A Visibility-Based Approach to Computing Nondeterministic Bouncing Strategies

Alexandra Q. Nilles[1], Yingying Ren[1], Israel Becerra[1], and Steven M. LaValle[2,1]

[1] Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA
{nilles2, yren17, israelb4, lavalle}@illinois.edu
[2] Faculty of Information Technology and Electrical Engineering
University of Oulu, Oulu, Finland

Abstract. Inspired by motion patterns of some commercially available mobile robots, we investigate the power of robots that move forward in straight lines until colliding with an environment boundary, at which point they can rotate in place and move forward again; we visualize this as the robot "bouncing" off boundaries. Different boundary interaction rules can be defined for such robots, such as one that orients the robot relative to its heading prior to collision, or relative to the normal of the boundary. We introduce a new data structure, the *bounce visibility graph*, which is generated from a polygonal environment definition. The bounce visibility graph can be queried to determine the feasibility of path-based tasks such as navigation and patrolling, assuming we have unavoidable nondeterminism in our actuation. If the task is feasible, then this approach synthesizes a strategy (a sequence of nondeterministic rotations). We also show how to compute stable cyclic trajectories and use these to limit uncertainty in the robot's position. [3]

1 Introduction

Mobile robots have rolled smoothly into our everyday lives, and can now be spotted vacuuming our floors, cleaning our pools, mowing our grass, and moving goods in our warehouses. These tasks require that the robot's path achieve certain properties; for example, a vacuuming robot should cover the whole space while not visiting any particular area more frequently than others. A robot that is monitoring humidity or temperature in a warehouse should repeat its path consistently so data can be compared over time. In many such examples, the strategies for controlling the robot's path may be decoupled from the specific application, so we envision building a library of useful high-level path controllers based on desired dynamical system properties.

Current algorithmic approaches to these tasks take two flavors: 1) maximizing the information available to the robot though high-fidelity sensors and map-generating algorithms such as SLAM, or 2) minimizing the information needed

[3] Code and documentation available at https://github.com/alexandroid000/bounce_viz

by the robot, such as the largely random navigation strategies of the early robot vacuums. The first approach is powerful and well-suited to dynamic environments, but also resource-intensive in terms of energy, computation, and storage space. The second approach is more resource efficient, but does not immediately provide general-purpose strategies for navigation and loop-closure. We propose a combined approach: first, global geometry of environment boundaries is provided to the system, either *a priori* or calculated online by an algorithm such as SLAM. The global geometry is then processed to produce a strategy providing strong formal guarantees and which can be executed with minimal processing power and only low bandwidth local sensors, such as bump and proximity sensors, instead of high bandwidth sensors such as cameras.

We consider simple robots with "bouncing" behaviors: robots that travel in straight lines in the plane, until encountering an environment boundary, at which point they rotate in place and set off again. The change in robot state at the boundary is modelled by what we call *bounce rules*. These interactions may be mechanical (the robot actually makes contact with a surface), or may be simulated with virtual boundaries (such as the perimeter wire systems used by lawn mowing robots [**?**]). Physical implementations of the bouncing maneuver have been validated experimentally [**?**,**?**]. Often these lines of work consider a subset of the strategy space, such as an iterated fixed bounce rule. In this work, we present a tractable approach to reasoning over *all possible* bounce strategies, generalizing previous tools for analyzing a few given bounce rules.

This paper presents three ideas for simplifying the characterization and generation of paths of such mobile robots. The first crucial aspect of our approach is that we assume that the robot has some intrinsic nondeterminism, so we generate nondeterministic strategies such that if the robot executes any action in a family at each stage, the strategy will succeed. We can then analyze the size of the bounce rule sets at each stage to determine the minimum required accuracy of a successful robot design. Our second contribution is that by using the geometric structure of a polygonal environment, we can create a combinatorial representation of the environment that lets us reason over a finite number of families of paths, instead of the infinite collection of all possible paths. Finally, we provide results on when transitions in this robotic system are *contracting*: two potential robot states under the transition move closer together. This property can be used to control uncertainty through actuation, and we provide the first steps toward leveraging this property.

2 Related Work

We incorporate techniques from computational geometry, specifically visibility [**?**]. Visibility has been considered extensively in robotics, but usually with the goal of avoiding obstacles [**?**,**?**]. To plan over collisions, we use the edge visibility graph, analyzed in [**?**], and shown to be strictly more powerful than the vertex visibility graph. Our work is also related to problems that consider what parts of a polygon will be illuminated by a light source after multiple reflections (as

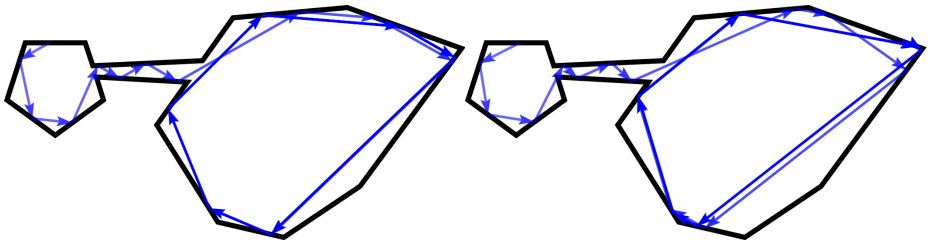

Fig. 1: Two paths produced by different sequences of bounces, which visit different points of ∂P, yet have the same sequence of edge collisions and high-level dynamical behavior (escape the room on the left, travel through hallway, then patrol the room on the right in a periodic orbit).

if the edges of the polygon are mirrors) [**?**], or with diffuse reflections [**?**], which are related to our nondeterministic bounces.

Our robot motion model is related to *dynamical billiards* [**?**]. Modified billiard systems have attracted recent interest [**?**,**?**,**?**]. One similar work was inspired by the dynamics of microorganisms; in [**?**], the authors show that *Chlamydomonas reinhardtii* "bounce" off boundaries at a specific angle determined by their body morphology. They characterize periodic and chaotic trajectories of such agents in regular polygons, planar curves, and other environments.

