space, applying the associated filter, and performing additional computation as events happen. For the nondeterministic formulation, we call such filters *combinatorial filters*.

### B. The bipartite I-space

As mentioned earlier, generic LP algorithms may not explore the full structure of our problem. One interesting property of our problem is that the distribution of targets in the shadows mimics network commodity flow. Another intrinsic and key property of our problem is that, in many cases, the relative order of component events does not affect the possible target distribution in the shadows. For example, the two shadow sequences in Fig. 8 are equivalent: the set of shadows at  $t = t_f$  are basically the same. This allows us

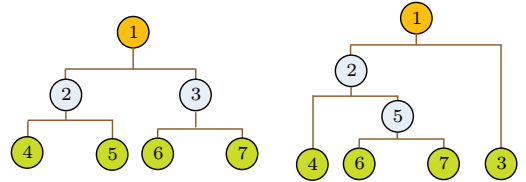


Fig. 8. Two shadow sequences that are equivalent for task of estimating lower and upper bounds on the number of targets in the shadows at time  $t = t_f$ .

to safely discard the intermediate shadows to obtain a more compact I-space  $\mathcal{I}_{bip}$ , the *bipartite I-space*. The basic idea



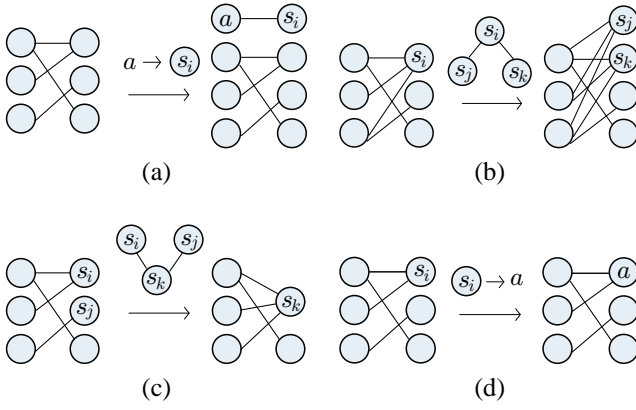


Fig. 9. Incrementally computing I-states in  $\mathcal{I}_{bip}$ : a) An appear component event in which  $a$  targets goes into shadow  $s_i$  adds two vertices and an edge, with  $a$  associated with the left vertex. b) A split event splits a vertex and all edges pointing to that vertex. c) A merge event collapses two vertices into one and collapses their ingoing edges. d) A disappear event in which  $s_i$  is revealed to have  $a$  targets in it only associates  $a$  with the vertex on the right side.

behind compressing  $\mathcal{I}_{ss}$  into  $\mathcal{I}_{bip}$  is that, since the robots' sensors cannot obtain information from the shadows as the robots move around, the information that really matters is how shadows from the beginning and the current time are related, while discarding the shadows from intermediate times. By conservation of targets in the environment, the number of targets in the shadows at  $t = t_0$  and appeared shadows must be equal that in the shadows at  $t = t_f$  and disappeared shadows. This hints towards a bipartite graph structure, which is why we denote the space of such I-states the bipartite I-space. To do the filtering, the component events are processed individually according to the procedure shown in Fig. 9. By the construction of  $\mathcal{I}_{ss}$  and  $\mathcal{I}_{bip}$ , we have shown that  $\mathcal{I}_{hist}$ ,  $\mathcal{I}_{ss}$ , and  $\mathcal{I}_{bip}$  describe the same ILP problem:

**Proposition 5** *Given that targets move nondeterministically, information from  $\mathcal{I}_{hist}$  and the corresponding  $\mathcal{I}_{ss}$ ,  $\mathcal{I}_{bip}$  describe the same ILP problem of the form (6).*

**PROOF.** The invariance from Observation 2 gives us that  $\mathcal{I}_{hist}$  and  $\mathcal{I}_{ss}$  are equivalent in capturing the distribution of hidden targets. To see that  $\mathcal{I}_{ss}$  and  $\mathcal{I}_{bip}$  are equivalent, we may consider each hidden target individually: Any flow of a target along a shadow sequence is possible in the corresponding bipartite structure, by construction. ■

A graphical illustration of relationship between I-spaces and I-states, which summarizes the I-space discussion, is given in Fig. 10. We point out that such hierarchical structures exist regardless of whether the formulation is nondeterministic or probabilistic; it so happens that for our filtering problem, the nondeterministic formulation leads to one more level of natural structure than the probabilistic formulation (see Section VI).

### C. Tracking targets as a max-flow problem

With the bipartite I-state structure, we are ready to illustrate the complete combinatorial filtering process with a concrete

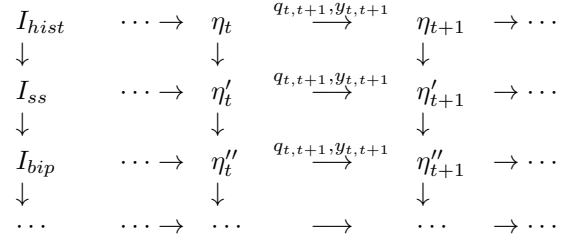


Fig. 10. Although it is possible to obtain  $\eta'_{t+1} \in \mathcal{I}_{ss}$  from  $\eta_t \in \mathcal{I}_{hist}$ , it is also possible to derive it from  $\eta_t$  and  $q_{t,t+1}, y_{t,t+1}$ . Same holds for  $\eta''_{t+1} \in \mathcal{I}_{bip}$ .

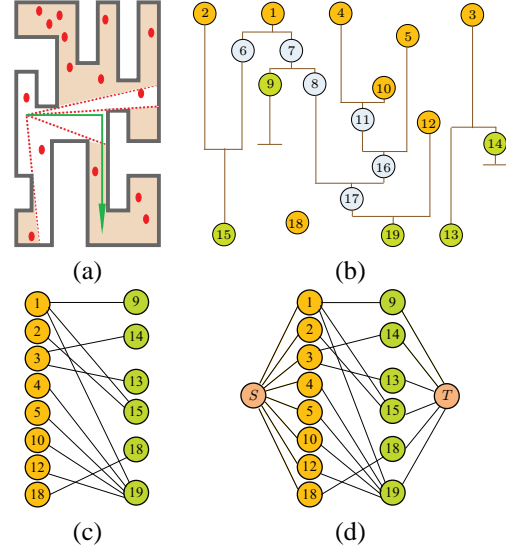


Fig. 11. a) A 2D office like environment. A single robot follows the green path. Red dots are illustrations of possible targets in the environment. b) The shadow sequence I-state for the environment and path. The orange indexed shadows are these at  $t = t_0$  or appearing; the green ones are these at  $t = t_f$  or disappearing. Shadow  $s_{18}$  are both appearing and existing at  $t = t_f$ . c) The bipartite I-state. d) Augmented graph for running max-flow algorithm.

example (the procedure was first introduced in [51]). After obtaining the bipartite structure, the rest of the algorithm is nothing more than applying a maximum flow subroutine (such as Edmonds-Karp) [10]. For the environment given in Fig. 11(a), a visibility cell decomposition procedure [6] will give us the shadow sequence I-state in Fig. 11(b). Applying the  $\mathcal{I}_{bip}$  filter then gives us the bipartite graph in Fig. 11(c). Note that each shadow becomes a vertex (sometimes two vertices) of the bipartite graph. Once the bipartite graph is constructed, the task of determining lower and upper bounds on shadows at  $t = t_f$  can be transformed into a max-flow problem. To achieve this, we first augment the graph by adding a source vertex  $S$  and sink vertex  $T$ . An edge is added between  $S$  and each shadow at  $t = t_0$  as well as each appeared shadow, and an edge is added between  $T$  and each shadow at  $t = t_f$  as well as each disappeared shadow. The end result of doing this to the graph in Fig. 11(c) is Fig. 11(d).

