Clearing an Orthogonal Polygon to Find the Evaders

Salma Sadat Mahdavi

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
e-mail: `ss.mahdavi110@gmail.com`

Mohammad Ghodsi[1]

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Institute for Research in Fundamental Sciences (IPM) School of Computer Science
e-mail: `ghodsi@sharif.ir`

Abstract

In a multi-robot system, a number of autonomous robots would sense, communicate, and decide to move within a given domain to achieve a common goal. In the pursuit-evasion problem, a polygonal region is given and a robot called a pursuer tries to find some mobile targets called evaders. The goal of this problem is to design a motion strategy for the pursuer such that it can detect all the evaders. In this paper, we consider a new variant of the pursuit-evasion problem in which the robots (pursuers) each moves back and forth along an orthogonal line segment inside a simple orthogonal polygon P. We assume that P includes unpredictable, moving evaders that have unbounded speed. We propose the first motion-planning algorithm for a group of robots, assuming that they move along the pre-located line segments with a constant speed to detect all the evaders with unbounded speed. Also, we prove an upper bound for the length of the paths that all pursuers move in the proposed algorithm.

Keywords: Computational Geometry, Art Gallery, Motion Planning, Pursuit Evasion, Multi Robot Systems, Sliding Robot.

1. Introduction

The mathematical study of the "pursuit-evasion" problem was first considered by Parson [1]. After that, the watchman route problem was introduced as a variation of the art gallery problem, which consists of finding static evaders in a polygon. The visibility-based motion-planning problem was introduced in 1997 by Lavalle et al. [2]. The aim was to coordinate the motions of one or more robots (pursuers) that have omnidirectional vision sensors to enable them to eventually "see" an evader

☆Fully documented templates are available in the elsarticle package on CTAN.

[1]This author's work was partially supported by IPM under Grant No. CS-1392-2-01

that is unpredictable, has an unknown initial position, and is capable of moving arbitrarily fast. The process of detecting all evaders is also known as clearing the polygon. The pursuit-evasion problem has a broad range of applications such as in air traffic control, military strategy, and trajectory tracking [2].

In 2011, Katz and Morgenstern introduced sliding camera guards for guarding orthogonal polygons [3]. A security camera slides back and forth along a horizontal (vertical, respectively) track and views every point along the track, directly upwards (leftwards, respectively) and directly downwards (rightwards, respectively). We define our "Robots" to be the same as the security cameras, where a robot r would travel back and forth along an axis-aligned line segment s inside an orthogonal polygon P. A point p is seen by s if there exists a point $q \in s$ such that $\overline{pq}$ is a line segment perpendicular to s and is completely inside P. The set of all points of P that can be seen by s is its sliding visibility polygon (see Figure 1). The point p is seen by the robot r, if r is at point q on s (e.g., $r = q$).

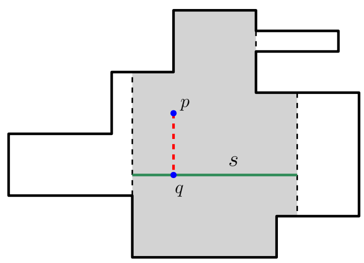

Figure 1: The shaded area shows the sliding visibility polygon of s.

The important reason for defining our robots is that, in spite of the definition of sliding camera in all the previous papers about this concept ([3][4][5]), it is assumed that a sliding camera can see all parts of its sliding visibility polygon simultaneously (which is a contradiction). So, we define our robots which move along the sliding cameras tracks and can see along the line segment perpendicular to its moving path. Therefore, the definition of our robots is more realistic than sliding cameras of the previous papers.

According to the visibility-based motion-planning problem and our defined robots, we study the new version of planning the motions for a group of robots for clearing an orthogonal polygon when robots are modeled as sliding cameras. We call our defined robots as "**Sliding Robots**". The given orthogonal polygon P has unpredictable, moving evaders with unbounded speed. Motion planning for a group of sliding robots to clear P means presenting a sequence of motions for the sliding robots such that any point of P is viewed by at least one robot. Moreover, a set of pre-located line segments, S, is given such that the union of their sliding visibility polygons is P.

Previous Works

Generally, in the pursuit-evasion problem, the pursuer is considered as an l-searcher with l flashlights and rotates them continuously with a bounded angular rotation speed [6]. Thus, an ∞-searcher (also known as an omnidirectional searcher) is a mobile robot equipped with a 360° view sensor for detecting evaders. Lavalle et al. proposed the first algorithm for solving the pursuit-evasion problem for an l-searcher [2]. They decomposed P into cells based on visibility properties and converted the problem to a search on an exponential-sized information graph. Durham et al. [7] addressed the problem of coordinating a team of mobile robots with limited sensing and communication capabilities to detect any evaders in an unknown and multiply connected planar environment. They proposed an algorithm that guarantees the detection of evaders by maintaining a complete coverage of the frontier between cleared and contaminated regions while expanding the cleared region.

The art gallery problem is a classical problem in computational geometry. Over the years, many variants of this problem have been studied [8, 9, 10, 11]. Most of these have been proved to be NP-hard [12], including the problem when the target region is a simple orthogonal polygon, and the goal is to find the minimum number of vertex guards to guard the entire polygon (e.g., [8, 11]). Some types of them, which consider the limited model of visibility, use polynomial time algorithms [13, 14].

The study of the art gallery problem based on the sliding camera was started in 2011 by Katz and Morgenstern [3]. They studied the problem of guarding a simple orthogonal polygon using minimum-cardinality sliding cameras (MCSC). They showed that, when the cameras are constrained to travel only vertically inside the polygon, the MCSC problem can be solved in polynomial time. They left the computation of the complexity of the MCSC problem as an open problem. In 2013, Durocher and Mehrabi [4] studied these two problems: the MCSC problem and the minimum-length sliding camera (MLSC) problem, where the goal was to minimize the total length of the trajectories along which the cameras travel. They proved that the MCSC problem is NP-hard, when the orthogonal polygon has holes. They also proved that the MLSC problem is solvable in polynomial time even for orthogonal polygons with holes. In 2014, De Berg *et al.* [5] presented a linear-time algorithm for solving the MCSC problem in an x-monotone orthogonal polygon. The complexity of the MCSC problem on a simple orthogonal polygon remains as an open problem.

