Park Rangers' Problem: Motion Planning for Sequential Visibility Requirements

John Phillips, Sihui Li[1 [0000−0003−1766−4316]], and Neil T. Dantam[1 [0000−0002−0907−2241]]

Department of Computer Science, Colorado School of Mines, USA
{johphill,li,ndantam}@mines.edu

Abstract. Many robot tasks may involve achieving visibility (such as to observe areas of interest) or maintaining occlusion (such as to avoid disturbing other agents). We generally formulate such sequential visibility tasks for 3D worlds, termed the Park Rangers' Problem, and we develop an approach to solve such tasks offering completeness under certain requirements. Our approach constructs an abstraction based on an exact test for visibility between areas, and multiple tests and relaxations for the nonconvex problem of determining occlusions between areas. We apply a constraint-based planning approach and iteratively refine the abstraction. Finally, we evaluate the approach on simulated visibility scenarios.

1 Introduction

Many robot applications involve tasks based on visibility requirements. For example, common information acquisition scenarios such as exploration, inspection, and monitoring, require one or more robots to achieve visibility of certain areas of interest, and safety or communication requirements may require members of a robot team to maintain visibility of each other. Further, some tasks may require maintaining occlusion, e.g., a household robot avoiding disrupting the home's inhabitants during its operation or an outdoor robot remaining hidden from animals to avoid causing them stress. Such applications with dynamic visibility and occlusion requirements generalize established robotics scenarios [9,1,23] as well as classic problems from computational geometry [2,21].

In this paper, we characterize multi-agent scenarios with sequential visibility and occlusion requirements as the Park Rangers' Problem, and we develop solution techniques for such problems through an abstraction refinement approach incorporating computational geometry, optimization, and constraint satisfaction. We first formally define the Park Rangers' Problem in terms of multiple agents moving in a 3D workspace that must achieve a sequence of visibilities and occlusions satisfying a logical specification or formal language (see section 3). Then, we develop an approach to solve such problems (see section 4). We abstract the environment and its visibility and occlusion properties using a subset-ordered lattice (see subsection 4.1). While determining visibility between abstraction

elements is directly computable using computational geometry techniques, determining occlusion status is more challenging, notably posing nonconvexities. We develop multiple tests and relaxations to address nonconvexity of occlusion determination (see subsection 4.2) and then apply constraint-based approaches to satisfy the sequential visibility requirements (see subsection 4.3). If a current abstraction does not admit valid plans, we iteratively refine the abstraction (see subsection 4.4). We show that this overall apporach is complete under certain assumptions (see subsection 4.5). Finally, we empirically evaluate our approach on scenarios with sequential visibility requirements (see section 5).

2 Related Work

Several works touch on the subject matter of this paper. First, there is the classical art gallery problem [21], which is easily solvable in 2D but not the 3D world in which many robots operate. The art gallery problem considers how to position static "guards" in an art gallery, defined as a polyhedron, such that at least one guard has visibility to all points in the polyhedron.

The watchman route problem generalizes the art gallery problem to moving guards where we must find paths for agents to observe a desired set of areas [2].

In 3D graphics, several works consider the problem of visibility. The "visibility skeleton" [7] and "visibility complex" [8] are analytical methods for finding visibility information. These methods have very high-polynomial running times and do not lend themselves to planning problems, where *a-priori* information about relationships between arbitrary points in a world are required. Compared to these classic formulations and graphics results, we address cases of agents operating in a 3D world where requirements may involve both visibility and occlusion over time.

In robotics, several works consider planning based on visibility constraints, especially in 2D. Much work has done in the area of "Pursuit-Evasion"-type problems, where some pursuing agents must achieve visibility of a moving evader agent [10]. There are many variations on this problem, for example, Fletcher et al. consider a "visibility-based escort problem" [9], where a single VIP agent must be escorted by an escort robot through a 2D environment; the VIP agent must not be seen by known enemy positions. Additionally several works in the robotics world treating 3D visibility problems exist; these are often sampling-based approaches which consider particular problems. For example, [22] introduces the concept of a "Visibility Integrity Roadmap", a datastructure that is based on sampling the visibility at various 3D points, and solving the "group following problem" with it. This approach relies on sampling to satisfy visibility constraints and notably can encounter problems with narrow "corridor" type obstacles, and is focused on visibility (although this approach could likely be extended to occlusion-based problems). Additionally, [19] uses sampling to create a convex decomposition of the workspace to aid in locating objects. In contrast, we are interested in an approach that can allow for exact constraints on visibility over a plan. This means we cannot rely on sampling, while we also seek to allow

for a generic approach to planning that allows for a wide variety of problems to be specified.

We note the visibility graph [13] as a a motion planning concept whose name suggests it may handily address the subject of this work. However, the "visibility" in visibility graphs refers to the incidental fact that nodes in visibility graphs must have line-of-sight (LOS) to one another. Visibility graphs do not, however, address line-of-sight visibility or occlusion information about an entire robot world, so they are not ideal for visibility-constrained planning.

3 Problem Definition

We find motion plans for robot teams who must satisfy visibility requirements. For example, a team of (robot) park rangers may need to observe wildlife without being seen while moving from their initial position, and we may also want, for safety, each ranger to see at least one other ranger. We define this problem by extending the classic motion planning formulation to include visibility information and then a sequence of general visibility and occlusion constraints.

Classically, motion planning problems consist of a configuration space $\mathcal{C}$ of dimension n, with a start configuration $\mathbf{q}_{\text{start}} \in \mathcal{C}$ and goal configuration $\mathbf{q}_{\text{goal}} \in \mathcal{C}$ [13]. We divide the configuration space $\mathcal{C}$ into $\mathcal{C}_{\text{free}}$, the open set *free space*, and $\mathcal{C}_{\text{obs}}$, the closed set *obstacle space*. $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}}$ may be derived from a workspace obstacle region $\mathcal{W}_{\text{obs}}$, where $\mathcal{C}_{\text{free}}$ is the set of configurations not colliding with $\mathcal{W}_{\text{obs}}$, and $\mathcal{C}_{\text{obs}} = \mathcal{C} \setminus \mathcal{C}_{\text{free}}$. A feasible plan is a path σ, such that $\sigma[0,1] \in \mathcal{C}_{\text{free}}$, $\sigma[0] = \mathbf{q}_{\text{start}}$, and $\sigma[1] = \mathbf{q}_{\text{goal}}$. We will extend this definition to now address sequential visibility.

