Counting Targets with Mobile Sensors in an Unknown Environment

Beat Gfeller[1], Matúš Mihalák[1], Subhash Suri[2*], Elias Vicari[1], and Peter Widmayer[1]

[1] Department of Computer Science, ETH Zurich, Zurich, Switzerland
{gfeller,mmihalak,vicariel,widmayer}@inf.ethz.ch
[2] Department of Computer Science, University of California, Santa Barbara, USA
suri@cs.ucsb.edu

Abstract. We consider the problem of counting the number of *indistinguishable* targets using a simple binary sensing model. Our setting includes an unknown number of point targets in a (simple or multiply-connected) polygonal workspace, and a moving point robot whose sensory input at any location is a binary vector representing the cyclic order of the polygon vertices and targets visible to the robot. In particular, the sensing model provides no coordinates, distance or angle measurements. We investigate this problem under two natural models of environment, friendly and hostile, which differ only in whether the robot can walk up to them or not, and under three different models of motion capability.
In the friendly scenario we show that the robots can count the targets, whereas in the hostile scenario no $(2-\varepsilon)$-approximation is possible, for any $\varepsilon > 0$. Next we consider two, possibly minimally more powerful robots that can count the targets exactly.

1 The Problem and the Model

Simple, small and inexpensive computational and sensing devices are currently at the forefront of research interest in computer science. These devices promise to bring computational capabilities into areas where previous approaches (usually consisting of complex and bulky hardware) are not feasible or cost-effective. Such devices are being successfully used in various monitoring systems, military tasks, and other information processing scenarios. Their main advantages are quick and easy deployment, scalability, and cost-effectiveness. However, in order to realize the full potential of these technologies, many new and challenging research problems must be solved, because the classical schemes designed for centralized and desktop computational hardware are inapplicable to the lightweight and distributed computational model of sensor nodes. The inherent limitations of the systems based on these simple devices have inspired the research community to consider the computation with a minimalistic view of hardware complexity, sensing and processing, energy supply, etc.

In this paper we use such a minimalistic approach in the area of mobile sensors – simple robots. We consider and define robots of unsophisticated sensing and mobile capabilities

[*] Work done while the author was a visiting professor at the Institute of Theoretical Computer Science, ETH Zurich

and investigate their computational power on an elementary yet natural problem of counting objects of interest in the robots' environment. We model the environment by a polygon P (simply or multiply-connected) in the plane and the objects of interests, namely, *targets* are modeled as a set of points inside P.

We assume that the robot is a (moving) point, equipped with a simple camera that can *sense* just the combinatorial features of the surrounding. In particular, the robot can see a vertex of P or a target, can distinguish a target from a vertex, but the vertices and the targets are otherwise indistinguishable, i.e., all vertices are visually identical and all targets are visually identical. It is only the cyclic order in which the robot sees the objects that distinguishes them from each other. We assume that the ordering is always consistent, which we take, without loss of generality, to be counterclockwise. We model such a discrete vision by a *point identification vector* (piv), which is a binary vector defined by the cyclically ordered list of targets and polygon vertices that are visible from the current robot's position, where each bit indicates whether the corresponding point is a target (value 1) or a vertex of the polygon (value 0). Sitting at a vertex of P, we assume that the cyclic order of the visible points (vertices and targets) starts with the neighboring vertex, i.e., the first component of the piv always represents the neighboring vertex (and therefore has value 0). For the robot located on a target, we make no assumption about the position of the first vertex – it is chosen by an adversary.

Moreover, the robot can see the edges of the polygon. This is modeled by a *combinatorial visibility vector* (cvv), a binary vector of length k whose i-th bit encodes whether there is an edge between vertex $i-1$ and vertex i of the k vertices visible from the robot's position. See Fig. 1 for illustration.

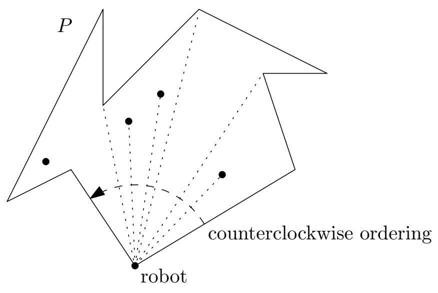

Fig. 1. An illustration of a point identification vector (piv) and a combinatorial visibility vector (cvv) in polygon P (with 4 targets); piv is $(0, 1, 0, 0, 1, 1, 0, 0)$ and cvv is $(1, 1, 0, 1, 0, 1)$

The robot has no other sensing capability, and in particular has no information about distances, angles, or world coordinates. This also motivates our simplistic model of robot's movement. The robot can pick a direction based on its sensing system and move in that direction until the environment prevents the robot to go any further. The direction of a robot's movement is the direction to one of the visible points (a vertex of P or a target) in robot's piv, and the robot stops when it reaches that point. The robots can sense the environment only when not moving.

Due to these unsophisticated vision and motion primitives, seemingly easy tasks become difficult in this model. For instance, a robot sitting at a vertex u can specify a visible vertex v by its index in the cvv of u. However, if the robot moves from u to v, it is not possible in general to recover the position of u with respect to v. A way to circumvent this issue is to mark u with a *pebble* before moving to v. A pebble is visually distinguishable from vertices and targets. If no other pebbles are visible from v, the position of u can be recovered. For a detailed discussion of the implications of this minimalistic model, see [1].

