Guaranteed navigation with an unreliable blind robot

Jeremy S. Lewis and Jason M. O'Kane

***Abstract*— We consider a navigation problem for a robot equipped with only a map, compass, and contact sensor. In addition to the limitations placed on sensing, we assume that there exists some bounded uncertainty on rotations of our robot, due to precision errors from the compass. We present an algorithm providing guaranteed transitions in the environment between certain pairs of points. The algorithm chains these transitions together to form complete navigation plans. The simplicity of the robot's design allows us to concentrate on the nature of the navigation problem, rather than the design and implementation of our robotic system. We illustrate the algorithm with an implementation and simulated results.**

I. Introduction

The ability to navigate reliably through a cluttered environment is a fundamental capability for mobile robots. Navigation can be a challenging problem because of the dual difficulties of finding a path from the robot's starting location to its goal, and of executing such a path successfully in spite of unpredictable actuation and limited sensing. Typical navigation methods take a decoupled approach, in which *path selection* and *path execution* are handled separately. The former phase chooses a path for the robot to follow without considering sensing issues, and the latter uses the robot's sensors to execute the chosen path. The primary limitation of that approach is that it is unsuitable for situations in which the robot must choose its path, or portions thereof, specifically to reduce or eliminate uncertainty.

In this paper, we present a unified approach that considers uncertainty directly in the process of path selection. Our approach has parallels to prior work on coastal navigation [26], but applies in a *minimalist* setting, considering a robot equipped with no sensors other than a compass and a contact sensor. Our study of this very simple robot model is motivated by the obvious desire to understand how navigation problems can be solved with simple, inexpensive robots, but also by a broader interest in understanding what information is truly required to complete the navigation task.

Although prior work has considered similar robot models for other tasks [24], [25], in this paper, we consider a much more realistic model for robot motion that includes substantial errors, and show that many navigation problems can still be solved under this model.

The basic intuition of the algorithm is to find a sequence of jumps, called high-level transitions, between corners in the environment. Each high-level transition is composed of repeated back-and-forth motions between the incident edges of the target vertex. These motions make progress toward the target, but cannot be guaranteed to reach it at any particular step because of the possibility of motion errors. If progress is made with each motion, however, the robot can guarantee to become arbitrarily close to the target node as the number of motions increases. To determine whether such a jump is possible, we use a formal notion of the *preimage* of the target vertex. The interesting feature of these transitions is that they tolerate uncertainty well—during their execution, the robot does not know its own position exactly—but terminates only after the robot has re-localized itself in a new place.

J. S. Lewis and J. M. O'Kane are with the Department of Computer Science and Engineering, University of South Carolina, 301 Main St., Columbia, SC 29208, USA. {lewisjs4,jokane}@cse.sc.edu

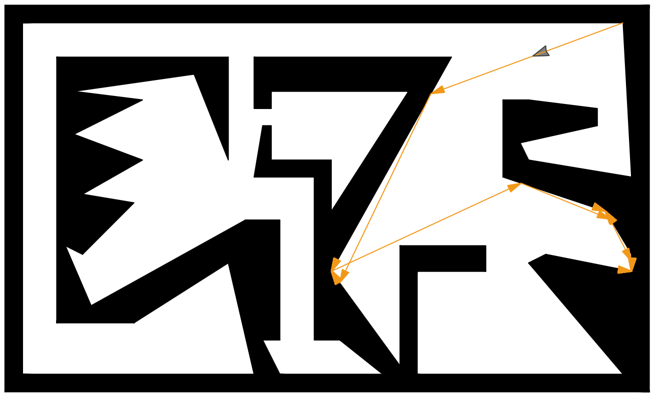

Fig. 1. A robot in a complex environment executing a plan generated by our algorithm. The robot uses convex corners to reduce its state uncertainty several times throughout the plan.

By finding pairs of environment vertices between which such high-level transitions can be made, the algorithm forms a directed graph of high-level transitions, through which it then searches for a complete navigation plan.

The remainder of the paper is structured as follows. In Section II, we discuss related research. Section III gives a formal problem statement. Our algorithm appears in Section IV, and we present an implementation in Section V. Section VI concludes the paper with discussion and a preview of future work.

II. Related work

Our planning algorithm is related to the idea of "preimage backchaining" introduced by Lozano-Pérez, Mason and Taylor [20]. Their research describes the notion of a *fine-motion strategy* as an effective counter to position uncertainty in compliant motions. Our approach is similar in that we consider an error cone—that is, a range of possible uncertainty values for each translation made by our robot—that increases the set of possible states from a single known state to some larger set of states derived from a known bound on error.

Erickson, et al, [15] also use this idea of an error cone to solve a global active localization problem. They describe a system whereby actions are carefully chosen to drive the probability of the robot's position toward a single cell in a coarse discretization of the environment. We are not, however, using a probabilistic approach, but rather a worst-case analysis. The other obvious difference is that we are solving a navigation problem and thus treat our points as landmarks, indirectly providing additional information about the robot's state.

The idea of landmark-based navigation was also proposed by Lazanas and Latombe [19]. They suggest the use of landmarks such that while the robot is in proximity of a landmark, the robot is able to execute error-free actions. They also assert that the robot is able to recognize when it has achieved its goal. In contrast, our robot has no sensor which would allow it to do so, nor does our planner depend on the robot explicitly sensing that it has achieved its goal state. Our planner, also, never assumes error-free actions by the robot nor an exact knowledge of any state after leaving the initial state. Instead, we use carefully a crafted plan that ensures the robot has reached its goal at plan completion, in spite of its lack of a goal-detecting sensor.