After obtaining the extended graph, capacities need to be assigned to edges of the graph before running max-flow. Let  $e(v_1, v_2)$  be an edge in the graph from vertex  $v_1$  to vertex  $v_2$ , and denote the capacity and flow on the edge

as  $c(v_1, v_2), f(v_1, v_2)$ , respectively. Suppose that we want to obtain the upper bound on the number of targets in shadow  $s_{19}$ . The edges of the original bipartite graph will always have infinite capacities, which we do not mention again. For each edge between  $S$  and a shadow indexed  $i$ , let  $c(S, i) = u_i$ . In our example these indices are 1-5, 10, 12, and 18. For each edge between a disappearing shadow indexed  $i$ , and  $T$ , let  $c(i, T) = l_i$ . These are 9 and 14 in our example. Since we want as many targets to go to  $s_{19}$  as possible, we let  $c(i, T) = 0$  for  $i = 13, 15, 18$  and  $c(19, T) = +\infty$ . After running the max-flow algorithm, the maximum possible number of targets that can end up in  $s_{19}$  is given by

$$f(19, T) + \sum_i f(i, T) - \sum_i c(i, T), \quad (7)$$

in which the summations are over indices of disappearing shadows. We need to consider disappearing shadows since these shadows should have flow equal to their capacity, which is not guaranteed by a max-flow algorithm. Filling in numbers into this example, assuming that the input lower/upper bound pairs for shadows 1-5, 9, 10, 12, 14, 18 are (2, 4), (0, 3), (5, 5), (2, 6), (4, 5), (2, 3), (1, 3), (3, 8), (2, 4), (5, 7), respectively, then (7) gives us that shadow 19 can have at most 24 targets. If we instead want the lower bound on the number of targets in  $s_{19}$ , we should let  $c(S, i) = l_i$ ,  $c(i, T) = l_i$  for  $i = 9, 14$ ,  $c(i, T) = +\infty$  for  $i = 13, 15, 18$  and  $c(19, T) = 0$ . After running the max-flow algorithm,  $s_{19}$ 's lower bound is given by

$$\sum_i c(S, i) - \sum_j f(j, T), \quad (8)$$

in which the first summation is over all shadows connected to  $S$  and the second summation is over all shadows connected to  $T$ . Using the earlier numbers, this minimum is 10 for shadow 19. The same procedure applies to an arbitrary set of shadows.

#### D. Incorporating FOV events

In the nondeterministic setting, there is no null FOV event. As mentioned in Observation 2, exit and enter FOV events can be handled by converting them into component events. To convert an enter FOV event of shadow  $s_i$  into component events, we simply create an appear component event of a single target and then merge the newly created shadow into  $s_i$ . Similarly, an exit FOV event can be converted into a split component event followed by a disappear component event. The rest of the algorithm stays the same. The problem is, however, if there is a large number of FOV events compared to the number of component events, this approach will slow down later steps of the algorithm since it will create two component events per FOV event. Fortunately, there is no reason to handle each FOV event individually; since each FOV event is associated with some shadow, we can group them based on this association. The only caveat is that we cannot just group all FOV events for one shadow into a single *batch* FOV event as this can introduce information loss. For example, if  $e_x, e_x, e_e, e_e$  happens to shadow  $s_i$ , this is not equivalent to nothing has happened: we know that  $s_i$  must have at least two targets in it originally (a ‘‘surplus’’). On the other hand, the just

mentioned surplus and net target flow are the only two pieces of information that FOV events of a shadow give us; hence up to two batch FOV events can summarize all information contained in all FOV events for a given shadow. Let  $\langle e_j \rangle$  be the sequence of FOV events for a shadow  $s_i$  in which  $e_j$  is either  $e_e$  or  $e_x$ , we build a counter to track the surplus of  $s_i$  as  $d_{\min} = \min\{d_j\}$ , with  $d_j$  defined as

$$d_j = \begin{cases} d_{j-1} + 1 & \text{if } e_j = e_e \\ d_{j-1} - 1 & \text{if } e_j = e_x \\ 0 & \text{if } j = 0. \end{cases}$$

Let  $d_{\text{tot}}$  be  $d_j$  for the last  $j$ , the net target flow from FOV events. We have four cases. If  $d_{\min} = d_{\text{tot}} = 0$ , we do nothing. If  $d_{\min} \geq 0$  and  $d_{\text{tot}} > 0$ , we only need to create one batch enter FOV event for  $s_i$  with  $d_{\text{tot}}$  number of targets. If  $d_{\min} < 0$  and  $d_{\text{tot}} = d_{\min}$ , we only need to create one batch exit FOV event with  $|d_{\min}|$  number of targets. In the last case, we need to create one batch exit FOV event with  $|d_{\min}|$  number of targets and then an enter FOV event with  $d_{\text{tot}} - d_{\min}$  number of targets. We can then apply the naive approach from the beginning of this subsection to convert these batch FOV events into component events. With this construction, we never need to handle more than  $5n$  events in which  $n$  is the maximum number of shadows.

#### E. Solving a variety of other tasks

The ability to obtain lower and upper bounds of the number of targets hiding inside a shadow easily extends to other useful tasks. We briefly cover a few of these variations.

*Refining initial bounds:* Max-flow computations can also be used to refine the lower and upper bounds from initial conditions if they are not tight. To get a refined lower bound for a shadow at  $t = t_0$ , say  $s_1$  from Fig. 11(b), let  $c(S, 1) = l_1$ ,  $c(S, i) = u_i$  for  $i \neq 1$ ,  $c(i, T) = u_i$  for disappearing shadows, and  $c(i, T) = 0$  for the rest. After running max-flow on this network, a tighter lower bound, if there is one, is given by

$$l'_1 = l_1 + \sum_i c(i, T) - \sum_j f(j, T). \quad (9)$$

The summations are done similar to that of (8). To refine  $u_1$ , let  $c(S, 1) = u_1$ ,  $c(S, i) = l_i$  for  $i \neq 1$ ,  $c(i, T) = u_i$  for disappearing shadows and  $c(i, T) = +\infty$ . After running max-flow,

$$u'_1 = f(S, 1) \quad (10)$$

This procedure also applies to a set of shadows.

*Counting:* In this case, the total number of targets,  $n$ , is unknown. For determining  $n$ , the lower and upper bounds on each shadow at  $t = t_0$  are set as  $l_i = 0, u_i = +\infty$ . As new component or FOV events are observed by the robots moving in the environment, the previous procedure is run to keep refining the initial bounds. Once we have  $l_i = u_i$  for each initial condition,  $n$  has been determined. Note that if the free space is not completely explored, then the upper bound remains at infinity. Another instance of counting is knowing  $n$ . For example, in a wild animal preserve, it may be required that the total number of a species is verified periodically. This reduces to the problem of being given  $n$  and wanting to

account for all of them. To verify the count, we can keep track of the lower bounds on the total number of targets, and if the number agrees with  $n$ , then the task has been accomplished.

