Bouncing Robots in Rectilinear Polygons

Onur Çağırıcı,[1] Yeganeh Bahoo,[1] and Steven M. LaValle[2]

Abstract— **In this paper, we describe a bouncing strategy (smart strategy) for a mobile robot that uses one bit of memory for feedback, and guarantees that the robot will traverse all the rooms (and doorways) of a 2D environment. The environment is modeled as a rectilinear polygon (also called orthogonal polygon), and the rooms and the doorways are defined by the decomposition algorithm we describe. Such a decomposition helps the robot to not go back to a room after leaving. We also define the notion of "virtual doors" that have the ability to let the robot through, or make the robot bounce from them. We compared three different types of bouncing rules: smart, random, billiard. The smart strategy grantees to reach to target. Although the random strategy on average behaves the same as the smart strategy, there are rectilinear polygons in which the robot cannot reach the target in the expected time steps. On the other hand, the billiard bouncing strategy can cause the robot to become trapped.**

I. INTRODUCTION

We study the problem of navigating a polygonal environment. This environment can be interpreted as a large warehouse with multiple rooms. The robot is not given any cameras or depth sensors that could otherwise measure distances or other geometric properties of the environment, and its goal is to reach to a particular area inside the polygon (or a particular room inside the warehouse). The robot is simply commanded to move straight, and does not change its direction of motion until it hits a wall. After it hits a wall, the robot "bounces" off of that wall, and continues to move to another direction. Given the angle of incidence α, the robot rotates, and orients itself to move to a new direction based on the *bouncing rule*. The bouncing rule can be a function of α, or can be independent.

In Figure 1, a robot R (denoted by a black circle) is moving in the direction $\vec{d_i}$ towards a wall. After hitting the wall, the robot changes its direction to $\vec{d}_{i+1}$. The angle α_i between $\vec{d_i}$ and the wall is referred to as *angle of incidence* or *hitting angle*, and the angle β_i between $\vec{d}_{i+1}$ and the wall is referred to as the *bouncing angle*.

Problem definition In this paper, we study the effect of the bouncing rule on the time taken by the robot to achieve its goal. Given a rectilinear polygon, the goal is to determine a bouncing rule for the robot, such that the robot definitely reaches a pre-defined target area without begin equipped by any type of environmental sensors (cameras etc.).

We concentrate on two well-known bouncing rules, and then propose our own bouncing rule. The first rule is *reflective bounce*, in which the angle of incidence is equal to the angle

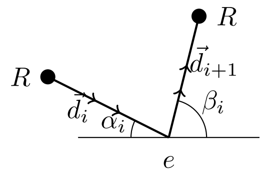


Fig. 1: The robot R with the direction $\vec{d_i}$ hits the wall with the hitting angle α, and then bounces back with the bouncing angle β, changing its direction to $\vec{d}_{i+1}$. This bounce is denoted by (e, α_i, β_i)

of reflection (i.e., billiard ball reflection). The second rule is *random bounce*, in which the robot bounces off of a wall with a random angle, regardless of the angle of incidence. In this sense, the bounce of a robot can be mathematically equivalent to a light ray bouncing off of a (not necessarily flat) mirror. Thus, we can reduce the problem of a robot bouncing off of the wall and reaching at a particular area to a ray reflecting off of a sequence of mirrors and lighting up any point inside a particular area.

II. RELATED WORK

There are several studies in which the problem of navigating a bouncing robot is considered a combinatorial problem. Sanai has described the setting as a dynamic system, and computed the physical and statistical aspects of the environment [1]. Boldrighini et al., on the other hand, studied the computational aspects of the reflection by interpreting the setting as a billiard ball bouncing around in a polygon [2]. As opposed to a billiard ball's reflection (or bouncing) with respect to the rules of physics, there are studies which consider the random reflection inside a polygon [3]. Takorsky studied the possible shots on a polygonal billiard table to shoot the billiard ball to a target point [4], Finding that not every point is reachable for a robot (can be illuminated by a light ray) unless the bouncing rule is random (the diffuse reflection is used).