Our Result

In this paper for a given set S of orthogonal line segments, we propose an algorithm to plan the motion of at most $|S|$ sliding robots along certain segments of S so that the entire polygon is guarded.

Owing to the difficulty of having multiple cooperating robots executing common tasks, we present a new method by storing some information on each reflex vertex. We assume that the sliding robots have map of the environment (a simple orthogonal polygon) and they are capable of broadcasting a message to all other robots by sending signals. This way, the robots can have some communications with each other to maintain the coordination process. The main result of our algorithm is that, if S is a set of MCSCs that guard the whole P, then our algorithm will detect all evaders with the ***minimum number of sliding robots***, assuming that the sliding robots just move along line segments of S. We implement our proposed algorithm and present an examples in Section 6, the Implementation Section.

We can assume that the input of the algorithm is just an orthogonal polygon P. Then, we compute set S of line segments using the algorithm in [3] and [4]. The only restriction of the set of line segments which we use is that it should guard all parts of P. So, we can use the proposed algorithms of [3] and [4], which find the set of sliding cameras that guards the entire P. In some of the algorithms the aim is to minimize the total length of sliding cameras [3] and in some of them the aim is to minimize the number of sliding cameras [4].

In this paper, we assume P and S as the inputs of the algorithm.

2. Preliminaries and Notations

Let P be an orthogonal polygon and $V(P) = \{v_1, v_2, ..., v_n\}$ be the set of all vertices of P in counterclockwise order. So, n is the number of vertices of P. We consider $V_{ref}(P)$ to be all of the reflex vertices of P and assume a general position such that no four reflex vertices are collinear. The reason of assuming the general position is explained in the Appendix in Section 8.

Suppose that $P(a, b)$ is a sub-polygon of P whose boundary is from a to b (a and b are two points on the boundary of P) in counterclockwise order.

Let v_j be a reflex vertex of P. v_j has two edges, $e_{j-1} = \overline{v_{j-1}, v_j}$ and $e_j = \overline{v_j, v_{j+1}}$, that can be extended inwardly until reach the boundary of P (See Figure 2).

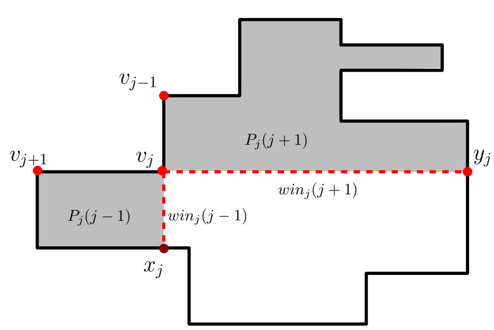

Figure 2: The windows and the sub-polygons of v_j are shown.

We call these extensions as the windows of v_j and denote them as $win_j(j-1) = \overline{v_j x_j}$ and $win_j(j+1) = \overline{v_j y_j}$, respectively. $win_j(j-1)$ and $win_j(j+1)$ are two line segments whose endpoints are on the boundary of P. $win_j(j-1)$ partitions P into two sub-polygons. Let $P_j(j-1)$ be a sub-polygon that consists of v_{j+1}, and let $P'_j(j-1)$ be $P \setminus P_j(j-1)$. Therefore, $P_j(j-1)$ and $P'_j(j-1)$ are denoted by $P(v_j, x_j)$ and $P(x_j, v_j)$, respectively. Similarly, let $P_j(j+1)$ be a sub-polygon that is separated from P by $win_j(j+1)$ and consists of v_{j-1}, and let $P'_j(j+1)$ be a sub-polygon that includes v_{j+1}. Therefore, $P_j(j+1)$ and $P'_j(j+1)$ are denoted by $P(y_j, v_j)$ and $P(v_j, y_j)$, respectively. Let L be the set of all lines which pass through the windows of P. L partitions P into orthogonal rectangles.

For each $v_j \in V_{ref}(P)$, we store an array called $FF_j(i)$ $(1 \le i \le 4)$ of size four in which the cells (of type Boolean) indicate whether the sub-polygons $P_j(j-1)$, $P_j(j+1)$, $P'_j(j-1)$, and $P'_j(j+1)$ are cleared (true), respectively.

3. The Proposed Algorithm

In this section, we present an algorithm for solving the pursuit-evasion problem using sliding robots. Assume that an orthogonal polygon P and a set of orthogonal line segments $S = \{s_1, s_2, \ldots, s_k\}$ are given. We present a path-planning algorithm for finding the unpredictable evaders using a set of sliding robots $R = \{r_1, r_2, \ldots, r_k\}$ in which r_i can move along the line segment s_i (k is the number of line segments).

In our algorithm at each time one sliding robot moves and clears some portion of P. The other robots are divided in two groups. The set of the robots which are waiting to clear some parts of P and the rest of them which are stopped until some request arrive. To distribute the movements of the robots, we define the "event points" as below:

Definition 1. An **event point** happens when r_i sees a reflex vertex, sees its corresponding waiting sliding robot, or reaches an endpoint of s_i.

3.1. Overview of the Algorithm

Our algorithm has six steps. The "start step," the "decision step," the "sending a signal step," the "update step," the "move back step,"and the "termination step." We assume that P is initially contaminated and we should clear the whole of it. To present our path-planning method, we start with an arbitrary sliding robot $r_i \in R$, which can move along $s_i \in S$. The first robot, r_i, starts moving from one endpoint of s_i. When r_i reaches an event point, it updates the cleared sub-polygons. By the time that r_i finishes its clearing, it moves back along s_i. Moreover, at each event point, r_i stops and, according to the cleared sub-polygons of P, decides to continue its movement or wait and send a signal to the other robots to clear a specific sub-polygon of P. When r_i sends a signal to the other robots to clear a sub-polygon, such as P_1, an arbitrary robot that can clear some parts of P_1 starts moving along its corresponding line segment. When all parts of P become cleared, the algorithm terminates.