Our approach is similar to Erdmann and Mason's work on sensorless manipulation [12]. Our work follows suit with an inspection of the robot's environment, rather than any engineering of the environment as in [20]. The synthesis of these works results in a planner that uses parts of the environment as landmarks, by describing a careful iterative motion process to eliminate uncertainty periodically throughout the robot's execution. By determining landmarks from plentiful environment features, in this case, convex vertices, we show that a very simple robot is able to solve problems previously considered only through changing the environment in some way or the addition of more sensors.

Our goal of considering simplified sensing and actuation systems while solving meaningful problems is not new. A number of different tasks have been addressed with this approach, including manipulation in general [3], [13], [14], [20], part orientation specifically [2], [4], [12], [16], [22], [28], navigation [5], [10], [17]–[19], [21], and mapping [1], [8], [9], [23], [27]. More generally, others have explored the question of the minimal sensing requirements to complete a given task [6], [11], [14]. This methodology of minimalist robotics research can arguably be traced back to Whitney [29]. The idea of the approach is that it is often useful to minimize the complexity of a robotic system in order to focus instead on the problem the robot intends to solve.

III. Problem statement

This section formalizes the navigation problem we consider.

A point robot moves in a closed, bounded, polygonal region $W \subset \mathbb{R}^2$ of the plane. The robot has a complete and accurate map of its environment. A vertex v of W is *convex* if the neighborhood of v in W is convex. Formally, let $B(v,\epsilon)$ denote the open ball with radius ϵ centered at v. A vertex v is defined as convex if there exists some $\epsilon > 0$ such that $B(v,\epsilon) \cap W$ is a convex set. Informally, notice that convex vertices are formed whenever the two incident edges of a vertex form an angle less than or equal to π radians.

The robot is equipped with a compass and a contact sensor, but no other sensors. Note specifically that the robot has no clock nor any method of odometry, and consequently cannot measure the distances it moves. Using its compass, the robot can orient itself in a desired direction relative to a global reference frame, but because of noise in the sensor, this rotation is subject to potentially large, bounded error. Using its contact sensor, can translate in this direction until it reaches the boundary of the environment.

Our model for the motions of this robot has the following elements:

1) The *state space* $X = W$ is simply the robot's environment. Because we encapsulate the robot's use of its compass as part of the actions, we need not record the robot's orientation as part of the state.
2) The *action space* $U \in [0, 2\pi)$ is the set of planar angles. To execute an action $u \in U$, the robot orients itself in direction u, subject to the error described below, then moves forward in this direction until it reaches the environment boundary.
3) Time proceeds in a series of *stages*, numbered $k = 1, 2, \ldots$. In each stage, the robot chooses and completes a single action. At stage k, the robot's state is denoted x_k and its action is denoted u_k.
4) Rotation errors are modeled as interference by an imaginary adversary called *nature*. In each stage, nature chooses a *nature action* $\theta_k \in \Theta$. Nature's action space $\Theta = [-\theta_{max}, +\theta_{max}]$ is an interval of possible error values. Note that because we are interested in worst-case guarantees of success, we need not consider any probabilities over Θ. The robot has no knowledge of nature's choice, nor any way to observe it directly or indirectly.
5) The *state transition function* $f : X \times U \times \Theta \to X$ describes how the state changes in response to the robot's actions, so that the current state x_k, combined with the robot's action u_k and nature's action θ_k, determines the next state x_{k+1}:
$$x_{k+1} = f(x_k, u_k, \theta_k). \tag{1}$$
Specifically, $f(x_k, u_k, \theta_k)$ is defined as the opposite endpoint of the longest segment in X, starting at x_k and moving in direction $u_k + \theta_k$. Note that, due to error, the robot does not know x_{k+1} exactly. For convenience, we occasionally abuse this notation to apply several stages' worth of actions at once, so that
$$x_{k+i} = f(x_k, u_k, \theta_k, u_{k+1}, \theta_{k+1}, \ldots, u_{k+i}, \theta_{k+i}). \tag{2}$$

The robot's goal, given W and θ_{max}, along with initial and goal states $x_I, x_G \in W$ and an accuracy bound δ, is to

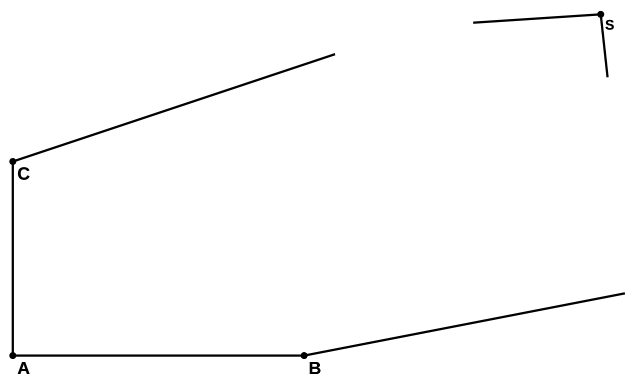

Fig. 2. The system of points: A, B, C, S

choose a sequence of actions $u_1, \ldots, u_K$ so that

$$||x_G - f(x_I, u_1, \theta_1, \ldots, u_K, \theta_K)|| < \delta \tag{3}$$

for all possible nature action sequences $\theta_1, \ldots, \theta_k \in \Theta$. That is, we seek actions that drive the robot from x_I to a point close to x_G, regardless of nature's actions. The accuracy bound δ is needed because the robot's motion error and sensor limitations prevent it from knowing that is has reached x_G exactly.