The studies we mentioned so far consider the reflection of a single ray (or bouncing of a single object). In addition, there are studies which assume that the rays are shot from a single source and reflect everywhere. This notion is called *diffuse reflection*. Ghosh et al. studied the properties of diffuse reflection, and described algorithms to compute the shortest path of a ray from a given source to a given target [5]. Aronov et. al. studied diffuse reflection assuming that the mirror on the boundary of a polygon is a line segment instead of the whole edge [6], [7]. Fox-Epstein et. al. showed that a single light source is able to illuminate a polygon with n edges after at most $\lfloor (n-2)/4 \rfloor$ diffuse reflections [8].

Foldes et. al. studied combinatorial aspects of illuminating a set of object describing a partially ordered set [9]. This

*This research has been supported by the Ryerson University Faculty of Science Dean's Research Fund

[1]Department of Computer Science Ryerson University, Toronto, Canada (bahoo,cagirici)@ryerson.ca

[2] Center for Ubiquitous Computing, University of Oulu, Oulu, Finland steven.lavalle@oulu.fi

study, although not directly related to reflection, introduces the difficulty of computing the "hardness" of a robot to reach to a particular region in a polygon.

The robots that have no environmental sensors were studied deeply in Nilles's PhD dissertation [10], [11]. Alam and Bobadilla also studied the problem of determining an efficient bouncing rule in rectilinear polygons [12] for multiple robots to monitor the environment. However, the goal of these works are slightly different than ours. They aim to find trajectories such that multiple robots patrol in their relative areas without colliding with each other.

The studies we mention all focus on the geometry of reflection (in our case, bouncing) in a polygonal environment. We focus on the geometry of the polygon, and aim to answer the question "What kind of bouncing rule should a robot use to reach its target after a reasonable number of bounces?" instead of "Can the robot ever reach to its target?" In this paper, we describe a bouncing rule that uses only one bit of memory, and lets the robot reach a particular area in a rectilinear polygon given the starting position of the robot, and the polygon.

III. MOTIVATION

Similar to Foldes et. al., we also focus on the "hardness" of a robot to reach to a particular region in a polygon. We use a data structure called "reachability graph" to analyze how hard it is for a robot to reach to a region (target region). To obtain the reachability graph, we first decompose the polygon into subpolygons, and then analyze the relationship between two neighboring subpolygons. The decomposition (or partitioning) of a polygon is done by *bottleneck segments*.

Definition 1 (Bottleneck segment): A bottleneck segment the shortest line segment $\overline{pq}$ drawn between a reflex vertex in the polygon and the boundary of the polygon, such that $\overline{pq}$ lies completely inside the polygon, and does not intersect the boundary except the endpoints.

The main reason behind this decomposition is to define the subpolygons obtained by the decomposition as (virtual) rooms, the bottleneck segments as (virtual) doors, and determine by the central system a sequence of rooms that the robot has to pass through. Initially, all the doors that lead to rooms that are not in the sequence are closed. To navigate the robot from one room to another, we can place doors on the bottleneck segments, and close them once the robot leaves the room. These doors might be physical automatic doors, as well as "virtual" doors that use RFID system to prevent robot from passing through. We explain the decomposition algorithm in detail in Section V.

IV. TERMINOLOGY

This paper considers 2D Euclidean geometry. A *line segment* $\overline{pq}$ is two points p and q, and the set of points on the shortest distance between them. The *angle* $\nabla(\overline{pq})$ of a line segment $\overline{pq}$ is the smallest angle between x-axis and $\overline{pq}$. A *polygonal chain* is a set $\overline{p_1q_1}, \ldots, \overline{p_nq_n} =$ of non-intersecting line segments where $q_i = p_{i+1}$ for $i \in \{1, ..., n\}$. A *simple polygon* is the set of points enclosed by the polygonal chain $\overline{p_1q_1}, \ldots, \overline{p_nq_n}$, plus the line segment $\overline{q_np_1}$. A rectilinear polygon is when $\nabla(\overline{p_iq_i}) \in \{0, \pi/2\}$. These line segments are called *edges* of the polygon, and the endpoints of line segments are the *vertices* of the polygon. We denote a simple polygon by P throughout this paper. The *vertex set*, and the edge set of P are denoted by $V(P)$ and $E(P)$, respectively. The *boundary* of a polygon P is the set of all points that are on $E(P)$, and is denoted by $\partial(P)$. The *interior* of P is the set of points $P \setminus \partial(P)$.

