Simple robots with minimal sensing

From local visibility to global geometry

Simple Robots with Minimal Sensing: From Local Visibility to Global Geometry

ETH Technical Report 547

Subhash Suri*
Department of Computer Science
University of California
Santa Barbara, USA 95106

Elias Vicari
Institute of Theoretical Computer Science,
ETH Zurich,
8092 Zurich, Switzerland

Peter Widmayer
Institute of Theoretical Computer Science,
ETH Zurich,
8092 Zurich, Switzerland

February 8, 2007

Abstract

We consider problems of geometric exploration and self-deployment for simple robots that can only sense the combinatorial (non-metric) features of their surroundings. Even with such a limited sensing, we show that robots can achieve complex geometric reasoning and perform many non-trivial tasks. Specifically, we show that one robot equipped with a single pebble can decide whether the workspace environment is a simply-connected polygon or not; with sufficiently many pebbles, it can also count the number of holes in the environment. Highlighting the subtleties of our sensing model, we show that a robot can decide whether the environment is a convex polygon, yet it cannot resolve whether a particular vertex is convex. Finally, we show that using such local and minimal sensing, a robot can compute a proper triangulation of a polygon, and that the triangulation algorithm can be implemented collaboratively by a group of m such robots, each with $\Theta(n/m)$ memory. As a corollary of the triangulation algorithm, we derive a *distributed* analog of the well-known Art Gallery Theorem [1]: a group of $\lfloor n/3 \rfloor$ (bounded memory) robots in our minimal sensing model can self-deploy to achieve visibility coverage of an n-vertex art gallery (polygon). This resolves an open question raised recently in [4].

1 Introduction

The study of simple robot systems, with minimal sensory input, is of fundamental interest in both theory and practice. In theory, a minimalistic model provides a clean conceptual framework for performance analysis and lends insights into the inherent complexity of various tasks: the positive results identify the easy problems, while the negative results help isolate the difficult problems that necessitate richer functionality and sensing. Models with simple sensing are also robust against noise and help simplify the information (belief) space of the robot systems, whose complexity is often a source of much difficulty in planning [7]. On the practical side as well, robots with a simple sensing architecture have many advantages: they are inexpensive, less susceptible to failure, robust against sensing uncertainty, and useful in a broad set of applications. With the emergence of *wireless sensor networks* [8], a group of simple micro-robots also offers an attractive and scalable architecture for large-scale collaborative exploration of

*Work done while the author was a visiting scholar at the Institute of Theoretical Computer Science, ETH, Zurich.

unknown environments.

Motivated by these considerations, we investigate several fundamental computational geometry problems in the context of simple robots, with minimal sensing ability. In particular, we assume that the robot is a (moving) point, equipped with a simple camera that allows the robot to sense just the *combinatorial features* in its field—these features are *unlabeled*, however, meaning that the sensor can detect a vertex of the polygon, but all vertices look the same to it. The features are represented as a binary vector, called the *combinatorial visibility vector* and denoted cvv. This vector is defined by the ordered list of vertices visible from the robot's current position, and the binary bits indicate whether the consecutive vertices form an edge of the environment or join two non-neighbor vertices (i.e. hides a portion of the polygon not seen from the current location). Figure 1 shows an example of a combinatorial visibility vector. The robot has no other sensing capability, and thus receives no information about distances, angles, or world coordinates. The robot, however, does possess "handedness," meaning it can decide clockwise or counterclockwise order. We assume idealized sensing throughout, since our primary goal is to understand theoretical limits of our model, and our algorithms are deterministic and analyzed in the worst-case model.

We are interested in discovering what non-trivial or complex tasks such a robot can accomplish. Since the robot's information is highly local, we are particularly interested in tasks that are seemingly *global*. For instance, can one or more robots decide whether a (unknown) polygonal environment (the robots' workspace) is *simply connected*, or does it have holes? We show that a single robot with a single pebble is able to decide the simplicity of the workspace; equivalently, a secondary passive robot can play the role of the pebble. Generalizing this, we show that if the workspace is a polygonal environment with k holes, then a single robot with $k+1$ pebbles can detect all the holes.