Our motion model is a form of *compliant motion*, in which task constraints are used to guide task completion, even when the exact system state is not known. Our use of contraction mappings and nondeterministic reasoning is related to the idea of funnelling: using the attraction regions of a dynamical system to guide states into a goal region. These ideas have been developed in the context of manipulation and fine motion control by Whitney [**?**], Mason [**?**], Erdmann [**?**], Goldberg [**?**], Lozano-Pérez, Mason, and Taylor [**?**], Lynch and Mason [**?**], and Burridge, Rizzi, and Koditschek [**?**], among many others.

Our intentional use of collisions with environment boundaries is enabled by the advent of more robust, lightweight mobile robots. Collisions as information sources have also been recently explored for multi-robot systems [**?**]. The first-class study of the dynamical properties of bouncing was proposed in [**?**] and continued in [**?**]. Here we extend and improve these analysis tools, and incorporate visibility properties to discretize the strategy space.

In [**?**] and [**?**], the authors describe navigation, coverage, and localization algorithms for a robot that rotates a fixed amount relative to the robot's prior heading. Due to the chosen state space discretization, periodic trajectories may exist which require bounce angles not allowed by the discretization. By using a discretization induced by the environment geometry, we are able to find all possible limit cycles, and the controllers necessary to achieve them. Another closely related work is [**?**], which considered navigation for a robot that has our same motion model, with uncertainty given as input to the algorithm. Instead of taking error bounds as input, our approach considers all possible amounts of uncertainty, and returns bounds on the required accuracy for strategies.

We are working toward a generalization and hierarchy of robot models, in a similar spirit to [**?**]. Their *simple combinatorial robot* is able to detect all visible vertices and any edges between them, and move straight toward vertices. By augmenting this basic robot model with sensors they construct a hierarchy of these robot models. Our questions are related but different: if the robot is given a compact, purely combinatorial environment representation, what tasks can it accomplish with minimal sensing?

3 Model and Definitions

We consider the movement of a point robot in a simple polygonal environment, potentially with polygonal obstacles. All index arithmetic for polygons with n vertices is mod n throughout this paper. We do not require polygons in general position. We do not consider trajectories that intersect polygon vertices. We define our robot to move forward in a straight line, until encountering an environment boundary (such as when a bump or range sensor is triggered). Once at a boundary, the robot is able to rotate in place. More formally, the model is:

- The *configuration space* $X = \partial P \times S^1$. P is a simple polygon, potentially with holes. P has boundary ∂P, the union of external and obstacle boundaries. S^1 is the robot's orientation in the plane. Let s refer to a point in ∂P without an associated robot orientation.
- The *action space* $U = [0, \pi]$, where $u \in U$ is the orientation the robot takes when it reaches an environment boundary, before moving forward again. u is measured counterclockwise relative to the boundary tangent.
- The *state transition function* $f : X \times U \to X$, which describes how actions change the state of the robot. We will often lift this function to act nondeterministically over sets of states and actions, propagating each state forward under each possible action, and unioning the resulting set of states.
- We model time as proceeding in stages; at each stage k, the robot is in contact with the environment boundary, executes an action u_k, and then moves forward until the next contact with the boundary at stage $k + 1$.

Definition 1. *Let $\alpha \in (0, \pi)$ be the robot's incoming heading relative to the environment boundary at the point of contact. Let $\theta \in (0, \pi)$ be a control parameter. A **bounce rule** b maps α and θ to an action u, and determines how the robot will reorient itself when it collides with a boundary. Bounce rules are defined in the frame in which the environment normal is aligned with the positive y axis and the robot's point of contact with the boundary is the origin.*

Definition 2. *A **nondeterministic bounce rule** is a bounce rule lifted to return a set of actions. We will restrict nondeterministic bounce rules to return a convex set of actions (a single interval in $(0, \pi)$).*

For example, a specular bounce (laser beams, billiard balls) has bounce rule $b(\alpha, \theta) = \pi - \alpha$. See Figure **??** for more examples of bounce rules.

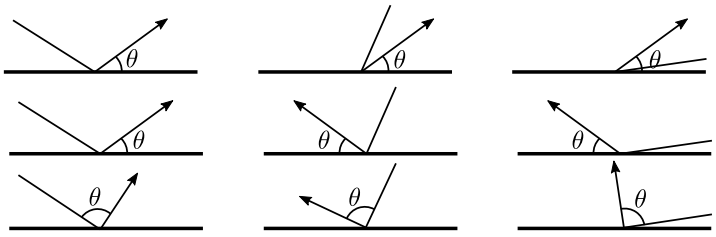

Fig. 2: Examples of different "bounce rules" that can be implemented on mobile robots. In the first row, $b(\alpha, \theta) = \theta$, which we refer to as a **fixed** bounce rule. In the second row, we have a **monotonic fixed** bounce rule, in which $b(\alpha, \theta) = \theta$ or $\pi - \theta$, depending on what is necessary to preserve monotonicity of travel direction along the x axis. In the third row, we have a **relative** bounce rule, $b(\alpha, \theta) = \alpha - \theta$, rotating α through θ in the clockwise direction. If this rotation causes the heading of the robot to still be facing into the boundary, the robot performs the rotation again until its heading points into the free space.

Definition 3. *A **bounce strategy** is a sequence of nondeterministic bounce rules.*

Of course, robots rarely move perfectly, so our analysis will assume the robot has some nondeterminism in its motion execution. Instead of modelling explicit distributions, we assume the robot may execute any action in a set. Planning over such nondeterminism can result in design constraints for robots; for example, if a robot has an uncertainty distribution over its actions in the range $u \pm \epsilon$, the largest allowable ϵ will be half the width of the smallest convex bounce rule interval in a strategy.

4 Visibility-Based Boundary Partitioning

Given a polygonal environment (such as the floor plan of a warehouse or office space), we would like to synthesize bounce strategies that allow a robot to perform a given task (such as navigation or patrolling). To do so, we first discretize the continuous space of all possible transitions between points on ∂P. We find a visibility-based partition that encodes the idea of some points on ∂P having different available transitions.

Definition 4. *The **visibility polygon** of a point s in a polygon P is the polygon formed by all the points in P that can be connected to s with a line segment that lies entirely within P.*

Imagine a robot sliding along the boundary of a polygon, calculating the visibility polygon as it moves. In a convex polygon, nothing exciting happens. In a nonconvex polygon, the reflex vertices (vertices with an internal angle greater than π) cause interesting behavior. As the robot slides, its visibility polygon

mostly changes continuously. Edges shrink or grow, but the combinatorial structure of the polygon remains the same, until it aligns with a reflex vertex r and another vertex v (visible from r). At this point, either v will become visible to the robot, adding an edge to the visibility polygon, or v will disappear behind r, removing an edge from the visibility polygon.