*Pursuit-evasion:* Suppose there is a single evader and the task is to determine where it might be. In this case,  $l_i = 0, u_i = 1$  for each shadow at  $t = t_0$ . There are three possibilities for each shadow at  $t = t_f$ : 1)  $l_i = u_i = 0$  (the evader is not in  $s_i$ ), 2)  $l_i = u_i = 1$  (the evader is definitely in  $s_i$ ), and 3)  $l_i = 0, u_i = 1$  (the evader may or may not be in  $s_i$ ). Note that this is a passive version of the pursuit-evasion problem. We do not determine a trajectory that is guaranteed to detect the evader. In general, this problem is NP-hard [14]. Nevertheless, the calculation method proposed in this paper can be used with heuristic search techniques (or even human operators) to correctly maintain the status of the pursuit.

### F. Incorporating distinguishability

So far we only considered the case of a single attribute, which is the fully indistinguishable case. What about multiple attributes? We consider two important cases of distinguishability based on whether attributes get mixed up or not. If attributes are not intertwined, i.e., each  $a_i$  in (1) is a single attribute, it is straightforward to see that for  $m$  attributes, all we need to do is to run the algorithm for a single attribute  $m$  times, once for each attribute. Additional computation can then be performed to calculate more complicated combinations. For example, if we want the lower and upper bounds on the number of all targets for a shadow, then we can simply add up individual lower and upper bounds.

For the second case in which we may have multiple attributes for some  $a_i$ , above approach does not work. Using the example from Fig. 11, suppose that there are two teams, red and blue, and the initial conditions of shadows at  $t = t_0$  are of the form  $(red \text{ or } blue, l_i, u_i)$ . Suppose that we want to get the lower and upper bound of the number of targets in  $s_{19}$  again. For lower bounds, four computations are needed: first we set red capacities to 0 and blue capacities to  $l_i$  for all edges starting from  $S$ . The capacities for each color for edges ending in  $T$  are set as before. Running two max-flow computations, one for red and one for blue, gives us one possible lower bound  $l_{r1}, l_{b1}$ . Switching red and blue and repeat above procedure gives us another lower bound  $l_{r2}, l_{b2}$ . We should have  $l_{r1} + l_{b1} = l_{r2} + l_{b2}$ . The lower bound on  $s_{19}$  is then  $l_{r1} + l_{b1}$  red or blue targets with between  $l_{r1}$  and  $l_{r2}$  red targets. The upper bound can be obtained similarly.

## VI. IMPERFECT SENSORS, PROBABILISTIC EVENTS, AND BAYESIAN FILTERS

Now consider the case of *probabilistic* uncertainty. So far we have assumed that shadows and events are always reported without any error, which is unrealistic in practice. For detecting shadows, we already mentioned that true sensing range may be unavailable for some sensors and sometimes it is simply computationally impractical to obtain the exact visible/shadow region. However, if we settle for partial correctness, then probabilistic models can be applied. For example,

when we deal with sensor networks, conservative, probabilistic estimates of sensing range may suffice.

The same principle applies to FOV events. For each of the three FOV events, we assume that the sensors on the robots may correctly observe it or mistake it for the other two events. An enter event for a component may be reported by the sensor as an enter, exit, or null event; the same applies to exit and null events. That is, the sensor mapping is given by  $h : E_{FOV} \rightarrow Y_{FOV}$ , with  $Y_{FOV}$  being the set of *FOV observations*

$$Y_{FOV} = \{y_e, y_x, y_n\},$$

in which  $y_e, y_x$ , and  $y_n$  are enter, exit, and null observations. The map  $h$  can be deterministic, nondeterministic, or probabilistic. In this section, the case of a probabilistic FOV event-sensor mapping is investigated, together with the assumption that the dynamics of a split event is provided.

Before moving on, we introduce some notations to facilitate the discussion of the probabilistic formulation. We use  $s_i$  to denote the shadow with label  $i$ , as well as the random variable for that shadow in the joint/multivariate distribution. For shadows  $s_1, \dots, s_n$ , the joint distribution is then  $P(s_1, \dots, s_n)$ , in which a specific entry is  $P(s_1 = x_1, \dots, s_n = x_n) \in [0, 1]$ . In writing formulas and outlining algorithms, we shorten the repeated variables to “...” on both the left hand side (LHS) and the right hand side (RHS) of an expression. In such cases, the combined “...” on the LHS and RHS denote the same set of random variables. For example,  $P(s_1, s_2, s_3, s_4) = P(s_1, s_2, s_k, s_3, s_4)$  is shortened to  $P(\dots) = P(\dots, s_k, \dots)$ .

### A. Problem formulation

In the basic setup, besides the availability of a sequence of component and FOV event observations (e.g. Fig. 7), the following assumptions are made:

- 1) Component events are observed without error.
- 2) Targets are indistinguishable. The initial condition is given as a joint probability distribution  $P(s_1, \dots, s_n)$  of targets in the  $n$  shadows at  $t = t_0$ .
- 3) When a split component event happens, a probabilistic *split rule* decides how the targets should redistribute.
- 4) Observations of FOV events follows distribution given by  $P(e = e | y = y), e \in E_{FOV}, y \in Y_{FOV}$ .

After general algorithms are presented, we discuss extensions relaxing the first two assumptions. The last two assumptions can be satisfied by collecting and analyzing sensor data from the same environment; the necessity of these two assumptions will become self-evident shortly. Given these assumptions, we want to obtain the target distribution in the  $m$  shadows,  $P(s'_1, \dots, s'_m)$ , at time  $t = t_f$ .

The resulting joint probability distribution is useful in solving many decision making problems; for example, in a fire evacuation scenario, knowing the the expected number of people trapped in various parts (shadows) of a building (possibly estimated through observations from infrared beam sensors or security cameras), firefighters can better decide which region of the building should be given priority when they look around. The expected number of people in each shadow is readily available from the joint probability distribution.

## B. Processing component events

To understand how observations affect target distributions in a probabilistic setting, let us first look at the component events (we do not distinguish between events and observations for these since they are the same by assumption). Among the four types of component events, split and disappear events are more important than appear and merge events.

**1) Split.** A split event introduces more uncertainty. As a shadow splits into two disjoint shadows, the probability masses in the newly spawned shadows cannot be predicted without additional information because the sensors can not see what happens within the shadow region during a split event. The issue is resolved by the introduction of a *split rule*, obtained from supporting data or an oracle, which dictates how the originating shadow's probability mass should be redistributed. For example, statistical data may support that the number of targets in the child shadows are proportional to their respective areas.

**2) Disappear.** When a shadow disappears, the targets hiding behind it are revealed. This information can be used to update our belief about the target distribution by eliminating some improbable distributions of targets. In particular, it can reduce the uncertainty created by split events. For example, suppose that a shadow  $s_i$ , having  $d_i$  targets in it (with 100% probability), splits into shadows  $s_j$  and  $s_k$ . It is possible that  $s_j$  has 0 to  $d_i$  targets in it, as does  $s_k$ . However, if  $s_k$  later disappears to reveal  $d_k$  targets in it and no other events happen to  $s_j$  and  $s_k$ , then  $s_j$  must have exactly  $d_i - d_k$  targets in it. In general, assuming that shadow  $s_k$  disappears with a target distribution  $P(s_k)$ , the update rule is given by

$$P'(s_1 = x_1, \dots, s_n = x_n) \propto \sum P(s_1 = x_1, \dots, s_k = x_k, \dots, s_n = x_n) P(s_k = x_k),$$

in which the summation is over all joint probability entries of  $P(s_1, \dots, s_n)$  such that  $s_k = x_k$ . Normalization is required.