IV. Algorithm description

This section describes our algorithm to solve the navigation problem introduced in Section III. The basic structure of the algorithm is to form a sequence of *high-level transitions*, each composed of several actions. Each high-level transition moves the robot between a pair of environment vertices. The key feature that makes such transitions useful is that, after each high-level transition completes, the robot has nearly eliminated its uncertainty about its position.

The algorithm proceeds by identifying pairs of vertices between which such a high-level transition can be made, then using graph search techniques to assemble a sequence of these high-level transitions into a complete plan. Section IV-A describes the basic strategy the robot uses to make its high-level transitions, and Section IV-B shows how to determine whether this approach can successfully make a high-level transition between two given vertices. Finally, Section IV-C describes how we use this vertex-pair transition test to build a directed graph, from which the complete plan can be generated.

A. Corner finding algorithm

Given two distinct environment vertices S and A, how can the robot use its unreliable motions to move reliably from S to A? Let B and C be the predecessor and successor of A in a counterclockwise ordering of the vertices of W respectively. We refer to the segment formed by A and B as AB and refer to the segment formed by A and C as AC.

To travel from S to A, the robot makes a series of motions back and forth between AB and AC. To simplify the description, we describe in detail the case in which the robot's first movement takes it from S to a point x_1 on AB. The complete algorithm considers both AB and AC as potential initial segments, making the obvious changes to the corner finding and preimage computation algorithms. After this first motion, the robot alternates between two actions:

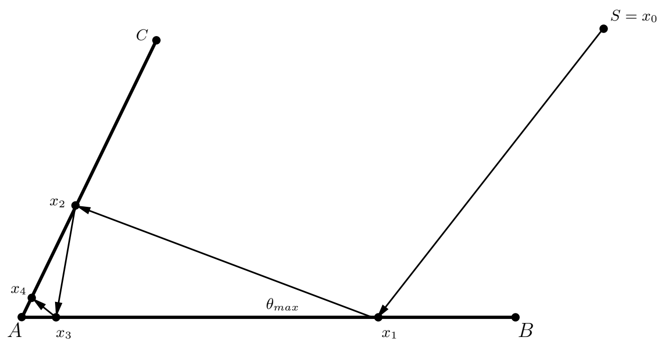

Fig. 3. Robot executing steps of corner-finding algorithm when AB is target segment.

Algorithm 1 FINDCORNER$(S, A, B, C, u_0, \theta_{max})$

```
1: $x_0 \leftarrow S$
2: for $k \leftarrow 1$ to $n$ do
3:     Execute action $u_{k-1}$
4:     if $k \bmod 2 = 1$ then
5:         $u_k \leftarrow \text{angle}(A - B) - \theta_{max}$
6:     else
7:         $u_k \leftarrow \text{angle}(A - C) + \theta_{max}$
8:     end if
9: end for
```

1) Whenever the robot is on AB, it chooses

$$u = \text{angle}(A - B) - \theta_{max}. \tag{4}$$

2) Whenever the robot is on AC, it chooses

$$u = \text{angle}(A - C) + \theta_{max}. \tag{5}$$

The intuition is that, at each step, the robot seeks to move toward A as directly as possible. However, because of the possibility of rotation errors, the robot must aim outward from the edge on which it currently rests by an amount equal to the maximum possible magnitude of this error. See Figure 3. The robot repeats the process some specified number of times, denoted n. This process is similar to the angle adjustment method used by Erickson et al, [15]. Details appear in Algorithm 1.

B. Computing preimages

Algorithm 1 depends on given vertices A and S, along with an initial action u_0. To apply this corner-finding algorithm as part of a successful global plan, however, the robot must find a value for u_0 under which the corner-finding algorithm is guaranteed to succeed. This section presents our approach to finding such a u_1, based on the notion of preimages.

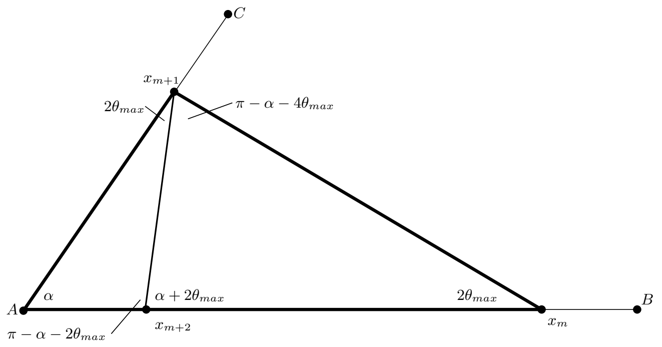

Fig. 4. Showing that $\triangle(x_{m+1}, x_{m+2}, A) \subset \triangle(x_m, x_{m+1}, A)$.

The *preimage* of vertex A from vertex S is defined as the set of actions the robot can execute as the first action u_0 of Algorithm 1, and be guaranteed not to collide with any obstacle in W except the two segments AC and AB. The following lemma provides the basis for the algorithm we use to compute preimages.

***Lemma** 1:* Let α denote the measure of angle formed at A with B and C. If $\alpha < \pi - 4\theta_{max}$, and the robot is guaranteed to make a collision free transition from x_1 to x_2, then the robot is also guaranteed to make the subsequent transitions to $x_3, \ldots, x_n$ without collision.