To compute all such points at which the visibility polygon changes structure, we compute the **partial local sequence**, defined in [**?**]. Each point in the partial local sequence marks the point at which a visible vertex appears or disappears behind a reflex vertex. The sequence is constructed by shooting a ray through each reflex vertex r from every visible vertex and keeping the resulting sequence of intersections with ∂P. See Figure **??** for an example of the vertices in the partial local sequence of v_0.

Once all the partial local sequences have been inserted into the original polygon, the resulting segments have the property that any two points in the segment can see the same edge set of the original polygon (though they may see different portions of those edges). See Algorithm **??** for a pseudocode description of this partitioning process. Algorithm **??** applies to polygons with or without holes; holes require more bookkeeping to correctly find visible vertices and shoot rays. See Figure **??** for an example partition of a polygon with holes. Let P' be the polygon P after application of Algorithm **??**.

Algorithm 1 PartitionPoly(P)

Input: A polygon P as a list of vertices in counterclockwise order.
Output: P': P with all partial local sequence points added as new vertices.

```
1: v_new ← {}
2: reflex_verts ← GetReflexVerts(P)
3: for v_r in reflex_verts do
4:     for v_vis in VisibleVerts(P, v_r) do
5:         v_new ← v_new ∪ ShootRay(v_vis, v_r)
6:     end for
7: end for
8: P' ← InsertVerts(v_new, P)
9: return P'
```

The ShootRay function takes two visible vertices v_1 and v_2 and compute the first intersection of ∂P and ray v_1v_2. This operation will take $O(\log n)$ time after preprocessing the polygon P in $O(n)$ [**?**]. The VisibleVerts function computes all visible vertices in the polygon given an input query vertex, and takes $O(n)$ [**?**]. So the total runtime of Algorithm 1 is $O(n^2 \log n)$.

Definition 5. *The **edge visibility graph** of a polygon P has a node for each edge of P, and has an arc between two nodes (e_i, e_j) if and only if there is a point s_i in the open edge e_i and a point s_j on the open edge e_j such that s_i and s_j can be connected with a line segment which is entirely within the interior of P.*

Definition 6. *Let the **bounce visibility graph** be the directed edge visibility graph of P'.*

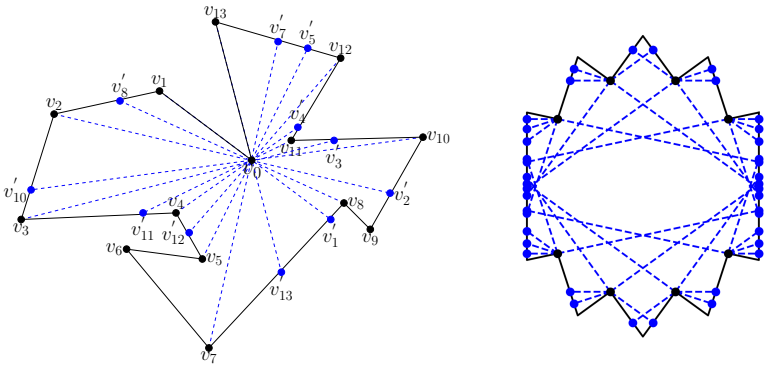

Fig. 3: On the left, the partial local sequence for v_0. On the right, an example polygon for which the bounce visibility graph has $O(n^4)$ edges.

Although visibility is a symmetric property, we use directed edges in the bounce visibility graph so that we can model the geometric constraints on the visibility from one edge to another, which are not symmetric and govern what actions allow the robot to accomplish that transition. See Section **??** for further exploration of this idea.

We now introduce the **bounce visibility diagram**, a helpful representation of the vertex-edge visibility structure of a partitioned polygon P'. If $|\partial P'|$ is the perimeter length of $\partial P'$, let the x axis of the bounce visibility diagram be the interval $[0, |\partial P'|)$, in which the endpoints of the interval are identified. Let the y axis of the diagram be a parameter θ, which is an angle between 0 and π. For a point $s \in \partial P'$, we can compute all the visible vertices and the angle at which they are visible. The graph of all angles at which vertices are visible from points in $\partial P'$ is the bounce visibility diagram.

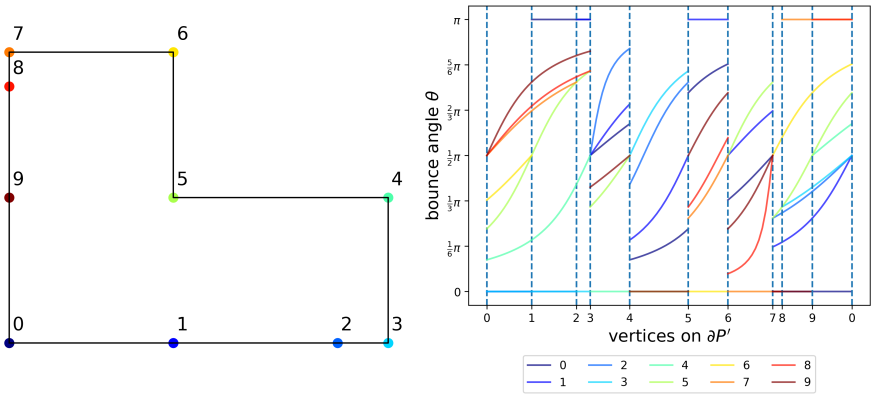

Fig. 4: A partitioned polygon and its corresponding bounce visibility diagram.

The bounce visibility diagram gives us some visual insight into the structure of the problem. First, it shows our motivation for the specific partitioning we have chosen: at each inserted vertex, another vertex appears or disappears from

view. Of course, such transitions also occur at the original vertices of P. Thus, the partition induced by the vertices of P and our inserted partial local sequences captures all combinatorial changes in the visible vertices.