**3) Appear.** An appearing shadow  $s_k$ , with distribution  $P(s_k)$ , can be joined with the rest via combining the independent distributions  $P(s_k)$  with  $P(s_1, \dots, s_n)$ :

$$\begin{aligned} P'(s_1 = x_1, \dots, s_n = x_n, s_k = x_k) \\ = P(s_1 = x_1, \dots, s_n = x_n) P(s_k = x_k). \end{aligned}$$

**4) Merge.** In this case, two probability masses are collapsed. We simply collect the joint distribution to form a single one,

$$\begin{aligned} P'(\dots, s_k = x_k) \\ = \sum_{x_i + x_j = x_k} P(\dots, s_i = x_i, \dots, s_j = x_j, \dots), \end{aligned}$$

in which  $s_k$  is the merged shadow of shadows  $s_i$  and  $s_j$ . A detailed example is given in Table I in which the original shadows are  $s_1, s_2, s_3$  and  $s_2, s_3$  merge to form shadow  $s_4$ .

## C. Processing FOV events and observations

Shifting to FOV events, we observe that an enter event only affects the shadow being entered by increasing the expected number of targets in the shadow. If there is a single shadow  $s$  and an enter event happens, we merely update  $P(s = d_i) = p_i$  to  $P(s = d_i + 1) = p_i$ . On the other hand, an exit event

TABLE I

before merge	$P(s_1 = 1, s_2 = 1, s_3 = 4) = 0.2$
	$P(s_1 = 1, s_2 = 2, s_3 = 3) = 0.2$
	$P(s_1 = 1, s_2 = 3, s_3 = 2) = 0.2$
	$P(s_1 = 2, s_2 = 1, s_3 = 3) = 0.2$
	$P(s_1 = 2, s_2 = 2, s_3 = 2) = 0.2$
after merge	$P(s_1 = 1, s_4 = 5) = 0.2 + 0.2 + 0.2 = 0.6$
	$P(s_1 = 2, s_4 = 4) = 0.2 + 0.2 = 0.4$

TABLE II

observation data structure used in Algorithm 1

<i>event</i>	event type, can be one of <i>appear, disappear, split, merge</i> component events and <i>enter, exit, null</i> FOV events
$s_s$	the originating shadow in a split event
$s_{s1}$	the first new shadow after a split event
$s_{s2}$	the second new shadow in a split event
$s_{m1}$	the first shadow in a merge event
$s_{m2}$	the second shadow in a merge event
$s_m$	the newly merged shadow
$s_e$	the newly appeared shadow from an appear event
$P(s_e = n_e)$	probability that $s_e$ contains $n_e$ targets
$s_v$	the disappearing shadow in a disappear event
$P(s_v = n_v)$	probability that $s_v$ contains $n_v$ targets

does the opposite and we change  $P(s = d_i) = p_i$  to  $P(s = d_i - 1) = p_i$ . A complication arises here: If shadow  $s_i$  splits into shadows  $s_j, s_k$  and an  $e_x$  event happens to shadow  $s_j$ , it suggests that it is impossible for  $s_j$  to have 0 target before the  $e_x$  event. The affected probability mass needs to be removed and the remaining values renormalized. The null event does not change the target distribution.

Now, to propagate a probability mass through a FOV observation,  $y$ , we essentially break the entry into three pieces according to above rules, multiplying each resulting entries with the probability  $P(e = e_e | y = y)$ ,  $P(e = e_x | y = y)$ , and  $P(e = e_n | y = y)$ , respectively. If an enter event is not possible for the observation, the two remaining entries are renormalized.

## D. Accurately propagating probability masses

The first algorithm we introduce in this section is one that solves the probabilistic formulation from subsection VI-A exactly. As events happen, the probability mass,  $P(s_1, \dots, s_n)$ , is updated according to Algorithms 1 and 2 based on earlier analysis, in which the *observation* data structure is defined in Table II.

As a demonstration, we work through the observation sequence given by Fig. 12, with the following assumptions: 1) Initially there are 2 targets each in shadow  $s_1, s_2$ , 2) the split rule is that each target has 0.5 probability of going into each of the two split shadows, 3) there is no null event or observation, with the true positive rate for any observation being  $p = 0.9$ , and 4)  $a_5 = 1$  with probability 0.5 and  $a_5 = 2$  with probability 0.5. The extra assumptions are made so that the calculation of the probability mass entries is limited and the entries can be listed in a table. The iterative processing of observations is shown in Table III. The distribution is represented using a table of joint probabilities, which is always practical when there are not too many targets and events. Renormalization is

---

**Algorithm 1** PROCESSPROBABILITYMASS
 

---

**Input:**  $P(s_1, \dots, s_n)$ , the initial target distribution  
 $Q$ , the queue of observation sequences  
 a **split rule**  
 $P(e | \mathbf{y})$ , the sensor statistics  
**Output:** the target distribution after all observations

**foreach** event *observation*  $o$  in  $Q$   
**switch**( $o.event$ )  
**case** *appear*:  
   update all  $P(s_1 = x_1, \dots, s_n = x_n) = p_j$  entries to  
    $P(s_1 = x_1, \dots, s_n = x_n, o.s_e = n_e) = p_j * P(o.s_e = n_e)$   
**case** *disappear*:  
   set  $P(s_1 = x_1, \dots, s_n = x_n)$  to  
    $\sum P(s_1 = x_1, \dots, o.s_v = n_v, \dots, s_n = x_n) * P(o.s_v = n_v)$   
   remove stale entries and renormalize the probability masses  
**case** *split*:  
   add two new shadows  $o.s_{s1}, o.s_{s2}$   
   split prob. mass in  $o.s_s$  into  $o.s_{s1}, o.s_{s2}$  by **split rule**  
**case** *merge*:  
   add a new shadow  $o.s_m$  and set  $P(\dots, o.s_m = n)$  to  
    $\sum_{n_1+n_2=n} P(\dots, o.s_{m1} = n_1, \dots, o.s_{m2} = n_2, \dots)$   
**case** *enter, exit, null*:  
   **call** PROCESSFOVEVENT  
**return** the updated target distribution

---



---

**Algorithm 2** PROCESSFOVEVENT
 

---

**Input:**  $P(s_1, \dots, s_n)$ , the target distribution  
 $P(e | \mathbf{y})$ , the sensor statistics  
 $y \in \{y_e, y_x, y_n\}$ , the FOV observation  
 $s_i$ , the affected shadow  
**Output:** the target distribution after the observation

**foreach**  $P(\dots, s_i = x_i, \dots) = p_j$  entry in the distribution  
**let**  $P'(\dots, s_i = x_i + 1, \dots) = p_j * P(e = e_e | \mathbf{y} = y)$   
**let**  $P''(\dots, s_i = x_i - 1, \dots) = p_j * P(e = e_n | \mathbf{y} = y)$   
**if**  $x_i > 0$   
   **let**  $P'''(\dots, s_i = x_i - 1, \dots) = p_j * P(e = e_e | \mathbf{y} = y)$   
**else**  
   normalize  $P', P''$  such that  $P' + P'' = p_j$   
   remove  $P(\dots, s_i = j, \dots) = p_j$  entry  
   store entries  $P', P''$  and  $P'''$  if applicable  
**return** the updated target distribution

---

performed in the third step for the first and sixth entries, as well as in the last step. In the merge step, the third and seventh entries from previous step are combined, as are the fifth and ninth entries. A graphical illustration of the probability masses during each step of the run is given in Fig. 13. Note that the dimensions change as component events happen.