Interestingly, in our minimalistic model of sensing, there are simple topological questions that seem undecidable. For instance, while it is quite easy for a single robot to decide whether the workspace is a convex polygon, it is not possible to decide whether a *particular vertex* is convex or reflex! Similarly, a robot cannot decide which is the *outer boundary* of the multiply connected polygonal workspace, although it can discover and count all the components in the environment.

Finally, we show that using such local and minimal sensing, a robot can compute a proper triangulation of a polygon. Furthermore, our triangulation algorithm can be implemented distributively by a group of m robots, each with $\Theta(n/m)$ memory. As a corollary of the triangulation algorithm, we derive a distributed analog of the well-known Art Gallery Theorem [1]: a group of $\lfloor n/3 \rfloor$ robots can self-deploy to guard a simple polygon. This improves the recent result of [4], where they showed that $\lfloor n/2 \rfloor$ distributed guards can achieve the coverage, raising the tantalizing question whether distributed nature of the problem is the source of the gap between the centralized optimum of $\lfloor n/3 \rfloor$ and their bound. Our result shows that even with minimal sensing and distributed robots, $\lfloor n/3 \rfloor$ guards suffice. Indeed, triangulation is a fundamental data structure, used as a building block of many geometric algorithms, and so we expect that this basic result will find other applications as well.

1.1 Related Work

Combinatorial geometric reasoning is key to many motion planning and exploration tasks in robotics [6, 7]. Our work is similar in spirit to that of [10, 11], in that we aim to explore the power and limitations of a minimal model of robot systems. However, there are key differences both in the abstraction—their sensing model assumes that important features of the environment are uniquely *labeled*, allowing sensors to distinguish these *landmarks*; our sensing model is more basic than this and works with unlabeled and indistinguishable features—and the nature of problems addressed—the main focus of previous work [5, 9, 11] is navigation and pursuit evasion, while we are concerned with geometric and topological structure of the environment and collaborative and distributed self-deployment, which do not seem to have been addressed in the past.

2 A Minimal Sensing Model

We consider robot systems with a simple model of "visual" sensing. At any position p, the sensory input of the robot is a *combinatorial visibility vector* $\mathrm{cvv}(p)$, which is a vector of zeroes and ones. This vector is defined by the vertices of the polygonal environment that are visible from the point p, and the binary bits encode whether or not the consecutive vertices form an edge of the polygon. In other words, the visibility vector tells the robot the number of vertices visible to it, and a cyclic order of the *edge types* (boundary or diagonal) defined by consecutive vertices. Fig. 1 shows an example, with the combinatorial visibility vector of vertex p.

We emphasize that the visibility vector is different from the visibility polygon—the former only uses the vertices of the original polygon, and is typically a strict subset of the visibility polygon. We believe that the visibility vector, despite being less informative than the corresponding visibility polygon, is better suited for our simple sensing model: in our coordinate-free, binary sensing model, there is no obvious way to represent or communicate entities other than polygon vertices or edges. Even polygon vertices and edges have no "names" or labels, and as such they can only be described in relative terms: e.g., ith counterclockwise vertex in the visibility vector of vertex q. In this regard, our sensing model is more basic and weaker than the minimalism assumed by [10, 11], who require presence of labeled features and distinguishable landmarks in the environment.

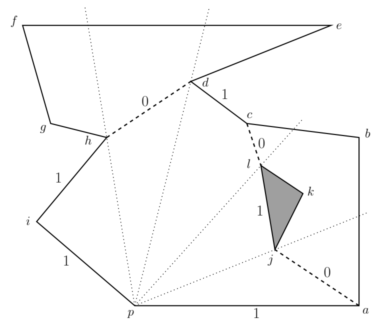

Figure 1: Illustration of the combinatorial visibility vector. In counterclockwise order, the vertices visible from p are p, a, j, l, c, d, h, i, and its visibility vector is $\mathrm{cvv}(p) = (1, 0, 1, 0, 1, 0, 1, 1)$.

In terms of the robot's motion abilities, we only assume that (1) it can sense when it is on a boundary, (2) it can move clockwise or counterclockwise along a boundary, and (3) at a position p, it can choose to move towards a vertex visible from position p. For the workspace environment, we assume that it is a polygon (simply or multiply-connected), with an unknown geometry and an unknown number of vertices.