We are interested in how the robots can solve various environment exploration tasks and what are the limitations implied by our simplistic assumptions on robots capabilities. In this paper we consider the problem of determining the number of targets in an unknown polygon P. Throughout this paper we refer to this problem as the *counting problem.* By n we denote the number of vertices of P and by m the number of targets therein. For simplicity we assume that the targets and polygon vertices are in a general position, i.e., no three points are collinear. In this paper we consider two different scenarios to model two basic classes of applications. In the *friendly* environment, the robot is allowed to walk to any target. In the *hostile* environment, the robot is not allowed to walk to targets. This scenario models the situation where a target represents an unsafe entity and coming into an imminent closeness to targets is dangerous.

For a friendly scenario we show that a single robot with a single pebble (marker) can count the number of targets in any polygon P. In contrast, we show that in a hostile scenario, the robots cannot count the targets in general. Thus, requiring the robots to count targets only from afar is a more complicated problem, and we must endow the robots with some additional capabilities. Surprisingly, we show that these additional capabilities are quite minor, yet subtle. In fact, we consider two possible models, and show their implications on our problem. We consider robots that can walk along edge or diagonal extensions, i.e., if a robot picks a visible vertex u as the direction of its walk, the robot can continue its walk in the same direction after it reaches u, if there is no polygonal edge to prevent it. In the second model we consider one additional global direction in which the robot can walk from any vertex of P. In both models the robots can solve the counting problem.

We are interested in deterministic algorithms and their worst-case analyses, which we express in terms of the number of steps (movements) of robots and in the amount of used memory. We work with word-memory units, where one word of memory has $\theta(\log(\max\{m, n\}))$ bits. We are also interested in approximation algorithms, i.e., in algorithms that deliver a (provably good) estimate on the number of targets. Further, we look for estimates that are never smaller than the actual number of targets. We say that an algorithm is a ρ-approximation for the counting problem if for the setting with m targets, $m \in \mathbb{N}$, the algorithm estimates the number of targets by z, for which $m \leq z \leq \rho \cdot m$. [3]

To demonstrate the notion of approximation and to justify our sensing model we illustrate that for the following weaker sensing model no non-trivial approximation exists: consider the sensing of the vertices in the same way as we defined before, but consider the sensing of the targets only by their presence. Thus, the only information the robot gets is

[3] We can generalize the approximation concept and consider α-approximation algorithms, for which $\frac{1}{\alpha}m \leq z \leq \alpha \cdot m$ holds. Observe that these two approximation concepts are equivalent in the following sense: for any solution z that is a ρ-approximation there is a solution z' that is an α-approximation, and vice versa. It is enough to set $z' = \frac{z}{\sqrt{\rho}}$, $\alpha = \sqrt{\rho}$.

the number of visible targets (but not their ordering within the vertices of P). Consider Fig. 2. It depicts two different scenarios, one scenario with $m = 1$ target and the second scenario with $m = n/3$ targets. In both scenarios the robot senses from every vertex exactly one target and therefore cannot distinguish the scenarios. Hence, for this simple sensing model no approximation algorithm can guarantee a ratio better than $n/3$.

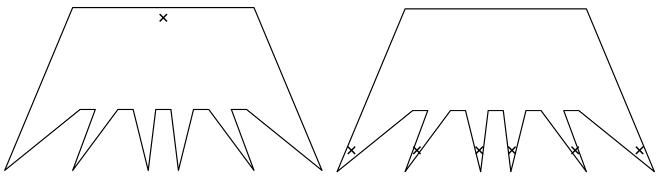

Fig. 2. If a robot only senses the number of targets then the number of targets cannot be approximated within $o(n)$

1.1 Related Work

The work of [1] considered simple robots with combinatorial sensing of the environment and investigated some elementary questions of what information about the topology of the environment can be deduced by simple robots. In our paper, we consider the same robots, but enlarge the complexity of the environment by adding the targets into the environment. The paper [1] demonstrates that a robot, although being strongly limited, can solve non-trivial tasks. A robot cannot decide whether a vertex is convex, but can decide whether the polygon is convex. Also, a robot cannot decide which is the outer boundary of a multiply connected polygon P, although it can discover and count all the boundary components in P. The paper also shows that a robot with one pebble can virtually label the vertices of any (simple or multiply-connected) polygon P, which allows the robot to navigate from any vertex i to any vertex j. The navigation result is an important building block in our paper. Further, the paper shows that the robot can compute a triangulation of P and solve a distributed version of the famous Art Gallery Problem with $n/3$ robots.

Combinatorial geometric reasoning is used in many motion planning and exploration tasks in robotics [2, 3]. There are other papers dealing with minimalistic models for robots [4–7]. However, the nature of problems investigated in our paper does not seem to be addressed in the past. They deal mainly with navigation and pursuit evasion and not with recognition of important points (targets) in the environment. [4] assumes *labeled* features of the environment (which is not a polygon), allowing sensors to distinguish these *landmarks*.

2 The Friendly Environment

In this section we show that in a friendly environment a robot with two pebbles can count the targets in any simply or multiply connected polygon.

We consider simple polygons first. In the beginning the robot counts n, the number of vertices of the polygon. This is an easy task: the robot leaves a pebble on the starting vertex and walks around the polygon's boundary, counting the vertices until it returns to the pebble. Let $1, 2, \ldots, n$ denote the vertices of the polygon, ordered in the counterclockwise direction, starting at the robot's position.

The idea of the algorithm is to go to every vertex i, $i = 1, 2, \ldots, n$, and count the targets that are visible from i and that are not visible from any vertex j, $j < i$. We call these targets *newly visible* at vertex i. Thus, the robot can go through vertices $i = 1, 2, \ldots, n$ and sum up all newly visible targets. Clearly, no target will be counted twice, and therefore the resulting sum is the total number of targets.