To verify the correctness of the outcome, Monte Carlo trials are also run, in which individual targets are propagated through the observation one by one. Since it is not an exact method, we leave the details of it to the next subsection. After 1000 successful random trials (this is the number of trials used for all Monte Carlo simulations in this paper), we obtained  $P(s_4 = 0) = 0.079, P(s_4 = 1) = 0.154, P(s_4 = 2) = 0.767$ , which matches closely the results of the exact algorithm.

### E. Efficiently propagating probability masses

Although the algorithm PROCESSPROBABILITYMASS is exact, its performance directly depends on the number of probability mass entries of a particular problem. When there are few targets and events, this is not a problem; but what if this is not the case? For a slightly more complicated

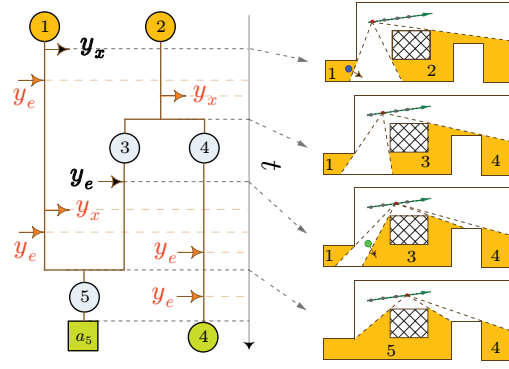


Fig. 12. A simple event observation sequence is generated (on the left, with only two FOV observations marked with bold faced font) when a robot carrying omni-directional, infinite range sensor follows the dotted path in a polygonal environment with a hole (the four figures on the right). The last event, disappearing of shadow  $s_5$ , is not shown on the right; we note that additional resource is needed to make  $s_5$  disappear (say, a sub search team). A slightly more complicated sequence is also possible with six additional FOV observations (on the left, marked with lightened font).

TABLE III

observation	probability masses
<i>initial</i>	$P(s_1 = 2, s_2 = 2) = 1$
$y_x, s_1$	$P(s_1 = 1, s_2 = 2) = 0.9$ $P(s_1 = 3, s_2 = 2) = 0.1$
<i>split,</i> $s_2 \rightarrow$ $s_3, s_4$	$P(s_1 = 1, s_3 = 0, s_4 = 2) = 0.9 * 0.25 = 0.225$ $P(s_1 = 1, s_3 = 1, s_4 = 1) = 0.9 * 0.5 = 0.45$ $P(s_1 = 1, s_3 = 2, s_4 = 0) = 0.9 * 0.25 = 0.225$ $P(s_1 = 3, s_3 = 0, s_4 = 2) = 0.1 * 0.25 = 0.025$ $P(s_1 = 3, s_3 = 1, s_4 = 1) = 0.1 * 0.5 = 0.05$ $P(s_1 = 3, s_3 = 2, s_4 = 0) = 0.1 * 0.25 = 0.025$
$y_e, s_3$	$P(s_1 = 1, s_3 = 1, s_4 = 2) = 0.225$ $P(s_1 = 1, s_3 = 0, s_4 = 1) = 0.45 * 0.1 = 0.045$ $P(s_1 = 1, s_3 = 2, s_4 = 1) = 0.45 * 0.9 = 0.405$ $P(s_1 = 1, s_3 = 1, s_4 = 0) = 0.225 * 0.1 = 0.0225$ $P(s_1 = 1, s_3 = 3, s_4 = 0) = 0.225 * 0.9 = 0.2025$ $P(s_1 = 3, s_3 = 1, s_4 = 2) = 0.025$ $P(s_1 = 3, s_3 = 0, s_4 = 1) = 0.05 * 0.1 = 0.005$ $P(s_1 = 3, s_3 = 2, s_4 = 1) = 0.05 * 0.9 = 0.045$ $P(s_1 = 3, s_3 = 1, s_4 = 0) = 0.025 * 0.1 = 0.0025$ $P(s_1 = 3, s_3 = 3, s_4 = 0) = 0.025 * 0.9 = 0.0225$
<i>merge,</i> $s_1, s_3$ $\rightarrow s_5$	$P(s_4 = 2, s_5 = 2) = 0.225$ $P(s_4 = 1, s_5 = 1) = 0.045$ $P(s_4 = 1, s_5 = 3) = 0.405 + 0.005 = 0.41$ $P(s_4 = 0, s_5 = 2) = 0.0225$ $P(s_4 = 0, s_5 = 4) = 0.2025 + 0.0025 = 0.205$ $P(s_4 = 2, s_5 = 4) = 0.025$ $P(s_4 = 1, s_5 = 5) = 0.045$ $P(s_4 = 0, s_5 = 6) = 0.0225$
<i>disappear,</i> $s_5$	$P(s_4 = 0) = 0.0769$ $(= 0.0225 * 0.5 / ((0.0225 + 0.045 + 0.225) * 0.5))$ $P(s_4 = 1) = 0.1538$ $P(s_4 = 2) = 0.7692$

event observation sequence (Fig. 12), with 5 targets each in shadow  $s_1$  and  $s_2$  to start, 135 joint probability table entries are obtained before the merge step, as shown in Fig. 14. The probability mass entries increase rapidly because of the split events and the FOV events. For a split event, if the originating shadow contains up to  $n$  targets, then the number of probability mass entries can multiply by up to a factor of  $n + 1$ . For FOV observations, each has certain probability to be enter, exit, and null events, which may cause the number of probability mass entries to triple in the



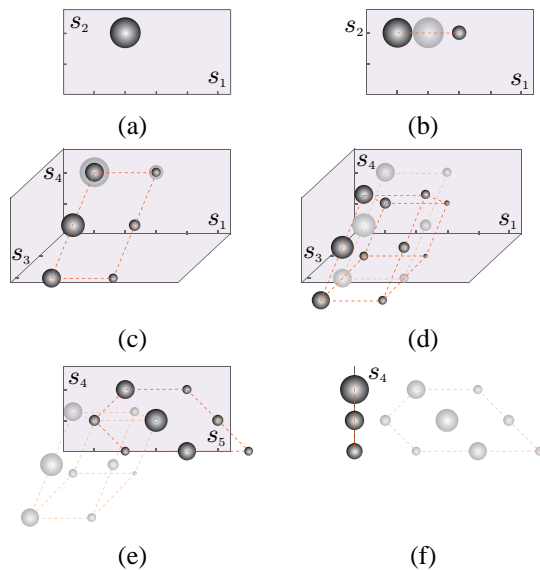


Fig. 13. Graphical display of the target probability mass as the algorithm PROCESSPROBABILITYMASS is run over the simple event observation sequence in Fig. 12. Each figure corresponds to one step in Table III. Lighter (if any) and darker balls represent probability masses before and after an event, respectively. The volumes are proportional to the magnitude of the probability mass entries.