3 Convexity of the Environment

We begin with a simple example highlighting the severe limitations of our sensing model: a robot in our model cannot decide whether a particular vertex of the environment is convex or reflex. Consider two polygons shown in Figure 2 (i) and (ii), where the combinatorial visibility vectors of the corresponding vertices are identical in the two cases. For instance, $\mathrm{cvv}(a) = (1, 0, 1)$, and $\mathrm{cvv}(b) = (1, 0, 1, 1, 1)$. The key observation is that the two vertices, c and g, both have the same visibility vectors, namely, $\mathrm{cvv}(c) = \mathrm{cvv}(g) = (1, 0, 1, 1, 0, 1)$, and yet one is convex and the other is reflex. Thus, an algorithm cannot decide whether c is convex: indeed, since the types of c and g are flipped in Figures (i) and (ii), any algorithm that distinguishes between them will be incorrect for at least one of these two cases. (Alternatively, one could also argue that in Fig. 2(i), h and d have identical vectors, but have different types.)

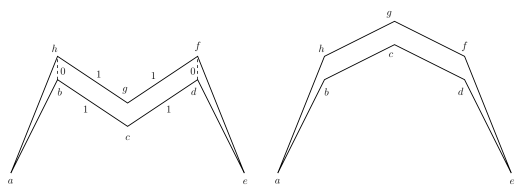

Figure 2: An example showing that a robot in our model cannot resolve whether a vertex is convex or not. The sensing information (combinatorial visibility vectors) for the corresponding vertices in the two polygons is identical, yet c and g have different types (convex, reflex) in the two cases.

This may seem surprising in light of the fact that such a robot *can* decide whether the entire polygon is convex or not. Observe that a necessary condition for the polygon to be convex is that the visibility vector of any vertex v must be all 1's—the presence of a 0 bit implies that there exists a "pocket" in the polygon not seen by v. If the visibility vectors of *all vertices* are all 1's, then the polygon is convex—this follows because every nonconvex polygon must contain a reflex vertex, and if r is reflex vertex them the visibility vector of the (either) neighbor of r cannot be all 1's. Thus, an algorithm to decide the convexity of a polygon P is the following.

Start at any vertex, say, v_0, and consider the visibility vector $\mathrm{cvv}(v_0)$. If this vector is not all 1's, stop—the polygon is ostensibly nonconvex. Otherwise, compute the size of the polygon, say, n by counting the number of 1's in the vector. Repeat the convexity test from each of the remaining $n-1$ vertices, by moving clockwise. If the visibility vectors of all the vertices are all 1's, then the polygon is convex.

We, thus, have the following theorem.

Theorem 1 *A single robot in our binary sensing model can decide whether a given polygon (workspace) is convex or not. However, it is not possible to decide if a particular vertex is convex.*

4 Topological Structure of the Environment

Next, we consider another fundamental task related to the geometry of the environment: is the workspace of the robot simply-connected, or does it have holes? If the polygon is multiply-connected, then how many holes does it have? We first show that one robot, equipped with a single pebble, can decide the simplicity question. The pebble, of course, can be replaced by another (passive) robot. Subsequently, we show that with enough pebbles (or passive robots), the robot can also identify and count the number of topological holes in the workspace.

4.1 Is the polygon simply-connected?

In order to decide the topological simplicity of its environment, the robot has to determine whether the 1 edges in its combinatorial visibility vector belong to two (or more) different component boundaries. As an example, how does the robot decide when located at vertex p in Figure 1, whether the edge (l, j) is part of a "hole" or does the outer boundary connect to it in the back (near vertex k, which is invisible to p)? We begin with a simple geometric fact that plays an important role in our algorithm.

Lemma 1 *Suppose P is a multiply-connected polygon, and C is a cycle representing one of the components. Then, there always exists a vertex $u \in C$ that sees a vertex $v \in P \setminus C$. That is, there must be a vertex $u \in C$ whose visibility vector includes a vertex from a different component of P.*

Proof: The proof follows from the fact that any polygonal domain admits a triangulation (using only diagonals that connect vertices). Such a triangulation is a connected graph, and so the boundary C of any component includes at least one diagonal in the triangulation that connects it to another component. ▮