Additionally, this diagram gives us some insight into types of possible segment-to-segment transitions under ranges of departure angles. For example, slicing the diagram vertically at a specific $s \in \partial P'$ produces a list of vertices visible from that point on the boundary (the combinatorial visibility vector) [**?**]. The vertical dotted lines, vertices of P', are the exact points at which the combinatorial visibility vector (cvv) changes. Within a segment (v_i, v_{i+1}), if two successive elements of the cvv are adjacent vertices, then *some* transition exists from (v_i, v_{i+1}) to a point on the edge between those vertices.

If for a segment (v_i, v_{i+1}), if there is an interval of θ such that no vertex lines are crossed from left to right, this corresponds to our notion of *safe actions*; the robot can transition from segment to segment under some amount of nondeterministic error in position and action. For example, in Figure **??**, we see that a band of intervals exists at the top and bottom of the diagram with no "crossings" as you move from left to right (moving the robot around ∂P). These bands correspond to the safe actions to be formally described in Proposition **??**.

Proposition 1. *The bounce visibility graph for a polygon with n vertices has $O(n^2)$ vertices and $O(n^4)$ edges.*

The proof of Proposition **??** is left to the appendix.

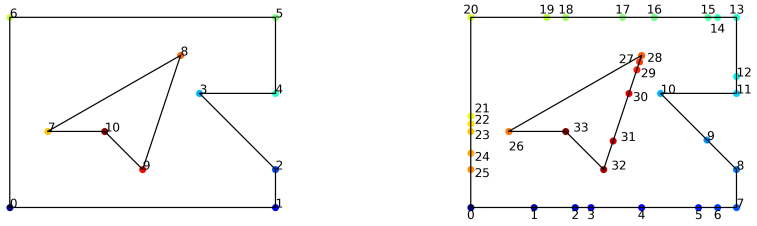

Fig. 5: An example partition for a polygon with holes; our discretization scheme extends naturally to handle static obstacles.

5 Safe Actions

To characterize some families of paths, we will use the boundary partition technique defined in Section **??**, then define *safe actions* between segments in the partition that are guaranteed to transition to the same edge from anywhere in the originating edge. Such actions define transitions which keep the robot state in one partition under nondeterministic actions.

Definition 7. *Two edges e_i, e_j of a polygon are **entirely visible** to each other if and only if every pair of points $s_i \in e_i$ and $s_j \in e_j$ are visible (the shortest path between s_i and s_j lies entirely within P).*

Definition 8. *A **safe action** from edge e_i to edge e_j in a polygon is an action u such that $f(s,u) \in e_j$ for any $s \in e_i$ and u in some interval of actions $\tilde{\theta} \subseteq (0,\pi)$.*

Proposition 2. *Given two entirely visible line segments $e_i = (v_i, v_{i+1})$ and $e_j = (v_j, v_{j+1})$ in $\partial P'$, if a safe action exists from e_i to e_j, the maximum interval of safe actions is $\tilde{\boldsymbol{\theta}} = [\boldsymbol{\theta_r}, \boldsymbol{\theta_l}]$ such that $\theta_r = \pi - \angle v_j v_{i+1} v_i$ and $\theta_l = \angle v_{j+1} v_i v_{i+1}$.*

For full proof, see the appendix. The sketch of the proof is that if θ_r is smaller than θ_l, every ray shot from e_i at $\theta \in [\theta_r, \theta_l]$ must intersect with e_j.

Note that this definition of a safe action is similar to the definition of an interval of safe actions from [**?**]; the main differences in approach are the generation of boundary segments, methods of searching the resulting graph, and how we generate constraints on robot uncertainty instead of assuming uncertainty bounds are an input to the algorithm.

Lemma 1. *If two edges in $\partial P'$ are entirely visible to each other, then there will be at least one safe action between them.*

See appendix for proof.

Corollary 1. *If two edges in $\partial P'$ share a vertex that is not reflex, and the two edges are not collinear, then there exist safe actions from one to the other in both directions.*

Proof. See Figure **??**.

Definition 9. *Given two entirely visible segments e_i and e_j, rotate frame such that e_i is aligned with the $x-$axis with its normal pointing along the positive $y-$axis, such that segment e_j is above segment e_i. If the intersection of segment e_i and e_j would be on the left of segment e_i, then call the transition from e_i to e_j a **left transition**; if the intersection would be on the right of segment e_i, then call the transition a **right transition**.*

Proposition 3. *For every polygon P and the resulting partitioned polygon P' under Algorithm **??**, each edge $e \in P'$ has at least two safe actions which allow transitions away from e.*

Proof. Let $e_i = (v_i, v_{i+1})$. Consider right transitions from e_i to some e_k, where the safe action interval $\tilde{\theta} = (0, \theta_l)$ for some nonzero θ_l. We will show that such a transition must exist.

By Corollary **??**, if an edge e_i has an adjacent edge e_{i+1} which is not collinear or separated by a reflex angle, $e_k = e_{i+1}$ and a safe transition exists between e_i and e_{i+1} with $\theta_l = \angle v_{i+2} v_i v_{i+1}$. See Figure **??**.

If the adjacent edge is at a reflex angle, Algorithm **??** will insert a vertex in line with e_i on the closest visible edge, forming edge e_k. If v_i is an original vertex of P, e_k will be entirely visible from e_i, since Algorithm **??** will otherwise insert a point in v_i's partial local sequence.

If v_i is itself inserted by Algorithm **??**, e_k will still be entirely visible. There must be some original vertex of P clockwise from v_i and collinear with e_i, which

would insert a vertex visible to e_i through Algorithm **??** if there were any reflex vertices blocking transitions to e_k. See Figure **??** for an example of the geometry.

If the adjacent edge e_{i+1} is collinear with e_i, take the first non-collinear edge and apply the above reasoning to find e_k. All the same arguments extend symmetrically to left transitions, which will have safe actions of the form $\tilde{\theta} = [\theta_r, \pi)$. Thus, each edge will have two guaranteed safe actions leading away from it. □

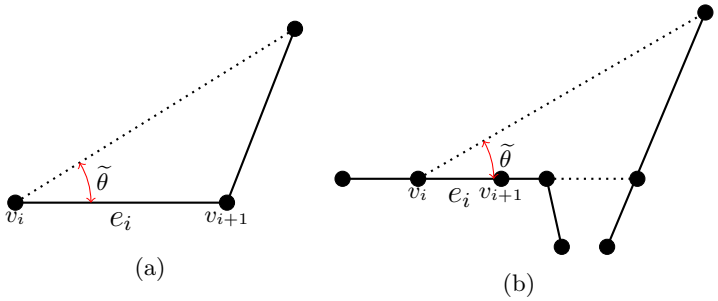

Fig. 6: Examples of geometrical setup for some guaranteed safe actions.