V. DECOMPOSITION

Our decomposition algorithm utilizes the bottleneck segments while partitioning a given polygon. We refer to the partitions as "rooms" or "subpolygons" throught the paper. Such a decomposition allows us to tackle the combinatorial problems by dividing them into sub-problems, i.e., the robot leaving a rectangular room. Since we close the door after the robot, if we can guarantee that the robot will leave any kind of rectangular room. Then, we can apply the same solution to every room in the polygon to navigate the robot to its target.

Let us describe our algorithm for partitioning a rectilinear subpolygon. Afterwards, we give examples on the utilization of the decomposition.

```
Input: A simple polygon $P = (V(P), E(P))$
Output: A set $\mathcal{P}$ of polygons where $\bigcup_{P_i \in \mathcal{P}} = P$
1  $\mathcal{P} \leftarrow \emptyset$;
2  $\Gamma \leftarrow \emptyset$ /* Set of virtual doors */
3  foreach reflex vertex $v$ of $P$ do
4      $d_{min} \leftarrow \infty$;
5      $s_{min} \leftarrow$ NULL;
6      foreach edge $e$ of $P$ do
7          $p \leftarrow$ closest point on $e$ to $v$;
           // $p$ can be a vertex of $P$
8          $s \leftarrow \overline{pv}$;
9          if $s$ completely lies inside $P$ AND $s \notin E(P)$ AND
             $s \notin \Gamma$ then
10             if $s.length < d_{min}$ then
11                 $d_{min} \leftarrow s.length$;
12                 $s_{min} \leftarrow s$;
13             end
14         end
15     end
16     $\Gamma \leftarrow \Gamma \cup \{s_{min}\}$;
17 end
18 Let $u, v, w, x$ be arbitrary vertices;
19 if $\exists \overline{uv} \in \Gamma \mid \nabla(\overline{uv}) \notin \{0, \pi/2\}$ then $\Gamma \leftarrow \Gamma \setminus \{\overline{uv}\}$ ;
   /* Remove the doors that are not
      horizontal or vertical to preserve
      rectilinearity. */
20 if $\exists (u, v, w, x) \mid \overline{uv}, \overline{vw}, \overline{wx} \in \Gamma$ then
21     $\Gamma \leftarrow \Gamma \setminus \{\overline{uv}, \overline{wx}\}$;
       /* If there exists a polygonal chain
          of three virtual doors, then
          remove two of them. */
22 end
23 foreach segment $s \in \Gamma$ do $\mathcal{P} \leftarrow \mathcal{P} \cup$ SPLIT(P,s);
   /* Iteratively, decompose the polygon
      into subploygons by cutting it
      through $s$. */
24 return $\mathcal{P}$
```

Algorithm 1: Decomposition algorithm

Algorithm 1 takes a simple polygon as input, and return a set of subpolygons derived from partitioning the input polygon. In Lines 1 and 2, we initialize the set of subpolygons and the set of virtual doors as empty sets, respectively. From Line 3 to Line 17, we iterate on each reflex vertex v in the polygon and find the minimum line segment from that vertex to a point inside the polygon. In Lines 4 and 5, we initialize an integer as inifinity, and a line segment as null for storing the virtual door. From Line 6 to Line 15, we iterate on the edges of the polygon to determine which one is the closest to the current reflex vertex. For each edge, we find the closest point p in Line 7, and we draw the segment $\overline{pv}$, and store it in s. In Line 9, we check if the computed segment s lies completely inside the polygon, is not an edge, and is not already a virtual door. If s passes all tests, then finally we check if the length is minimum, and store the minimum distance and minimum length segment in variables d_{min} and s_{min}, respectively in Lines 11 and 12. After we find the minimum segment for a reflex vertex, we add that segment into the set of virtual doors in Line 16.

When the algorithm reaches 17, we have a set of virtual doors which are drawn from every reflex vertex to the boundary of the polygon. There might be two problems with this set of virtual doors. The first problem is when the slope of a virtual door is not 0, $\pi/2$, $3\pi/2$, or 2π. Since we work with rectilinear polygons, we remove such doors in Line 19 to preserve rectilinearity. The second problem is that there might be a polygonal chain of three virtual doors, rendering two of them redundant. In this case, we remove the first and the last edge of this chain in Line 21. After we finish our post-processing on the doors, we iterate on each virtual door, and split the polygon to create subpolygons in line 23.