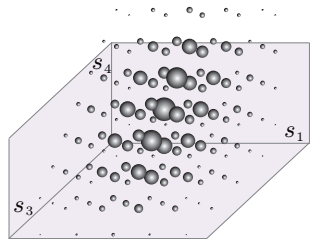


Fig. 14. Probability masses for the slightly more complicated observation sequence in Fig. 12 before shadows  $s_1, s_3$  merge to form  $s_5$ . The axes  $s_1, s_3$ , and  $s_4$ , as shown in the figure, have ranges  $[1, 9]$ ,  $[0, 7]$ , and  $[0, 5]$ , respectively, starting from the origin.

worst case. Therefore, as the number of targets and events increase, the space required to store the probability masses may grow exponentially. Since processing each observation requires going through all the entries, computation time will also explode, which means that the exact algorithm will not work efficiently. On the bright side, when a large number of probability mass entries are present, some of these entries must have very low weights; making approximations by trimming away these low probability entries is unlikely to greatly affect the final target distribution.

*Monte Carlo trials:* Since our task is to probabilistically track targets, sequential Monte Carlo methods are a natural choice. As a first heuristic, we perform simple trials such that each trial starts with the initial distribution of targets. These targets are propagated through the event observations by querying a Monte Carlo simulator. During each trial, the outcome of simulation may contradict an observation, in which case the trial is simply discarded. After a certain number of successful trials are completed, the final target distribution is obtained. For example, the mean of the number of targets in

a shadow at  $t = t_f$  is simply the average of the number of targets in that shadow over all successful runs. For simulations in this paper, we require 1000 successful trials. Note that since the particular Monte Carlo simulation we perform in this paper does not depend on data, its result is probabilistically correct and therefore can serve as baselines for verifying results from other algorithms.

*Improving the PROCESSPROBABILITYMASS algorithm:* Observing that the computation is burdened by storing the sheer amount of probability mass entries when there are many targets and observations, an obvious simplification is to *resample* the entries and keep the important ones. For example, we may choose to retain the first 1000 probability mass entries of largest value. With each step of processing looking at each entry once, the processing time per step becomes a constant, albeit a large one. With this approximation, the earlier algorithm then runs in time *linear* in the number of observations. We call this heuristic *basic truncation*.

The problem with basic truncation, however, is that the trimmed away entries may turn out to be important. Take the processing in Table III for example, if the fourth entry after the merge step,  $P(s_4 = 0; s_5 = 2) = 0.0225$ , is truncated, then the second entry in the end,  $P(s_4 = 0) = 0.0769$ , will be lost, which is significant. The issue becomes problematic very quickly as the number of coexisting shadows increases, since each shadow creates one dimension in the joint distribution and sampling a high dimensional space is inherently inefficient. To alleviate this problem, in addition to the basic truncation approach of keeping fixed amount of entries with highest probability after each update, we also employ the following:

- 1) Randomly allow probability mass entries with low value to survive truncation. In doing this, we hope to allow enough low probability yet important entries to survive truncation. We denote this heuristic as *random truncation*.
- 2) Retain more entries during update steps right before merge and disappear events. Since disappear events usually cause the the number of probability mass entries to decrease dramatically (concentrating the distribution, or reducing the uncertainty), we can afford to keep more entries right before these events, without incurring much extra computational cost. Merge events also cause the number to decrease as some entries can be combined after merging. We combine this with random truncation and denote the resulting heuristic *random truncation with event lookahead*.

By construction, it is straightforward to see that the additional heuristics do not need asymptotically more time. There is a clear similarity between these heuristics and particle filtering: They all begin with a discrete set of probability masses, push the set through an update rule, and resample when necessary. They also share the same weakness: If the key sample points with low probabilities are truncated, the end result may be severely skewed. Unlike in typical particle filter problems, the number of random variables in our problem keeps changing with split and merge events.

## F. Extensions

In subsection VI-A, assumptions 1) and 2) are made to simplify the presentation of the probabilistic algorithms. The first assumption is that component events are observed without error. Although component events can be observed with high accuracy in many environments, it is not always the case. Sensor network is such an example: sensing range can be hard to know precisely. The extension to handle such uncertainties is relatively straightforward, at least in theory: all we need to do is to maintain a probability distribution over all possible sequence of shadows consistent with the robot's observations. Obtaining expectations of the number of targets in any shadow can then be done by also calculating the expectation over all possible sequences of shadows that contains the target shadow. The computation effort will certainly increase; resampling can alleviate the burden somewhat.

Various distinguishability assumptions can also be handled. Recall that in the nondeterministic formulation, two distinguishability cases are investigated. When there are only teams with single attributes, the approach from the nondeterministic setting applies by simply carrying out one computation per team. If the teams have multiple attributes (for example, the initial condition may be given as a joint distribution of red and blue teams), a direct extension is performing one computation for each joint probability entries in the initial condition. This is clearly more work and resampling may be necessary depending on the granularity of the initial target distribution. On the plus side, although we lose some accuracy with resampling (to save computation time), a richer class of problems can now be handled because any initial condition can be described as a joint probability distribution.

## VII. SIMULATION RESULTS AND COMPLEXITY ANALYSIS

The simulation programs were developed adhering to the Java 1.6 language standard under the Eclipse environment. The computations were performed on a workstation with an Intel Core 2 Quad processor running at 3.0 GHz. The JavaVM has a maximum memory of 1.5GB. Although many parts of our algorithm can be easily parallelized, no multi-threading was used in this implementation.

### A. Nondeterministically moving targets

For the nondeterministic case, we implemented and tested the algorithms for a single robot that moves in a simply connected polygonal region in  $\mathbb{R}^2$  using an omnidirectional visibility sensor. For such environment, the shadows are completely characterized by bitangents and inflections. When the environment is known and the robot can localize itself, efficient 2D cell decomposition algorithms can be readily applied to obtain the sequence of shadows for each location of the robot, allowing the shadows to be continuously tracked. This setup also enables us to construct an oracle (not available to the algorithm) for distributing targets inside the free space to simulate their nondeterministic behavior. For max-flow, we implemented the  $O(VE^2)$ -time Edmonds-Karp max-flow algorithm [10], in which  $V$  and  $E$  are the numbers of vertices and edges in the flow graph, respectively.

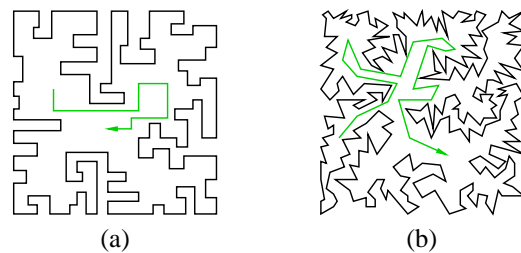


Fig. 15. Complicated examples that were used to test our approach. The given robot trajectories are shown as green lines in the direction of the arrow.

For the environment in Figure 15(a), the trajectory generates 85 component events. Our oracle randomly distributed 100 targets in the free space as the component events occur. This setting yields a bipartite graph that has 41 vertices and 60 edges. Calculating the lower and upper bounds for the 18 final shadows for a single team took 0.1 seconds. The second free space, shown in Figure 15(b), has 385 component events, 491 total shadows, 124 vertices in the bipartite graph with 339 edges. The example involves a million targets with 5 teams that intersperse. The bounds on the 12 final shadow components for all 5 teams were computed under one second.