3.1 Visibility problems

In this paper, we extend motion planning to incorporate visibility requirements. We consider an $\mathcal{SE}(3)$ robot workspace $\mathcal{W}$ containing obstacles that both cause collision and occlude visibility. For this current work, we consider only opaque (not translucent or transparent) obstacles. Specifically, workspace $\mathcal{W}$ consists of disjoint *workspace obstacle region* $\mathcal{W}_{\text{obs}}$ and *workspace free region* $\mathcal{W}_{\text{free}}$. Configurations solely in $\mathcal{W}_{\text{free}}$ are traversable; however, visibility requirements may further restrict valid paths.

We consider visibility based on line-of-sight between workspace points. Two points $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{W}$ are mutually visible if and only if the straight line between them passes only through $\mathcal{W}_{\text{free}}$. Using $\overline{xy}$ to denote the line segment between x and y, points x and y are mutually visible when $\overline{xy} \subseteq \mathcal{W}_{\text{free}}$ and mutually occluded when $\overline{xy} \not\subseteq \mathcal{W}_{\text{free}}$ or equivalently $\overline{xy} \cap \mathcal{W}_{\text{obs}} \neq \emptyset$.

3.2 Motion Planning for Sequential Visibility

We address motion planning for sequential visibility constraints. An individual visibility constraint is based on visibility between regions or agents. For example,

consider a workspace with regions A, B and robots x, y; we then have Boolean propositions which may be visible $(x,\ A)$, visible $(x,\ B)$, etc. The proposition visible $(a,\ b)$ is true when the line segment(s) between a and b are not occluded. We extend these propositions to sequential constraints representing visibility requirements over time. For example, we may want visible $(x,\ y)$ to always hold, visible $(x,\ A) \vee$ visible $(y,\ A)$ to never hold, and visible $(x,\ B) \vee$ visible $(y,\ B)$ to hold during at least one point. Such sequences of discrete constraints form a *formal language* of the traces of proposition assignments satisfying the problem requirements. We may specify such a formal language using notations such as state machines or temporal logics [17]. Using formal languages allows us to leverage work that has been done in robot planning where valid plans are thought of as strings in a language, for example [4],[15]. Formal languages allow users to specify visibility requirements that change over the course of the plan. Thinking of problems as a formal language allows us to solve both problems using the same approach.

Definition 1. *We define the Park Rangers' Problem as* $V = (\mathcal{W}, \mathcal{C}, P, \Psi)$*, where,*

- $\mathcal{W} = \mathcal{W}_{\text{free}} \cup \mathcal{W}_{\text{obs}}$ *the 2D or 3D world consisting of disjoint free space* $\mathcal{W}_{\text{free}}$ *and obstacle space* $\mathcal{W}_{\text{obs}}$*. Free space* $\mathcal{W}_{\text{free}}$ *is traversible and permits visibility, while* $\mathcal{W}_{\text{obs}}$ *is not traversible and occludes visibility.*
- $\mathcal{C} : m \times m \cdots \times m$ *is our multi-robot configuration space, where each* m *is the position of one robot in the 2D or 3D world.*
- P *is the set of atomic visibility propositions—e.g.,* visible $(\alpha,\ \beta)$*—that could be true of a particular configuration.*
- Ψ *is a formal language over symbols* 2^P*, where each symbol denotes the true and false visibility predicates at some step and each string* $\psi \in \Psi$ *is a trace satisfying visibility constraints.*

The goal of our planning problem is to find a path σ for the robot team that is traversible, $\sigma[0, 1] \in \mathcal{C}_{\text{free}}$, and which corresponds to some string $\psi \in \Psi$, indicating that the path satisfies the sequential visibility requirements.

4 Algorithm

We develop an iterative abstraction, planning, and refinement approach for the Park Rangers' Problem (see Figure 1). First, we create a visibility abstraction $\mathcal{T}$ to represent visibility and occlusion between sets of points in a 2D or 3D world $\mathcal{W}$ (see subsection 4.1 and subsection 4.2). Then, we attempt to use the abstraction $\mathcal{T}$ to generate a sequence of region traversals ψ that satisfy visibility specification Ψ (see subsection 4.3). However, coarseness and indeterminate visibility of $\mathcal{T}$ may not admit a valid plan $\psi \in \Psi$. For such cases when no plan ψ is found, we then repeatedly refine $\mathcal{T}$ to deal with finer-grained regions (see subsection 4.4).

4.1 Visibility Abstraction

We construct a discrete abstraction to facilitate planning for multi-agent visibility requirements. The abstraction is a convex partitioning of world $\mathcal{W}$, labeled

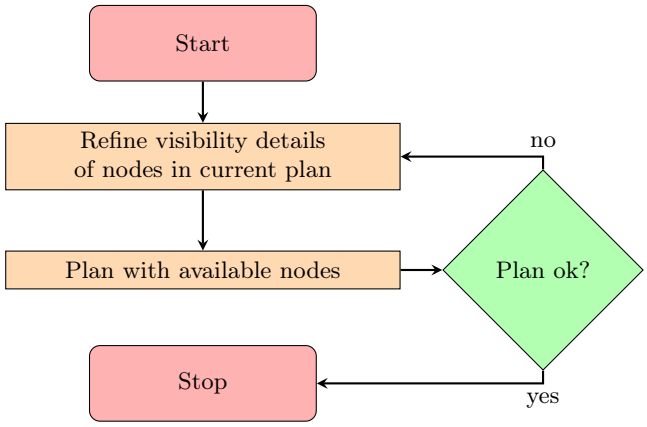

Fig. 1: The general algorithm flow.

with visibility or occlusion information between pairs of partitions. Such as partitioning enables (1) lookup of instantaneous visibility or occlusion requirements and (2) task-level planning for the sequential visibility and occlusion Ψ.

Visibility Categories The abstraction represents visibility or occlusion between convex partitions. Two *points* are either visible or occluded based on whether the line segment between them passes through only $\mathcal{W}_{\text{free}}$ or intersects $\mathcal{W}_{\text{obs}}$. Two convex *partitions* may be fully visible or fully occluded if all points between them are visible or occluded; it is also possible that two partitions contain both visible and occluded points (see Figure 2).