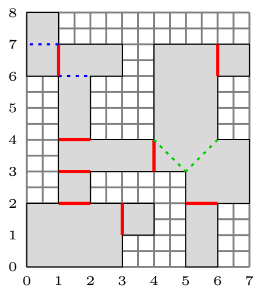


Fig. 2: A polygon decomposed by Algorithm 1. The virtual doors are denoted by bold, red line segments. The dotted line segments indicate that they were eliminated by the algorithm (Lines 20 – 22). The blue segments were eliminated because they form a polygonal chain of length 3, and the green segments were eliminated because they are not orthogonal.

Proposition 1: Given a (not necessarily rectilinear) simple polygon P and three non-collinear points $a, b, c \in \partial(P)$, if there exist two virtual doors ab and bc that are generated by Algorithm 1, then the triangle $\triangle abc$ lies completely inside P.

Proof: Since the algorithm picks virtual doors from the segments that lie inside P completely, then only ac has the potential to intersect $\partial(P)$. Let i be an intersection point between $\partial(P)$ and ac. Without loss of generality, let i be the first intersection point while traveling from a to c. Then, ai is shorter than ac since the intersection point i lies on ac, somewhere in-between. Therefore, the algorithm would pick ai as a virtual door rather than ac. ∎

Proposition 2: The bottleneck segments computed by Algorithm 1 do not intersect, even for a non-rectilinear polygon.

Proof: Denote two intersecting virtual doors by ab and cd. Without loss of generality, let (a, c, b, d) be the clockwise order of the endpoints of the virtual doors, let a and d lie on x-axis of the coordinate system, and let a be at $(0, 0)$. We proceed by analyzing the cases of this scenario.

***Case 1** ad and bc are edges:* If $|ab| < |ac|$, then by triangle inequality, $|bd| < |ac|$ must hold. Therefore, the algorithm picks bd instead of cd as a virtual door. Analogously, if $|cd| < |bd|$ holds, then ac is picked as a virtual door.

***Case 2** ad is an edge, c is a vertex, and b is a point on an edge:* If $|cd| < |bd|$, then by triangle inequality, $|ac| < |ab|$ holds (remember the clockwise order). Thus, the algorithm picks ac instead of ab. If, on the other hand, $|ab| < |ac|$ holds, then $|bd| < |cd|$ holds.

***Case 3** Neither ad nor bc is an edge:* This case is equivalent to Case 2. In addition, there is possibility of picking bc and ad as virtual doors. ∎

We could equip the polygonal environment with a central system which controls whether a robot goes through or bounces back from a virtual door. Although there are many ways to operate such a system, our virtual sensor could be implemented by a single RFID system. Our proposed system is as follows: Every robot is equipped with an RFID receiver, and every virtual doors is equipped with a device that is capable of sending RFID signals with multiple frequencies. The RFID receivers have a very low threshold such that they activate once the robot is very close to a virtual door. If the robot receives an RFID signal from a bandwidth which it listens to, then it bounces back. Otherwise, the robot continues to move as if there are no obstacles.

Since we do not want a robot to return a subpolygon from which it got out, we close the door after it leaves. Thus, the robot will avoid redundant bounces and carry on.

When there is a single robot bouncing around, the system is fairly simple. The virtual doors become active if the door is closed, and passive otherwise. However, once there are multiple robots, this approach does not work since we do not want to close the door for every other robot once one of them passes through.

VI. Bouncing Strategy

We describe the simplest form of bouncing strategy for the robot, which uses one bit of memory, and then show that how we can extend the strategy if additional resources are available.

Our goal is to make the robot traverse every sub-polygon obtained by Algorithm 1. The strategy is deterministic, uses only the information on the length of the narrowest virtual

door ℓ, the distance between the door and the opposite wall h, and guarantees the robot to traverse every sub-polygon. Whenever the robot hits a wall, it is able to flip a bit in its memory.
Let k be the bit which the robot flips every time it hits a wall. We call this rule as "smart bouncing" as it lets the robot go slow and steady for very narrow doors. The rule is

$$(1-k)\cdot(\pi/2-\alpha)+k\cdot(\pi/2-\alpha+\theta), \qquad (1)$$

where α is the hitting angle, and $\theta = \arcsin\left(\frac{\ell}{\sqrt{\ell^2+h^2}}\right)$, ℓ is the length of the door, and h is the distance between the door and the opposite wall. If this information is not updated, then the robot is given the minimum value of θ for every ℓ and h, in the beginning.
For the sake of simplicity, let us pose the problem as follows.