3.2. Details of the Algorithm

Now, we explain the steps of the algorithm in detail. We store the status of the regions in their corresponding reflex vertices, which are updated by the robots during the movements to keep track of the contaminated regions, which is helpful in the decision-making process.

For each $r_i \in R$, we consider an array, which is called $D_i(j), 1 \leq j \leq 3$. Each storage includes an interval such as (a, b), which indicates the boundary of P between a and b in counterclockwise order. These storage are updated at each event points. The first storage, $D_i(1)$, indicates the cleared sub-polygon of P when r_i is clearing ($D_i(1)$ maybe cleared partly by r_i). The second storage, $D_i(2)$, indicates the sub-polygon of P that should be cleared by r_i and maybe some other robots. The third storage, $D_i(3)$, specifies the sub-polygon that should be cleared until r_i can continue its movement. Initially, we assume that all parts P are contaminated; therefore, $\forall_{r_i \in R} D_i(1) = \emptyset$ and $\forall_{v_j \in V_{ref}(P), 1 \leq i \leq 4}$ $FF_j(i) = false$. Also, we assume that $\forall_{r_i \in R} D_i(2) = D_i(3) = \emptyset$. Note that, except the start and termination steps, there is no order for the other steps and they can be done in any order.

Start Step

As mentioned earlier, we start with one of the endpoints of an arbitrary s_i (r_i moves along s_i).

- If r_i is going to start from an endpoint that is on the boundary, r_i can see two consecutive vertices (suppose the endpoint is on the edge $e_k = \overline{v_k v_{k+1}}$).
  - If v_k and v_{k+1} are convex (for example, the left endpoint of s_1 in Figure 3), then r_i starts clearing P by its movement and updates $D_i(1) = (v_k, v_{k+1})$ and $D_i(2) = (v_{k+1}, v_k)$. r_i continues its movement along s_i until an event point happens. At each event point, r_i does the update step then the decision step.
  - If at least one of v_k or v_{k+1} is a reflex vertex (for example, the lower endpoint of s_1 or s_2 in Figure 4), then r_i cannot start clearing P and stops on the endpoint. Suppose that the maximal line segment passes through edge e_k is l. Let x and w be the first intersection of l at the boundary of two sides. s_i can be inside the sub-polygon corresponding to (x, w) or (w, x). Assume that s_i is inside (w, x). Therefore, r_i stops on the endpoint and does the decision step and updates $D_i(2) = (w, x)$.
- If r_i is going to start from an endpoint that is not on the boundary, then r_i cannot start clearing P; it therefore stops on the endpoint and does the decision step. Suppose that the maximal normal line segment to s_i that passes through r_i is lr. Let x and w be the first intersection of l at the boundary of two sides. s_i can be inside the sub-polygon corresponding to (x, w) or (w, x). Assume that s_i is inside (w, x). Therefore, r_i sends a signal to the other robots to clear (x, w),

and updates $D_i(2) = (w, x)$ and $D_i(3) = (x, w)$. As shown in Figure 3, if r_2 is going to start from z, it stops and sends a signal to the other robots to clear the sub-polygon corresponding to (x, w).

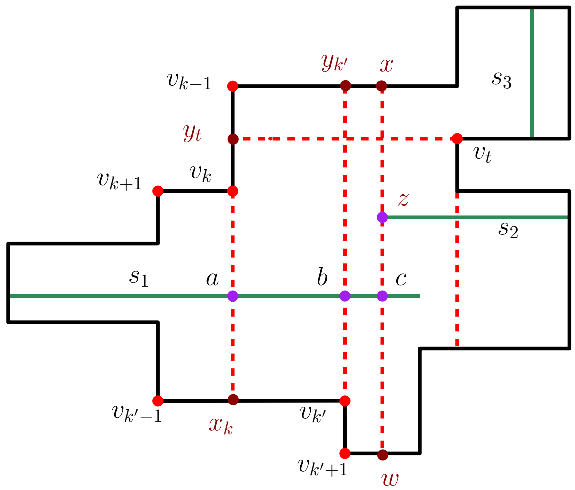

Figure 3: Sliding robots r_1, r_2, and r_3 move along line segments s_1, s_2, and s_3, respectively.

Update Step

Assume that r_i moves along s_i. When an event point happens, r_i stops and updates $D_i(1)$ (increases the cleared region) and $D_i(2)$ (decreases the sub-polygon that should be cleared). See Figure 3; when r_1 starts moving from left endpoint and reaches a, it updates $D_1(1) = (v_k, x_k)$ and $D_1(2) = (x_k, v_k)$. When r_1 reaches b, it updates $D_1(1) = (y_{k'}, v_{k'+1})$ and $D_1(2) = (v_{k'+1}, y_{k'})$. These updates can be done in $\mathcal{O}(1)$ time by changing two endpoints of $D_i(1)$ and $D_i(2)$.

When r_i sees a reflex vertex, v_k, during its movement, it updates $FF_k(j)$ for $1 \leq j \leq 4$ as detailed below:

Note that if $(v_k, x_k) \in D_i(1)$ and $(y_k, v_k) \in D_i(1)$, then $v_{k+1} \in D_i(1)$ and $v_{k-1} \in D_i(1)$, respectively.

- If $(v_k, x_k) \in D_i(1)$, then $P_k(k-1)$ is cleared and r_i updates $FF_k(1) = true$ (See Figure 3; when r_1 moves back from left to right and reaches a, it sees v_k).

- If $(x_k, v_k) \in D_i(1)$, then $P'_k(k-1)$ is cleared and r_i updates $FF_k(3) = true$ (See Figure 3; when r_1 moves back from right to left and reaches a, it sees v_k).

- If $(y_k, v_k) \in D_i(1)$, then $P_k(k+1)$ is cleared and r_i updates $FF_k(2) = true$ (See Figure 3; when r_1 moves back from left to right and reaches b, it sees $v_{k'}$).

- If $(v_k, y_k) \in D_i(1)$, then $P'_k(k+1)$ is cleared and r_i updates $FF_k(4) = true$ (See Figure 3; when r_1 moves back from right to left and reaches b, it sees $v_{k'}$).