Definition 2 (Partition Visibility). *Two convex partitions A, B, are either fully visible, fully occluded, or partially visible/occluded.*

$$\mathcal{V}(A,\ B) = \begin{cases} \text{FULL}, & \forall\,(x,\ y) \in (A,\ B)\,,\ \overline{xy} \subset \mathcal{W}_{\text{free}} \\ \text{OCCLUDED}, & \forall\,(x,\ y) \in (A,\ B)\,,\ \overline{xy} \not\subset \mathcal{W}_{\text{free}} \\ \text{PARTIAL}, & \exists\,(x,\ y) \in (A,\ B)\,,\ \overline{xy} \subset \mathcal{W}_{\text{free}} \\ & \wedge \exists\,(x,\ y) \in (A,\ B)\,,\ \overline{xy} \not\subset \mathcal{W}_{\text{free}} \end{cases}$$

Proposition 1 (Full Visibility). *Two convex polytopes A, B are mutually fully visible if and only if their convex hull does not intersect workspace obstacle region $\mathcal{W}_{\text{obs}}$. Denoting the convex hull of A, B as* $\mathbf{conv}\,(A,\ B)$,

$$\mathbf{conv}\,(A,\ B) \subseteq \mathcal{W}_{\text{free}} \quad \Longleftrightarrow \quad \mathcal{V}(A,\ B) = \text{FULL}\,. \tag{1}$$

Proof. Consider all line segments $\overline{xy}$ between A and B. Every part of $\mathbf{conv}\,(A,\ B)$ must be covered by some $\overline{xy}$, so when every $\overline{xy}$ does not intersect any occlusion, we know $\mathbf{conv}\,(A,\ B) \subseteq \mathcal{W}_{\text{free}}$. Every $\overline{xy}$ must pass only through $\mathbf{conv}\,(A,\ B)$, so when $\mathbf{conv}\,(A,\ B) \subseteq \mathcal{W}_{\text{free}}$, no $\overline{xy}$ can intersect an occlusion.

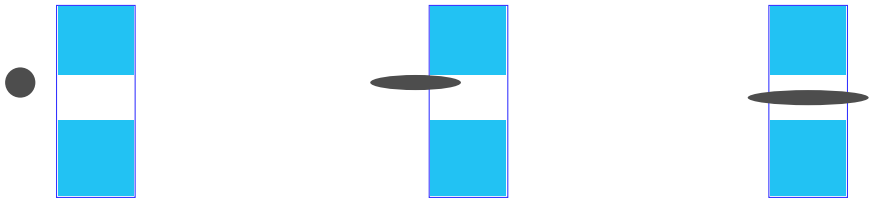

Fig. 2: 2D representations of $\mathcal{S}$. FULL, PARTIAL, UNKNOWN from left to right. The dark blue line represents a convex hull around partitions.

While any two regions must be fully visible, fully occluded, or partially visible/occluded, it is possible that our tests to check the visibility between specific partitions will not converge (see subsection 4.2). We address such cases with a fourth label, $\mathcal{V}(A,\, B)$ = UNKNOWN, and we address both PARTIAL and UNKNOWN through iterative refinement (see subsection 4.4).

Visibility Lattice We organize convex partitions and refinements (i.e., sub-partitions) as a bounded *visibility lattice* $\mathcal{T}$, offering a discrete abstraction for planning based on visibility constraints. Lattice nodes $\mathcal{Q}$ are partitions of world $\mathcal{W}$. The lattice partial order relation is subset-equals ($\subseteq$). This lattice permits inference of visibility between subset-ordered partitions.

Definition 3 (Visibility Lattice). *A visibility lattice $\mathcal{T}$ is a tuple $\mathcal{T} = (\mathcal{Q},\, \cap,\, \cup,\, \mathcal{W},\, \emptyset,\, \mathcal{S},\, \mathcal{V})$ forming a bounded lattice coupled with visibility information,*

- *$\mathcal{Q}$ are the lattice elements representing partitions, $\cup_{q \in \mathcal{Q}} = \mathcal{W}$,*
- *$\cap$ (set intersection) is the lattice meet,*
- *$\cup$ (set union) is the lattice join,*
- *$\mathcal{W}$ (world) is the lattice maximum,*
- *$\emptyset$ (empty set) is the lattice minimum,*
- *$\mathcal{S} = \{\text{FULL, OCCLUDED, PARTIAL, UNKNOWN}\}$ is the possible visibility,*
- *$\mathcal{V} : \mathcal{Q} \times \mathcal{Q} \to \mathcal{S}$ represents visibility between regions.*

Proposition 2 (Derived Visibility & Occlusion). *Subset ordered lattice nodes derive visibility or occlusion: when $(A' \subseteq A) \wedge (B' \subseteq B) \wedge (\mathcal{V}(A,\, B) \in \{\text{FULL}, \text{OCCLUDED}\})$, we know $\mathcal{V}(A',\, B') = \mathcal{V}(A,\, B)$.*

We use Definition 3 and Proposition 2 to construct and refine a visibility abstraction. Beginning with the lattice maximum, a refinement will subdivide some node $q \in \mathcal{Q}$ when either q contains occlusions or for some other partition q', $\mathcal{V}(q,\, q') \in \{\text{PARTIAL, UNKNOWN}\}$. We refine when the current abstraction does not admit a valid task plan (see subsection 4.3). Though general abstraction as convex partitions is possible, our particular refinement approach described in subsection 4.4 represents and refines partitions using octrees.

4.2 Visibility and Occlusion Tests

While testing for full visibility is directly — and exactly — computable from a convex hull (see Proposition 1), determining full or partial occlusion is more difficult, and in particular poses nonconvex problems (see Figure 3). We test occlusion cases through a combination of approaches (see Algorithm 1): convex decomposition of the free space, a linear relation to test possible lines-of-sight, and a nonlinear (nonconvex) program to fit a line-of-sight. This combination of approaches enables determination of full or partial occlusion in many cases, and we addresses the indeterminate cases (partial or unknown occlusion) through the refinement approach of subsection 4.4.

Algorithm 1: Visibility & Occlusion Test

Input: Convex Partitions A, B
Output: Visibility or Occlusion Status
1 **if conv** $(A,\ B) \in \mathcal{W}_{\text{free}}$ **then return** FULL;
2 **else if** *No path* $A \to B$ *in convex decomposition of* **conv** $(A,\ B) \setminus \mathcal{W}_{\text{obs}}$ **then**
3 | **return** OCCLUDED;
4 **else if** *No valid line-of-sight direction in linear relaxation* **then**
5 | **return** OCCLUDED;
6 **else if** *Valid line-of-sight in nonlinear program* **then return** PARTIAL;
7 **else return** UNKNOWN;

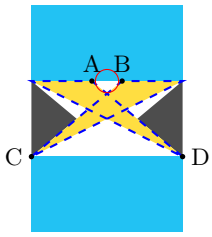

Fig. 3: Nonconvexity of fitting a LOS between convex regions. Dotted blue shows valid solutions. The red circle indicates a solution-space gap.