Definition 2 (Room escape problem): Given a rectangular room with a door, and a bouncing robot inside this room, how can we guarantee that the robot will reach to the door regardless of its starting position and initial movement vector?

If the robot uses symmetrical bouncing rule (i.e., the angle of incidence is equal to the angle of bounce), then the robot can be trapped in the room, bouncing off of the same walls at the same points. In Figure 3a, we see such a case. The robot can never reach to the door since it makes the same six bounces until its battery is drained.

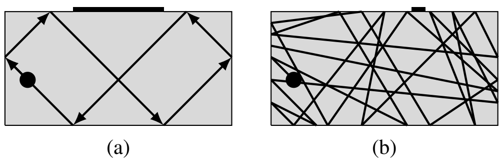


Fig. 3: (a) An example of a robot getting stuck in a rectangle with the symmetrical bouncing strategy $\beta_i = \alpha_i$. (b) An example of a robot having a very hard time to get out of a rectangle with the random bouncing strategy.

Remark 3: The random bouncing robot will eventually escape from any polygonal room as long as there is a door. However, when the number of bounces is taken into account, it is not optimal to use the random bouncing rule if the door is too narrow.

We propose the bouncing rule given in Equation (1), which guarantees that the robot to reach the door. The idea behind that strategy is as follows. The robot goes back and forth between two parallel horizontal (respectively, vertical) walls. While doing so, it covers the distance horizontally (respectively, vertically) on the walls. The distance that the robot covers is slightly smaller than the length of the door. This behavior continues when the robot changes from horizontal (respectively, vertical) to vertical (respectively, horizontal) walls. Because the distance covered each time is less than the length of the door, the robot will reach the door after finite number of bounces. This strategy can also be used for a robot to patrol all the subpolygons (decomposed by Algorithm 1) in a rectilinear polygon.

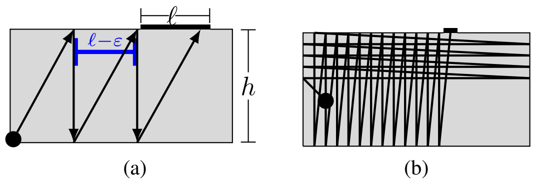


Fig. 4: (a) Smart bouncing for a robot R. ℓ denotes the length of the virtual door, h denotes the distance between the door and the opposite wall. (b) An example of a robot reaching to a narrow door with the smart bouncing strategy.

Definition 4 (Polygon patrolling problem): Given a decomposed rectilinear polygon, and a bouncing robot inside this polygon, how can we guarantee that the robot will enter every room regardless of its starting position and initial movement vector?

Proposition 3: Given a rectilinear polygon that is decomposed into subpolygons by Algorithm 1, a robot that follows the smart bouncing rule is guaranteed to traverse all the subpolygons.

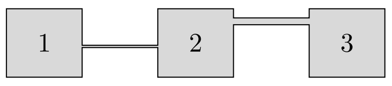
If we set the angle of rotation according to the smallest value of $\frac{\ell}{\sqrt{\ell^2+h^2}}$ for all ℓ and h, then the robot is guaranteed to leave any room it is in, because the distance it covers on a wall will always be smaller than the length of the door. Leaving a certain room means entering another one, since the subpolygons produced by the decomposition algorithm pairwise share exactly one edge (which is the virtual door). Following this pattern, the robot will leave every room it is in, and go into adjacent room. Once the robot leaves a room i, and enters the room j, the virtual door between i and j is closed behind the robot, if j has another adjacent room except i. In case the only room that is adjacent to j is i, then this means that j is a dead-end, and there are no other rooms to go except i. In that case, the virtual door between i and j is left open until the robot goes back to i.