We utilize this lemma to decide simplicity of the workspace as follows: the robot moves to a boundary, and traverses it in cyclic order; at every vertex, it checks each vertex of its visibility vector to see if it lie on the same boundary or not. By the preceding lemma, if the polygon has a hole, then one of these tests will necessarily fail. The following pseudo-code describes our algorithm. The step that checks whether a vertex v is on the same boundary as vertex u is done through a function EXPLORE, which is described right after the main algorithm.

```
SIMPLICITY $(P)$

1. Initially, the robot moves to an arbitrary vertex of the boundary.

2. The robot marks that vertex with a pebble, and walks along the boundary
   (finishing when it returns to the pebble), say in counterclockwise order,
   to determine the number of edges on this boundary. Let this number be $n_0$.

3. The robot now begins the main phase of the algorithm. The vertex with the
   pebble currently on it represents the vertex being scanned. Let $u_0$ be the
   initial vertex with the pebble. Repeat the following steps $n_0$ times.

    (a) Let $u_i$ be the vertex with the pebble on it, and let $\mathrm{cvv}(u_i)$
        be the visibility vector of $u_i$. Perform EXPLORE $(u_i, v_j, n_0)$, for
        each vertex $v_j$ visible from $u_i$. (The robot moves to vertex $v_j$,
        performs the EXPLORE operation, and then returns to $u_i$.)
    (b) If any call to EXPLORE returns $simple = false$, then terminate; the
        polygon is multiply-connected. Otherwise, once all the vertices visible
        from $u_i$ have been explored, advance the pebble from $u_i$ to $u_{i+1}$.

EXPLORE $(u, v, n)$

1. Move clockwise from $v$ along the component boundary containing $v$.

2. If a pebble is encountered within $n$ steps, set $simple = true$; and stop
   (the robot is back at vertex $u_i$).
```

3. Otherwise, set $simple = false$; retrace n steps backwards (returning to vertex v), and move to the vertex u (identified with a pebble).

For instance, in Figure 1, the call EXPLORE $(p, a, 10)$ returns true, while EXPLORE $(p, j, 10)$ would return false. We summarize this result in the following theorem.

Theorem 2 *A single robot with a pebble can decide if a polygonal environment is simply connected or not.*

The processing time of the algorithm can be improved in several ways, though that's not our main intent here. For instance, instead of checking all vertices of the combinatorial visibility vector, it suffices to limit the EXPLORE to only the endpoint of the 0-edges. We next show how a robot can discover all the components (holes) in its workspace environment.

4.2 Counting the number of holes

The algorithm is a recursive extension of the simplicity algorithm: when the robot discovers a new hole, it recursively scans its boundary to discover new holes. The pebbles are used to mark the holes that have been already discovered. The key difference from the simplicity algorithm is that the robot needs to maintain state information to go back in the recursion. The following pseudo-code describes the algorithm.

```
COMPONENTS $(P)$

1. The robot starts on the boundary of an arbitrary component. It marks this component with one pebble, computes its size (number of edges), by walking around the boundary, using the pebble as a marker. Let $n_0$ be the size of the current component.

2. For the component being scanned, the robot maintains the index (in counterclockwise order, from the starting vertex) of the current vertex. Let $u_i$ be the current vertex. For each vertex $v_j \in \mathrm{cvv}(u_i)$, say, in counterclockwise order, the robot invokes EXPLORE$(u_i, v_j, n_0)$.

3. If EXPLORE$(u_i, v_j, n_0)$ returns true, then scan of the current component continues.

4. If EXPLORE$(u_i, v_j, n_0)$ returns false, then we have discovered a new component.

    (a) The robot then uses a new pebble to mark this component; increments the component counter; computes the size, $n'$, of the new component, and sets $n_0 = \max\{n_0, n'\}$, as the maximum length of the walk in EXPLORE.

    (b) The robot saves state for the old component (containing the vertex $u_i$), along with how to return to $u_i$ from $v_j$ (using pebbles),[1] and then recursively works on the new component.
```

Theorem 3 *If the boundary of the polygonal environment consists of k components, then a robot with k pebbles can discover all the components.*

In our algorithm, we assumed that the robot has $O(k)$ memory, where k is the number of components, to save the state of the recursion. Alternatively, the same result can be obtained with $O(k)$ robots, each with constant (word) memory.