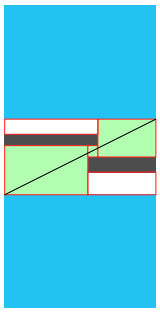

Fig. 4: Convex decomposition with solution valid LOS.

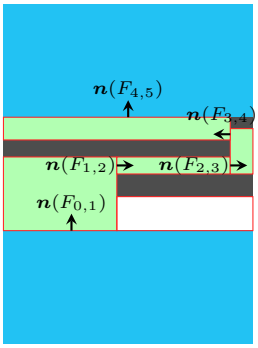

Fig. 5: Path p admits no solution for the linear relaxation (2).

Convex Hulls and Line-of-Sight Nonconvexity The first test checks whether the convex hull of two partitions contains only free space (line 1). Existing algorithms and libraries support testing for collisions between this convex hull and

$\mathcal{W}_{\text{obs}}$ [16]. Under Proposition 1, a convex hull only in free space implies full visibility.

Any occlusions within the convex hull of two partitions indicate that the partitions are either fully or partially occluded. However, the direct formulation to fit a line-of-sight is a nonconvex optimization problem. Figure 3 shows an example. Consider a line-of-sight parameterized by endpoints in each partition. This figure shows two valid endpoints for the line-of-sight, which are separated by an area of invalid endpoints, indicating the nonconvexity. We address this nonconvexity, through the decomposition and linear relaxation described in this section and the iterative refinement described in subsection 4.4.

Free Space Convex Decomposition Our first step to address line-of-sight nonconvexity is a *convex freespace decomposition graph* (line 2), which offers both an initial occlusion test and supports creating the subsequent linear relaxation to test full occlusion. The convex freespace decomposition of two nodes $A, B \in \mathcal{Q}$ is a convex decomposition D of $\mathbf{conv}(A, B) \setminus \mathcal{W}_{\text{obs}}$. A convex freespace decomposition graph is a graph $G = (N, E)$ where the nodes N are the individual convex spaces in D, and edges E exist between nodes in N when the nodes share a common face (see Figure 4). Any line-of-sight A, B must necessarily pass through adjacent nodes of N.

We test this necessary condition via graph search on G for path p starting in A and ending in B. If no path p exists, then A and B are fully occluded. Otherwise when we find such a path p, we proceed to linear relaxation and full nonlinear program described below.

Linear Relaxation to Test Occlusion Next, we develop linear relaxations for fitting line-of-sight that offer necessary conditions for visibility (line 4). In other words, when these linear relaxations admit no solution, we have shown full occlusion. Consider as described above path p through convex decomposition G.

A line-of-sight passing through p is parameterizable by origin point $\boldsymbol{o}$ and direction vector $\boldsymbol{v}$. Fitting both $\boldsymbol{o}$ and $\boldsymbol{v}$ yields nonconvexities such as in Figure 3. However, we observe necessary conditions from G and p producing a linear relaxation. First, we identify along p a set of hyperplanes (lines in 2D or planes in 3D worlds) that the line-of-sight must cross through the same direction. Second, we identify a set of halfspaces containing p, (determined by lines in 2D or planes in 3D worlds) in which the line-of-sight must fully reside. We describe each of these linear constraints below. Our relaxed constraint for the line-of-sight to cross hyperplanes in desired directions ensures valid angles between the line-of-sight and each hyperplane normal $\boldsymbol{n}$ (see Figure 5). Each angle must be within interval $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$, linearly expressible as a positive inner product, $\boldsymbol{v} \cdot \boldsymbol{n} > 0$, where $\boldsymbol{v}$ is a decision variable and each $\boldsymbol{n}$ is constant. The crossing hyperplanes and normals $\boldsymbol{n}$ come from G and p. For any consecutive regions $i, i+1$ along p, the line-of-sight must cross the facet (hyperplane) between those regions in the direction of i to $i+1$, yielding constraints,

$$\boldsymbol{v} \cdot \boldsymbol{n}(F_{i,i+1}) > 0, \ \forall i \,, \tag{2}$$

where $\boldsymbol{n}(F_{i,i+1})$ is the normal vector of facet $F_{i,i+1}$.

Our relaxed constraint for the line-of-sight to lie within halfspaces checks that two points along the line are both within the halfspace. We consider one point as origin $\boldsymbol{o}$ and another point as direction and magnitude $\boldsymbol{v}$ (i.e., $\boldsymbol{v}$ need not be a unit vector). Each halfspace H is bordered by hyperplane $\boldsymbol{n}(H) = d(H)$, so that each point $\mathbf{x}$ in the halfspace satisfies linear constraint $\mathbf{x} \cdot \boldsymbol{n}(H) < d(H)$. The necessary containing halfspaces arise from pairs of regions along p, including source A and destination B. For each pair of regions a and b, we identify half spaces as each region boundary through which the line-of-sight does not cross (i.e., boundaries that do not form facets used in (2)). The constraints are,

$$\begin{aligned} \boldsymbol{o} \cdot \boldsymbol{n}(H_{a,i}) &< d(H_{a,i}),\ \forall i \\ (\boldsymbol{o} + \boldsymbol{v}) \cdot \boldsymbol{n}(H_{b,j}) &< d(H_{b,j}),\ \forall j\,, \end{aligned} \tag{3}$$

where $H_{a,i}$ are the non-crossing boundaries of convex region a and $H_{a,i}$ are the non-crossing boundaries of convex region b.

The necessary condition for partial visibility is that there exists some path p where for every pair of regions a, b, a solution exists for constraints (2) and (3). Thus, if for every path p, we find some pair of regions a, b that admits no solution to the constraints, we may conclude regions A and B are fully occluded.