Move Back Step

Assume that an event point happens when r_i moves along s_i. Then, r_i updates $D_i(1)$ and $D_i(2)$. At each time that $D_i(2)$ becomes empty while $D_i(1) \neq \emptyset$, r_i finishes its clearing and moves back along s_i. It moves back until it sees a waiting robot or reaches an endpoint of s_i. While it is moving back, if r_i sees its corresponding waiting robot (supposedly r_j) and $D_i(1) = D_j(3)$, then $D_i(2) = \emptyset$. Therefore, r_i updates $D_j(3) = \emptyset$, $D_j(1) = D_j(1) \bigcup D_i(1)$, and $D_j(2) = D_j(2)/D_i(1)$. If r_i sees a reflex vertex v_k during moving back, it updates $FF_k(j)$ for $1 \leq j \leq 4$ as explained in "Update Step". Since $D_i(2)$ is empty, r_i finishes its clearing and r_j starts moving back. r_j can be collinear with the endpoint of s_i. See Figure 3; when r_1 moves back from left to right and reaches c, it sees the waiting robot r_2 at z. So, r_1 updates the storage of r_2 and r_2 moves back.

Decision Step

When r_i sees an event point, it stops, does the "Update Step" or may do the "Move Back Step". In the case that r_i is on the endpoint of s_i, we do as below.

Let ep be the endpoint which r_i is on that. ep can be on the boundary (I) or inside (II) P.

$(I.)$ Suppose that ep is **on the boundary** of P, it lies on an edge of P called $e_k = \overline{v_k v_{k+1}}$. In this situation do as below:

1. When $v_k \in V_{ref}(P)$, (See Figure 4; assume that r_3 is on the blue point of s_3.)

   (a) If $P_k(k+1)$ is contaminated (i.e., $FF_k(2) = false$), then $P_k(k+1)$ should be cleared. Therefore, r_i waits and sends a signal to the other robots to clear $P_k(k+1)$ (indicated by (y_k, v_k)) and updates $D_i(3) = (y_k, v_k)$.

   (b) Else, $P_k(k+1)$ is cleared (i.e., $FF_k(2) = true$). So, add $P_k(k+1)$ to the cleared parts, $D_i(1) = D_i(1) \cup (y_k, v_k)$ and decrease it from the parts that should be cleared, $D_i(2) = D_i(2) \setminus (y_k, v_k)$.

2. When $v_{k+1} \in V_{ref}(P)$, (See Figure 4; assume that r_1 or r_2 is on the blue point of s_1 or s_2, respectively.)

   (a) If $P_{k+1}(k)$ is contaminated (i.e., $FF_{k+1}(1) = false$), then $P_{k+1}(k)$ should be cleared. Therefore, r_i waits and sends a signal to the other robots to clear $P_{k+1}(k)$ (indicated by (v_{k+1}, x_{k+1})) and updates $D_i(3) = (v_{k+1}, x_{k+1})$.

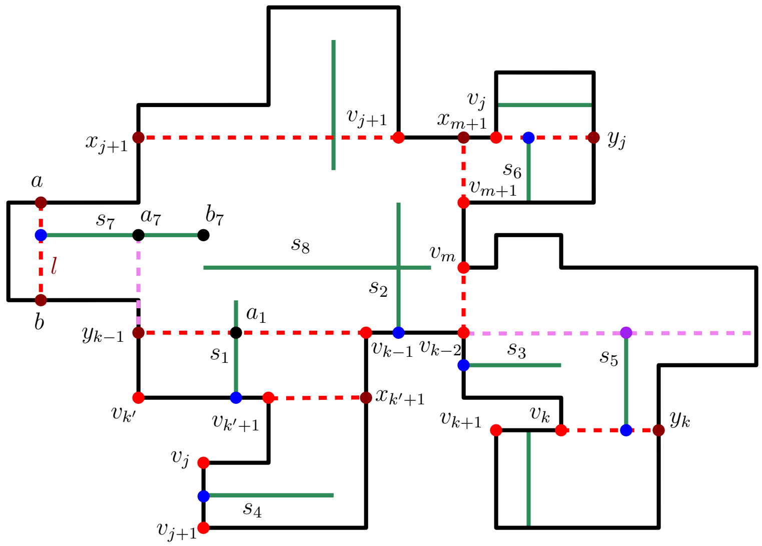

Figure 4: Sliding robot r_i moves along line segment s_i.

(b) Else, $P_{k+1}(k)$ is cleared (i.e., $FF_{k+1}(1) = true$). So, update $D_i(1) = D_i(1) \cup (v_{k+1}, x_{k+1})$ and $D_i(2) = D_i(2) \setminus (v_{k+1}, x_{k+1})$.

3. When at least one of v_k and v_{k+1} is a reflex vertex, then ep is aligned a window of P, assume that ep is on $\ell \in L$. (See Figure 4; assume that r_3 is on the blue point of s_3.)

   (a) If ℓ includes two consecutive reflex vertices v_m, v_{m+1}, where $m \neq k$ (suppose that the nearest one to r_i is v_m), then

      i. If $P_{m+1}(m)$ is contaminated, then do same as 2a.

      ii. Else, do same as 2b.

4. When v_k and v_{k+1} are convex, (See Figure 4; assume that r_4 is on the blue point of s_4.)

   - If r_i is going to start moving from ep (i.e., if $D_i(1) = \emptyset$), then r_i updates $D_i(1) = (v_k, v_{k+1})$ and starts moving along s_i.
   - If r_i reaches the endpoint of s_i (i.e., if $D_i(1) \neq \emptyset$), then $D_i(2)$ is $\emptyset$ and r_i moves back.

$(II.)$ Suppose that ep is **inside** and not on the boundary of P, ep can be collinear with at most 3 reflex vertices (due to general position assumption no four reflex vertices are collinear). According to the number of these reflex vertices do as below.