Proof: Use induction on the stage index k. As a base case, note that the conclusion is given for $k = 1$. For the induction step, assume that the statement is true for $k = m$ to show that it is true for $k = m + 1$. Refer to Figure 4. In the transition from x_m to x_{m+1}, the robot may pass through any point in the triangle formed by x_m, x_{m+1}, and A. By the inductive hypothesis, therefore, we know that the interior of this triangle does not contain any obstacles. Straightforward reasoning about the angles in this arrangement shows that $\angle(x_m, x_{m+1}, x_{m+2}) = \pi - \alpha - 4\theta$, which by supposition is greater than 0. As a result, the triangle formed by x_m, x_{m+1}, and x_{m+2} is non-degenerate, and x_{m+2} is closer to A than x_m. This implies that the triangle formed by x_{m+1}, x_{m+2}, and A is fully contained within the triangle formed by x_m, x_{m+1} and A. Since the latter triangle contains no obstacles, the former must also contain no obstacles. This ensures that the transition from x_{m+1} to x_{m+2} is collision free, completing the proof. ■

The implication of Lemma 1 is that there are only two ways in which the robot can have a collision while executing Algorithm 1: colliding with an obstacle on its initial translation from S to x_1, or along its second transition from x_1 to x_2. Our algorithm proceeds by finding intervals of actions that are guaranteed to safely complete these first two transitions.

1) From S to x_1: This section extensively references Algorithm 2. To check for instances of obstacles between S and AB, we define the robot's error cone (lines 2:2-8). For a given action u, this cone is defined as the region is bounded by rays originating at S with directions $u + \theta_{max}$ and $u - \theta_{max}$. We sweep the error cone around S and note all the angles at which the leading or trailing edge of the cone intersects some vertex $v \in W$. An example appears in Figure 5. We refer to the actions that generate these intersections as *critical actions*.

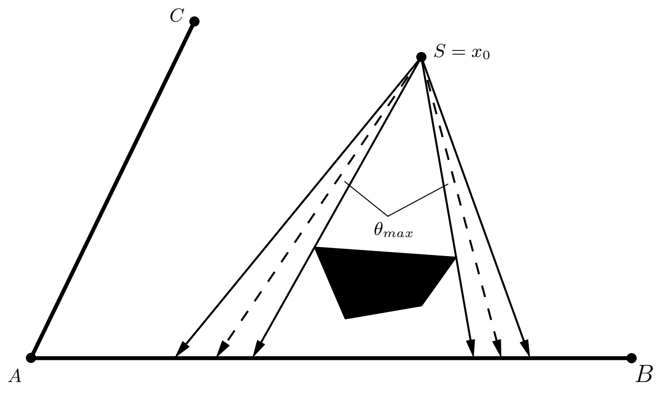

Fig. 5. Determining two critical actions of the system.

Algorithm 2 FIRSTTRANSITIONPREIMAGE(A, B, C, S)

```
 1: for $i \leftarrow 0$ to $1$ do
 2:     for all vertices $v \in W$ do
 3:         $u \leftarrow \mathrm{angle}(v_j - S) + (-1)^i \theta_{max}$
 4:         if Intersects(ray$(S, u)$, $AB$) then
 5:             $critActions$.insert($u$)
 6:         end if
 7:     end for
 8: end for
 9: sortClockwise($critActions$)
10: for $s \leftarrow 1$ to $j$ do
11:     $f \leftarrow s - 1$
12:     for all vertices $v \in W$ do
13:         $mid \leftarrow \frac{critActions[f] + critActions[s]}{2}$
14:         if $v \in$ triangle($S, mid + \theta_{max}, mid - \theta_{max}, AB$)
            then
15:             delete($critActions[f]$)
16:             break loop
17:         else
18:             $preimage$.add($critActions[f], critActions[s]$)
19:         end if
20:     end for
21: end for
22: return $preimage$
```

Critical actions represent directions at which a preimage segment might begin or end. Once all the critical actions are known, for each consecutive pair in an ordered clockwise sequence, a mid-direction is chosen and along that direction, an error cone is drawn (line 2:14). If the error cone contains no vertices of W, then the area between those two critical actions is collision free and the segment formed by the intersection of rays along the two critical actions and the target segment is included in the preimage (line 2:18). If any vertices are found, then collision avoidance cannot be guaranteed and thus the area is excluded from the preimage (line 2:15).

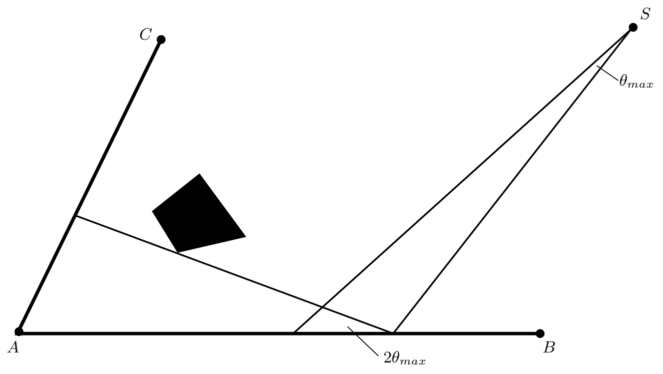

Fig. 6. Determining right side of preimage along segment AB.

Algorithm 3 SECONDTRANSITIONPREIMAGE(A, B, C, S)

```
1:  $\delta \leftarrow -(\text{angle}(A - B) - 2\theta_{max})$
2:  $q \leftarrow B$
3:  for all vertices $v \in W$ do
4:      $p \leftarrow$ intersect(ray($v, \delta$), $AB$)
5:      if dist($A, p$) $\geq$ dist($A, q$) then
6:          $q \leftarrow p$
7:      end if
8:  end for
9:  $preimage$.insert(angle($S - A$), $q$)
10: return $preimage$
```

2) From x_1 to x_2: This section extensively references Algorithm 3. To ensure that all jumps between the two target segments are obstacle-free, we must determine a triangle representing the largest collision-free error cone for the jump from x_1 to x_2. To find the triangle, we determine a vector representing the outermost edge of the cone (line 3:1). We then draw rays from each vertex in W in the opposite direction (line 3:4). The points at which transitions from S would cause these rays intersect with the target segment result in critical actions. To check these points, however, it is only necessary to note which of the critical actions lie closest to A along AB (lines 3:5-6). This point becomes the farthest end of our preimage segment(s) (line 3:9).