6 Dynamical Properties of Paths

Many tasks required of mobile robots can be specified in terms of dynamical properties of the path the robot takes through space: stability, ergodicity, etc. Our motion strategy allows us to disregard the robot's motion in the interior of P. Instead, the robot's state is an interval along the boundary ∂P, and we can formulate transitions as a discrete dynamical system under the transition function f. The properties of this dynamical system can be used to engineer paths corresponding to different robot task requirements.

One generally useful property of mapping functions is *contraction*: when two points under the function always become closer together. We can use this property to control uncertainty in the robot's position by entering stable limit cycles.

Definition 10. *Let $\boldsymbol{\phi_{i,j}}$ be the interior angle between two edges $e_i, e_j \in \partial P$.*

Definition 11. *A function g that maps a metric space M to itself is a **contraction mapping** if for all $x, y \in M$, $|g(x) - g(y)| \leq c|x - y|$, and $0 \leq c < 1$.*

Lemma 2. *If the transition from segment e_i to segment e_j is a left transition, then the transition function $f(x, \theta)$ between segments e_i and e_j is a contraction mapping if and only if $\theta > \frac{\pi}{2} + \frac{\phi_{i,j}}{2}$; if a right transition, the transition function $f(x, \theta)$ is a contraction mapping if and only if $\theta < \frac{\pi}{2} - \frac{\phi_{i,j}}{2}$.*

In our case, we will always take M to be an interval in $\mathbb{R}$ and we will use the L1 norm. The proof is a straightforward application of the law of sines, and is included in the appendix.

Corollary 2. *For all pairs of adjacent segments with internal angle less than π, there exists a range of actions for which f is a contraction mapping.*

Definition 12. *The **contraction coefficient** of a mapping is the ratio of the distance between points before and after the mapping is applied. Let $C(\theta, \phi_{i,j})$ be the contraction coefficient of a transition from e_i to e_j in ∂P For a left transition, $C(\theta, \phi_{i,j}) = |\frac{\sin(\theta)}{\sin(\theta - \phi_{i,j})}|$; for a right transition, $C(\theta, \phi_{i,j}) = |\frac{\sin(\theta)}{\sin(\theta + \phi_{i,j})}|$.*

See the proof of Lemma **??** in the appendix for derivation of C. For a sequence of transitions $f_0, \ldots, f_k$, we can construct the overall mapping from the domain of f_0 to the range of f_k through function composition. Since f is a linear mapping, the contraction coefficient of a composition of multiple bounces can be determined by multiplying the contraction coefficients of each bounce.

Definition 13. *Given a sequence of m feasible transitions $F = \{f_0, f_1, \ldots, f_{m-1}\}$, at stage k the robot will be located on edge $e(k)$ and will depart the edge with action θ_k; the contraction coefficient of the overall robot trajectory $f_{m-1} \circ \ldots \circ f_0$ is $C(F) = \prod_{k=0}^{m-1} C(\theta_k, \phi_{e(k),e(k+1)})$.*

If $C(F) < 1$, the composition of F is a contraction mapping. This is true even if some transition along the way is not a contraction mapping, since $C(F)$ is simply the ratio of distances between points before and after the mapping is applied. Furthermore, the value of $C(F)$ tells us the exact ratio by which the size of the set of possible robot states has changed.

Limit Cycles: A contraction mapping that takes an interval back to itself has a unique fixed point, by the Banach fixed point theorem [**?**]. By composing individual transition functions, we can create a self-mapping by finding transitions which take the robot back to its originating edge. A fixed point of this mapping corresponds to a stable limit cycle. Such trajectories in regular polygons were characterized in [**?**]. Here, we present a more general proof for the existence of limit cycles in all convex polygons.

Definition 14. *$\phi_{P,min}$ is the smallest interior angle in a polygon P.*

Theorem 1. *For all convex polygons with n edges, there exist constant fixed-angle bouncing strategies which result in a period n limit cycle regardless of the robot's start position.*

Proof. See Figure **??** in the appendix for the geometric setup.

Without loss of generality, assume $\theta \in (0, \frac{\pi}{2}]$. The robot will always bounce to the next adjacent edge if and only if $\theta \in (0, \min_i(\angle v_{i+2} v_i v_{i+1}))$.

Suppose we have two start positions s_1 and s_2 on edge e_0 and a constant fixed bounce rule $b(\alpha, \theta) = \theta$. At stage k, s_1 and s_2 will be located at $f^k(s_1, \theta)$ and $f^k(s_2, \theta)$.

By Definition **??**, the distance between s_1 and s_2 changes after one orbit of the polygon by the ratio $\frac{|f^n(s_1,\theta) - f^n(s_2,\theta)|}{|s_1 - s_2|} = \prod_{i=0}^{n-1} C(\theta, \phi_{i,i+1})$. If this ratio is less than one, then $f^n(s, \theta)$ has a unique fixed point by the Banach fixed-point

theorem [**?**]. By Lemma **??**, this constraint is satisfied if $\theta < \frac{\pi}{2} - \frac{\phi_{i,i+1}}{2}$ for all i. We can guarantee that this condition holds for the orbit by requiring

$$\theta \in (0, \min(\min_{i=0,1,\ldots,n-1}(\angle v_{i+2}v_iv_{i+1}), \min_{i=0,1,\ldots,n-1}(\frac{\pi}{2} - \frac{\phi_{i-1,i}}{2}))),$$

in which case the fixed-angle bouncing strategy with $b(\alpha, \theta) = \theta$ leads to a convex cycle which visits each edge of P sequentially, regardless of the robot's start position. Symmetry applies for orbits in the opposite direction. □

Proposition 4. *For all points s on the boundary of all polygons, a constant fixed-angle controller exists which will cause the robot's trajectory to enter a stable limit cycle.*

Proof. First we observe that by Proposition **??**, for every segment $e \in \partial P'$, safe actions always exist for two action intervals. These intervals are the ones bordering the segment itself: by staying close enough to the boundary, the robot may guarantee a safe transition. By Lemma **??**, these safe actions will also admit contraction mappings. Thus, we may choose a constant fixed-angle controller such that it results in safe, contracting transitions from all segments in P'. Since there are a finite number of segments in P', this controller must result in limit cycle from every point in P. If s is on a hole of the polygon, this procedure will cause the robot to leave the hole and enter a cycle on the exterior boundary. □

7 Applications

7.1 Navigation

Here, we define the *navigation* task, assuming that the robot has unavoidable uncertainty in its actuation. A navigation strategy should take a robot from any point in a starting set to some point in a goal set, as long as some action in the bounce rule set is successfully executed at each stage.

