Exploring Polygonal Environments by Simple Robots with Faulty Combinatorial Vision

Anvesh Komuravelli[1,⋆] and Matúš Mihalák[2]

[1] Department of Comp. Science and Engineering, Indian Institute of Technology Kharagpur, India
`anvesh@cse.iitkgp.ernet.in`
[2] Institute of Theoretical Computer Science, ETH Zurich, Switzerland
`matus.mihalak@inf.ethz.ch`

Abstract. We study robustness issues of basic exploration tasks of simple robots inside a polygon P when sensors provide possibly faulty information about the unlabelled environment P. Ideally, the simple robot we consider is able to sense the number and the order of visible vertices, and can move to any such visible vertex. Additionally, the robot senses whether two visible vertices form an edge of P. We call this sensing a *combinatorial vision.* The robot can use pebbles to mark vertices. If there is a visible vertex with a pebble, the robot knows (senses) the index of this vertex in the list of visible vertices in counterclockwise order. It has been shown [1] that such a simple robot, using one pebble, can virtually label the visible vertices with their global indices, and navigate consistently in P. This allows, for example, to compute the map or a triangulation of P. In this paper we revisit some of these computational tasks in a *faulty* environment, in that we model situations where the sensors "see" two visible vertices as one vertex. In such a situation, we show that a simple robot with one pebble cannot even compute the number of vertices of P. We conjecture (and discuss) that this is neither possible with two pebbles. We then present an algorithm that uses three pebbles of two types, and allows the simple robot to count the vertices of P. Using this algorithm as a subroutine, we present algorithms that reconstruct the map of P, as well as the correct visibility at every vertex of P.

1 Introduction

Nowadays one of the main research areas in microrobotics is the study of simple mobile autonomous robots. The recent technological development made it possible to build small mobile robots with simple sensing and computational capabilities at a very low cost, which has launched an interest in the study of distributed robotic systems – computation with *swarms* of robots, not unlike the computational paradigm of wireless sensor networks (where a lot of simple, small and inexpensive devices are spread in the environment, the

⋆ The work was done while the author was an internship student at ETH Zurich.

devices self-deploy in a working wireless network, gather data from the environment and provide simple computational tasks). Simple robots promise to bring mobile computational capabilities into areas where previous approaches (usually of bulky construction) are not feasible or cost-effective. The main advantages are quick and easy deployment, scalability, and cost-effectiveness. This new concept raises new research problems, as the classical schemes designed for centrally operated, or overwhelmingly equipped robots are inapplicable to the lightweight and/or distributed computational models of simple robots.

In this paper we consider one particular model of simple robots, the so called *simple combinatorial robot.* In this model the robot is modeled as a moving point inside a simple polygon P, and the sensing provides only "combinatorial" information about the surroundings. In particular, the robot does not sense any metric information (such as angles, distances, coordinates, or direction). Also, the robot can only move to visible vertices. Study of simple robots with possibly minimum requirements on the sensed information is an attractive topic both in theory and practice, as minimalistic assumptions provide robots that are less susceptible to failures, they are robust against sensing uncertainty and can be very inexpensive to build. In theory, a minimalistic model allows a worst-case computational analysis and provides insights about complexity of various tasks: the positive results identify the easy problems, while the negative results identify the difficult problem for which a richer functionality and sensing is necessary.

The *simple combinatorial robot* was first defined and studied by Suri et al. [1]. The robot operates inside a polygon P. We denote the set of vertices of the polygon P by $V = \{v_0, v_1, \ldots, v_{n-1}\}$, where two vertices v_i and v_{i+1}, $i \geq 0$, form an edge $e_i = v_i, v_{i+1}$ of P.[1] The robot, initially placed at vertex v_0, can only move to a visible vertex, and while moving, the robot does not sense anything about the environment. When the robot lands at a vertex of P, it senses all visible vertices, but only the presence of vertices – the vertices are unlabelled. The robot senses the vertices in a cyclic order, which is the only way the robot can distinguish the vertices from each other. Thus, a movement operation of the robot is simply of the form "move to the i-th visible vertex". The order of visible vertices is assumed to be counterclockwise (ccw). Additionally, the robot senses whether two visible vertices form a boundary edge of P. Positioned at vertex v, this is modelled by a *combinatorial visibility vector* $\text{cvv}(v) = (c_0, \ldots, c_k)$, which is a binary vector that encodes, given there are $k+1$ visible vertices, whether the i-th and $(i+1)$-th visible vertex, $i = 0, 1, \ldots$, form an edge of P ($c_i = 1$) or not

[1] To avoid notational overhead, we assume all operations on the indices to be modulo the corresponding number (n in this case).