Algorithm 4 shows how to compute the preimage. The preimage computed by Algorithm 4 is a set of zero or more disjoint sets from which the initial action, u_0, is chosen. It is safe for the robot to choose any action in any of the sets. In our algorithm, we do not suggest an explicit method for choosing some *correct* action, u_0. That decision would be based on factors which we are not considering, such as optimality, thus any action in a non-empty preimage is a *correct* u_0 for our algorithm.

C. Finding a global path

The two above sections define how we compute a preimage between two vertices. If that preimage is non-empty, then it could be said that there exists a directed edge between the given vertices, S and A. By noting which vertices are connected by which directed edges we represent the problem

Algorithm 4 COMPUTEPREIMAGE(A, B, C, S)

```
1: return FirstTransitionPreimage($A, B, C, S$) $\cap$
   SecondTransitionPreimage($A, B, C, S$)
```

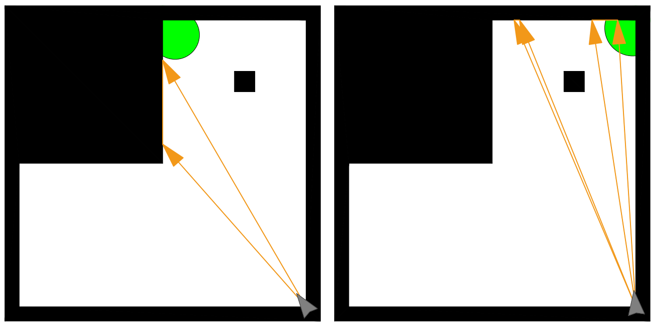

Fig. 7. A trivial environment depicting two preimages. The first, simply a contiguous set of critical actions. The second, a more complicated example with a disjoint set of critical actions.

of moving through an environment via any number of high-level transitions as a graph search. Using one of the usual methods for computing paths through graphs, in our case a breadth-first search, we search the graph and if a path through the environment is obtained, the robot has a guarantee that it can, using its corner-finding routine and preimages, transition from its initial state and into its goal state.

V. IMPLEMENTATION AND EXPERIMENTS

We implemented this algorithm in simulation using CGAL [7] as the geometry engine modeling our robot and environment. The algorithm was implemented in C++ and all animations were performed with OpenGL. Throughout all of these initial experiments, θ_{max} is set to $\frac{\pi}{50}$ radians–an arbitrarily chosen value. We present two non-trivial environments in these initial simulations. The environment from Figure 8 is a rectilinear environment with 44 vertices. The graph for this environment took 3 minutes 16 seconds to complete. The non-rectilinear environment in Figure 12 has 62 vertices and its graph was computed in 6 minutes 24 seconds. It is given to illustrate a more extreme example of the sorts of problems our algorithm can solve.

Figure 7 is an example of two of these preimages, given for a simple environment. Each illustration depicts the starting point of the system S as the point denoted by the triangular icon. The target vertex, A, is given as the shaded vertex. The arrows originating at S represent the two ends of a set or sets of angles forming the preimage of the system. The first illustration is the most simple case, a single set of critical actions which the robot can execute to ensure a collision-free traversal.

The second illustration uses an obstacle to demonstrate a slightly more complex example. The preimage now contains disjoint sets. Contrast Figure 7 with Figure 8. Figure 8 illustrates a more complicated system. Without error, the most obvious course of action for the robot would be to drive directly into its goal vertex. This figure illustrates an example

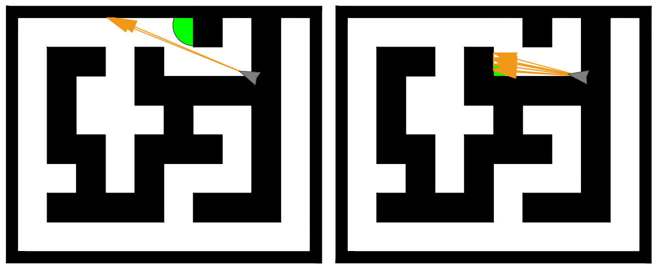

Fig. 8. A less obvious preimage in a more realistic environment and a preimage containing multiple sets of critical angles, all of which are included in the preimage.

of a system that wouldn't be solved by that plan, even in an error-free environment. The second illustration in Figure 8 shows a situation in which multiple critical angles are found due to the arrangement of vertices in the environment. Here, all of the sets of critical angles should be included in the preimage.

Knowing what the preimages between vertices look like, now allows the robot to make plans that guarantee successful traversal through its environment. For the experiments involving the simulated robot, we begin the robot with certain knowledge of its position. From that point, we simulate its use of a map, contact sensor, and compass to allow it to execute its corner-finding algorithm. Using the graph determined previously, the robot is able to make plans, i.e. search the graph, to find high-level transitions which carry it from $x_{Initial}$ to x_{Goal}. It is important to note that for the simulation to be accurate, we do not ever reset the robot's position during any portion of its corner-finding algorithm, nor during any point during or after high-level transitions. Uncertainty is allowed to accumulate naturally during each movement in a transition, using a pseudo-random number to generate each θ_k. Each time the robot makes a decision to execute some action, it is offset by nature a random amount bounded by $\pm\theta_{max}$. The number of iterations of the corner-finding routine, n, is set to 20.