Definition 15. ***Navigation**: Given a polygonal environment P, a convex starting set $S \subset \partial P$ of possible robot positions along the environment boundary, and a convex goal set $G \subset \partial P$, determine a strategy π which will be guaranteed to take the robot from any point in S to a point in G, or determine that no such strategy exists.*

Using the formalisms built up so far, we can perform the navigation task as a search over a discrete bounce visibility graph. Shortest paths in the graph will correspond to paths with the fewest number of bounces. It is important to note that an arc $e_i \to e_k$ in the bounce visibility graph only implies that for each point $x \in e_i$ there exists an action taking x to some point in e_k. For our navigation task, we require a *range* of angles which can take *any* point in e_i to a point on e_k, so we restrict the arcs of the bounce visibility graph to those corresponding to safe actions only. As may be expected, not all edges of $\partial P'$ are reachable using safe actions.

Proposition 5. *There exist simple polygons such that under Algorithm* **??**, *there exist edges in the partitioned polygon* P' *that are* ***not*** *reachable by a safe action from any other edge in* P'.

Proof. See Figure **??** for an example in which the edge counterclockwise from vertex 10 is not reachable via safe actions. Equivalently, node 10 in G_{safe} has no incoming arcs.

The only such segments will be segments for which both endpoints are reflex vertices or vertices inserted by Algorithm **??**, since by Corollary **??**, segments adjacent to other vertices of P will be reachable by a safe action. Thus, navigation using paths in G_{safe} is not complete: we cannot get everywhere safely, at least under this partitioning of ∂P.

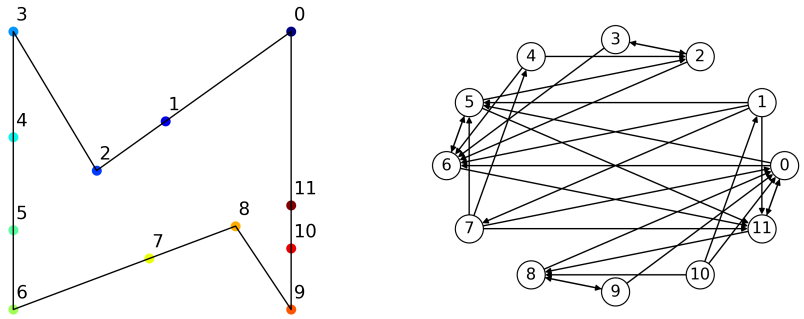

Fig. 7: A polygon after Algorithm **??** and its safe bounce visibility graph BVG_{safe}.

However, it is still useful to explore what is possible under nondeterministic control of such a robot. Here, we will show how to search for a constant fixed-angle bounce controller, where at every stage the robot executes a fixed-angle bounce in some range $\tilde{\theta}$. This is an extension of the controllers analyzed in [**?**] for regular polygons. We define a few helper functions:

- MKBVG: Uses the visibility information generated in Algorithm **??** to generate the bounce visibility graph in $O(n^4)$ time.
- MKSAFEBVG: Using Definition **??** and Proposition **??**, we can create a *safe roadmap*, G_{safe}, out of the bounce visibility graph by traversing all edges and removing edges with an empty $\tilde{\theta}$, and labelling the remaining edges with the interval of safe actions.
- SEARCHCONSTANTFIXEDSTRATS: performs breadth-first search from start to goal, starting with $\tilde{\theta} = (0, \pi)$ and intersecting $\tilde{\theta}$ with the safe action intervals along each path, terminating when all start states reach a goal state at the same stage with overlapping $\tilde{\theta}$ intervals. Returns the resulting constant controller $\tilde{\theta}$ or that no such strategy is possible.

Code implementing Algorithm **??** and many example polygons can be found at https://github.com/alexandroid000/bounce_viz.

Algorithm 2 SafeConstantFixedNavigate(P, S, G, k)

Input: A polygon P, intervals S and G in ∂P, and an integer bound k.
Output: A nondeterministic bounce rule, or None if no strategy can be found.

$P' \leftarrow$ PartitionPoly(P)
$BVG \leftarrow$ mkBVG(P')
$BVG_{safe} \leftarrow$ mkSafeBVG(BVG)
$\tilde{\theta} \leftarrow$ SearchConstantFixedStrats(BVG_{safe}, S, G, k)
return $\tilde{\theta}$

7.2 Patrolling

Note that there are exponentially many possible limit cycles, since a cycle may contain transitions which are not contraction mappings, as long as the overall contraction coefficient is less than one. Also, Proposition **??** shows that at least one type of limit cycle is reachable from all points on ∂P for all polygons, under an open interval of constant fixed controllers. As more becomes understood about the structure and robustness of such limit cycles, we can sketch some possibly useful robotic tasks under the umbrella of *patrolling*:

Definition 16. *Given an environment P, a set of possible starting states S, and a sequence of edges of the environment $E = \{e_1, \ldots, e_k\}$, determine a strategy which causes the robot to enter a stable cycle visiting each edge of the sequence in order from any point in S.*

This task is related to the Aquarium Keeper's Problem in computational geometry [**?**]. It may be solved by coloring nodes in the bounce visibility graph by which edge of P they belong to, and then searching for safe cycles which visit edges in the correct order. If a cycle is contracting, it will result in a converging stable trajectory.

Interesting open questions remain on how to incorporate other useful properties of a patrolling cycle, such as coverage of a space with certain sensors, or guarantees about detecting evaders while patrolling.