Assume that ep is **not** collinear with any reflex vertex. Consider the maximal orthogonal line segment normal to s_i at ep and call it l. Let a and b be two endpoints of l. Line segment l partitions P into two sub-polygons. One of them consists of s_i. Therefore, r_i sends a signal to the other robots to clear the sub-polygon that does not include s_i and that is between a and b $\big($ r_i updates $D_i(3)$ depending on its position to $D_i(3) = (a, b)$ or $D_i(3) = (b, a)\big)$. See Figure 4; if r_7 is on the blue point of s_7, then the sub-polygon that is between (a, b) in counterclockwise order should be cleared.

Assume that ep is collinear by **at least one** reflex vertex. So, ep is on a window, say $\ell \in L$.

- If ℓ consists of one reflex vertex v_k (assume that the consecutive vertex of v_k on ℓ is v_{k+1}) and s_i is inside $P_k(k+1)$, then (See Figure 4; assume that r_5 is on the blue point of s_5.)
  - If $P'_k(k+1)$ is contaminated (i.e., $FF_k(4) = false$), then r_i sends a signal to the other robots to clear $P'_k(k+1)$ and updates $D_i(3) = (v_k, y_k)$.
  - Else, $D_i(1) = D_i(1) \cup (v_k, y_k)$ and $D_i(2) = D_i(2) \setminus (v_k, y_k)$.
- if ℓ consists of one reflex vertex v_k and s_i is inside $P'_k(k+1)$, then
  - If $P_k(k+1)$ is contaminated, then do same as 1a.
  - Else, do same as 1b.
- If ℓ consists of two consecutive reflex vertices v_k and v_{k+1} (suppose that the nearest one to ep is v_k) and s_i is inside $P_k(k+1)$, then (See Figure 4; assume that r_6 is on the blue point of s_6.)
  - If $P_{k+1}(k)$ is contaminated, do same as 2a.
  - Else, r_i sends a signal to the other robots to clear $P'_{k+1}(k) \bigcap P'_k(k+1)$ and updates $D_i(3) = (x_{k+1}, y_k)$.
- If ℓ consists of two consecutive reflex vertices v_k and v_{k+1} and s_i is inside $P'_k(k+1)$, then
  - If $P_{k+1}(k)$ is contaminated, do same as 2a
  - If $P_k(k+1)$ is contaminated, do same as 1a
  - If $P_{k+1}(k)$ and $P_k(k+1)$ are cleared, then $D_i(1) = D_i(1) \cup (y_k, x_{k+1})$ and $D_i(2) = D_i(2) \setminus (y_k, x_{k+1})$ (do same as 1b and 2b).

Now, suppose that an event point happens and r_i sees at least one reflex vertex. If there are no two consecutive reflex vertices on ℓ, then r_i continues its movement along s_i. If there are two consecutive reflex vertices on ℓ, do same as I1, I2. See Figure 4; assume that r_1 moves from blue point until point a_1 of s_1.

Sending a Signal Step

Assume that r_i waits and sends a signal to the other robots to clear sub-polygon P_1, which is between a and b in counterclockwise order ($D_i(3) = (a, b)$). Robot r_i can wait whenever it sees a reflex vertex or it reaches an endpoint.

When r_i sends a signal, a robot that can clear some parts of P_1 consisting of a starts clearing. If more than one robot can start clearing, choose one of them arbitrarily. At each time, one robot is clearing the polygon. Suppose that r_j sees a and starts clearing P_1.

Let $l_a(s_j)$ be the orthogonal line segment which passes through a and intersects s_j. Also, let a_j be the intersection of s_j and $l_a(s_j)$. r_j can be inside or outside P_1. r_j starts its movement from a_j on s_j. $D_j(1)$ is the intersection of the boundary of P_1 (that is on the boundary of P) and $l_a(s_j)$. Also, $D_j(2) = D_i(3)$. See Figure 4; suppose $r_i = r_5$ is on purple point and sends a signal to the other robots to clear $P_1 = P_{k-1}(k-2)$ (i.e., $D_5(3) = (y_{k-1}, v_{k-1})$). $r_j = r_7$ is a robot which is outside of P_1. r_7 starts clearing from a_7 towards the right endpoint of s_7 and set $D_7(2) = D_5(3), D_7(1) = (y_{k-1}, v_{k'})$. Here $l_a(s_j)$ is a vertical line segment. $r_j = r_1$ is a robot which is inside of P_1. r_1 starts clearing from a_1 towards the down endpoint of s_1 and set $D_1(2) = D_5(3), D_1(1) = (v_{k-1}, y_{k-1})$. Here $l_a(s_j)$ is a horizontal line segment.

Note that when r_j moves it can clear some parts of P except P_1 but we only consider the cleared parts which is inside P_1.

As mention before, in some cases r_i can wait and send a signal when it sees a reflex vertex v_g and $FF_g(x) = false, x \in \{1, 2, 3, 4\}$. In this case P_1 is the corresponding sub-polygon of $FF_g(x)$. See Figure 4; when $r_i = r_5$ and $FF_{k-1}(1) = false$, r_5 waits until $P_1 = P_{k-1}(k-2)$ becomes cleared. At the time that a robot (supposedly r_u) updates $FF_g(x)$ to $true$, r_u finishes its clearance and updates $D_i(1) = D_i(1) \bigcup D_u(1)$ and $D_u(2) = D_u(2) \setminus D_u(1)$ and $D_i(3) = \emptyset$. Then, r_i continues its movement.

Termination Step

We assume that, initially, all parts of P are contaminated. So, $\forall_{r_i \in R} D_i(1) = \emptyset$ and $\forall 1 \leq i \leq 4$, $FF_j(i) = false$. When (1) there is no waiting robot ($\forall_{r_i \in R} D_i(3) = \emptyset$), (2) all robots have cleared their corresponding sub-polygons ($\forall_{r_i \in R} D_i(2) = \emptyset$), and (3) all parts of P have been cleared ($\bigcup_{i=1}^{|R|} D_i(1) = P$), the motion-planning algorithm is finished. These three conditions should happen simultaneously.