Figure 9 is a plan the simulated robot devised to transition from the initial state, again represented by the triangle icon, to the goal state, the shaded vertex. The arrow heads which occur at each of the convex vertices into which the robot uses its corner-finding algorithm to drive itself are depicting the repeated transition back-and-forth between the two segments AB and AC. Our robot has no sensor to detect goal achievement, so it is forced to execute n iterations of the corner-finding routine.

The arrows in this figure are drawn along the actual paths the robot follows, offset by θ. Notice that the initial transition from some x_0 to x_1 isn't as close to the goal vertex as it could be. Since the preimage is a set of angles and possibly multiple sets of disjoint angles, the robot has a decision to make, "along which direction should I translate?" Because our algorithm is concerned only with feasibility, rather than optimality, we decided the robot should aim for the center angle of the largest contiguous set of angle in the preimage.

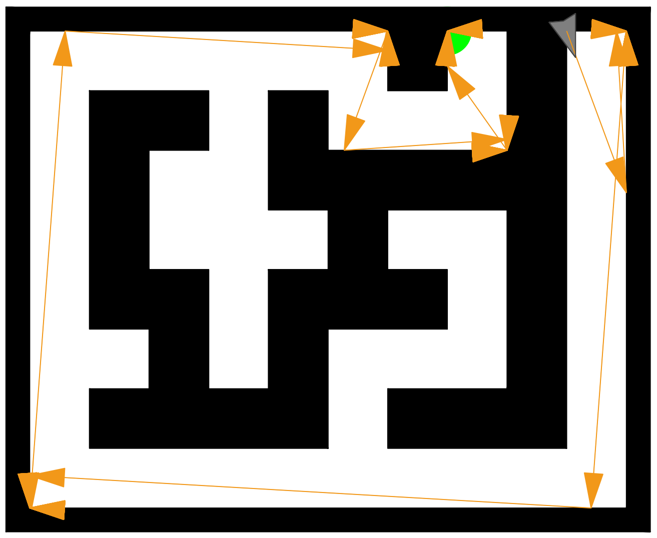

Fig. 9. A plan generated by our algorithm in a realistic environment.

This has the effect of always aiming as far from obstacles as the robot can.

Figure 10 is given as the analogue to Figure 9. That is, in the graph of nodes and edges, the nodes representing the initial and goal vertices in, Figure 9 and Figure 10 are connected with two edges. Contrast these figures with Figure 11. Again, the initial state is marked by the triangle icon, the goal state is marked by the shading, but these two vertices only have one edge between them. It is, in fact, true that there is no way for the robot to get back into that portion of the environment once it exits. This missing edge is only partially a consequence of error, but is also weakness of our algorithm's dependence on its corner-finding routine. The corner-finding routine must have some "line of sight" between the convex vertex describing its initial state and one of the segments forming the convex vertex of its target state. There is a small direct path between a convex vertex on the bottom right of the environment and the segment forming the initial convex vertex of that area. The path is, however, too small to allow our algorithm to generate actions guaranteeing a successful traversal.

In spite of limitation illustrated by Figure 10, our algorithm does not need to be restricted to rectilinear environments. Figures 12 and 13 show an environment with unusual features. The two plans devised by the robot allow guaranteed traversals between some of the more extreme features. We did include in this environment features which would provide limitations based both on the uncertainty involved and the limitations of the corner-finding routine. Note that due to the length of the path traversing from the left side of the environment to the right, that with $\theta_{max} = \frac{\pi}{50}$ there is no guaranteed safe path. There is, also, a portion of the map for which the corner-finding routine fails—it isn't possible to get into nor out of the center section of the environment.

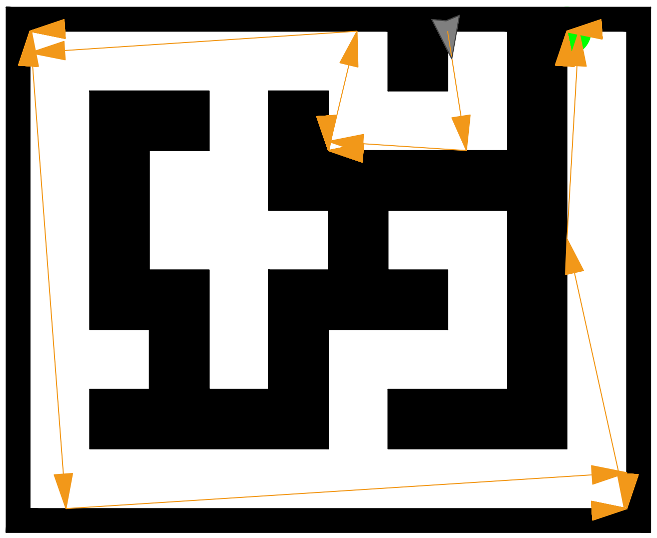

Fig. 10. This figure illustrates the second edge between x_I and x_G depicted and x_I and x_G from Figure 9.