We now describe how the robot can identify whether a target is newly visible. Being at vertex i, the robot wants to identify whether a k-th target in its visibility vector is newly visible. The robot goes to the target, leaves the pebble there, and checks for every vertex $j < i$, whether the pebble is visible from j (the navigation from the target back to the vertex i can be done by leaving a pebble at i and checking the position of i in the visibility vector of the target). Obviously, the target is newly visible if and only if the pebble is not visible from any vertex j, $j < i$. Overall, the robot needs one pebble and constant memory (to remember the number of vertices, the current position i, the position j and the position k of the considered target at i, and to mark the newly visible targets in the visibility vector of vertex i). Hence, in time $2i$ we can check whether a target visible from the i-th vertex is newly visible. To check all targets at position i we need at most $2mi$ steps. Thus, the robot needs $O(mn^2)$ steps to count the targets in P.

If the time is crucial, one can achieve a $O(mn)$ running time at the expense of used memory. For each vertex i the robot maintains the piv with the additional information stating whether a given target is newly visible. In the beginning, every target in the piv is marked as newly visible. Then for every vertex i the robot marks each newly visible target with a pebble and walks around the boundary towards vertex n and at every vertex i, if the robot sees the pebble, it marks the corresponding bit in the bit array of vertex i as not newly visible. Thus, the robot walks m times around the boundary (for each target it walks exactly once and at most n steps), resulting into $O(mn)$ steps of the robot. The robot needs $O(n)$ of memory.

Theorem 1. *In the friendly environment a robot with two pebbles can count the targets in a simple polygon in $O(mn^2)$ steps and with $O(1)$ memory, or in $O(mn)$ steps and with $O(n)$ memory.*

The result can be easily extended to polygons with holes (multi connected polygons), if we can navigate through the vertices in a consistent way. In [1], a navigation in an arbitrary multiply connected polygon was demonstrated with a robot and one pebble. Our robot has all the capabilities of the robot in that work, therefore the robot can first compute the navigation instructions, which are then stored in the robot's memory. Alternatively, we can use an additional, globally distinguishable pebble and perform the vertex navigation on the fly.

Theorem 2. *In the friendly environment a robot with two pebbles can count the targets in any polygon.*

3 Hostile Environment

After solving the counting problem in the scenario where robots can walk to targets, we consider now the scenario where robots walk only on vertices of P.

3.1 Inapproximability and Approximation

Inapproximability. We show that the counting problem cannot be approximated within a factor $2-\varepsilon$, for any $\varepsilon > 0$, even if the polygon P is simple. Consider the polygon in Fig. 3. The polygon consists of four spikes attached to the four sides of a rectangle. It depicts two scenarios with a different number of targets. In the first scenario there are 6 targets and in the second scenario there are 4 targets. Considering any vertex of the polygon, the vectors cvv and piv are the same in both scenarios. Hence, the robot cannot distinguish the two scenarios, which shows a lower-bound of $6/4 = 3/2$ for the approximation ratio.

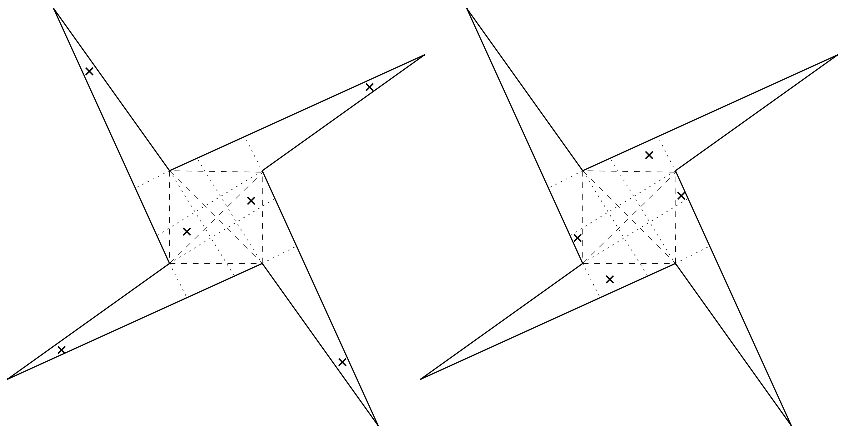
Fig. 3. The counting problem cannot be approximated within a factor 3/2.

This construction can be extended to a general-sized polygon, where $2k$ spikes are attached to a regular $2k$-gon, using $2k$ and $4k-2$ targets in two different scenarios, thus giving the desired lower-bound $(2-\varepsilon)$. See the Appendix for details.

Theorem 3. *The counting problem cannot be approximated within a factor* $2-\varepsilon$ *even in a simple polygon.*

Note that this inapproximability result relies only on the visibility limitations of the robots and not on their limited navigation capabilities.

Approximation. Since the counting problem cannot be solved optimally, it is natural to look for approximate solutions, i.e., for good estimates of m, the number of targets.

Observe first that m is at least the number of targets visible from any vertex of P. Let m_i denote the number of targets that are visible from vertex i. We have $m \geq \max_i m_i$. On the other hand, clearly, $m \leq \sum_i m_i$. Since every target is visible from at least three vertices of P (consider a triangulation of P and the vertices of the triangle, in which the target lies), we have $m \leq \frac{1}{3} \sum_i m_i$. A robot can compute the sum $z = \sum_i m_i$ with one pebble that allows the robot to navigate through all vertices of P (even with holes). Obviously, reporting $\frac{1}{3}z$ as the estimate for the number of targets yields an $\frac{n}{3}$-approximation. Alternatively, if we denote by k the number of vertices with non-zero m_i, the value z becomes a $\frac{k}{3}$-approximation.