Nonlinear Program to Test Partial Visibility When the previous tests do not prove occlusion, we test for partial visibility by attempting to solve the nonlinear program for line-of-sight (line 6). Consider a line-of-sight parameterized as $\boldsymbol{x} = \boldsymbol{o} + t\boldsymbol{v}$, where $\boldsymbol{x}$ is any point on the line, $\boldsymbol{o}$ is a fixed point on the line, $\boldsymbol{v}$ is a direction vector, and t is a scalar factor. We determine line parameters $\boldsymbol{o}$ and $\boldsymbol{v}$ by constructing the nonlinear program from path p through convex decomposition G. As with the linear relaxation, the line-of-sight must cross through successive facets $F_{i,i+1}$ along p, and we now further (nonlinearly) constraint the line-of-sight to be contained with the facet $F_{i,i+1}$. We define the point $\boldsymbol{x}_{i,i+1}$ on the facet $F_{i,i+1}$ as $\boldsymbol{x}_{i,i+1} = \boldsymbol{o} + t_{i,i+1}\boldsymbol{v}$, introducing additional decision variable $t_{i,i+1}$. Each point $\boldsymbol{x}_{i,i+1}$ must lie on its facet's hyperplane, introducing the constraints,

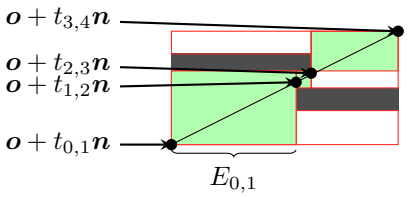

Fig. 6: Illustration of Equation 4 and Equation 5.

$$\overbrace{(\boldsymbol{o} + t_{i,i+1}\boldsymbol{v})}^{\boldsymbol{x}_{i,i+1}} \cdot \boldsymbol{n}(F_{i,i+1}) = d(F_{i,i+1}),\ \forall i\,. \tag{4}$$

Each point must further lie within boundaries of the facet, meaning it is contained in the set of edge halfspaces $E_{i,j}$ parallel to $\boldsymbol{v}$ and containing the endpoints $\boldsymbol{e_0}, \boldsymbol{e_1}$ of each edge e of $F_{i,j}$. This introduces the constraints,

$$\overbrace{\boldsymbol{o} + t_{i,i+1}\boldsymbol{v}}^{\boldsymbol{x}_{i,i+1}} \quad \cdot \quad \overbrace{\boldsymbol{n}(e)}^{\text{facet edge plane normal}} \quad < d(e),\ \forall i,\ \forall e \in (E_{i,i+1})\,. \tag{5}$$

A solution to (4) and (5) produces an unobstructed line-of-sight, meaning that the two partitions are partially occluded. If the nonlinear program does not converge to a valid solution, then we have an unknown case. We handle both partial and unknown visibility through our iterative refinement procedure.

4.3 Task Planning

In this section we describe our task planning approach. First we summarize key background on Linear Temporal Logic (LTL) and Satisfiability Modulo Theories (SMT), which are the logical formalism and reasoning we use to describe constraints for robot plans. Then we describe the constraint construction for visibility problems.

Background on SMT and LTL

Linear Temporal Logic (LTL) We specify visibility requirements Ψ using co-safe Linear Temporal Logic (LTL). Classically, LTL formulae define ω-languages (over infinite strings), and the co-safe LTL subset we employ addresses finite string requirements [12]. We summarize LTL semantics for membership of string π in the language of LTL formula ϕ. We say $\pi \vDash \phi$ to indicate that π *holds* in ϕ, and say that π consists of symbols $\sigma_0, \sigma_1, \sigma_2, \ldots$ where each σ_i is a member of 2^{AP}. The notation π^k means the suffix of π starting at k.

Definition 4 (LTL Semantics).

- *For all* π, $\pi \vDash \text{true}$ *and* $\pi \nvDash \text{false}$.
- *For an atomic proposition* $p \in AP$, $\pi \vDash p$ *if and only if* $p \in \sigma_0$ *and* $\pi \vDash \neg p$ *if and only if* $p \notin \sigma_0$
- $\pi \vDash \phi_1 \wedge \phi_2$ *if and only if* $\pi \vDash \phi_1$ *and* $\pi \vDash \phi_2$
- $\pi \vDash \phi_1 \vee \phi_2$ *if and only if* $\pi \vDash \phi_1$ *or* $\pi \vDash \phi_2$
- $\pi \vDash \phi_1 \mathcal{U} \phi_2$ *there is some* $k \geq 0$ *such that* $\pi^k \vDash \phi_2$ *and* $\pi^i \vDash \phi_1$ *for* $0 \leq i < k$

Co-safe LTL languages are distinguished by a finite prefix that marks a string as being in the language, followed by an infite suffix which does not affect the string's membership. For convience, we define two additional shorthand operators and their mappings to co-safe LTL. Practically always ($\blacksquare$) means an expression holds for the finite prefix and practically eventually ($\blacklozenge$) means an expression holds at some point during the prefix. We mark the end of the prefix with a supplemental proposition $\mathcal{E}$.

$$\blacksquare \phi \rightsquigarrow \phi \mathcal{U} \mathcal{E} \qquad \blacklozenge \phi \rightsquigarrow (\neg \mathcal{E}) \mathcal{U} \phi \tag{6}$$

These operators serve as convenience for an 'always-like' and 'eventually-like' specification in a co-safe LTL.

Satisfiability Modulo Theories (SMT) We take a constraint-based planning approach [11,18] represented using Satisfiability Modulo Theories (SMT) [6].

The constraint based encoding constructs a formula that represents possible plans over some finite horizon h. Decision variables in the formula indicate values of each fluent or state variable f at each step k, which we write as $f^{\langle k \rangle}$. Formula constraints represent valid state changes between successive steps k,$k+1$. Specifically, the formula consists of (1) a start constraint s which is the start configuration of fluents, (2) some formula c defining the transition function of the system at each step k, and (3) the goal configuration $g^{\langle h \rangle}$ which requires all the relevant fluents to have the desired values at the end of the plan. We then pass the formula to an SMT solver [5], which either produces a satisfying assignment encoding a plan or determines the formula is unsatisfiable. If the formula is unsatisfiable, one may choose to increase the number of steps h, or, especially in our case, find more information about the environment for the formula (see subsection 4.4).

Transforming LTL to SMT formulae Since we specify the visibility requirements using co-safe LTL, we must translate this specification to SMT [3,14]. We summarize this procedure in Algorithm 2. We directly handle practically always (■) and practically eventually (⧫) to eliminate the decision variable for $\mathcal{E}$ by setting values for $\mathcal{E}$ of $\mathcal{E}^{\langle h \rangle} = \top$ and $\mathcal{E}^{\langle k \rangle} = \bot,\ \forall k \in 1, \ldots, h-1$.