($c_i = 0$). The convention is that the vertex v is visible to itself, and v is the 0-th visible vertex of v. Figure 1 illustrates the concept of cvv's. The robot can use pebbles to mark vertices. If there is a visible vertex with a pebble, the robot also senses the index of this vertex in the list of visible vertices in ccw order. In case the robot uses pebbles of different types, the robot also senses the type of the pebble. Naturally, the goal of computation with pebbles is to use few pebbles and few different types of pebbles.

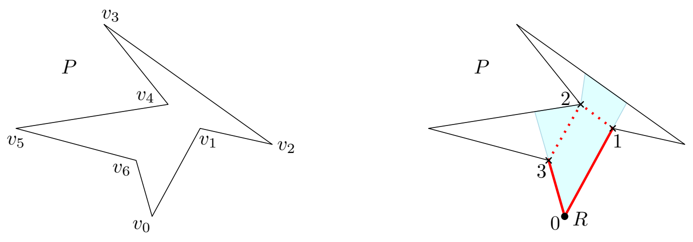


Fig. 1. The left figure depicts a polygon P on vertices $v_0, \ldots, v_6$. On the right figure, a robot R is placed on vertex v_0 of the same polygon. The visible region of the polygon is shaded. The visible vertices (ordered ccw from the robot's position) have only local identifiers $0, 1, 2$, and 3 (no global information) stating their position in the ccw order, and the combinatorial visibility vector of v_0 is $\text{cvv}(v_0) = (1, 0, 0, 1)$, as the visible vertices $0, 1$ form an edge, vertices $1, 2$ form a diagonal, vertices $2, 3$ form a diagonal, and vertices $3, 0$ form an edge of P

To understand capabilities of minimalistic robots, one studies what problems are solvable and which not, i.e., one is interested in the possibility only, and does not primarily aim for the best running time of algorithms. Learning and exploring the environment is a prime problem for any robotic system. The results of Suri et al. [1] show that a simple combinatorial robot without a pebble can decide whether the polygon P is convex. On the other hand, without a pebble the robot cannot count the number of vertices, as shows the result of [2]. Allowing the robot to use one pebble, the robot can virtually label the vertices of P and construct a map of P, i.e., the visibility graph $G = (V_{\text{vis}}, E_{\text{vis}})$ of P, a graph with $V_{\text{vis}} = V$ and with an edge between every two vertices that are visible to each other in P. This then allows the robot to consistently navigate inside the polygon, and, for example, compute a triangulation of P. Computing the visibility graph of P is essentially everything the simple combinatorial robot can do with one pebble, as was shown in [2].

In this paper we study the robustness issues of the simple combinatorial robot in scenarios where the sensing does not provide accurate information.

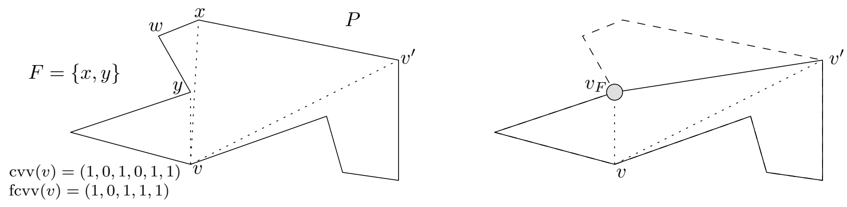
In practice, two vertices visible from vertex v can be "seen" as being very close to each other (e.g., they span a very tiny angle with v). If a very "simple" sensory device is used, these two vertices may wrongly be recognized as a single vertex. This creates a faulty sensing for the robot. In this section we model such situations formally and study conditions in which the simple combinatorial robot can reconstruct the visibility graph of a simply-connected polygon P (the visibility graph of P is often called the *map* of the environment). We show in Section 2 that even counting the number of vertices of P is not possible with one pebble. We conjecture that this is still not possible with two pebbles. We then show that using three pebbles of two different types allows the simple combinatorial robot to count the number of vertices of P. In Section 3 we present an algorithm that allows a simple robot with three pebbles of two different types to compute the visibility graph of P, using the algorithm for counting as the main part. We conclude the paper and outline some future work in Section 4.

Modeling Vertex Faults

For a given simply-connected polygon P on vertices V, a *vertex fault* is a set $F \subsetneq V$, $|F| = 2$. We will sometimes refer to a *vertex fault* simply as a *fault.* A vertex that belongs to a vertex fault is called a *faulty vertex.* We denote by $\mathcal{F}$ a collection of vertex faults, i.e., a set $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$, where every F_i, $i = 1, 2, \ldots, m$, is a vertex fault. We assume that the vertex faults in $\mathcal{F}$ are mutually disjoint, i.e., no vertex belongs to more than one vertex fault.