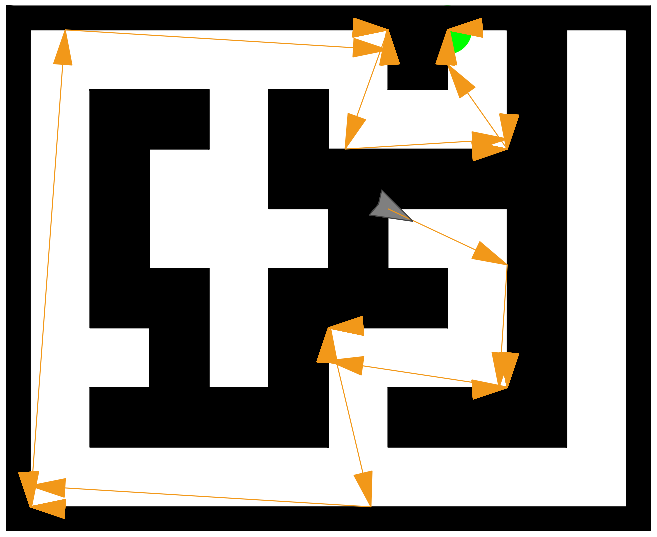

Fig. 11. A plan from x_I to x_G, with no way for the robot to return to its initial state.

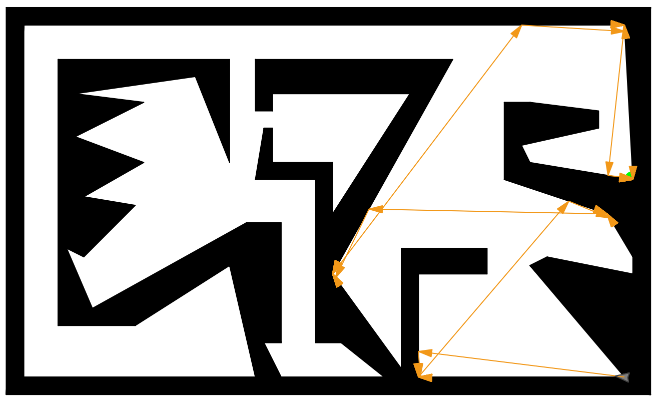

Fig. 12. A full plan devised by the algorithm to navigate a non-rectilinear environment.

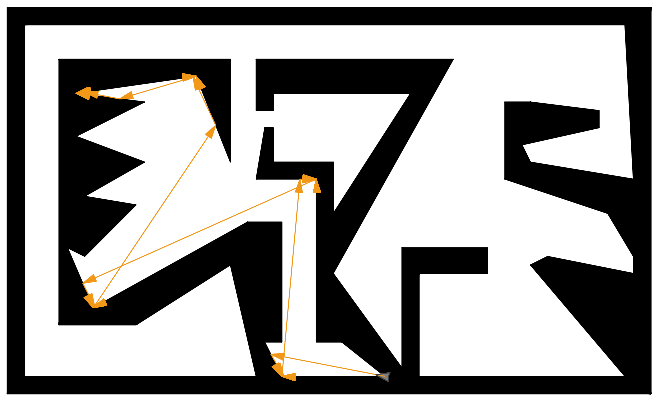

Fig. 13. This figure illustrates a very extreme problem, which our algorithm was able to successfully determine a plan to navigate.

VI. Discussion and Conclusion

In this paper we presented a strategy whereby a robot having only a map, contact sensor, and compass navigates between vertices in a planar environment. We presented a corner-finding routine and using an analysis of that routine, were able to determine all critical actions for the robot–critical actions defined as indicators of a possible collision with some part of its environment. By determining which of these critical actions, when paired in a clockwise order, give a guaranteed set of angles along which such a collision does not occur, we were able to compute a preimage. The preimage between two vertices in the environment was then used to map the vertices of the environment to the nodes of a graph and the presence of a non-empty preimage as an edge between those nodes. The complete plan is generated by a graph search on this graph.

A. Future Work

*1) **Bounds on Uncertainty**:* In Section V we arbitrarily choose a value for θ_{max} for our experiments. The value of θ_{max} has a great deal of impact on the results of each preimage calculation; a preimage may be empty for one value of θ_{max} and not for another. The preimage's dependence on θ_{max} means that for any given system, A, B, C, and S, there are bounds on θ_{max} itself which determine the largest value for which a non-empty preimage exists. If this value was known for each vertex pair in the environment, we could then calculate the largest θ_{max} for which a given instance of the navigation problem can be solved.

This line of reasoning could also lead to a probabilistic model of the preimage. Larger values of θ_{max} could be used and would lead to changes in probabilities associated, then, with the edges of the graph. In this case, it would be necessary to consider the area of each error cone as it grows with increased uncertainty. By calculating how much of this error cone's area is taken up by obstacles it would be possible to determine the probability of a collision with its environment during a transition. This, of course, assumes that each $\theta \in \Theta$ has an equally likely chance to occur.

*2) **Completeness:*** Though our current corner-finding routine is robust enough to navigate realistic environments with guarantees of success, it is not complete. The most obvious example of this incompleteness is the corner-finding routine's dependence on a large enough direct path from any given vertex to one of the segments forming the convex vertex into which the robot transitions. A complete algorithm would need to, at least, overcome this problem.

One approach would be to note that we treat the entire environment as obstacle, except the two segments for which we're aiming and A. Instead, every edge of the environment could function as target segments and rather than aiming to avoid them, we could use them to reach places previously unattainable.

Another consideration for completeness is the corner-finding routine's method of computing direction. Currently, we attempt to drive as directly into A from any given x_k as θ_{max} allows. Its easy to imagine a situation in which an obstacle lies close enough to A along one of its target segments to cause a collision. Though our overall algorithm isn't so sensitive to this, as we can choose which segment is our initial, there are situations in which this would cause the preimage to be empty.