Algorithm 2: Transforming an LTL formula to SMT constraints

```
   Input: ψ; // LTL formula
   Input: R; // Robots
   Input: N; // The plan horizon
   Output: SMT formula F_ψ
 1 Function ExpandLTL(φ, min, max):
 2     if Proposition?(φ) then return φ;
 3     else if oper(φ) ∈ {∧, ¬, ∨} then
 4         F ← ();
 5         for o in operands(φ) do F ← F oper(φ) ExpandLTL(o, min, max);
 6         return F
 7     else if oper(φ) = ⧫ then
 8         F ← () ;
 9         for k from min to max do
10             F ← F ∨ ExpandLTL(expression(φ), k, max);
11         return F
12     else if oper(φ) = ■ then
13         F ← () ;
14         for k from min to max do
15             F ← F∧ ExpandLTL(expression(φ), k, max);
16         return F
17 return ExpandLTL(ψ, 0, N);
```

Task Planning for Visibility Now we describe the task planning approach for Definition 1. Using a constraint-based formulation, there are three parts to the constraints: motion between partitions in the visibility abstraction, visibility status based on the current set of partitions, and the visibility specification from Definition 1. Once we have constructed the formula, we find a satisfying assignment representing the plan.

Mobility Constraints Mobility constraints encode the movement of robots through the environment. The possible locations are partitions $\mathfrak{p} \in \mathcal{Q}$, which we represent using an SMT enumerated type. For each robot r, we define the fluent $\mathrm{at}(r)$ that maps to the robot's location $\mathfrak{p} \in \mathcal{Q}$. Movement is possible between adjacent partitions according to the following constraint,

$$\left(\mathrm{at}(r)^{\langle k+1 \rangle} = i\right) \implies \left(\mathrm{at}(r)^{\langle k \rangle} = i\right) \vee \left(\bigvee_{j \in \mathrm{nbrs}(i)} \left(\mathrm{at}(r)^{\langle k \rangle} = j\right) \right), \ \forall r, i, k \ , \quad (7)$$

where k indicates the current timestep, i and j are partitions, and $\mathrm{nbrs}(i)$ are neighboring partitions of i .

Visibility Constraints Visibility constraints determine which robots and partitions are visible or occluded. We construct the constraints from known visibilities described in subsection 4.1. A robot r_1 in partition $\mathfrak{p}_1$ inherits visibility status of the partition. First, considering one robot, other partitions $\mathfrak{p}_2$ are visible or occluded according to the status with respect to $\mathfrak{p}_1$,

$$\left(\mathrm{at}(r_1)^{\langle k \rangle} = \mathfrak{p}_1\right) \implies \left(\mathcal{V}(r_1, \mathfrak{p}_2)^{\langle k \rangle} = \mathcal{V}(\mathfrak{p}_1, \mathfrak{p}_2)\right) \ . \quad (8)$$

Second, considering another robot r_2 in partition $\mathfrak{p}_2$, visibility status is similarly inherited,

$$\left(\mathrm{at}(r_1)^{\langle k \rangle} = \mathfrak{p}_1\right) \wedge \left(\mathrm{at}(r_2)^{\langle k \rangle} = \mathfrak{p}_2\right) \implies \left(\mathcal{V}(r_1, r_2)^{\langle k \rangle} = \mathcal{V}(\mathfrak{p}_1, \mathfrak{p}_2)\right) \ . \quad (9)$$

Visibility Specification The final part of the constraints is the visibility specification Ψ from Definition 1. Considering Ψ as a formal language specified in co-safe LTL, we construct the SMT constraints using Algorithm 2.

Once we have built the full set of constraints, we check satisfiability. If a satisfying assignment is found, the plan, in the form of successive locations for robots, is extracted from the assignments to the $\mathrm{at}(\alpha, \beta)$ variables. Otherwise, we refine the abstraction and iterate (see Figure 1).

4.4 Refinement

Although our tests for $(\mathrm{FULL}, \mathrm{OCCLUDED})$ will only label a pair of partitions with these labels when they are exactly in those states as defined in Definition 2, the $(\mathrm{PARTIAL}, \mathrm{UNKNOWN})$ states are less useful and difficult to plan with. We therefore develop a refinement procedure based on the visibility lattice (see Definition 3) properties (see Proposition 2) and optimization formulations

(subsection 4.2) to address cases where a plan may need to traverse these indeterminate regions. Algorithm 3 summarizes the procedure. Refining convex partition $\mathfrak{p}$ subdivides $\mathfrak{p}$ into multiple convex children $\mathfrak{p}_0, \ldots, \mathfrak{p}_n$ whose union is $\mathfrak{p}$ (line 1); our implementation (see section 5) subdivides partitions as octrees, though other partitioning methods, such as binary-space partitioning, are possible; all facet edges of the partitions must decrease during refinement steps. Under Proposition 2, when $\mathfrak{p}$ and another partition $\mathfrak{p}'$ are fully visible or occluded, then for each subpartition $\mathfrak{p}_i$ of $\mathfrak{p}$, we may conclude $\mathfrak{p}_i$ and $\mathfrak{p}'$ are equivalently fully visible or occluded (line 4). Otherwise, when $\mathfrak{p}$ and $\mathfrak{p}'$ are indeterminate, we perform the occlusion tests of Algorithm 1 (line 5). The refined partitioning may indicate that a planned sequence traversals are valid, invalid, or still indeterminate. When the traversals are valid, we have found the plan and terminate. When the traversals are invalid, we find an alternative task plan. When the traversals are still indeterminate, we iteratively continue to refine.

Algorithm 3: Visibility Refinement

Input: $\mathcal{W} = \mathcal{W}_{\text{free}} \cup \mathcal{W}_{\text{obs}}$, $\mathfrak{p}$; // Workspace, Partition to Refine
InOut: $\mathcal{Q}$, $\mathcal{V}$; // Set of Partitions, Visibility

1 $\mathfrak{p}_0, \ldots, \mathfrak{p}_n \leftarrow$ `subdivide`$(\mathfrak{p})$;
2 **foreach** $\mathfrak{p}_i \in \mathfrak{p}_0, \ldots, \mathfrak{p}_n$ **do**
3 **foreach** $\mathfrak{p}' \in$ `leaves`$(\mathcal{Q})$ **do**
4 **if** $\mathcal{V}(\mathfrak{p}, \mathfrak{p}') \in \{\text{OCCLUDED}, \text{FULL}\}$ **then** $\mathcal{V}(\mathfrak{p}_i, \mathfrak{p}') \leftarrow \mathcal{V}(\mathfrak{p}, \mathfrak{p}')$;
5 **else** $\mathcal{V}(\mathfrak{p}_i, \mathfrak{p}) \leftarrow$ `ComputeOcclusion`$(\mathfrak{p}_i, \mathfrak{p})$; // Algorithm 1
6 $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mathfrak{p}_0, \ldots, \mathfrak{p}_n\}$;

4.5 Completeness

In this section we analyze the completeness properties of our algorithm. For infinitesimal (point) robots, we show that our approach is complete under clearance assumptions.