We define and study the *simple combinatorial robot with vertex faults* (*faulty robot* for short) – a model derived from the simple combinatorial robot that reflects our discussion on unreliable sensing. For a given polygon P and a given set of vertex faults $\mathcal{F}$, a faulty robot sitting at some vertex $v \in V$ senses its surrounding in P via the *faulty combinatorial visibility vector* $\text{fcvv}(v)$ which is defined from the cvv in the following way (consult Fig. 2 for illustration). Let $\text{cvv}(v)$ be the combinatorial visibility vector of vertex v in polygon P. For any two visible vertices x and y, $x, y \neq v$, that belong to the same vertex fault F and that appear consecutively in the "vision" of vertex v (recall that the visible vertices of v are considered in ccw order), we remove from the cvv the information about x and y (i.e., we remove the bit that encodes whether they form an edge or a diagonal in P). Doing so for any such pair of vertices defines the faulty combinatorial visibility vector fcvv of vertex v.

Thus, if vertex v does not see any vertex from a vertex fault, the cvv and the fcvv are the same. Notice also that according to the definition, the robot at vertex v cannot distinguish between vertices x and y from F only if they lie consecutively next to each other (as observed from vertex v). The reason for this is that from different positions the vertices x and y may cause sensing problems, and from others not. Especially, if from some position the vertices x and y do not appear consecutively, i.e., there is at least one vertex w between them, then the robot's sensing can distinguish between x and y. The concept of the fcvv can also be seen as treating the two vertices of a vertex fault as one "virtual" vertex (as observed by a robot), and then defining the fcvv as the cvv with the virtual vertices. In this understanding, the robot thus senses less vertices (than there really are). We will assume that every vertex fault $F = \{u_F, v_F\}$ is visible from a vertex of P, i.e., there is a vertex v in P which sees both u_F and u_F with the correct vision (as otherwise such a vertex fault does not give any faulty vision).



w
x
P
v'
$F = \{x, y\}$
y
$\mathrm{cvv}(v) = (1, 0, 1, 0, 1, 1)$
v
$\mathrm{fcvv}(v) = (1, 0, 1, 1, 1)$
v_F
v'
v

Fig. 2. Illustration of a faulty combinatorial visibility vector. The left figure depicts a polygon P with one vertex fault $F = \{x, y\}$. The vertices x and y appear consecutively in the ccw order as seen from v (the dotted lines are the "vision" lines) and therefore fcvv(v) differs from cvv(v) – the 0 encoding that x and y form a diagonal in P is removed. The right figure depicts an alternative view on fcvv's. The vertex fault $\{x, y\}$ is seen by a robot at v as one virtual vertex v_F, and fcvv(v) is then the cvv of this new (faulty) vision with v_F in it

It remains to specify what happens if a robot decides to move to a virtual vertex v_F. In our model we assume that the robot can land non-deterministically at either vertex of F. We study the worst-case behavior of algorithms, and thus assume an *adversary* that decides where the robot lands.

Finally, if a pebble is left at a faulty vertex of a vertex fault F, a robot that sees the virtual vertex v_F also sees the pebble as being placed at the virtual vertex v_F.

Related Work

The concept of simple, deterministic robots that sense no metric information (distances, angles, coordinates, etc.) is a relatively new research area. The simple combinatorial robot, the model we consider in this paper, was defined and studied in [1]. The robot was shown be able to compute the visibility graph of P using one pebble. A similar approach to minimalism was studied for example by Yershova et al [3]. They study pursuit-evasion problems with a robot that can only sense the type of the current vertex (reflex or convex angle) and can only move along the boundary edges, but can continue in the same direction after reaching a vertex with reflex angle. In these and similar models (see e.g. [4] or [5] for other examples of similar models) the considered sensing is very simple, yet the reliability of such sensing is crucial for the solutions of the studied problems.

A recent, not directly related, but well studied area of fault-tolerance with mobile robots addresses the computation issues with imprecise compasses. In this model, a set of asynchronous autonomous robots are placed in a plane (i.e., not in a polygon) equipped with a sense of direction (and distance) and capability to move an arbitrary distance in an arbitrary direction. An imprecise compass delivers a direction that can deviate from the actual value, but the error is bounded. In this model, mainly the gathering problem was studied [6,7]. Also for the gathering problem, the issue of not obtaining perfectly accurate sensory input, and not having a perfectly accurate movement was studied in [8] for asynchronous robots.