By extending the algorithm in a way that guarantees completeness, then we can make stronger statements about the bounds on uncertainty. If we were to find a complete an algorithm for our problem description, then preimages could be directly mapped to the above-mentioned bound on θ_{max}—that is, preimages would depend completely on the uncertainty of the system.

*3) **Optimality:*** In our presentation, we have made no claims of optimality. Currently we are investigating the problem of optimal navigation for this robot.

Acknowledgments

This work is partially supported by a grant from the University of South Carolina, Office of Research and Health Sciences Research Funding Program.

References

[1] E. U. Acar and H. Choset, "Complete sensor-based coverage with extended-range detectors: A hierarchical decomposition in terms of critical points and voronoi diagrams," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[2] S. Akella, W. Huang, K. M. Lynch, and M. T. Mason, "Parts feeding on a conveyor with a one joint robot," *Algorthmica*, vol. 26, no. 3, pp. 313–344, Mar. 2000.

[3] S. Akella and M. Mason, "Posing polygonal objects in the plane by pushing," *International Journal of Robotics Research*, vol. 17, no. 1, pp. 70–88, Jan. 1998.

[4] R.-P. Berretty, K. Goldberg, M. Overmars, and F. V. der Stappen, "Trap design for vibratory part feeders," *International Journal of Robotics Research*, vol. 20, no. 11, Nov. 2001.

[5] A. Blum, P. Raghavan, and B. Schieber, "Navigating in unfamiliar geometric terrain," *SIAM Journal on Computing*, vol. 26, no. 1, pp. 110–137, 1997.

[6] M. Blum and D. Kozen, "On the power of the compass (or, why mazes are easier to search than graphs)," in *Proc. IEEE Symposium on Foundations of Computer Science*, 1978, pp. 132–142.

[7] "CGAL, Computational Geometry Algorithms Library," http://www.cgal.org.

[8] H. Choset and J. Burdick, "Sensor based motion planning: Incremental construction of the hierarchical generalized Voronoi graph," *International Journal of Robotics Research*, vol. 19, no. 2, pp. 126–148, 2000.

[9] ——, "Sensor based motion planning: The hierarchical generalized Voronoi graph," *International Journal of Robotics Research*, vol. 19, no. 2, pp. 96–125, 2000.

[10] X. Deng, T. Kameda, and C. H. Papadimitriou, "How to learn an unknown environment I: The rectilinear case," *Journal of the ACM*, vol. 45, no. 2, pp. 215–245, 1998.

[11] B. R. Donald, "On information invariants in robotics," *Artificial Intelligence*, vol. 72, pp. 217–304, 1995. [Online]. Available: citeseer.ist.psu.edu/article/donald95information.html

[12] M. Erdmann and M. T. Mason, "An exploration of sensorless manipulation," *IEEE Transactions on Robotics and Automation*, vol. 4, no. 4, pp. 369–379, Aug. 1988.

[13] M. A. Erdmann, "Using backprojections for fine motion planning with uncertainty," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 19–45, 1986.

[14] ——, "Understanding action and sensing by designing action-based sensors," *International Journal of Robotics Research*, vol. 14, no. 5, pp. 483–509, 1995.

[15] L. Erickson, J. Knuth, J. M. O'Kane, and S. M. LaValle, "Probabilistic localization with a blind robot," in *Proc. IEEE International Conference on Robotics and Automation*, 2008.

[16] K. Y. Goldberg, "Orienting polygonal parts without sensors," *Algorthmica*, vol. 10, pp. 201–225, 1993.

[17] I. Kamon and E. Rivlin, "Sensory-based motion planning with global proofs," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 6, pp. 814–822, Dec. 1997.

[18] I. Kamon, E. Rivlin, and E. Rimon, "Range-sensor based navigation in three dimensions," in *Proc. IEEE International Conference on Robotics and Automation*, 1999.

[19] A. Lazanas and J. C. Latombe, "Landmark-based robot navigation," in *Proc. National Conference on Artificial Intelligence (AAAI)*, 1992.

[20] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor, "Automatic synthesis of fine-motion strategies for robots," *International Journal of Robotics Research*, vol. 3, no. 1, pp. 3–24, 1984.

[21] V. J. Lumelsky and S. Tiwari, "An algorithm for maze searching with azimuth input," in *Proc. IEEE International Conference on Robotics and Automation*, 1994, pp. 111–116.

[22] M. Moll and M. Erdmann, "Manipulation of pose distributions," *International Journal of Robotics Research*, vol. 21, no. 3, pp. 277–292, 2002.

[23] C. Ó. Dúnlaing and C. K. Yap, "A retraction method for planning the motion of a disc," *Journal of Algorithms*, vol. 6, pp. 104–111, 1982.

[24] J. M. O'Kane and S. M. LaValle, "Localization with limited sensing," *IEEE Transactions on Robotics*, vol. 23, pp. 704–716, Aug. 2007.

[25] ——, "On comparing the power of robots," *International Journal of Robotics Research*, vol. 27, no. 1, pp. 5–23, Jan. 2008.

[26] N. Roy and S. Thrun, "Coastal navigation with mobile robots," in *Advances in Neural Processing Systems*, 1999, pp. 1043–1049.

[27] B. Tovar, L. Guilamo, and S. M. LaValle, "Gap Navigation Trees: Minimal representation for visibility-based tasks," in *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2004.

[28] A. F. van der Stappen, R.-P. Berretty, K. Goldberg, and M. H. Overmars, "Geometry and part feeding," in *Sensor Based Intelligent Robots*, 2000, pp. 259–281.

[29] D. E. Whitney, "Real robots don't need jigs," in *Proc. IEEE International Conference on Robotics and Automation*, 1986.