4.3 Which is the outer boundary?

Finally, as another example of a simple task that is difficult in our minimal model, we argue that even though a robot can discover all the components (holes) in the environment, it cannot decide *which* one is the outermost boundary. Consider the centrally symmetric polygon shown in Figure 3. We observe that every vertex in this polygon has the same combinatorial visibility vectors, namely, $(1, 0, 1, 1, 0, 1)$, irrespective of whether it is on the outer cycle or the inner cycle. Because the robot cannot sense or measure angles or distances, its world view looks the same whether it is on the outer cycle or the inner one, implying that it cannot resolve between those two boundaries.

In the next section, we show a single robot, using a constant number of pebbles, is able to compute a proper

[1] Recall that the robot is able to identify v_j from the visibility vector of u_i, but not vice versa. In order to identify u_i from v_j, the robot must place a "distinct" pebble at u_i, move to v_j, and then record the index of u_i in the $\mathrm{cvv}(v_j)$. This pebble must be distinct in the sense that robot can distinguish it from any other pebbles that are marking other components. One can achieve this in several way: by using two pebbles, instead of one, if the robot is able to make this distinction; or using a special pebble that is used for this signaling.

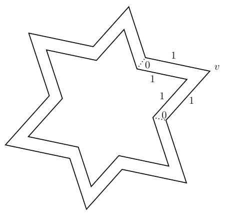

Figure 3: An example showing that a robot in our model cannot resolve which is the outer boundary.

triangulation of a simple polygon. This result, in addition to providing a important data structure for geometric reasoning and exploration, is also the basis for solving the art gallery problem in our sensing model.

5 Polygon Triangulation

In this section, we describe our algorithm for triangulating a simple polygon, which is the key step in solving the art gallery problem as well as a geometric structure with broad applicability [2]. We describe our algorithm for a single robot with a constant number of pebbles in the minimal model, with the assumption that this robot has $\Theta(n)$ memory to store the triangulation. Later we show that the same result can be obtained by a collaborative group of m robots, each with $\Theta(n/m)$ memory. As a corollary of this distributed triangulation, we obtain that a group of $\lfloor n/3 \rfloor$ robots, each with $\Theta(1)$ memory, can collectively build the triangulation and solve the art gallery problem.

Our polygon triangulation algorithm is recursive in nature, and works as follows. See Figure 4. The robot places an initial pebble at a vertex, call it v_0. The combinatorial visibility vector of v_0, $\mathrm{cvv}(v_0)$, consists of a sequence of 0-edges intermixed with 1-edges. Each 0-edge is a diagonal that separates a "pocket" from v_0, and the pockets defined by different 0-edges are pairwise disjoint. The robot will *recursively* triangulate the pockets formed by the 0-edges (say, by visiting them in counterclockwise order), and then complete the triangulation by drawing diagonals from v_0 to all the vertices in its visibility vector. Thus, in high-level pseudo-code, the triangulation algorithm can be described as follows:

TRIANGULATION (P)

1. The robot begins at an arbitrary vertex v_0. Let $e_1, e_2, \ldots, e_k$ denote the 0-edges in the combinatorial visibility vector of v_0 (in ccw order), and let P_i denote the pocket of the polygon defined by the edge e_i.
2. The robot recursively computes the triangulation of P_i, for $i = 1, 2, \ldots, k$.
3. The robot finishes the triangulation by adding diagonals from v_0 to all the vertices in its combinatorial visibility vector (endpoints of both 0 and 1-edges).

Turning this high level description into a correct algorithm that fits in our minimal sensing model, however, requires several careful steps. We use the illustration of Figure 4 to explain the key steps of the algorithm.

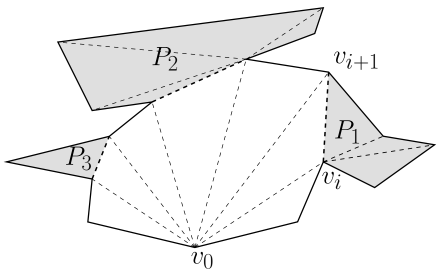

Figure 4: The triangulation algorithm. Starting at v_0, the algorithm triangulates the pockets P_1, P_2, P_3, in that order, and finally completes the triangulation by drawing edges from v_0 to all vertices visible from it.