We define a "phase" to be the movement between two consecutive event points. Because of our algorithm, one robot can move and clear some parts of P at any time. In each phasethe cleared parts are increased. When a robot r_i finishes its clearing, it transfers its cleared parts to another robot and its $D_i(2)$ is $\emptyset$. Only the last robot do not transfer anything. When $D_i(1)$ indicates P for any robot and $D_i(2) = D_i(3) = \emptyset$, that robot is the last robot and the algorithm is finished. So, for checking

$\bigcup_{i=1}^{|R|} D_i(1) = P$, we only need to check $D_i(1)$ at each phasein $\mathcal{O}(1)$ time. When $D_i(1) = (v_{k+1}, v_k)$ for any $r_i \in R, v_k \in V(P)$ and $D_i(2) = D_i(3) = \emptyset$, the algorithm is finished (P can be shown by (v_{k+1}, v_k)).

4. Correctness

In this section, we show the correctness of the proposed algorithm. One advantage of the proposed algorithm is that the algorithm is not simple but its correctness is simple. We prove that the proposed algorithm is deadlock free (it is not trapped in a loop). At each time one robot moves. Since S guards all parts of P, then the algorithm will be terminated. Then, we will prove Lem.2 which is another advantage of our algorithm. Starting with any arbitrary sliding robot, the algorithm can clear P completely.

Lemma 1. *The proposed algorithm is deadlock free.*

PROOF. Assume that r_i is waiting for sub-polygon P_i to be cleared by a sequence of robots. Inside P_i, r_j may be waiting for sub-polygon P_j to be cleared. Therefore, there may exist a chain of waiting robots, say, $r_{seq}(i) =< r_j, r_t, \ldots, r_m >$, for clearing P_i. If $r_i \in r_{seq}(i)$, a deadlock occurs and the algorithm will not get terminated. Therefore, we shall show that the relation $r_i \in r_{seq}(i)$ will never become valid.

Owing to the definition of the window and its corresponding sub-polygons, when r_i waits for the clearance of P_i, it cannot see any points of P_i, except its window. Since the sub-polygons corresponding to the other robots of $r_{seq}(i)$ are inside P_i, none of the waiting robots of $r_{seq}(i)$ can wait for r_i. Hence, the algorithm is deadlock free.

Lemma 2. *A simple orthogonal polygon can be completely cleared starting with an arbitrary sliding robot.*

PROOF. Assume that we start with an arbitrary robot r_i. Because of Lemma 1, the proposed algorithm is deadlock free. Moreover, since S guards all parts of P, the termination step will happen. Based on the termination step, the relation $\bigcup_{i=1}^{|S|} D_i(1) = P$ becomes valid; therefore, there is no contaminated point in P and it gets cleared completely.

5. Analysis

In this section, we analysis the total length that all sliding robots move. Let m be the total length of the edges of P. Let R_{out} be the set of the sliding robots which are reported by the algorithm for clearing P (output of the algorithm). We call the set of the line segments which the robots of R_{out}

move on them as S_{out}. So, $S_{out} \subset S$ is the output set of the line segments that sliding robots move along them and clear P. First, we show that there is no recontamination in our process. We then prove that the length of the paths which sliding robots move and clear P is at most $2m + n|e_{max}|$, where $e_{max} = \max_{e_i \in S} e_i$.

Lemma 3. *There is no recontamination in the proposed algorithm.*

PROOF. Due to our algorithm when a robot finishes its clearing it stops at that point. So, it keeps safe its cleared parts. In some cases a robot does the clearance process more than one time. In this cases another robot keeps the cleared parts safe. When r_i stops and waits for the other robot to clear sub-polygon P_1, it keeps safe the cleared parts since that time. According to Lem.1 the algorithm is deadlock (i.e., there is no loop in the sequence of the waited robots). So, r_i will never clear any parts of P_1 and no point of $D_i(1)$ becomes contaminated again. Therefore, there is no recontamination during our algorithm.

Lemma 4. *The total length of the paths which sliding robots move and clear P is at most $2m+n|e_{max}|$.*

PROOF. Let X be the total length of the paths which sliding robots move and clear P. The paths can be partitioned into three sets (Y, Z and W). The set of the movements that robots move and clear some parts of P simultaneously, called Y. The set of the movements in moving back steps, called Z. The set of the movements when a robot receives a signal and moves until reaches its start point, called W. Due to the algorithm the cleared parts of P always increase and never decrease. Only when a robot moves back to transfer its information to its corresponding waited robot, it does not increase the cleared parts. As Lemma 3 and this fact that the cleared parts always increase (except moving back process), the algorithm clears each part of P once. So, for clearing P its sufficient to clear boundary of P once. Therefore, $Y \leq m$. As the paths where a robot r_i moves and clears some part and the paths where r_i moves back are equal, $Z \leq m$. As mentioned before the algorithm clear boundary of P once. So, in the worst-case for clearing each vertex of P one robot should be called (send a signal step). In worst-case, at each call a robot r_i should move along s_i and reach its start point. So, the length of the movement at each call is at most $|s_i|$. As e_{max} is the maximum length line segment in S_{out}, $|s_i| \leq |e_{max}|$. So, for each call a robot moves at most $|e_{max}|$ and for all calls ($\mathcal{O}(n)$ vertex), we have $W \leq n|e_{max}|$. Therefore, $X \leq 2m + n|e_{max}|$. The worst-case number of calls is shown in Fig. 5. So, the tight upper bound for the number of calls is $\mathcal{O}(n)$.

Corollary 5. *If S is the set of minimum cardinality sliding cameras that guard the whole P, then our algorithm clears P with the minimum number of sliding robots (considering that the robots should move along line segments of S).*