Although the approximation is not sound at first sight (consider a convex polygon with a single target in it), it gives some insight into the complexity of the counting problem. Notice that the derived approximation ratio depends solely on the number of vertices n (or on k, the number of vertices with a view on at least one target) and not on the number of targets. Hence, if m grows in comparison to n or k, the approximation ratio gets better. In other words, the approximation ratio does not grow with the number of targets, but is determined by the structure of the polygon (i.e., by n) and by the way how the targets are placed in this structure (i.e., by k).

For the sake of completeness, we would like to mention that a 2-approximation algorithm can be designed under a slightly stronger model (but weaker than the one described in Section 3.3). In this model, the $2 - \varepsilon$ inapproximability result still holds. We omit the precise description of the algorithm due to space limitations.

3.2 More Power to the Robots

We have seen in the previous subsection that a simple robot cannot count the targets in a simple polygon. We therefore look at possible robot enhancements, which keep the robots as simple as possible and at the same time enable the robots to count the targets. We consider two such extensions.

In the first one we allow the robot to walk along edge-extensions and diagonal-extensions, i.e., if a robot at vertex v picks a vertex w as the direction of the robot's walk, the robot is allowed to walk in the same direction after it reaches w, and stops on the boundary at a point w'. Fig. 4 illustrates this enhancement. The line vw' is called an *edge-extension* (*diagonal-extension*) if vw is an edge (diagonal) of P. If we do not need to distinguish whether vw' is an edge- or diagonal-extension, we shall call vw' simply an *extension*. If a pebble is placed at w', it is then visible in the same way as a vertex of P, but visually distinguishable.

In the second extension one additional, global direction is introduced, in which a robot can move. Without loss of generality we assume that it is the direction of a vertical line going through the robot's position. For simplicity of presentation we assume that the polygon does not have vertical edges. On top of that we assume the robot can tell whether a visible point (a vertex or a target) is to the left or right of the vertical line, and whether it is above or below the robot, i.e., above or below the horizontal line going through the robot's position. Such an enhancement can be viewed as a navigation with compass. If a robot walks from a vertex v in the vertical direction we say that it walks along the *vertical extension* of v.

For both extensions we present algorithms that allow robots to count the number of targets inside the polygon P.

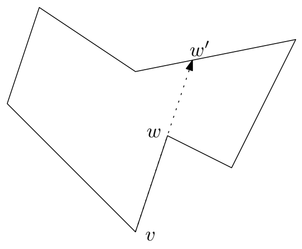

Fig. 4. A robot at position v chooses vw as the direction of the robot's walk. After the robot reaches w, it can continue in the same direction until it hits the boundary of P at point w'

Partition of the Polygon and Counting. The algorithms are based on the idea of partitioning the polygon into special triangles and counting the targets in the triangles exactly. To illustrate the idea, consider a triangulation of P with the property that every triangle has at least one side identical to a side of P. We call such a triangle a *baseline* triangle, and the edge of the triangle that lies on the boundary of P a *baseline edge.* A triangulation with all baseline triangles is called a Hamiltonian triangulation, as its dual is a path[4].

In the case of a Hamiltonian triangulation a robot can count the targets with the following algorithm. For every triangle the robot moves to the vertex opposite the edge of the triangle which is also an edge of P, and counts the targets that are visible between the two vertices of the edge. Clearly, in this way every target is counted exactly once. Unfortunately, such a triangulation does not always exist, see Fig. 5. However, if we add

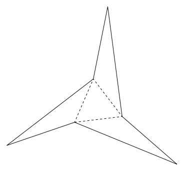

Fig. 5. A polygon and its unique triangulation with a triangle consisting solely of diagonal edges. The triangulation is depicted by dashed lines

new vertices on the boundary of P at particular positions, we can always partition a polygon P into baseline triangles. Such a partition does not necessary form the "usual"

[4] A dual of a triangulation is a graph, where each triangle corresponds to one vertex and there is an edge between two vertices if the two corresponding triangles share an edge.

triangulation of P in the sense that a triangle can consist of more than 3 vertices of P (see Fig. 6 and the triangle vw_1v_2 which also contains a vertex v_1 on its boundary). We call such a partition a *baseline triangulation*. Either of the two robot extensions which we have introduced allows robots to use additional points of P to compute such a triangulation.

The result of [8] presents an algorithm that recognizes whether a polygon has a Hamiltonian triangulations and computes one. The algorithm can be adapted by a robot with an additional capability, which allows the robot to recognize the reflex vertices of P (which is not possible in our model, see [1]).

3.3 Walking Along Edge- and Diagonal-Extensions

In this section we consider robots that can walk along edge- and diagonal-extensions. We show that such robots can partition any simple polygon into baseline triangles, and thus can count the targets.

Consider a robot at a vertex v of the polygon P. Let $v_1, v_2, \ldots, v_i \ldots$ denote the visible vertices from v, cyclically ordered in the counterclockwise direction. Observe that the lines vv_i partition the visible part of P into triangles (all with a common point v), each with at least one baseline edge. See Fig. 6 for an illustration, where the triangles vw_1v_2, vv_2w_2, vv_3v_4 and vv_4v_5 partition the visible part of P. Thus we can partition the visible part of P. Observe that the invisible part of P is a set of disjoint simple sub-polygons. In the example from Fig. 6 the sub-polygons P_1 and P_2 form the invisible part of P. We call such a sub-polygon a *pocket* of P. Observe that a pocket is created by a line which is an edge-extension or a diagonal-extension. Applying a recursive partitioning approach on the pockets, we create a partition of P into triangles with at least one edge on the boundary of P. Let T denote this triangulation.