At the top level of the recursion, the base visibility vector is defined for vertex v_0. The robot consistently scans the edges of a visibility vector in counterclockwise order. Let the first 0-edge in this visibility vector be $v_i v_{i+1}$, and let P_i denote the pocket (subpolygon) defined by this edge. The vertices v_i and v_{i+1} are *identified* by the robot as the ith and the $(i+1)$st vertices in $\mathrm{cvv}(v_0)$. However, as the robot enters the pocket P_i for recursive triangulation, it no longer "sees" the polygon from v_0, and needs a way to distinguish (identify) v_{i+1} from the "new base" v_i. The robot does this through the use of a pebble as follows.

Lemma 2 *Using 2 pebbles, the robot can compute the indices k and j such that the vertices $v_0, v_{i+1} \in$ $\mathrm{cvv}(v_0)$ are the kth and jth vertex, respectively, in the combinatorial visibility vector* $\mathrm{cvv}(v_i)$.

Proof: From v_0, which is marked by the first pebble, the robot heads straight towards v_{i+1}, drops a second pebble there, and returns to v_0. It then, heads to v_i, and identifies v_0 and v_{i+1} as the first[2] counterclockwise vertex in $\mathrm{cvv}(v_i)$ that has a pebble. The second vertex marked with a pebble is v_0. The robot stores these indices k and j as the identity of v_0 and v_{i+1} when based at v_i, respectively. Having identified the relevant nodes in the local visibility of v_i, the robot can then remove the pebble from v_{i+1}. This ensures that we only use at most two pebbles throughout the algorithm. ▮

Once the robot can identify v_{i+1} from its local view, it knows the extent of the pocket P_i: the edge $v_i v_{i+1}$ marks the end of the pocket and the recursive call to the triangulation.

Secondly, by always visiting the pockets in a cyclic order, the robot can consistently compute a "vertex labeling" that serves to identify and store the triangulation globally. In particular, while at position v_0, the robot can see all the vertices in its visibility vector, that "view" is entirely local—the jth ccw vertex in $\mathrm{cvv}(v_0)$ has no meaning to the robot when it is located at another vertex v_k. Thus, during the triangulation algorithm, the robot computes a global labeling, which is a cyclic ordering of the vertices in P, starting from the base vertex v_0. This labeling is computed easily as follows. The robot assigns the label 0 to the vertex v_0; that is, $\ell(v_0) = 0$. It then assigns increasing labels to all the vertices in $\mathrm{cvv}(v_0)$ until it comes to the first 0-edge, say, $e_i = (v_i, v_{i+1})$. As the robot recursively computes the triangulation of P_i, it assigned labels in the pocket, starting with $\ell(v_i)$, and ending with the label $\ell(v_{i+1})$. At this point, the robot continues the labeling in $\mathrm{cvv}(v_0)$ until the next 0-edge, and so on. In the end, the triangulation is stored in the robot's memory as a collection of diagonals, where diagonal (i, j) means the presence of a triangulation edge between the vertices are have indices i and j in the ccw walk along the polygon starting at v_0.

So far we have described the triangulation algorithm using a single robot with $\Theta(n)$ memory (necessary to stored the triangulation). But it is easy to convert this into a distributed implementation, using a group of m robots, each with $\Theta(n/m)$ memory. In the distributed model, we only assume that each robot has a unique ID and that there is communication protocol that permits leader election as well as communication between the leader and any slave.

[2]When at v_i, the robot uses the first edge incident to v_i in the pocket P_i as the beginning of its visibility vector.

Theorem 4 *In our minimal sensing model, one robot with two pebbles can compute a triangulation of a simple polygon. The algorithm is easily turned into a distributed implementation using m robots, each with $\Theta(n/m)$ memory.*

6 The Art Gallery Problem

The well-known Art Gallery Theorem [1, 3] asserts that every n-vertex polygon can be "guarded" by placing $\lfloor n/3 \rfloor$ guards at vertices of the polygon, and this bound is the best possible in the worst-case. The classical setting of the art gallery theorem assumes full knowledge of the polygons, including vertex coordinates. In a recent paper, [4] showed that, given an unknown polygon, $\lfloor n/2 \rfloor$ mobile guards, each with only local views, can "self-deploy" at vertices to achieve the art gallery coverage. Their result raises the interesting question whether this gap is inherently due to the lack of global geometry. We show that this is not so, and in fact $\lfloor n/3 \rfloor$ guards in our minimal sensing model can self-deploy to guard the polygon. Unlike [4], our algorithm is based on triangulation, mimicking the original proof of Fisk [3].