2 Counting the Number of Vertices

In this section we consider the elementary problem of inferring the number of vertices of a polygon P by a faulty robot. We shall see that this problem, being trivial in the fault-free case using one pebble, becomes non-trivial in the presence of faults even with two pebbles. We will show, however, that a robot with three pebbles of two types can compute the number of vertices of P.

2.1 Counting with 1 Pebble

It is illustrative to consider first the case when there are no vertex faults. In such a case the robot simply leaves a pebble on the current vertex and moves around the boundary, always moving to its first visible vertex (which is its “right” neighbor), counting the number of visited vertices, until the robot comes back to a vertex with the pebble. In case of vertex faults this simple strategy does obviously not work. Consider for example a convex polygon on four vertices v_0, v_1, v_2, v_3 and one vertex fault $F = \{v_1, v_2\}$. Assume that the robot initially sits at vertex v_0. The robot drops the pebble to mark v_0 and moves to its right neighbor, which is a virtual vertex v_F. The adversary makes the robot land on v_2. The robot then continues to v_3 and v_0, visiting only three vertices in total. One could probably easily derive a correct algorithm for this simple case, nonetheless we show that in general, using only one pebble, there is no algorithm for the problem of counting the number of vertices in the presence of vertex faults.

Theorem 1. *Any simple robot with one pebble cannot count the number of vertices of a polygon P with vertex faults.*

Proof. Let $\mathcal{A}$ be an arbitrary (deterministic) algorithm for the simple robot with one pebble. We will show that $\mathcal{A}$ cannot count the number of vertices in every polygon P.

Consider polygons P_1 and P_2 in Fig. 3 with a different number of vertices. The left polygon P_1 is a square and the right polygon P_2 is a convex polygon on six vertices. P_1 has no vertex fault, and P_2 has three vertex faults $F_1 = \{v_0, v_1\}$, $F_2 = \{v_2, v_3\}$ and $F_3 = \{v_4, v_5\}$. Thus, if we consider a robot placed at a vertex of the vertex fault $\{v_0, v_1\}$ for example, it can visually distinguish between the vertices v_0 and v_1, but from either of these vertices the robot sees v_2, v_3 as a single virtual vertex, and v_4, v_5 as another virtual vertex. Let us denote by v_{F_1}, v_{F_2}, and v_{F_3} the virtual vertices that correspond to the vertex faults F_1, F_2, and F_3, respectively.

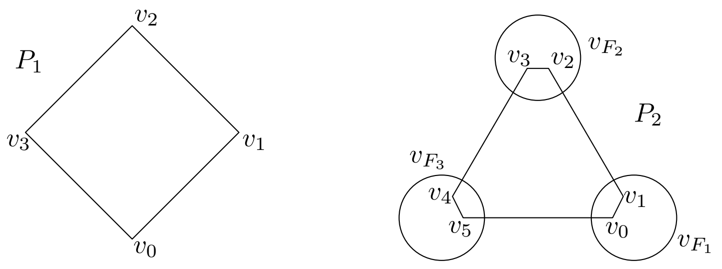


Fig. 3. Polygons used for the proof of Theorem 1

Observe first that a robot has the same view in both polygons, i.e., $\mathrm{fcvv}(v) = (1, 1, 1, 1)$ for any vertex v in both polygons. Thus, if the robot does not use the pebble, it cannot count the number of vertices because if after ℓ moves and observations in P_1 it determines that polygon has four vertices, then the same movements and observations can be made in the second polygon, and thus the deterministic robot has to claim P_2 has four vertices, which is obviously wrong.

Let us consider the situation when a robot executing $\mathcal{A}$ (in both polygons) drops a pebble. As P_1 and P_2 are symmetric we can, without loss of generality, assume the robot drops the pebble at vertex v_0 when run on any of the two polygons. We now show that any movement of a robot executing $\mathcal{A}$ in P_1 can be mimicked in P_2 as well, by appropriate choices (by the adversary) of a vertex the robot lands at, when moving to a virtual vertex v_{F_i}, $i = 1, 2, 3$, such that the observed fcvv's remain the same, together with the position of the pebble therein. If a robot in P_1 moves to its first visible vertex (i.e., to vertex v_1 in our case), then robot in P_2 attempts to move to v_1 as well, and thus the robot in P_2 lands at v_1 as well. Hence, the position of the pebble in both cases is the same – the pebble is on the vertex which is the robot's left neighbor. Similarly, if the robot in P_1 moves to its last visible vertex (i.e., to vertex v_3), then robot in P_2 attempts to move to vertex v_{F_3} and lands at vertex v_5. If the robot in P_1 moves to the second visible vertex (vertex v_2), then the robot in P_2 lands at vertex v_3. It is easy to check that the position of the pebble is the same in both cases. Now (assuming the pebble is still at vertex v_0) for any position of the robot in P_1 and any movement of the robot to a visible vertex, the adversary can make the robot in P_2 mimic the movement by an appropriate choice of landings in P_2. We do not list all possible movements here, but give one more example only. Assume the robot in P_1 at vertex v_2 moves to vertex v_1 and then to vertex v_3. If the robot in P_2 is at vertex v_3, the algorithm $\mathcal{A}$ moves the robot first to vertex v_2, and then attempts to move the robot to vertex v_{F_3}, and lands at vertex v_5 (by the choice of the adversary).