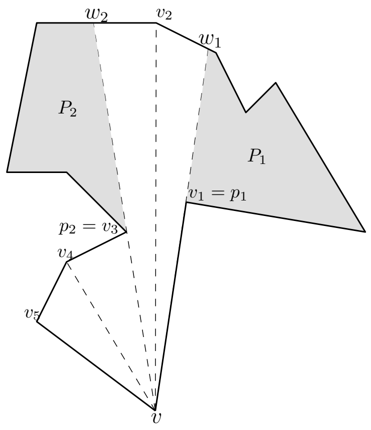

Fig. 6. The extensions of a vertex v define baseline triangles and pockets of v

The main idea of the algorithm is to count all vertices from the robot's position v and then proceed recursively in the corresponding pockets of the polygon, thus navigating through T and counting the targets in the triangles of T. We begin with a high-level description. For a vertex v let $P_1, \ldots, P_\ell$ denote pockets of P defined by all extensions originating at v. Let p_i, $i = 1, \ldots, \ell$, denote the visible vertex whose extension defines P_i. Let w_i be the point of P for which vw_i is the extension of vp_i.

Counting in Simple Polygons
1. Count all the targets that are visible from the robot's position at vertex v.
2. Put a pebble at v and remember the position of v in the respective piv of every vertex p_i and of every point w_i.
3. Recursively count the targets in P_i, $i = 1, \ldots, \ell$, by marking the point w_i with a pebble and going to p_i.

When a robot walks to vertex p_i to start a recursive call for pocket P_i, it first checks the position of the pebble that marks the point w_i. Next the robot determines which vertices (and targets) visible from p_i belong to pocket P_i. Let k be the number of vertices (including w_i) and targets visible from p_i. Let h be the index of w_i in the piv of vertex p_i. If pocket P_i lies to the right of p_iw_i, then P_i contains the vertices and targets from the piv of p_i with index $1, 2, \ldots, h$. If pocket P_i lies to the left of p_iw_i, then P_i contains the vertices and targets from the piv of p_i with index $h, h + 1, \ldots, k$. Observe that P_i lies to the right of p_iw_i if and only if p_i is the first end-point of the diagonal in piv of vertex v.

The robot at vertex p_i knows which part of its piv represents the sub-polygon P_i and it can therefore perform the same steps of the **Counting in Simple Polygons** algorithm on the pocket P_i only. Before that, the pebble from w_i is recollected as it is no longer needed. When the robot finishes the counting in P_i it returns to the vertex v (using the stored navigation information) and continues the algorithm there.

Theorem 4. *A robot with one pebble, able to walk along extensions, can count the number of targets in every simple polygon in $O(n)$ steps with $O(n)$ memory.*

Proof. Observe first that the algorithm provides a consistent navigation scheme through the triangulation T of P. Thus every target is counted exactly once. The dual of T is a tree. When traversing T in the way it was created, we encounter for every triangle a vertex that defines the triangle which was not visited before. Thus, the dual of T has $O(n)$ vertices which determines the number of steps of the algorithm (since the robot spends a constant time in every triangle of T). The robot needs to store the necessary information to return from a recursive call – the predecessor v of every vertex p_i. Hence, $O(n)$ memory is needed. □

3.4 Walking with a Compass

In this section we consider one additional fixed direction in which the robot can move. Without loss of generality, we assume that a robot sitting at a vertex can, additionally to moving to all visible vertices, move also along the vertical line going through the robot's position.

We present an algorithm that computes a baseline triangulation in any simple or multiply-connected polygon and navigates the robot such that each triangle is considered for counting exactly once and thus it allows the robot to count the number of targets in the polygon. To simplify the presentation we first use an arbitrary number of pebbles – we show later how to use only a constant number of pebbles.

The key observation is that all the vertical extensions of a polygon P partition the polygon into baseline cones and quadrilaterals for which two opposite sides are on the boundary of P (Figure 7). Each quadrilateral can be subsequently partitioned into two baseline triangles (by picking a diagonal as the common boundary of the triangles).

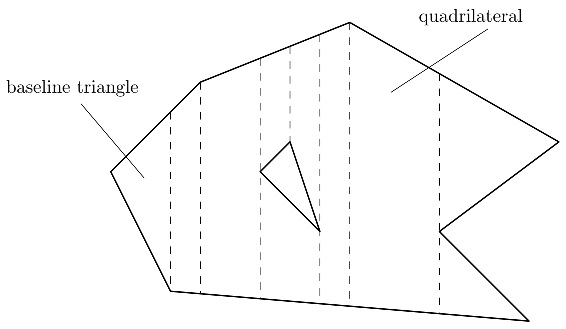

Fig. 7. A multi-connected polygon with its partition by vertical extensions.

Hence, using at most $2n$ pebbles, the robot can mark every end-point of every vertical extension which then imposes a baseline triangulation. This can be done by visiting every vertex of P (using one pebble) [1]. To count every target exactly once, the robot goes through every vertex or pebble p and considers only triangles lying *above* p and on its *right* (if any). Since every triangle has one vertical side, the robot can always reach the opposite vertex of the baseline side in one step and count the targets in the triangle, and return back.

Counting with a Compass – Go through every vertex of P. At any vertex v walk along the vertical extension of v and mark its endpoint(s) by a pebble. Go through every vertex or marked point p and count the number of targets in the triangles with common point p lying above p and to the right of p.

We now show how to reduce the number of used pebbles at the cost of an increased running time. The robot does not mark all the quadrilaterals at once, but one by one. Let us call an endpoint of a vertical extension a q-node. We shall show how to navigate through all the vertices and q-nodes in a consistent way. We begin with the navigation through vertices of P from [1], where every edge of P is visited exactly once. If a robot moves in this navigation along a polygonal edge uv, we compute all the q-nodes lying on this edge and before the robot moves to v it visits all the q-nodes in the order of increasing distance from u.