The proof of [3] uses the fact that a triangulation can be 3-colored: three colors can be assigned to vertices so that no diagonal has same color at both endpoints. Then, placing the guards at the *least frequent* color vertices solves the art gallery problem: there are n vertices, so the least frequent color occurs at most $\lfloor n/3 \rfloor$ times, and since each triangle must have all three colors, every triangle is visible to some guard. Thus, to solve the art gallery problem in our sensing model, we just need to show that the triangulation computed in the previous section can be 3-colored by our robots.

Lemma 3 *In our minimal sensing model, a robot can 3-color the triangulation of a polygon.*

Proof: See Figure 5 for illustration. The robot begins by coloring the initial vertex v_0, from which the triangulation began, as 1. It then colors all the vertices in $\mathrm{cvv}(v_0)$ alternately as 2 and 3. Thus, each 0-edge in the visibility vector of v_0 is colored $(2, 3)$. The robot now revisits the pockets P_i in the same order as in the triangulation step, and propagates the coloring. If the pocket P_i was triangulated by the robot from the vertex v_i, and the color of v_i is 2 then the diagonals incident to v_i can be colored by alternating between colors 3 and 1, and so on. It is easy to see that this coloring succeeds. ▮

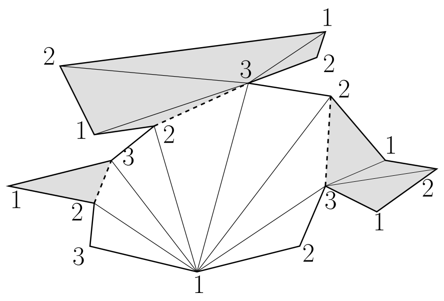

Figure 5: 3-coloring the triangulation.

Once colors have been assigned, and recorded by the robot, it can make one final tour of the polygon and determine the least frequently used color. Placing guards at those $\lfloor n/3 \rfloor$ vertices solves the art gallery problem. The robots reach their guarding position i by walking along the boundary for $i - 1$ steps.

We can implement this algorithm using a collaborative group of m robots, each with $\Theta(n/m)$ memory, using standard leader election protocols, so that at any point one robot servers the role of our main robot in the single-robot setting, while the remaining ones passively follow and act as storage devices. Thus, these passive robots can collaboratively store the $\Theta(n)$ information recording the triangulation and the coloring.

Theorem 5 *In our minimal sensing model, a single robot with $\Theta(n)$ memory can solve the art gallery problem for an n-vertex polygon. Alternatively, a group of $\lfloor n/3 \rfloor$ robots, each with $\Theta(1)$ memory, can solve the art gallery theorem and self-deploy to guard the polygon.*

7 Conclusions

We considered a simple and minimalistic model of visibility-based robot systems, and showed that despite severe sensory limitations, these robots can accomplish fairly complex tasks, and infer global attributes of their workspace. At the same time, the impossibility of some seemingly simple topological tasks also highlights the limitations of our minimal model. In future work, it will be interesting to explore which other global tasks are possible in this model, and which ones are not. It will also be interesting to study the power of additional simple primitives, such as local *angle sensing*, which circumvents the impossibility of deciding whether a vertex is convex and whether a component is the outermost boundary.

References

[1] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory*, 18:39–41, 1975.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications*. Springer-Verlag, 1997.

[3] S. Fisk. A short proof of Chvátal's watchman theorem. *Journal of Combinatorial Theory*, 24(B):374, 1978.

[4] A. Ganguli, J. Cortes, and F. Bullo. Distributed deployment of asynchronous guards in art galleries. In *Proceedings of the American Control Conference*, pages 1416–1421, June 2006.

[5] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. *IJCGA*, 9(5):471–494, 1999.

[6] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[7] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[8] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, 2000.

[9] 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, 2004.

[10] B. Tovar, L. Freda, and S. M. LaValle. Using a robot to learn geometric information from permutations of landmarks. *Contemporary Mathematics, to appear*.

[11] A. Yershova, B. Tovar, R. Ghrist, and S. M. LaValle. Bitbots: Simple robots solving complex tasks. In *AAAI National Conference on Artificial Intelligence*, 2005.