If the robot picks up the pebble in P_1 so can the robot in P_2, as we have maintained the same vision and the position of the pebble is the same for the robots in both polygons.

Thus, as the adversary can force the algorithm $\mathcal{A}$ to produce the same vision sequence in both polygons, the algorithm cannot compute the number of vertices in both polygons. □

2.2 Counting with 2 Pebbles

A natural question is to study the problem using two pebbles. While we do not know whether two pebbles suffice to compute the number of vertices of any polygon P, we outline the difficulties in designing such an algorithm.

Consider a (big) polygon which consists of "triangular cells" as depicted in Fig. 4. The triangular cell can be seen as a triangle whose tips were cut off. For the construction we cut off just a tiny bit so that the resulting two vertices of a loose end have distance ε (ε as small as needed). Also, the two vertices of every end of the cell form a vertex fault. We can glue the triangular cells together as depicted in the figure. Starting from a central triangular cell, we can grow the polygon to an arbitrary size by making the newly glued cells smaller and smaller. To make the construction finite, we just use triangular cells with no open ends. We make the construction such that the two vertices of every vertex fault F appear consecutively in ccw order as seen from any visible vertex, and thus the two vertices will be seen by the robot as a single virtual vertex v_F. For this to achieve, one has to set an appropriate ε and an appropriate angle at which the new cells are glued. For brevity we omit the precise description of the construction. We note that the depiction in Fig. 4 is only schematic. We call the resulting polygon *triangular*. For the moment we assume the polygon is big enough for "anything which follows", while the exact size will naturally become clear at the end of the section.

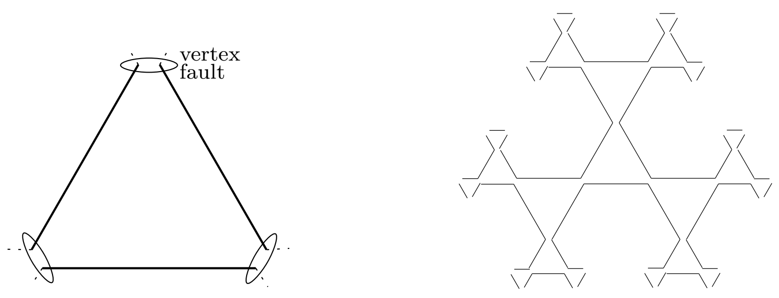


Fig. 4. Left: A "triangular cell" is a triangle with endpoints split into open ends. The two vertices of each open end form a vertex fault. Right: The whole polygon is build from these "triangular cells" by an appropriate rotation and scaling

We first prove a useful lemma that highlights the main technique for the proof of the main result of this subsection.

Lemma 1. *A simple robot with no pebbles can be made to stay within two neighboring cells in any triangular polygon P. Furthermore, if the initial vertex can be chosen by the adversary, the robot can be made to stay within one cell.*

Proof. The main trick is to choose the proper vertex $v \in F$ where the robot lands when it attempts to move to a virtual vertex v_F. We (the adversary) can choose this vertex arbitrarily (i.e., the robot does not notice the difference) as long as the vision from these two vertices is the same. Observe that if the robot is not at the ending triangular cell, the vision is everywhere the same, fcvv $= (1, 1, 1, 1, 1, 1)$. Our choice of the landing vertex will depend on what the robot wants to do after landing in v_F. For the following discussion, consult Fig. 5. Let s denote the vertex where the robot starts. Let e be the vertex for which $\{s, e\}$ is a vertex fault in P. Vertex s is a "gateway" to two neighboring triangular cells A and B, with vertices as depicted in the figure. We show how to make the robot stay in the cells A and B. Assume for example the robot wants to move to its right neighbor (which is the virtual vertex of the vertex fault $\{a, b\}$). The robot may land at a or b. We have the freedom to choose. Depending on the robot's next move we choose a or b such that after the next move the robot stays in A or B. The important observation is that a robot at a or b has the same sensing (the same fcvv) and thus, as the robot is deterministic, has to do the same movement, regardless of whether it lands at a or b. If the next move is "go to the i-th visible vertex in ccw order", where i is 1 or 2, then we make the robot land at b (as if it landed at a, the next movement would bring the robot out of A and B). Similarly, if the next move is "go to the i-th visible vertex in ccw order", where i is 4 or 5, then we make the robot land at a. Clearly, if the next move is "go to the 3rd visible vertex", the robot stays within the cells A and B regardless of us choosing a or b as the landing vertex. Thus, we only choose a or b according to the robot's first movement that is different from "go to the 3rd visible vertex". After we have chosen the proper vertex a or b for the robot to land, we can similarly argue for all subsequent movements.