Let us consider the situation where the robot is at a point p (a vertex u or a q-node) of the edge uv and it wants to move to the next q-node. The robot can find the next q-node by sequentially creating all q-nodes (by going to every vertex of P) and checking which one lies on the edge uv and closest to p. Specifically, using a pebble the robot marks the initial position p. The next pebble is used to mark the so-far closest q-node on the edge uv. The robot goes through every vertex w of P and creates q-nodes lying on the vertical extensions of w. For every such q-node the robot checks whether it lies on the edge uv and whether it is closer to u than the current best. The two pebbles make this operation easy for the robot.

Theorem 5. *A robot with 2 pebbles, able to walk along vertical extensions, can count the number of targets in every polygon in $O(n^3)$ steps and with $O(n)$ memory.*

4 Conclusions

We considered a minimalistic computational framework of mobile sensors – simple robots, whose visibility-based sensing reflects just the combinatorial character of the environment. We investigated their capabilities on the problem of counting points of interest (targets) in a polygon P and considered two scenarios. We have showed that in the friendly environment the robots can count the targets using one pebble. In the hostile environment the robots cannot count the targets and they cannot even approximate the number of targets by a multiplicative factor less than 2. We showed that a naive approach is an n-approximation, where n is the number of vertices of the polygon. We have looked at possible minimum extensions of the robots' capabilities that allow to count targets. We have considered two such extensions – walking along edge- and diagonal-extensions, and walking with compass.

We have not answered all interesting questions and many of these remain open for the future research. For example, what is the best approximation ratio of the problem? Is the lower-bound tight or is there a better approximation algorithm? What is the inherent power of pebbles: can we do anything without them? Are there simpler robots' enhancements that allow the robots count the targets? Can a collaboration of more robots do better than a single robot?

References

1. Suri, S., Vicari, E., Widmayer, P.: Simple robots with minimal sensing: From local visibility to global geometry. In: Proceedings of the 22nd Conference on Artificial Intelligence (AAAI07). (to appear)
2. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers, Norwell, MA, USA (1991)
3. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge, U.K. (2006)
4. Tovar, B., Freda, L., LaValle, S.M.: Using a robot to learn geometric information from permutations of landmarks. Contemporary Mathematics, to appear
5. Yershova, A., Tovar, B., Ghrist, R., LaValle, S.M.: Bitbots: Simple robots solving complex tasks. In: AAAI National Conference on Artificial Intelligence. (2005) 1336–1341
6. Guibas, L.J., Latombe, J.C., LaValle, S.M., Lin, D., Motwani, R.: A visibility-based pursuit-evasion problem. IJCGA **9**(5) (1999) 471–494
7. Sachs, S., Rajko, S., LaValle, S.M.: Visibility-based pursuit-evasion in an unknown planar environment. International Journal of Robotics Research **23**(1) (2004) 3–26
8. Narasimhan, G.: On hamiltonian triangulations in simple polygons. Int. J. Comput. Geometry Appl. **9**(3) (1999) 261–275

A The $(2 - \varepsilon)$ Lower Bound

We construct a simple polygon P with $2n$ vertices and place into the polygon $2n - 2$ targets in the first scenario and n targets in the second scenario such that the piv remains unchanged in both scenarios, at every vertex of P. We assume n is even, i.e., $n = 2k$.

The Fig. 8 outlines the construction for $n = 12$. The polygon consists of n *outer* vertices $y_1, y_2, \dots, y_n$ and n *inner* vertices $x_1, x_2, \dots, x_n$. It can be viewed as an n-gon, a regular polygon formed by vertices x_i, $i = 1, \dots, n$, connected on each side x_i, x_{i+1} to a triangular spike x_i, y_i, x_{i+1}. Here and further in the text, the indices are to be understood in a cyclic fashion. The line $y_i x_i$ intersects the segment $x_{i+1}x_{i+2}$ in the middle. Thus, the visibility region of y_i, i.e., the cone of y_i defined by lines y_i, x_i and y_i, x_{i+1}, intersects the visibility regions of vertices y_{i-1} and y_{i+1}, but not the visibility regions of other y_js.

Observe first that a robot at vertex y_i sees only two vertices of P, namely vertex x_i and vertex x_{i+1}. Further, a robot sitting at vertex x_i sees all vertices x_j, $j = 1, 2, \dots, n$, and vertices y_{i-1} and y_i.

The aim is to place the targets in a way that a robot sitting at vertex y_{2l+1} sees one target (the piv is $(0, 1, 0)$), and a robot sitting at vertex y_{2l} sees 2 targets (the piv is $(0, 1, 1, 0)$). For a robot at vertex x_i, $i = 1, \dots, n$, we want the robot to see exactly 1 target between each two consecutive vertices of its piv, i.e., we want the piv to be $(0, 1, 0, 1, 0, 1, \dots, 0, 1, 0)$. Observe that the consecutive vertices of piv at vertex x_i are $y_i, x_{i+1}, \dots, x_n, x_1, \dots, x_{i-1}, y_{i-1}$. We show how to achieve such visibility with two different number of targets. First we use only n targets and then we use $2n - 2$ targets.