VII. REACHABILITY GRAPH

Using the subpolygons $\mathcal{P} = \{P_1, \ldots, P_k\}$ obtained by Algorithm 1, we describe two metrics, called the *escapability factor* and the *reachability factor*. These metrics determine the "likelihood" of a robot to be able to escape the subpolygon it is currently in, and reach a certain subpolygon from its current position, respectively.
The escapability factor of a robot from a subpolygon P_i is computed by the formula $\Phi(i) = |\ell_j^i|/|w(\ell_j^i)|$, where $|\ell_j^i|$ is the length of the j^{th} virtual door in subpolygon P_i, and $|w(\ell_j^i)|$ is the length of the wall that ℓ_j^i is on. One can deduce that if $|\ell_j^i| = |w(\ell_j^i)|$, then the robot will surely escape the subpolygon, as one of our assumptions is that the hitting angle is never π.

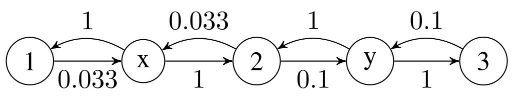
Remark 5: Since one end of every virtual door is on a reflex vertex, and the polygon is a simple polygon without holes, there is a unique order of subpolygons that a robot must pass through between any non-adjacent subpolygons.

These two metrics, namely the escapability factor and the reachability factor, have a relationship. The reachability factor is computed in a similar fashion of computing probability. We compute the reachability factor multiplying the values of the escapability factor of each subpolygon, regardless of the bouncing rules. Suppose that the robot is in P_1 and the goal is to reach P_k. Without loss of generality, suppose that P_1 and P_k do not share any virtual doors, and let $(P_2, \ldots, P_{k_1})$ be series of subpolygons for the robot to pass to reach P_k. Then, the reachability factor of the robot from P_1 to P_k is $\Psi(1,k) = \Pi_i^{k-1}\Phi(i)$.
To demonstrate this computation better, let us give an example. Let there be three big rectangular subpolygons side by side with gaps in-between, closest ones connected with a very narrow rectangular subpolygon. For the sake of simplicity, let us refer to the big subpolygons as chambers, and the narrow ones as doorways.



(a)

$$\begin{tikzcd} 1 \arrow[r, "0.033"'] & x \arrow[l, bend right, "1"'] \arrow[r, "1"'] & 2 \arrow[l, bend right, "0.033"'] \arrow[r, "0.1"'] & y \arrow[l, bend right, "1"'] \arrow[r, "1"'] & 3 \arrow[l, bend right, "0.1"'] \end{tikzcd}$$



(b)

Fig. 5: (a) Three rooms connected by two narrow doorways. (b) The reachability graph of the polygon given in.

In Figure 5a, there are three rooms whose dimensions are identical, numbered $1, 2, 3$, and two narrow doorways connecting them. When Algorithm 1 runs on this polygon, the virtual doors are two short openings at the ends of each doorway. Without loss of generality, let the doorways be a, b, c, d beginning from the leftmost door. For example, let the long edges of the polygons be 30 units, let the widths of a and b be 1 unit, and let the widths of c and d be 3 units. That is, $\Phi(1) = 0.03$, and $\Phi(2) = 0.1$. Since the doorways have virtual doors at both ends, and the virtual doors cover the whole wall, the escape factor of the doorways is 1. Now, let us analyze the example given. Suppose that the a robot starts moving in chamber 1. Then, the reachability factor of this robot to chamber 3 is $\Psi(1,3) = 0.033 \cdot 1 \cdot 0.1 \cdot 1 = 0.003$. The reachability graph is an edge weighted and directed graph built where each subpolygon, regardless of being a doorway or a chamber, is a vertex in the graph, and there exists an edge between two subpolygons that share a virtual door. The graph for the polygon given in Figure 5a, see Figure 5b. The chambers are labeled by $1, 2, 3$, and the doorways are labeled by x and y.
When we obtain the reachability graph, we can decide on the bouncing strategy of the robot. If the subpolygon that we want the robot to reach has low reachability factor, then having a random bouncing rule or the billiard bouncing rule might not be the best choice. Thus, we let our robot have the smart bouncing strategy given in Equation 1. On the other hand, if the reachability factor is high, then the smart bouncing might lead to some redundant bounces, wasting the limited power. In that case, the robot follows either billiard or random bounce. The best result in terms of time spent to reach a subpolygon would be to change the bouncing rule for each escapability factor of each subpolygon. This can be achieved via the central system feeding the robot with proper bouncing rule. However, the robot then needs more than one bit of memory.
In short, we cannot claim a particular bouncing rule to be the optimal. As each bouncing rule has trade-offs with the others, the optimum choice of the bouncing rule can be determined after analyzing the structural properties of the polygon.