Definition 5. *An r-clearance path is a set of partitions P needed to be traversed, visible, or occluded to satisfy Ψ where each partition has a non-infinitesimal volume of at least r.*

Lemma 1. *When an r-clearance path exists, then in finite time, $\mathcal{T}$ will contain leaf nodes covering the r-clearance path and which have fully determined status, that is, one of* OCCLUDED, FULL.

Proof. For the FULL case we have a direct proof. Since our path exists, for some partition in the path P_i where full visibility is required to some other partition $\mathfrak{p}'$, eventually, we will have some region $\mathfrak{p} \in \mathcal{Q}$, such that $\mathfrak{p} \subseteq P_i$, so that the FULL state will be found by our algorithm—i.e., $\mathbf{conv}\,(\mathfrak{p}',\, P_i) \cap \mathcal{W}_{\text{obs}} = \emptyset$. Since $\mathfrak{p} \subseteq P_i$, we can conclude $\mathbf{conv}\,(\mathfrak{p}',\, \mathfrak{p}) \cap \mathcal{W}_{\text{obs}} = \emptyset$ also.

Next we cover the OCCLUDED case:

- Since the r-clearance path exists, for some partition in the path P_i which must be OCCLUDED to some other partition $\mathfrak{p}'$, we must have a node $\mathfrak{p} \subseteq P_i$, $\mathfrak{p} \in \mathcal{Q}$.
- Necessary occluded partitions along an r-clearance path must be caused by non-infinitesimal obstacles.
- For two small-enough occluded partitions, all potential lines of sight are cut by some single obstacle, that is, an individual line from some point $p_1 \in \mathfrak{p}'$ and $p_2 \in \mathfrak{p}$ must be cut off by a face of $\mathcal{W}_{\text{obs}}$.
- When leaf partitions become small enough, we are eventually guaranteed to hit a special case in the graph search on line 2 of Algorithm 1. That is, no path in the freespace decomposition graph exists from $\mathfrak{p}$ to $\mathfrak{p}'$, since given a small enough size ($< r$), a single obstacle in $\mathcal{W}_{\text{obs}}$ will divide the freespace decomposition graph into disconnected parts and no path is possible. Since obstacles are non-infinitesimal, we will eventually find these regions, in finite time, even if the other steps of the occlusion-finding procedure fail to produce OCCLUDED.

Theorem 1. *Our algorithm is complete for point-robots.*

Proof. The proof follows from Lemma 1. If an r-clearance path exists, we will eventually find nodes in $\mathcal{T}$ that are in the path in finite time and construct a plan. Otherwise, we continue refining.

This concludes our completeness proof. Note that this proof does not cover non-point robots which may be larger than a region of occlusion or which may be too large to pass between regions of occlusion. For such non-point robots, we can still obtain completeness where Ψ only contains visibility constraints like $\text{visible}(\alpha, \beta)$ since the first part of Lemma 1 still applies. However, we have not proven completeness of this algorithm for non-point robots and occlusion constraints in Ψ of the form $\neg\, \text{visible}(\alpha, \beta)$.

5 Experiments

In this section we show our experimental results using our approach for two different types of problems, one with eventual visibility constraints (we must eventually find something visible) and another with always occlusion constraints (we must never be seen). All experiments were run on a multi-core system with a single AMD Ryzen 9 12-core processor with hyperthreading enabled, for a total of 24 usable cores, and 32 GB of RAM. For convex decompositions and collision detection, we use the CGAL project [20]. To construct and solve SMT expressions, we use the TMKit [4] software package to interface with the Z3 SMT solver [5].

To leverage multiple CPUs, line 5 in Algorithm 3 is run asynchronously on multiple threads so that the `ComputeOcclusion` algorithm is run in parallel for different pairs of regions in $\mathcal{Q}$. All experiments used 14 threads for this kind of refinement.

5.1 Industrial Survey

Here, we require a team of robots not bound to the ground (like drones) to survey an industrial site. The goal is for a team of drones to obtain visibility of wind turbines in a wind farm.

We created a scene of 16 wind turbines on a heightmap. The robots (two drones, x and y) were tasked with observing various "user regions". User regions are like regular regions in $\mathcal{Q}$ except they are specified by the user, and do not get subdivided during refinement. User regions were specified to surround all turbines so that if all user regions around a turbine were seen, most important details of the turbine should be visible. See Figure 7 and Table 1 for details. The LTL specification is,

$$\blacklozenge \left(\bigwedge_{r \in \text{user-regions}} (\text{visible}\,(x,\ r) \vee \text{visible}\,(y,\ r)) \right) . \tag{10}$$

This experiment shows the ability of our approach to find regions in space to observe given target regions, and the expressiveness of LTL for visibility requirements on multiple robots.

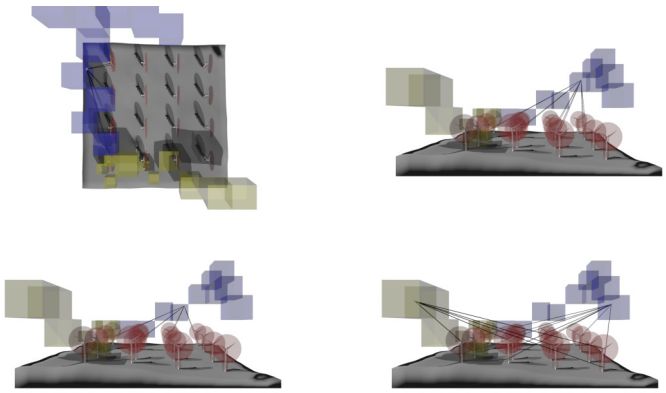

Fig. 7: A single plan requiring all user regions (red) to be fully visible to at least one drone. Partitions for different robots are drawn as translucent cubes. LOS is shown by a black line from the robot to the target.