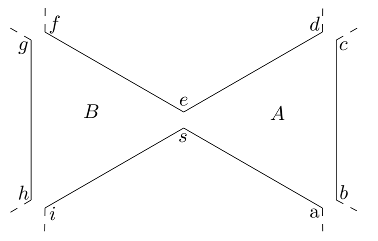


Fig. 5. A robot that does not use a pebble never leaves cells A and B

From the aforementioned arguments it is now an easy observation that if the adversary can choose the initial vertex (i.e., either s or e) then the robot can be made to stay within one cell (say, cell A in our case). □

Using the ideas of the previous lemma we show the following theorem

Theorem 2. *If a faulty robot with two pebbles can count the number of vertices of a triangular polygon P, then at any time of the computation the two pebbles are at most two moves (of the robot) apart.*

Proof. Let us consider the situation where the two pebbles B_1 and B_2 are more than two moves apart. Thus, the pebbles are in two cells A and B which do not share a single vertex. Let us consider the moment when the robot places the second pebble B_2 in cell B. We will show that the robot cannot count the number of vertices of P. We will argue that the adversary can choose landings in such a way that the robot will never come back to cell A (where the first pebble B_1 is placed). Thus, effectively, this will lead into a situation of a robot with one pebble only. In this situation, however, the robot cannot lose sight of the second pebble B_2, as otherwise the robot would end up in a situation of Lemma 1, according to which the adversary can make the robot stay in one cell (forever). Clearly, if the robot cannot lose the sight of the second pebble B_2, it cannot visit all vertices of P (as picking up the pebble B_2 results into the situation of Lemma 1, and thus we can make the robot to stay in one cell, never coming back to cell A), and thus it cannot count the number of vertices of P.

Consider the situation in Fig. 6, where B_1 denotes the first pebble, and B_2 denotes the second pebble. B_1 lies in cell A, B_2 lies in cell B, and there is at least one more cell X between the two cells (and B_1 and B_2 do not lie in X). We want to avoid the robot coming to a vertex of vertex fault F_1, the "gateway" to cell A. For this, we first argue about the position of pebble B_2 in cell B. It is placed at a vertex of a vertex fault $F_4 = \{g, h\}$. From the geometry of the setting and from our assumptions it follows that the robot *had* to came to F_4 from a vertex of P that did not see the pebble B_1. Hence, we (the adversary) can choose whether the robot lands at g or h – the visibility will be the same, so the robot decides to place a pebble in either case.

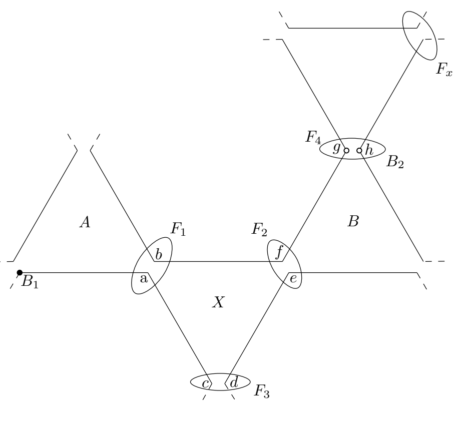


Fig. 6. Pebbles B_1 and B_2 are separated by at least 3 moves

This effectively means that we (the adversary) can decide the location of the pebble B_2 to be g or h. Our decision depends on the next step(s) of the robot. We may assume that the next step of the robot is a movement (as collecting the right-now dropped pebble is useless and does not help the robot to navigate or compute anything). Let us first consider the case in which we let the robot land at vertex g to place the pebble B_2 there. If the robot never leaves the sight of B_2 then the robot can clearly never come to cell A, and it also cannot count the number of vertices of P. Thus, assume the robot eventually leaves the sight of B_2. Clearly, for one of the choices of landing at g or h, the "leaving" of the robot does not happen at a vertex of F_2 (i.e., if for a particular choice of landing the "leaving" happens at a vertex of F_2, then for the other choice of landing the "leaving" happens at a vertex of F_x – the symmetrically placed vertex fault to F_2; this follows because the robot will do the same sequence of movements in either case). Thus, choosing the proper landing, the robot moves from a vertex of F_x to a cell with no sight of a pebble, and thus it ends up at the situation of Lemma 1, which guarantees that the robot will stay in one cell (forever). □

Thus, according to the theorem, the two pebbles have to be dropped in adjacent cells, or in the same cell. This hints us that the robot should keep track of the two pebbles such that they are not very far apart. Thus, as the robot moves, it should move the pebbles too. While this may help in visiting vertices, it is not obvious it helps in counting them exactly. This provokes us to make the following conjecture.