To place the n targets we proceed as follows. We place one target into each triangle y_i, x_i, x_{i+1}. Observe that the triangle is divided into three parts by the lines y_{i-1}, x_i and y_{y-1}, x_{i-1}. Let us label the parts P_1, P_2 and P_3, starting at a part containing the vertex x_{i+1}. Fig. 9 illustrates the partition. For odd i, we place the target into part P_2. For even i, we place the target into P_1. Observe now that a robot indeed sees one target from every vertex y_{2l+1} and two targets from every vertex y_{2l}. Observe also that any vertex x_j sees exactly one target between two consecutive vertices x_i, x_{i+1}, $i, i + 1 \neq j$, because the parts P_1 and P_2 of triangle y_i, x_i, x_{i+1} contain exactly one target and the parts are completely visible from x_j within the segment x_i, x_{i+1}. There is also one target visible in the segment y_j, x_{j+1} and in the segment x_{j-1}, y_j which shows the claim for n targets.

We now use $2n - 2$ targets in P to achieve the same visibility configuration. First, we place one target into every triangle x_i, y_i, x_{i+1} such that the target is visible only from vertices x_i, y_i and x_{i+1}. This can be easily achieved when the target is placed very close to y_i. This leads to piv being $(0, 1, 0)$ at vertices y_i and piv being $(0, 1, 0, 0, \dots, 0, 0, 1, 0)$ at vertices x_i. The remaining $n-2$ targets are placed in the following way. For the presentation purposes we label the targets $t_1, \dots, t_{n-2}$. Each target t_i is placed *close* to vertex x_i and in the cone C_i of x_i defined by the vertices x_{n-1}, x_n. More precisely, by placing t_i close to x_i we mean to place the target t_i into the triangle $T_i := x_{i-1}, x_i, x_{i+1}$. Observe now that for any placement of target t_i into $C_i \cap T_i$ the piv of vertex x_i is as desired, i.e., $(0, 1, 0, 1, 0, \dots, 0, 1, 0)$. Indeed, for vertex x_i, $i \leq n - 2$, the cone C_i contains t_i and thus the target is visible between x_{n-1} and x_n. For every other cone of x_i defined by vertices x_j and x_{j+1}, the target t_j lies in that cone. Also, for vertex x_{n-1} the cone of x_{n-1} defined by vertices x_i and x_{i+1} contains exactly one target – t_{i+1}. Similarly, the cone of vertex x_n defined by vertices x_i and x_{i+1} contains exactly one target – t_i. To achieve the desired piv