Leveraging Cycles to Reduce Uncertainty: Say no safe path exists between two subsets of ∂P. We may use limit cycles to reduce the uncertainty in the robot's position enough to create a safe transition. By Proposition **??**, limit cycles are reachable from any point $s \in \partial P$ under a constant fixed-angle controller (and we know the range of angles from which this constant controller may be drawn). Say the contraction coefficient of a given limit cycle is C, and the length of edge e_i is ℓ_i. After k iterations of the cycle, the distance between the robot's position and the fixed point of the cycle's transition function is less than $C^k \ell_i$. Thus, we may reduce the uncertainty in the robot's position to less than ϵ after $\lceil \log(\epsilon/\ell_i)/\log(C) \rceil$ iterations of the cycle. If instead of a deterministic fixed-angle controller, we consider a nondeterministic controller, we need to reason over a range of C coefficients and resulting attractor structure. The mathematical theory of these attractors and the best way to choose such nondeterministic controllers is still an open question.

8 Open Questions and Future Work

We have presented a visibility-based approach to reasoning about paths and strategies for a class of mobile robots. We are moving toward understanding nondeterministic control of such robots; however, many open questions remain!

Comparing bounce rules. Our approach can be used to compare different families of bounce strategies in a given polygon, by comparing the reachability of the transition systems induced by each strategy family. This would require either reducing the full bounce visibility graph given constraints on possible bounces, or constructing a more appropriate transition system. It is not clear the best way to analyze relative bounce rules, in which the outgoing angle of a robot after a collision is a function of the incoming angle.

Optimal Strategies. Say we wish to find the strategy taking the robot from region S to region G with the maximum amount of allowed uncertainty in the bounce rules (the sum of the interval sizes of each action set along the path). This problem can be framed as an optimization problem over the space of all transition function compositions.

Localization. A localization strategy is a nondeterministic strategy that produces paths which reduce uncertainty in the robot's position to below some desired threshold, from arbitrarily large starting sets. The use of limit cycles to produce localizing strategies has been explored in [**?**], and it would be interesting to take a similar approach with our environment discretization.

Ergodic Trajectories. We are interested in strategies that produce ergodic motion, where the robot does not spend "too much" time in any given part of the state space. Measures of ergodicity have recently been used in exploration tasks [**?**]. Chaotic dynamical systems have also been used directly as controllers for mobile robots [**?**].

Acknowledgement: This work was supported by NSF grants 1035345 and 1328018, and CONACyT post-doctoral fellowship 277028.

9 Appendix

A Proof of Proposition ??

Proposition 1. *The bounce visibility graph for a simple polygon with n vertices has $O(n^2)$ vertices and $O(n^4)$ edges.*

Proof. Consider a polygon P with n vertices, r of which are reflex vertices. To construct the bounce visibility graph, we insert the vertices of the partial local sequence for each vertex in P. For a convex vertex, its partial local sequence will not add any new vertices to P. However, a reflex vertex can add $O(n)$ new vertices.

Up to half of the vertices in the polygon can be reflex, so the complexity of r is $O(n)$. Therefore, the number of vertices in P', returned by Algorithm **??** is $O(n^2)$. Each vertex indexes an edge in P', and thus a node in the edge visibility graph of P'. At most, a given vertex in P' may see all other vertices, so in the worst case, the bounce visibility graph will have $O(n^4)$ edges. See Figure **??** for an example of such a polygon. □

A.1 Worst Case Example for Algorithm ??

We might hope that if r is large, then not all of the reflex vertices will produce a large number of new vertices, and we may bound the size of the edge set in the visibility graph. Unfortunately, the number of reflex vertices, the new vertices produced in their partial local sequence, and the new vertices' visibility can be large at the same time. We will present a family of input polygons with bounce visibility graph edge-set size of $O(n^4)$.

Let $n = 4t+2$, in which t is a positive integer. We design a polygon with $r = 2t$ reflex vertices. The polygon is symmetric with respect to its medium horizontal line. In the top half, the reflex vertices are uniformly located on a circle and thus they are visible to each other; the convex vertices are chosen so that they are outside the circle and the line through an edge will not intersect other edges. Each reflex vertex will have at least $t-1$ new vertices in its partial local sequence. There will be $2t(t-1)+n$ vertices in the polygon after we insert all new vertices in the partial local sequence for all reflex vertices. Each of them can see at least $t(t-1)+n/2$ other vertices. Thus the number of edges in the transition graph for the polygon with inserted vertices is $O((2t(t-1)+n)(t(t-1)+n/2)) = O(t^4) = O(n^4)$. Fig **??** shows the polygon for $t = 4$ with all the vertices in the partial local sequences.

B Proof of Proposition ??

Proposition 2. *Given two entirely visible line segments $e_i = (v_i, v_{i+1})$ and $e_j = (v_j, v_{j+1})$, if a safe action exists from e_i to e_j, the maximum range of safe actions is $\tilde{\boldsymbol{\theta}} = [\boldsymbol{\theta_r}, \boldsymbol{\theta_l}]$ such that $\theta_r = \pi - \angle v_j v_{i+1} v_i$ and $\theta_l = \angle v_{j+1} v_i v_{i+1}$.*

Proof. Let edge $e_i = (v_i, v_{i+1})$ be aligned with the positive x axis with the clockwise endpoint at the origin, without loss of generality. Due to the edges being entirely visible, $e_j = (v_j, v_{j+1})$ must be in the top half of the plane, above e_i.

Take the quadrilateral formed by the convex hull of the edge endpoints. Let the edges between e_i and e_j be $e_l = (v_i, v_{j+1})$ and the right-hand edge $e_r = (v_{i+1}, v_j)$. Let θ_l be the angle between e_l and the positive x axis ($0 < \theta_l < \pi$); similarly for e_r and θ_r. See Figure **??** for an illustration of the setup.

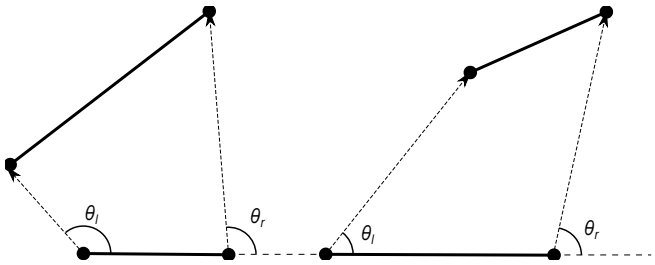

Fig. 8: Angle range such that a transition exists for all points on originating edge (left: such a range exists, right: such a range does not exist)

There are three cases to consider: if e_l and e_r are extended to infinity, they cross either above or below edge e_i, or they are parallel.