The inputs to the base algorithm (single attribute, no FOV events) are: 1) A sequence of  $n$  shadows, and 2) The initial condition which takes the form of a pair of lower and upper bounds for each shadow at  $t = t_0$ . In the worst case, there are  $O(n)$  vertices and  $O(n^2)$  edges in the bipartite graph. Edmonds-Karp max-flow then gives us  $O(n^5)$  running time in the worst case. Applying a push-relabel algorithm with FIFO vertex selection rule will cut the running time to  $O(n^3)$  [13]. Adding FOV events does not increase time complexity asymptotically, as discussed in subsection V-D. Adding partial distinguishability, on the other hand, will introduce another input parameter  $m$ , the number of teams, that contributes linearly to time complexity. The typical worst case running time for the nondeterministic case is then  $O(n^3m)$ . The number of targets in the system does not directly affect the performance.

### B. Probabilistic setup

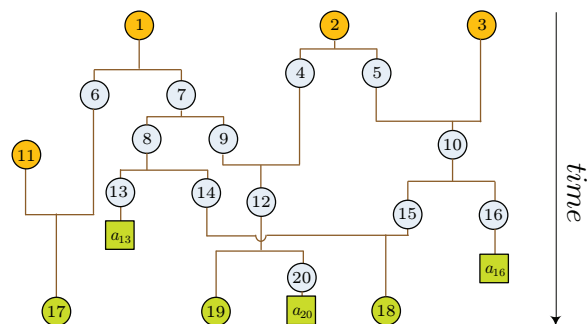


Fig. 16. An event observation sequence with a total 20 shadows in its life cycle. The FOV observations are not marked.

For the probabilistic case, we ran a simulation with the observation sequence in Fig. 16. The sequence contains 14

component event observations. We also included 32 FOV events scattered along the sequence, which are not marked in the figure. Shadows 1, 2, 3, 11, 13, 16, 20 are associated with 10, 7, 8, 9, 6, 9, 4 targets (with probability 1), respectively. For performance measures, we look at the time for one run of the algorithm to complete, as well as the expectation (mean and standard deviation for randomized methods) of targets in individual shadows at the end ( $s_{17}$ ,  $s_{18}$ , and  $s_{19}$ ). When we randomly pick entries to keep, the time result is averaged over 10 runs and the accuracy is given in the form of mean and standard deviation. In our implementation, we also make the following choices: 1) for random truncation, the entries are kept based on its probability multiplied with a random number in  $(0, 1)$ , 2) for event lookahead, we will not truncate the entries if there is a disappear event within the next four events or a merge event within the next two events. The outcome is summarized in Table IV. The heuristics basic truncation, random truncation, and random truncation with event lookahead are shortened as TR, RT, and RT-LA, respectively. The number following the method is the number of entries kept. By *frequent failure*, we mean that more than 1/3 of the times the heuristic fails to give a valid result. These are indicative of minimum number of entries needed for the method to work.

TABLE IV

heuristic	$s_{17}$	$s_{18}$	$s_{19}$	t(s)
none, precise	11.12	5.84	5.60	329.5
TR-10000	failure			
TR-20000	10.67	4.85	5.33	6.1
TR-50000	11.00	5.38	5.52	15.5
TR-100000	10.96	5.59	5.53	29.4
TR-200000	11.06	5.73	5.56	61.4
RT-10000	frequent failure			
RT-20000	11.38(0.20)	5.31(0.23)	5.67(0.20)	6.1
RT-50000	11.16(0.02)	5.36(0.03)	5.64(0.02)	14.6
RT-100000	11.03(0.01)	5.62(0.01)	5.56(0.01)	28.3
RT-LA-2000	frequent failure			
RT-LA-5000	11.32(1.30)	6.54(1.46)	5.28(1.28)	2.0
RT-LA-10000	11.17(0.56)	5.87(0.91)	5.02(0.48)	4.2
RT-LA-20000	11.62(0.26)	5.30(0.18)	5.57(0.16)	8.3
RT-LA-50000	11.32(0.01)	5.51(0.01)	5.60(0.01)	17.7
Monte Carlo	11.16	5.58	5.57	42.1

The result shows that when no heuristic is used, the algorithm takes much more time to finish. This is not surprising since the time complexity is induced by the space requirement for storing the probability mass entries. On the other hand, all of the truncation heuristics work reasonably well, with the randomized truncation plus event lookahead greatly reduces the number of entries to retain. The RT-LA-50000 run compares well with the TR-100000 run on accuracy, but uses one third less time. We expect the advantage to become more obvious as more targets are present in the system.

For a second test, we change the number of targets in shadows 1, 2, 3, 13, 16, 20 to 25, 22, 23, 8, 15, 9, while leaving other observations unchanged. With the increased number of targets, the basic algorithm runs out of memory after 10 minutes, before the third split is completed. At the peak of its memory usage during the failed run, there are more than  $2 \times 10^7$  probability mass entries. On the other hand,

the randomized methods do not have this problem: Both RT-100000 RT-LA-50000 yield good results compared to Monte Carlo trials, with similar running time. The result is summarized in Table V. Comprehensive performance analysis of

TABLE V

Heuristic	$s_{17}$	$s_{18}$	$s_{19}$	t(s)
none, exact	out of memory after 10 mins			
Monte Carlo	18.26	22.25	11.85	43.2
RT-100000	17.78(0.03)	21.83(0.03)	12.34(0.02)	66.3
RT-LA-50000	18.24(0.08)	21.99(0.07)	12.57(0.07)	40.6

probabilistic algorithm is hard since the performance depends on external factors such as the implementation of the specific split rule, random number generator, and so on. Nevertheless, for completeness, we discuss the performance at a higher level. To avoid the issue of external factors, we assume that at each step, each probability mass takes constant time to process. Unlike the nondeterministic case, running time of the PROCESSPROBABILITYMASS algorithm may depend heavily on the number of targets in the system via the split rule. If there are  $n$  split events with an average number of targets in the originating shadow being  $p$  and also  $n_f$  FOV events, with the reasonable additional assumption that merge, appear, and disappear events are on the same order as split events, the PROCESSPROBABILITYMASS algorithm can take time  $O(p^n 3^{n_f})$ . The running time of the resampling based algorithms has a big constant depending on the number of entries to keep, but otherwise depends only linearly on the number of critical events.

## VIII. CONCLUSION

In conclusion, we have formulated and solved, at a very general level, the problem of tracking targets moving in and out of the FOV of moving sensors. The resulting filters may be applied in numerous settings, such as pursuit-evasion, target enumeration, and situational awareness. When targets move nondeterministically, a combinatorial filter is proposed for the tracking task: We show that the naturally emerging integer linear programming problem is in fact solvable in polynomial time and provide an efficient max-flow based solution for it. For the probabilistic filtering problem in which targets move probabilistically and sensors are not reliable, we give both exact and efficient algorithms that handle the several possible scenarios depending on the number of targets and observations in a system. In solving the more general, probabilistic version of the tracking problem, a clear link is also established between combinatorial filtering and Bayesian filtering methods: The final target distribution is in essence associating the combinatorial solution, a polytope structure, with appropriate probabilities. Viewing it from another angle, the probabilistic shadow I-space extends naturally from its nondeterministic counterpart, by merely adding dimensions to record probabilities.