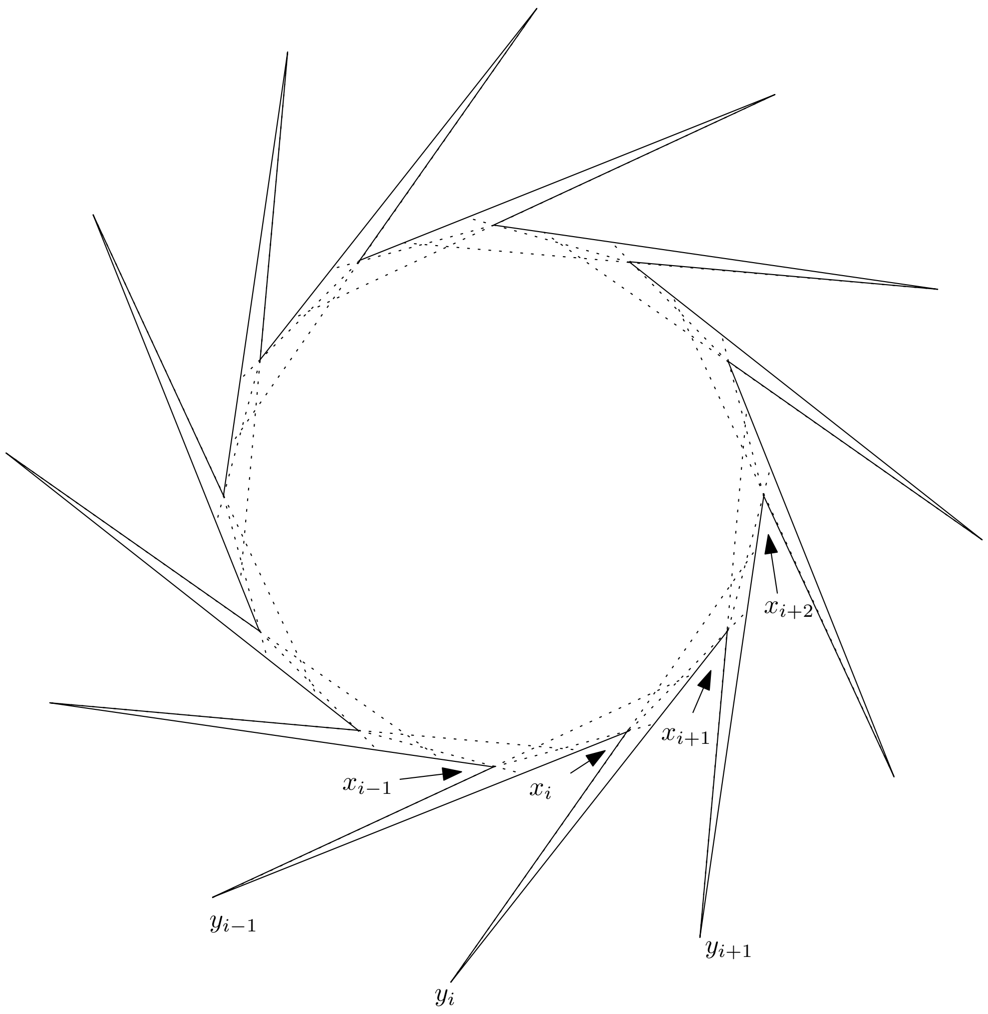

Fig. 8. Construction of a polygon P for the $(2-\varepsilon)$-inapproximability

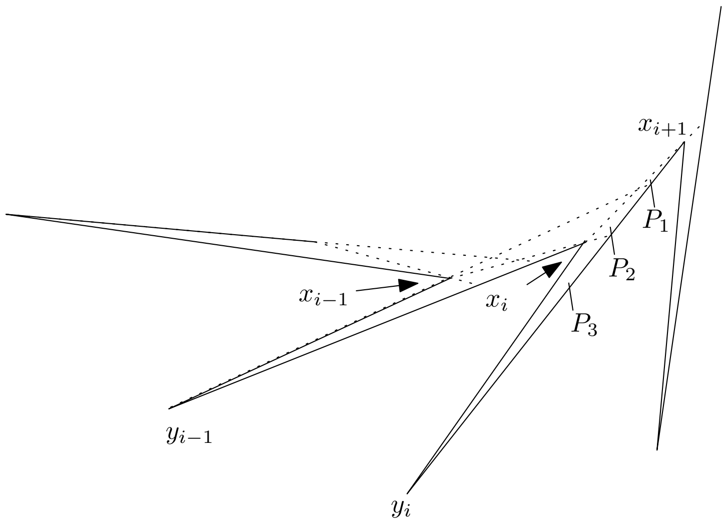

Fig. 9. The partition of the triangle y_i, x_i, x_{i+1} into three parts P_1, P_2 and P_3 by the lines y_{i-1}, x_{i-1} and y_{i-1}, x_i

from the vertices y_i, we place each target t_i within T_i either to the left or to the right of line y_{i-1}, x_{i-1}. For $i-1=2l$ we place t_i to the right of the line y_{i-1}, x_{i-1}, so that t_i is visible from y_{i-1} (i.e., into the cone of y_{i-1} defined by vertices x_{i-1} and x_i). For $i-1=2l+1$ we place t_i to the left of line y_{i-1}, x_{i-1}, so that t_i is not visible from y_{i-1}. It is easy to observe that for every vertex y_i, its piv is $(0,1,0)$ if $i=2l+1$, and $(0,1,1,0)$ if $i=2l$. A placement of $2n-2$ targets into the polygon P with $2n$ vertices, where $n=12$, is depicted in Fig. 10.

This ends the proof.

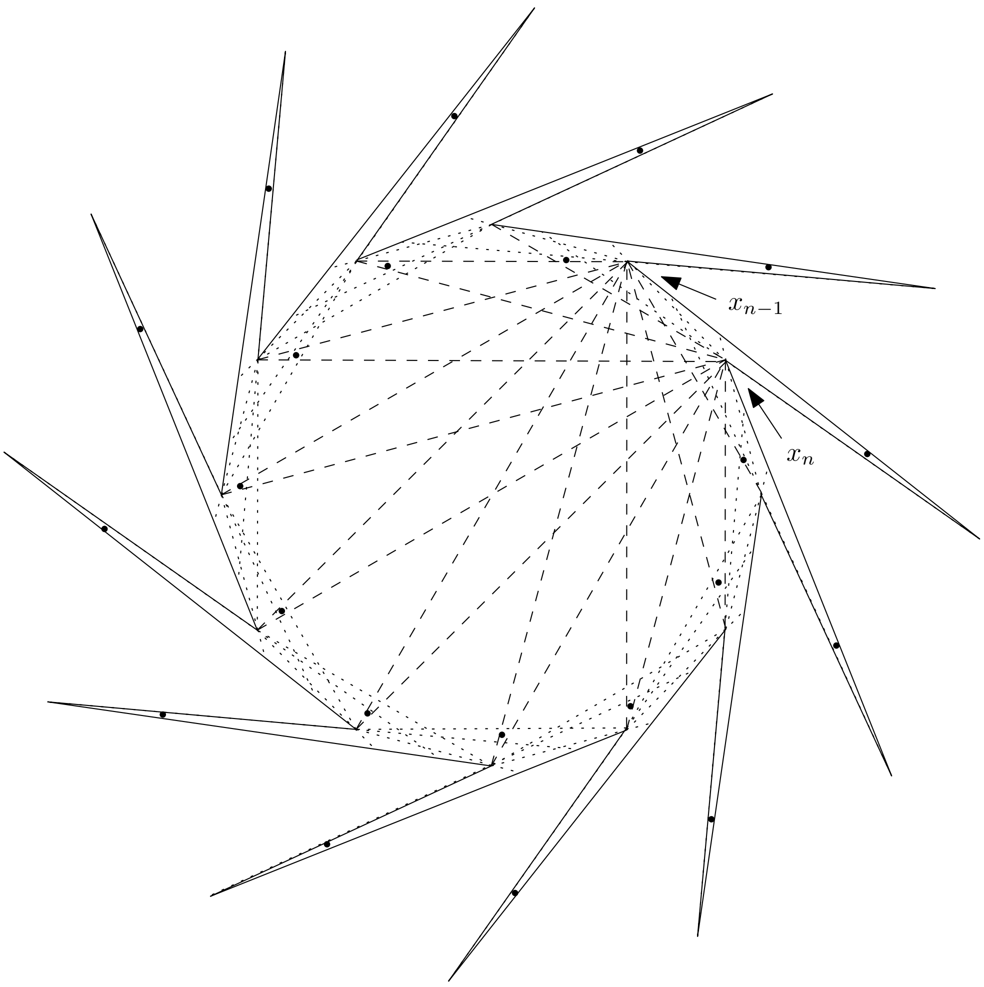

Fig. 10. A placement of $2n-2$ targets into the polygon P from Fig. 8