Case 1: e_l and e_r meet below edge e_i. In this case, $\theta_l > \theta_r$ and if a ray is cast from any point on e_i at angle $\theta \in [\theta_r, \theta_l]$, the ray is guaranteed to intersect e_j in its interior.

Case 2: e_l and e_r meet above edge e_i. In this case, $\theta_l < \theta_r$, and there is no angle θ such that a ray shot from *any* point on e_i will intersect e_j. To see this, imagine sliding a ray at angle θ_l across the quadrilateral - at some point before reaching v_{i+1}, the ray must stop intersecting e_j, else we would have $\theta_l > \theta_r$.

Case 3: e_l and e_r are parallel. This implies that $\theta_l = \theta_r$, which is the only angle for which a transition from any point on e_i is guaranteed to intersect e_j, and $\tilde{\theta}$ is a singleton set.

Thus, for each case, we can either compute the maximum angle range or determine that no such angle range exists. □

C Proof of Lemma 1

Lemma 1. *If two segments are entirely visible to each other, there will be at least one safe action between them.*

Proof. From the proof of Proposition **??**, we can see that if case one holds in one direction, case two will hold in the other direction, so a safe action must exist from one edge to the other in one direction. If case three holds, there is a safe action both directions but $\tilde{\theta}$ is a singleton set.

D Proof of Lemma ??

Lemma 2. *If the transition from segment e_i to segment e_j is a left transition, then the transition function $f(x, \theta)$ between segments e_i and e_j is a contraction mapping if and only if $\theta > \frac{\pi}{2} + \frac{\phi_{i,j}}{2}$; otherwise, the transition function $f(x, \theta)$ is a contraction mapping if and only if $\theta < \frac{\pi}{2} - \frac{\phi_{i,j}}{2}$.*

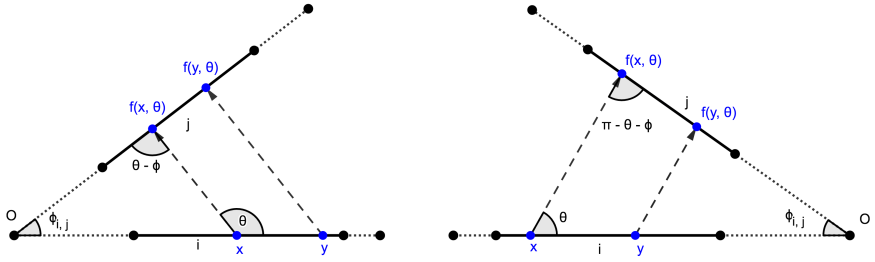

Fig. 9: The two cases for computing contraction mapping conditions.

Proof. We consider the two cases of transition separately:

1. For the transition shown in the left hand side of Figure **??**, $\overline{xf(x,\theta)} \parallel \overline{yf(y,\theta)} \Rightarrow \frac{|f(x,\theta)-f(y,\theta)|}{|x-y|} = \frac{|f(x,\theta)|}{|x|} = \frac{\sin(\pi-\theta)}{\sin(\theta-\phi_{i,j})} = \frac{\sin(\theta)}{\sin(\theta-\phi_{i,j})}$. The transition will be contraction if and only if $\frac{|f(x,\theta)-f(y,\theta)|}{|x-y|} < 1 \iff \sin(\theta) < \sin(\theta - \phi_{i,j})$. If $\theta < \frac{\pi}{2}$, then $\sin(\theta) > \sin(\theta - \phi_{i,j})$. Thus we need $\theta > \frac{\pi}{2}$. If $\theta - \phi_{i,j} > \frac{\pi}{2}$, then $\sin(\theta) < \sin(\theta - \phi_{i,j})$ and we are done; otherwise we need $\pi - \theta < \theta - \phi_{i,j} \Rightarrow \theta - \frac{\phi_{i,j}}{2} > \frac{\pi}{2}$. Combining all conditions, we have the transition will be contraction if and only if $\theta > \frac{\pi}{2} + \frac{\phi_{i,j}}{2}$.
2. Similarly, for a right transiton shown in the right diagram of figure **??**, $\overline{xf(x,\theta)} \parallel \overline{yf(y,\theta)} \Rightarrow \frac{|f(x,\theta)-f(y,\theta)|}{|x-y|} = \frac{|f(x,\theta)|}{|x|} = \frac{\sin(\pi-\theta)}{\sin(\pi-\theta-\phi_{i,j})} = \frac{\sin(\theta)}{\sin(\theta+\phi_{i,j})}$. The transition will be contraction if and only if $\frac{|f(x,\theta)-f(y,\theta)|}{|x-y|} < 1 \iff \sin(\theta) < \sin(\theta + \phi_{i,j})$. If $\theta > \frac{\pi}{2}$, then $\sin(\theta) > \sin(\theta + \phi_{i,j})$. Thus we need $\theta < \frac{\pi}{2}$. If $\theta + \phi_{i,j} < \frac{\pi}{2}$, then $\sin(\theta) < \sin(\theta + \phi_{i,j})$ and we are done; otherwise we need $\theta < \pi - \theta - \phi_{i,j} \Rightarrow \theta < \frac{\pi}{2} - \frac{\phi_{i,j}}{2}$. Combining all conditions, we have the transition will be contraction if and only if $\theta < \frac{\pi}{2} - \frac{\phi_{i,j}}{2}$.

□

E Supplementary Figure for Theorem ??

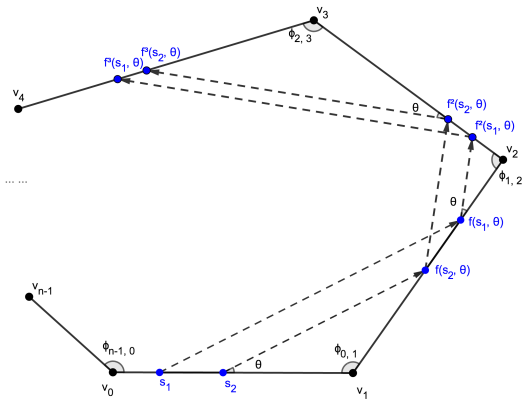

Fig. 10: The notation setup for the proof of contracting cycle in a convex polygon.