VIII. COMPUTED EXAMPLES

We conduct simulations with three different robot types. (i) **The reflecting robot**, which behaves like a billiard ball, i.e., angle of incidence is equal to the angle of reflection. (ii) **The random robot**, which can bounce off of an edge in any direction, and the angle of incidence does not play a role. (iii) **The smart robot**, which uses the bounce rule given in (1).
In our simulations, we select a starting room that the robots start in, and a target room that the robots are to reach. We compare three types of robots based on the number of bounces they make until they reach to the target room. We pick far away rooms as starting room and the target room. We run the simulations in the polygon given in Figure 2.
In Figure 6, we see the polygon partitioned by Algorithm 1, rooms colored differently and numbered from 0 to 8. The robot uses the smart bouncing rule. On the left hand side, it starts from Room 5 and goes to Room 8. On the right hand side, it starts from Room 2 and goes to Room 6. The green line segments represent the open doors. The sequence of subpolygons that the robot will follow is $5, 4, 7, 1, 0, 8$.
When starting from the same position with the same direction of motion, the reflecting bounce rule and the smart bounce rule output a deterministic series of bounces. Thus, we conduct the simulations that concern these rules only once.

Above simulations show that the smart bouncing rule, although guarantees to reach to the target room, might lead the robot to make extra bounces compared to reflecting bouncing rule. Since the random bouncing is not deterministic, above examples do not accurately represent a comparison between two bouncing rules. Therefore, we conduct 500 simulations and report the average results. We set a limit of 100 bounces, and after 500 bounces, we assume that the robot's battery is dead. The robot starts from a random point in the starting room, with random movement vector. In the first set of simu-

	5-8	2-6
Rf	16	25
Sm	26	40
Rn	27.4	32.7

TABLE I: The number of bounces for different types of bouncing rules.

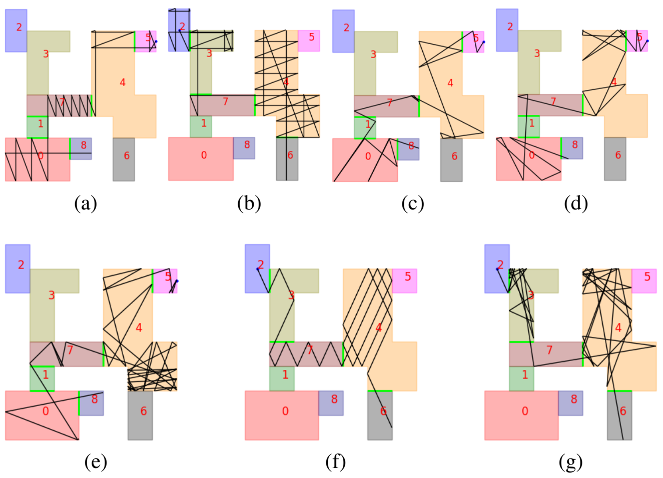


Fig. 6: (a) Smart bouncing rule. From Room 5 to Room 8. 26 bounces. (b) Smart bouncing rule. From Room 2 to Room 6. 40 bounces (c)-(e) Random bouncing rule. From room 5 to Room 8. 21, 25 and 46 bounces are done, respectively. (f)-(g) Reflective bouncing rule. From Room 2 to Room 6. 25 and 46 bounces are done, respectively.

lations, the robot starts from Room 5. The target is Room 8. The random bouncing robot reached to its destination in all 100 tests under 500 bounces. The average number of bounces is 27.4. This result indicates that the difference between the random bouncing rule and the smart bouncing rule is not very significant, and smart bouncing rule is preferable in case where the doors are too narrow. Moreover, due to the stochastic nature of random bouncing rule, the robot might make redundant bounces.

For the next set of simulations, the robot starts from Room 2, and the target is Room 6. With the smart bouncing rule, the robot reaches its target after 40 bounces, whereas the reflecting bounce reaches to the target after 26 bounces. On the average, the robot with the random bouncing rule makes 32.7 bounces.

A summary of the performances is given in Table I. Rf, Sm and Rn are the reflective, smart and random bouncing rules, respectively. The column labeled 5-8 indicate the simulations run when the robots start from 5 and the target is 8. Analogously, 2-6 indicate that the robot starts from Room 2 and the target is Room 6.