Conjecture 1. A simple robot with two pebbles cannot count the number of vertices of a polygon with vertex faults.

2.3 Counting with 3 Pebbles

Now, we present an algorithm for counting the vertices of a polygon with any number of vertex faults using three pebbles of two different types.

Theorem 3. *A simple robot with three pebbles of two different types can count the number of vertices of a polygon P with vertex faults.*

Proof. Our algorithm uses the distinct pebble (pebble of type 2) to mark the start vertex v_0, and two other identical pebbles (pebbles of type 1) to traverse consistently along the boundary of P in ccw order. Starting at vertex v_0, the algorithm's goal is to be able to go to the i-th vertex on the boundary, $i = 1, 2, 3, 4, \ldots$, until the pebble of type 2 is found again, and thus the number of vertices of P is inferred. The pebble of type 2 will not have any other usage in the algorithm.

As we have seen in the previous sections, going to the first vertex is already impossible if no pebble is used (recall, just set $\{v_1, v_2\}$ to be a vertex fault and let the robot land at vertex v_2 instead of landing at v_1). Using two pebbles, traversing the boundary consistently is possible. We will show how to make one *step* of the traversing, i.e., how to move to the next vertex on the boundary. The whole traversing is then just the repetition of these steps.

Assume the robot is at vertex v_0 and it wants to walk to vertex v_1. The robot leaves a pebble (which is always of type 1 from now on) at v_0 and it attempts to move to its first visible vertex (which may be a virtual vertex). Let us denote by v the vertex where the robot landed. Observe now that the robot landed at vertex v_1 if and only if the robot sees the pebble at the *left neighbor* of v – the last visible vertex (possibly virtual) when considered in ccw order. To see this, observe first that if the robot indeed landed at v_1, it sees a pebble at its left neighbor, even if v_0 belongs to a vertex fault F and the left neighbor is seen as a virtual vertex v_F. On the other side, consider the case when the robot did not land at v_1, but at some other vertex v. Thus, $\{v_1, v\}$ has to be a vertex fault of the polygon, and v_0, v_1, and v are mutually visible. We want to show that v does not see a pebble at its left neighbor (even if it is seen as a virtual vertex). This could only be possible if v_0 and the *true* left neighbor w of v formed a virtual vertex for v. However, as v sees vertices v_0, v_1, and w in this order, v_0 and w cannot form a virtual vertex, as v_0 and w do not appear consecutively in the combinatorial vision of v.

The robot can thus easily check whether it landed at the desired vertex. It just looks to its left neighbor and checks whether there is the pebble. If the robot does not land at vertex v_1, but at some vertex v instead, then clearly $\{v_1, v\}$ is a vertex fault in P. Observe that vertex v sees v_0 and v_1. The robot wants to move to v_1. From the view of vertex v, vertex v_1 is the neighboring vertex of v_0 (which is the vertex with the pebble) in ccw order. Thus, the robot attempts to move to this vertex, and, as v and v_1 form a vertex fault, the robot lands correctly at v_1. Let us call this strategy the *remedy procedure*.

Thus, a robot can move to the neighboring vertex on the boundary using one pebble. If the robot had more pebbles, it could just place a new one, move to the neighboring vertex (using the same algorithm), etc. In case of two pebbles only, the robot does the following. It leaves the second pebble at the vertex v_1 (it knows the current vertex is v_1) and moves to the left neighboring vertex, i.e., to v_0. This can be done in a similar way as when the robot walked from v_0 to v_1, just in the reverse, symmetrical order. The robot again checks whether it landed at v_0 (now it is easy, it just checks whether it landed at a vertex with a pebble on it), and performs the (slightly altered) remedy procedure, if needed. This time, if the robot does not land at v_0, the robot sees two pebbles, and thus it has to attempt to walk to the first visible vertex with a pebble (in the order of vertices as seen from the robot's position). A robot at vertex v_0 then collects the pebble and attempts to move to the vertex with the second pebble, the vertex v_1. If the robot lands at v_1, the robot can start the same algorithm again, thus getting from v_1

to v_2, etc. If the robot does not land at v_1, however, then it lands at vertex v which forms with v_1 a vertex fault. From v the robot sees v_1, and it can identify v_1 as the vertex with the pebble. Thus, the robot can attempt to move to v_1 where it also has to land.