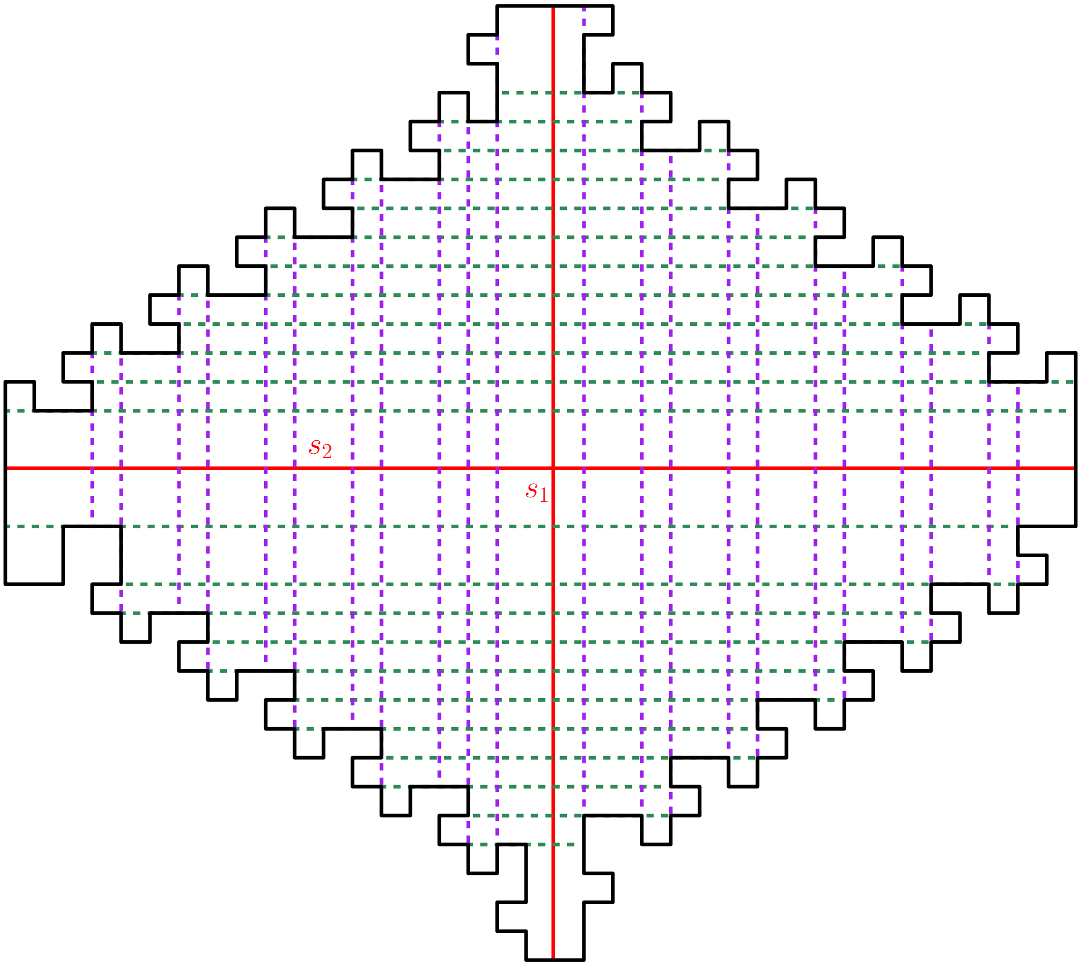

Figure 5: For clearing the polygon a sliding robot s_1 (or s_2) should be called $\mathcal{O}(n)$ times.

6. Implementation

The algorithm has been implemented in Java processing language using the Apple MF840 PC with processor 2.7 GHz, Intel Cori5 and Ram 8 GB. In the implementation, we assume that the pursuers can move along the given line segments which are placed in an environment that is bounded by a simple orthogonal polygon. All motions are determined using information only from the reflex vertices or the other visible robots. The algorithm successfully computed results for several examples. One example is shown in Figure 6. As lack of space, we do not present some similar steps in Figure 6. See the animations of applying algorithm on some examples and also, the report of the total length traversed by the robots using the algorithm, the number of Decision steps, and the number of calling other robots in a table, in `https://www.dropbox.com/sh/wskztugum1s411u/AAApfhkIjC1WkjRxOr_9xlPAa?dl=0`.

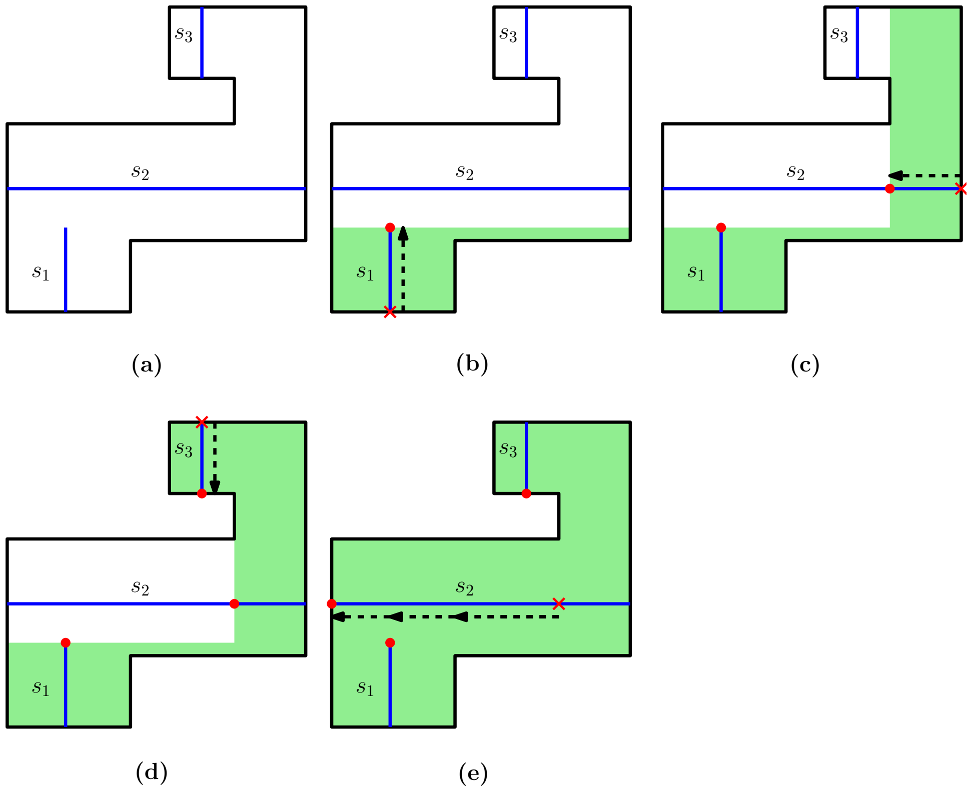

Figure 6: The robot's path, start-point and end-point of each step are shown by dashed arrow, cross and circle, respectively.