IX. CONCLUSION AND FUTURE WORK

In this paper we have studied the robot with limited sensors which can bounce of the surfaces once it hit them in the presence of (virtual) doors which can be manipulated. We have described an algorithm to identify the doors, and partition the polygon using these doors.

Utilizing these doors are crucial while navigating a robot which has no sensors. However, the doors alone will not suffice in some cases, such as the door between two rooms is too narrow (i.e., the escapability factor is less than 0.1). We introduced a bouncing rule, called smart bouncing, which uses only one bit of memory. With this rule, the robot is guaranteed to leave a rectangular room regardless of the size of the door. Thus, when the doors close automatically, a robot with the smart bouncing rule definitely reaches its target after finite number of bounces. With the same reasoning, the robot can patrol every room in a rectinlinear polygon with opening and closing the doors in the correct order.

The smart bouncing rule, compared with the other two type of bouncing rules, does not have a clear advantage in terms of number of bounces when the escapability factors are greater than 0.1 unit. What separates the smart bouncing rule from the others is that the guarantee for the robot to leave a room, even when the escapability factor is much smaller than 0.1 unit. We believe that our proposed bouncing rule paves the way for different application areas, and can also be utilized in different types of polygons (e.g., polygons with holes, non-rectilinear polygons) by some trivial modifications. Although the number of the bounces are equal, unlike others, the smart bouncing rule guarantees the escape.

For an extension of this work, we plan to work on the central system that controls the virtual doors for multiple robots bouncing around in the same polygon. Our future work includes simulating three types of bouncing rules in polygons with different characteristics, such as abundantly many rooms, very narrow doors, labyrinth-like shape.

REFERENCES

[1] Y. G. Sanai, "Dynamical Systems with Elastic Reflections," *Russian Mathematical Surveys*, vol. 25, no. 2, pp. 137–189, 1970.

[2] C. Boldrighini, M. Keane, and F. Marchetti, "Billiards in Polygons," *The Annals of Probability*, vol. 6, no. 4, pp. 532–540, 1978.

[3] A. Q. Nilles, Y. Ren, I. Becerra, and S. M. LaValle, "A Visibility-Based Approach to Computing Nondeterministic Bouncing Strategies," in *Algorithmic Foundations of Robotics XIII*, 2020, pp. 89–105.

[4] G. W. Tokarsky, "Polygonal Rooms Not Illuminable from Every Point," *The American Mathematical Monthly*, vol. 102, no. 10, pp. 867–879, Dec. 1995.

[5] S. K. Ghosh, P. P. Goswami, A. Maheshwari, S. C. Nandy, S. P. Pal, and S. Sarvattomanda, "Algorithms for computing diffuse reflection paths in polygons," *The Visual Computer*, vol. 28, no. 12, pp. 1229–1237, 2012.

[6] B. Aronov, A. R. Davis, T. K. Dey, S. P. Pal, and D. C. Prasad, "Visibility with One Reflection," pp. 13–23, 1992.

[7] B. Aronov, A. R. Davis, T. K. Dey, S. P. Pal, and D. Prasad, "Visibility with Multiple Reflections," in *Discrete and Computational Geometry*, vol. 20, 1998, pp. 61–78.

[8] E. Fox-Epstein, C. D. Tóth, and A. Winslow, "Diffuse Reflection Radius in a Simple Polygon," vol. 76, pp. 910–931, 2016.

[9] S. Foldes, I. Rival, and J. Urrutia, "Light sources, obstructions and spherical orders," *Discrete Mathematics*, vol. 102, no. 1, pp. 13–23, 1992.

[10] A. Q. Nilles, "Designing boundary interactions for simple mobile robots," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2020.

[11] A. Q. Nilles, A. Pervan, T. A. Berrueta, T. D. Murphey, and S. M. LaValle, "Information Requirements of Collision-Based Micromanipulation," in *Algorithmic Foundations of Robotics XIV*. Springer International Publishing, 2021, pp. 210–226.

[12] T. Alam and L. Bobadilla, "Multi-Robot Coverage and Persistent Monitoring in Sensing-Constrained Environments," *Robotics*, vol. 9, no. 2, 2020.