We have presented a procedure which allows the robot to move from a vertex to its neighboring vertex on the boundary, and keeps both pebbles with the robot. Thus, repeating this procedure until the robot reaches the pebble of type 2 allows the robot to count the number of vertices of P. □

3 Fault Detection and Map Construction

We have shown that a simple robot with three pebbles of two types can count the number of vertices of a polygon P. Using the traversing procedure we will show that the robot can also reconstruct the correct cvv at every vertex of the polygon, and thus it can reconstruct the visibility graph of P. This is actually a simple task to do while the robot traverses the boundary of P, as the robot at position v_i (before attempting to move to vertex v_{i+1}) can check whether it sees a vertex with the pebble of type 1 (the vertex v_0), and thus it can find out whether v_0 and v_i are visible in P. This proves the following theorem.

Theorem 4. *Simple robot with three pebbles of two different types can reconstruct visibility graph (and all* cvv*'s) of a polygon with vertex faults.*

A related reconstruction-question arises, namely, can a simple robot identify all vertex faults? We present a procedure that allows the robot to identify all vertex faults visible from the vertex v_0. Repeating this procedure for all other vertices v_i, $i = 1, 2, \ldots, n$, then allows to identify all vertex faults. Again, the robot can identify all vertex faults visible from v_0 using three pebbles of two different kinds. First, the robot computes the number of vertices of P using the algorithm of the previous section. After that, the robot at vertex v_0 takes the pebble of type 1 from v_0, and traverses the polygon (using the traversing procedure with the two identical pebbles). The robot checks for every visited vertex v_i, $i = 1, 2, \ldots, n$, whether it is visible from v_0 by leaving the pebble of type 1 at v_i, traversing back to v_0, and checking whether pebble of type 1 is visible from v_0. If vertex v_i is visible from v_0 at the same position (in the ordered list of visible vertices) as the previously visible vertex v', then (and only then) $\{v', v_i\}$ forms a vertex fault visible from v_0.

Theorem 5. *Simple robot with three pebbles of two different types can identify the vertices F of every virtual vertex v_F.*

4 Conclusions and Further Work

We have studied a particular model of faulty sensing of simple combinatorial robots in a polygon P. We have shown that even the otherwise trivial task of computing the number of vertices of P is not feasible for a robot with one pebble. We have demonstrated difficulties a robot with two pebbles has to count the number of vertices and conjectured that the robot cannot count. Finally, we have presented algorithms that allows a robot with three pebbles of two different types to count the number of vertices of P, to reconstruct the visibility graph of P, and to identify all vertex faults.

An obvious open problem left is Conjecture 1. Similarly, one might want to get rid of using two types of pebbles. As a future work we would like to study other models of faults for simple combinatorial robots (e.g., movement faults where the robot does not stop at reflex vertices), and possibly a combination of these faults.

References

1. Suri, S., Vicari, E., Widmayer, P.: Simple robots with minimal sensing: From local visibility to global geometry. International Journal of Robotics Research **27**(9) (September 2008) 1055–1067
2. Brunner, J., Mihalák, M., Suri, S., Vicari, E., Widmayer, P.: Simple robots in polygonal environments: A hierarchy. In: Proceedings of the 4th International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS). (2008) 111–124
3. Yershova, A., Tovar, B., Ghrist, R., LaValle, S.: Bitbots: Simple robots solving complex tasks. In: Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference. (2005) 1336–1341
4. Tovar, B., Freda, L., LaValle, S.: Using a robot to learn geometric information from permutations of landmarks. Contemporary Mathematics **438** (2007) 33–45
5. Ganguli, A., Cortés, J., Bullo, F.: Distributed deployment of asynchronous guards in art galleries. In: Proceedings of the American Control Conference. (June 2006) 1416–1421
6. Souissi, S., Défago, X., Yamashita, M.: Gathering asynchronous mobile robots with inaccurate compasses. In: Proceedings of 10th International Conference on Principles of Distributed Systems (OPODIS). (2006) 484–500
7. Katayama, Y., Tomida, Y., Imazu, H., Inuzuka, N., Wad, K.: Dynamic compass models and gathering algorithms for autonomous mobile robots. In: Proceedings of the 14th Colloquium on Structural Information and Communication Complexity (SIROCCO). (2007) 274–288
8. Cohen, R., Peleg, D.: Convergence of autonomous mobile robots with inaccurate sensors and movements. In: Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS). (2006) 549–560