5.2 Wildlife Count

In this experiment we solve the "Park Ranger Problem". An agent (x) (a park ranger drone) must avoid being seen by an area (R) known to have sensitive wildlife while observing some other target region (G)—perhaps a different set of wildlife to be observed, or a potential search and rescue target. The algorithm

successfully finds a plan where the "avoid" region is always fully occluded from the goal region. The LTL specification for this problem is,

$$\blacksquare\left(\neg \operatorname{visible}(x,\ R)\right) \wedge \blacklozenge(\operatorname{visible}(x,\ G))\ . \tag{11}$$

The results are in Figure 8 and Table 1. This experiment demonstrates that our approach also allows for the reverse of the experiment in subsection 5.1—we can stay fully occluded from some undesirable region.

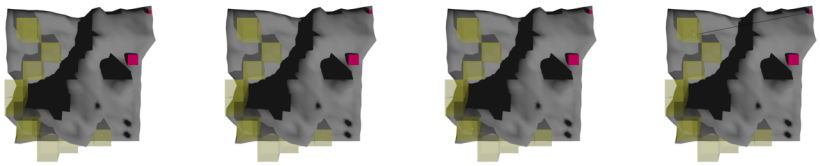

Fig. 8: Snapshots of the plan for the park ranger drone to see the target wildlife area (small red box) while avoiding the larger sensitive area (larger red box).

	Nodes Processed	*Nodes in SMT*	*Planning Time(s)*	*Total(s)*
Industrial Survey	703	617	14141.86	21574.83
Wildlife Count	253	223	62.23	2512.91

Table 1: Experimental metrics. Plans in both scenarios were 10 steps.

6 Conclusion

In this paper we formulated the Park Rangers' Problem for sequential visibility tasks, and we introduce an abstraction and refinement approach for such tasks. Our method subdivides the robot world into partitions to enable symbolic planning for the visibility requirements, and we analyze the method to show its completeness under clearance and robot size assumptions. Our experimental results demonstrate the approach's validity, and future work to combine abstraction steps (such as convex decompositions) and efficiently update constraints (such as incremental constraints solving) will support planning for larger environments and more agents.

Acknowledgments. This work was supported in part by ARL TBAM-CRP [W911NF-22-2-0235] and the National Science Foundation (NSF) under Grant No. CCF-2124010.

References

1. Atanasov, N., Le Ny, J., Daniilidis, K., Pappas, G.J.: Information acquisition with sensing robots: Algorithms and error bounds. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). pp. 6447–6454. IEEE (2014)

2. Chin, W., Ntafos, S.: Optimum watchman routes. Information Processing Letters **28**(1), 39–44 (1988). https://doi.org/10.1016/0020-0190(88)90141-X
3. Cimatti, A., Pistore, M., Roveri, M., Sebastiani, R.: Improving the encoding of LTL model checking into SAT. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 196–207. Springer (2002)
4. Dantam, N.T., Chaudhuri, S., Kavraki, L.E.: The task motion kit. IEEE Robotics & Automation Magazine **25**(3), 61–70 (2018)
5. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340. Springer (2008)
6. De Moura, L., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Communications of the ACM **54**(9), 69–77 (2011). https://doi.org/10.1145/1995376.1995394
7. Durand, F., Drettakis, G., Puech, C.: The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In: Proceedings of the 24th annual conference on Computer graphics and interactive techniques. pp. 89–100 (1997)
8. Durand, F., Drettakis, G., Puech, C.: The 3d visibility complex. ACM Transactions on Graphics **21**(2), 176–206 (apr 2002). https://doi.org/10.1145/508357.508362
9. Fletcher, L.G., Perali, P., Beathard, A., O'Kane, J.M.: A visibility-based escort problem. In: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2023)
10. Guibas, L.J., Latombe, J.C., Lavalle, S.M., Lin, D., Motwani, R.: A visibility-based pursuit-evasion problem. International Journal of Computational Geometry & Applications **09**(04n05), 471–493 (1999). https://doi.org/10.1142/S0218195999000273
11. Kautz, H.A., Selman, B.: Blackbox: A new approach to the application of theorem proving to problem solving. In: AIPS98 Workshop on Planning as Combinatorial Search. pp. 58–60 (1998)
12. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods in System Design **19**, 291–314 (2001)
13. LaValle, S.M.: Planning Algorithms. Cambridge University Press, USA (2006)
14. Li, J., Zhu, S., Pu, G., Zhang, L., Vardi, M.Y.: SAT-based explicit LTL reasoning and its application to satisfiability checking. Formal Methods in System Design **54**, 164–190 (2019)
15. Menghi, C., Garcia, S., Pelliccione, P., Tumova, J.: Multi-robot ltl planning under uncertainty. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) Formal Methods. pp. 399–417. Springer International Publishing, Cham (2018)
16. Pan, J., Chitta, S., Manocha, D.: FCL: A general purpose library for collision and proximity queries. In: 2012 IEEE International Conference on Robotics and Automation (ICRA). pp. 3859–3866 (2012)
17. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science. pp. 46–57. IEEE (1977)
18. Rintanen, J.: Engineering efficient planners with SAT. In: European Conference on Artificial Intelligence (ECAI). pp. 684–689 (2012)
19. Sarmientoy, A., Murrieta-Cidz, R., Hutchinsony, S.: A sample-based convex cover for rapidly finding an object in a 3-d environment. In: 2005 IEEE International Conference on Robotics and Automation (ICRA). pp. 3486–3491 (2005). https://doi.org/10.1109/ROBOT.2005.1570649
20. The CGAL Project: CGAL User and Reference Manual. CGAL Editorial Board, 5.6 edn. (2023), https://doc.cgal.org/5.6/Manual/packages.html

21. Urrutia, J.: Chapter 22 - art gallery and illumination problems. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 973–1027. North-Holland, Amsterdam (2000). https://doi.org/10.1016/B978-044482537-7/50023-1
22. Zhi, J., Hao, Y., Vo, C., Morales, M., Lien, J.M.: Computing 3-d from-region visibility using visibility integrity. IEEE Robotics and Automation Letters **4**(4), 4286–4291 (2019). https://doi.org/10.1109/LRA.2019.2931280
23. Zou, R., Bhattacharya, S.: On optimal pursuit trajectories for visibility-based target-tracking game. IEEE Transactions on Robotics **35**(2), 449–465 (2018)