7. Conclusion

When the environment is known for the sliding robots, we propose an algorithm for planning the motions of a group of sliding robots to detect all the unpredictable moving evaders with bounded speed ($\neq \infty$). We use a set of line segments S where the sliding robots move along them. In the case where S is a set of minimum-cardinality sliding cameras that guard P, the proposed algorithm uses the minimum number of sliding robots to clear P.

As an open problem, we can consider a case where the environment is unknown to the robots, and the robots can only plan their motions based on the local visible area. If the robots send the information to those that are visible to them, will be challenging problem in practice.

8. Acknowledgment

We would like to thank Ali Narenji for his helpful partnership and comments. Also, we would like to thank Ali Behrouz and MohammadReza Karegar for implementing the proposed algorithm.

References

[1] T. D. Parsons, Pursuit-evasion in a graph, in: Theory and applications of graphs, Springer, 1978, pp. 426–441.

[2] S. M. LaValle, D. Lin, L. J. Guibas, J.-C. Latombe, R. Motwani, Finding an unpredictable target in a workspace with obstacles, in: Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on, Vol. 1, IEEE, 1997, pp. 737–742.

[3] M. J. Katz, G. Morgenstern, Guarding orthogonal art galleries with sliding cameras, International Journal of Computational Geometry & Applications 21 (02) (2011) 241–250.

[4] S. Durocher, S. Mehrabi, Guarding orthogonal art galleries using sliding cameras: algorithmic and hardness results, in: Mathematical Foundations of Computer Science 2013, Springer, 2013, pp. 314–324.

[5] M. de Berg, S. Durocher, S. Mehrabi, Guarding monotone art galleries with sliding cameras in linear time, in: Combinatorial Optimization and Applications, Springer, 2014, pp. 113–125.

[6] I. Suzuki, M. Yamashita, Searching for a mobile intruder in a polygonal region, SIAM Journal on computing 21 (5) (1992) 863–888.

[7] J. W. Durham, A. Franchi, F. Bullo, Distributed pursuit-evasion without mapping or global localization via local frontiers, Autonomous Robots 32 (1) (2012) 81–95.

[8] J. O'rourke, Art gallery theorems and algorithms, Vol. 57, Oxford University Press Oxford, 1987.

[9] J. Urrutia, et al., Art gallery and illumination problems, Handbook of computational geometry 1 (1) (2000) 973–1027.

[10] F. Hoffmann, On the rectilinear art gallery problem, Springer, 1990.

[11] D. Schuchardt, H.-D. Hecker, Two np-hard art-gallery problems for ortho-polygons, Mathematical Logic Quarterly 41 (2) (1995) 261–267.

[12] D.-T. Lee, A. K. Lin, Computational complexity of art gallery problems, Information Theory, IEEE Transactions on 32 (2) (1986) 276–282.

[13] R. Motwani, A. Raghunathan, H. Saran, Covering orthogonal polygons with star polygons: The perfect graph approach, in: Proceedings of the fourth annual symposium on Computational geometry, ACM, 1988, pp. 211–223.

[14] C. Worman, J. M. Keil, Polygon decomposition and the orthogonal art gallery problem, International Journal of Computational Geometry & Applications 17 (02) (2007) 105–138.

Appendix

The assumption of the general position is very important for our algorithm. As defined before, in general position no four reflex vertices are collinear. If we omit this assumption, each line ℓ can have many pairs of consecutive reflex vertices. Then in the steps of our algorithm we should have a while loop for checking the cleared areas, which increases the running time of the algorithm. For example in Fig.7, assume that s_1 starts moving from a. When it reaches b, it should check the clearance of all gray sub-polygons. As $\mathcal{O}(n)$ reflex vertices can be on a line, s_1 should check $\mathcal{O}(n)$ sub-polygons. Our algorithm can not support these examples.

Our algorithm can be modified to handle this situation. How ever this will increase the space and time complexity. This can be done by storing the clearance of the sub-polygons using the floating storage. Therefore, we consider the general position for the number of collinear reflex vertices.

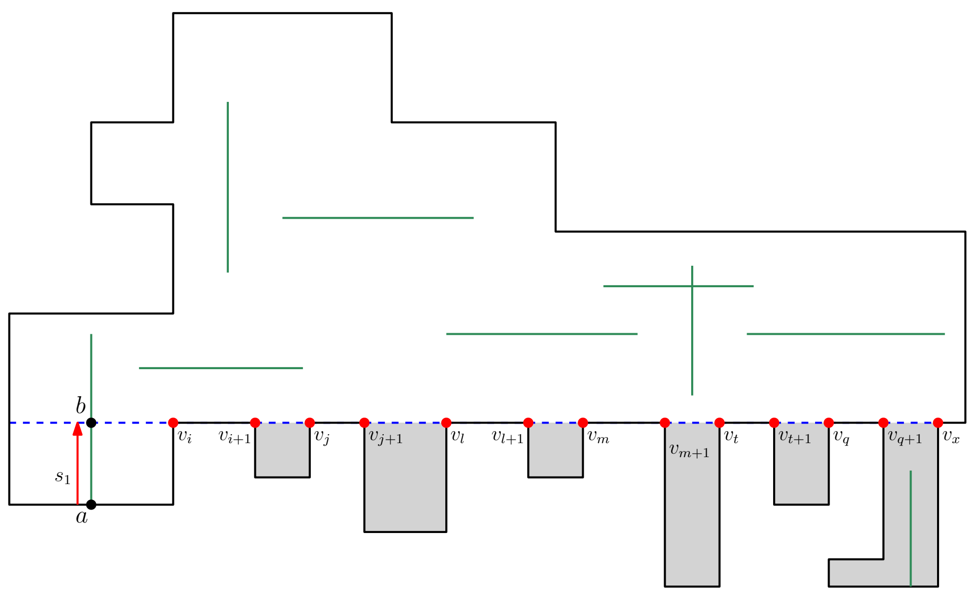

Figure 7: In this example more than three reflex vertices are collinear.