*Acknowledgments:* This work was supported in part by NSF grants 0904501 (IIS Robotics) and 1035345 (Cyberphysical Systems), DARPA STOMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

## REFERENCES

- [1] S. Alexander, R. Bishop, and R. Ghrist. Pursuit and evasion in non-convex domains of arbitrary dimension. In G. S. Sukhatme, S. Schaal, W. Burgard, and D. Fox, editors, *Robotics: Science and Systems II*. MIT Press, Cambridge, MA, 2007.
- [2] L. Alonso, A. S. Goldstein, and E. M. Reingold. Lion and man: Upper and lower bounds. *ORSA Journal of Computing*, 4(4):447–452, 1992.
- [3] Y. Baryshnikov and R. Ghrist. Target enumeration via euler characteristic integrals. *SIAM Journal of Applied Mathematics*, 70(4):825, August 2009.
- [4] P. E. Berry, C. Pontecorvo, and D. A. B. Fogg. Optimal search, location and tracking of surface maritime targets by a constellation of surveillance satellites. Technical report, DSTO Information Sciences Laboratory, Edingburgh, SA, 2003.
- [5] S. D. Bopardikar, F. Bullo, and J. P. Hespanha. On discrete-time pursuit-evasion games with sensing limitations. *IEEE Transactions on Robotics*, 24(6):1429–1439, 2008.
- [6] P. Bose, A. Lubiv, and J. I. Munro. Efficient visibility queries in simple polygons. In *Proceedings Canadian Conference on Computational Geometry*, pages 23–28, 1992.
- [7] F. Bourgault, T. Furukawa, and H.F. Durrant-Whyte. Coordinated decentralized search for a lost target in a bayesian world. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 48 – 53 vol.1, 2003.
- [8] W.-P. Chin and S. Ntafos. Optimum watchman routes. *Information Processing Letters*, 28:39–44, 1988.
- [9] W.-P. Chin and S. Ntafos. Optimum watchman routes in simple polygons. *Discrete and Computational Geometry*, 6:9–31, 1991.
- [10] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.
- [11] B. P. Gerkey, S. Thrun, and G. Gordon. Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research*, 25(4):299–322, 2006.
- [12] B. Gfeller, M. Mihalak, S. Suri, E. Vicari, and P. Widmayer. Counting targets with mobile sensors in an unknown environment. In *ALGOSEN-SORS*, July 2007.
- [13] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 136–146, New York, NY, USA, 1986. ACM.
- [14] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. *International Journal of Computational Geometry and Applications*, 9(5):471–494, 1999.
- [15] O. Hájek. *Pursuit Games*. Academic, New York, 1975.
- [16] Y. Ho, A. E. Bryson, and S. Baron. Differential games and optimal pursuit-evasion strategies. *IEEE Transactions on Automatic Control*, 10(4):385–389, 1965.
- [17] R. Isaacs. *Differential Games*. Wiley, New York, 1965.
- [18] V. Isler, S. Kannan, , and S. Khanna. Randomized pursuit-evasion in a polygonal environment. *IEEE Transactions on Robotics*, 5(21):864–875, 2005.
- [19] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME, Journal of Basic Engineering*, 82:35–45, 1960.
- [20] T. Kameda, M. Yamashita, and I. Suzuki. On-line polygon search by a seven-state boundary 1-searcher. *IEEE Transactions on Robotics*, 22:446–460, Jun 2006.
- [21] S. Kopparty and C. V. Ravishankar. A framework for pursuit evasion games in  $R^n$ . *Information Processing Letters*, 96(3):114–122, 2005.
- [22] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at <http://planning.cs.uiuc.edu/>.
- [23] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. *IEEE Transactions on Robotics and Automation*, 17(2):196–201, April 2001.
- [24] S. M. LaValle, D. Lin, L. J. Guibas, J.-C. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 737–742, 1997.
- [25] S.-H. Lim, T. Furukawa, G. Dissanayake, and H. F. Durrant-Whyte. A time-optimal control strategy for pursuit-evasion games problems. In *Proceedings IEEE International Conference on Robotics & Automation*, 2004.
- [26] R. Mahler. Objective functions for bayesian control-theoretic sensor management, ii: Mhc-like approximation. In S. Butenko, R. Murphey, and P. Paralos, editors, *New Developments in Cooperative Control and Optimization*, pages 273–316, Norwell, MA, 2003. Kluwer Academic Publishers.
- [27] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, 1987.
- [28] J. O'Rourke. Visibility. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 643–663. Chapman and Hall/CRC Press, New York, 2004.
- [29] M. Pachter. Simple motion pursuit-evasion differential games. In *Mediterranean Conference on Control and Automation*, Lisbon, Portugal, July 2002.
- [30] C. H. Papadimitriou and K. J. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.
- [31] S.-M. Park, J.-H. Lee, and K.-Y. Chwa. Visibility-based pursuit-evasion in a polygonal region by a searcher. Technical Report CS/TR-2001-161, Dept. of Computer Science, KAIST, Seoul, South Korea, January 2001.
- [32] T. D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Application of Graphs*, pages 426–441. Springer-Verlag, Berlin, 1976.
- [33] S. Petitjean, D. Kriegman, and J. Ponce. Computing exact aspect graphs of curved objects: algebraic surfaces. *International Journal of Computer Vision*, 9:231–255, Dec 1992.
- [34] M. Pocchiola and G. Vegter. The visibility complex. *International Journal Computational Geometry & Applications*, 6(3):279–308, 1996.
- [35] S. Sachs, S. Rajko, and S. M. LaValle. Visibility-based pursuit-evasion in an unknown planar environment. *International Journal of Robotics Research*, 23(1):3–26, January 2004.
- [36] A. Schrijver. *Combinatorial Optimization*. Springer-Verlag, 2003.
- [37] J. Sgall. A solution of david gales lion and man problem. *Theoretical Computational Science*, 259(1-2):663–670, 2001.
- [38] T. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 80(9):1384–1399, September 1992.
- [39] B. Simov, G. Slutzki, and S. M. LaValle. Pursuit-evasion using beam detection. In *Proceedings IEEE International Conference on Robotics and Automation*, 2000.
- [40] J. Singh, R. Kumar, U. Madhow, S. Suri, and R. Cagley. Tracking multiple targets using binary proximity sensors. In *Proc. Information Processing in Sensor Networks*, 2007.
- [41] L. D. Stone. *Theory of Optimal Search*. Academic Press, New York, NY, 1975.
- [42] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, October 1992.
- [43] B. Tovar, F. Cohen, and S. M. LaValle. Sensor beams, obstacles, and possible paths. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2008.
- [44] B. Tovar and S. M. LaValle. Visibility-based pursuit-evasion with bounded speed. *International Journal of Robotics Research*, 2007. Under review (invited submission from WAFR 2006).
- [45] B. Tovar, R. Murrieta, and S. M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23(3):506–518, June 2007.
- [46] V. Turetsky. Upper bounds of the pursuer control based on a linear-quadratic differential game. *Journal of Optimization Theory and Applications*, 121(1):163–191, April 2004.
- [47] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry. Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18(5):662669, 2002.
- [48] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte. Simultaneous localization, mapping and moving object tracking. *International Journal of Robotics Research*, 26(9):889–916, 2007.
- [49] I. Yan and G. L. Blankenship. Numerical methods in search path planning. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 1563 –1569 vol.2, 1988.
- [50] D. B. Yang, H. H. Gonzalez-Banos, and L. J. Guibas. Hitting people in crowds with a real-time network of simple image sensors. In *Proc. IEEE International Conference on Computer Vision*, volume 1, pages 122– 129, 2003.
- [51] J. Yu and S. M. LaValle. Tracking hidden agents through shadow information spaces. In *Proceedings IEEE International Conference on Robotics and Automation*, 2008.
- [52] J. Yu and S. M. LaValle. Probabilistic shadow information spaces. In *Proceedings IEEE International Conference on Robotics and Automation*, 2010.