Implementation and Improvement of: An Algorithm for Maze Searching with Azimuth Input

Greg Schmidt

Technical Report

December 21, 1995

Abstract

The work presented by Lumelsky and Tiwari[1] is a new algorithm for Maze Searching. This algorithm called the Angulus Algorithm incorporates directional information for nonheuristic maze searching for the first time. Their algorithm solves the problem of locating a target from a given starting point for a point automaton (robot). The work presented here, implements their algorithm and extends their work for the case of mazes with holes. This extension can be used for exploring parallel extensions to their algorithm. One parallel problem is to use a set of point automatons to search a maze for a target originating at the same start point. Another problem is to drop a set of point automatons at random locations in a maze and have them search for the target. The details in this report show the steps of the implementation, the extended unproved algorithm and proposed ideas for parallel extensions.

1 Background

The problem of motion planning for a point automaton (robot) in a planar scene with unknown obstacles has been around for awhile. The work done by Lumelsky and Tiwari[1] presented a new algorithm based on directional input as opposed to requiring knowledge of cartesian coordinates. This algorithm is a nonheuristic and on-line approach. The objective of the automaton is to generate a collision-free path from a starting point S to a target point T, or to prove that T is unreachable from S. The obstacle's boundaries are assumed to be simple, piecewise smooth closed curves of finite perimeter. This excludes obstacles with holes. Their particular algorithm called the Angulus Algorithm solves this problem, but appears to have a redundant step that may never be reached. This step assumes an obstacle environment containing at least one hole. Though this step can be easily removed without affecting the content of the algorithm, they also claim that their obstacle environment is very general and is satisfied by all obstacles encountered in robotics. This claim appears to exclude simple cases of floor layouts in a common house such as illustrated in Figure 1. This type of environment is considered a graph or maze with a hole in it. The research presented here disputes their generality claim and expands on the generality of environments encountered in robotics.

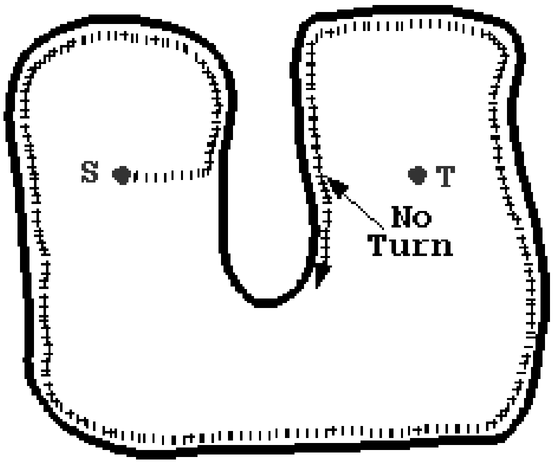

Figure 1: Maze with a Hole

2 Objective

One objective of our work is to implement and study the Angulus Algorithm. This is important for extending the algorithm to include graphs with holes. These algorithms are then proved. From this work extensions to distributed and parallel algorithms are developed. The algorithms have an input of the form: $\vec{S} = S_1, S_2, ..., S_n$ and T. $\vec{S}$ is a set of n point automatons working together to find the target T. There are other forms, but these are ignored since they are being developed currently by Lumelsky and Tiwari[1].

For each extended algorithm developed, they are implemented and proven. Some of the objectives for the implementation include quick development time and easy modification. The programming language C++ is great for expandibility (modification). The development time is slow at first, but fast thereafter. Another objective is to develop algorithms that may be reduced to others rather if at all possible. Domain transforms may then be used upon older algorithms. This, in effect, speeds up the implementation time, but creates some overhead in the algorithm development phase. The last objective is to generate a proof for each algorithm. Since these algorithms are targeted for analysis, it is essential to contain a proof for each.

3 Motivation

The motivation behind this work is an idea tossed around at NASA (National Aeronautics and Space Administration) regarding missions to Mars. The idea is to send a ship and distribute multiple robots on the surface. These robots could survey specific areas, gather information such as soil samples and return to the lander. The lander could emit a signal so that directional-based motion planning algorithms could be used. The robots would collaborate together to return to the lander. This sort of algorithm would require $\vec{S} = S_1, S_2, ..., S_n$ and T as input where $S_1 \neq S_2 \neq ... \neq S_n$. This algorithm form of algorithm is explored in our work.

Another form of algorithm involves the same input, but $S_1 = S_2 = ... = S_n$. This algorithm may represent the case that all the robots leave the lander at the same time and spread out to find some target. The target could be a lost or damaged robot that is emitting a distress call. This sounds funny, but it may be practical if money[1] or human lives are at stake.

4 Clarification of the Angulus Algorithm

The Angulus Algorithm has a few steps that are unclear. This section contains some clarification to the steps involved. *Step* 4 of the algorithm is rewritten for clarity. *Step* 5 is removed due to redundancy.

In *Step* 4 it states that the robot may leave when $\alpha_i = \beta_i = -2\pi$. The chances of this condition, as worded, occuring is near zero. The step continues and points this out. Then it gives an additional condition for the departure of the robot. To simplify this confusing step, a new *Step* 4 was generated. The new *Step* 4 is:

> If it occurs at least once that $\alpha \leq -2\pi$, then leave obstacle for T only when $\alpha_i = -2\pi$ and $\beta_i \in (-4\pi, -2\pi]$. If T is reached, stop; if an obstacle is encountered, go to *Step* 2.

Step 5 can never be reached for the case of mazes without holes. If holes were allowed in the mazes, then it can be shown that the algorithm fails in at least one instance. Refer to the maze in Figure 1. Therefore, *Step* 5 can be removed for sake of redundancy.

5 The Right-handed Angulus Algorithm

This algorithm is a simple extension of the Angulus Algorithm presented by Lumelsky and Tiwari[1]. The details simply take into consideration the fact that the Angulus Algorithm is set up for a left-handed point automaton. This extension is the equivalent algorithm but for a right-handed point automaton.

The Right-handed Angulus Algorithm

1. Proceed along the line ST and if the target is reached, stop.
2. If an obstacle is encountered at a hit point H_j, turn right and follow the obstacle boundary, while simultaneous measuring the cumulative angles α and β.
3. At a point C_i on the obstacle boundary, if it is physically possible to walk straight toward T, and the current velocity vector v_i points toward T, and both $\alpha_i, \beta_i \in (0, 2\pi)$, proceed directly toward T. If T is reached, stop; if an obstacle is encountered, go to *Step* 2.
4. If it occurs at least once that $\alpha \geq 2\pi$, then leave obstacle for T only when $\alpha_i = 2\pi$ and $\beta_i \in [2\pi, 4\pi)$. If T is reached, stop; if an obstacle is encountered, go to *Step* 2.

Proof: The proof is quite simple and may be considered a reduction. A reduction proof states a problem P can be reduced to a problem P', $P\ \alpha P'$, if the following conditions hold:

1. The input, x, of the problem P can be converted to input, x', for problem P'.
2. The solution for the input, x', of P' can be converted back to a solution for input, x, of P.

This may be extended for time complexity bounds as well[2]. The proof then follows.

[1] If money is at stake, this is known as the infamous "dad clause".

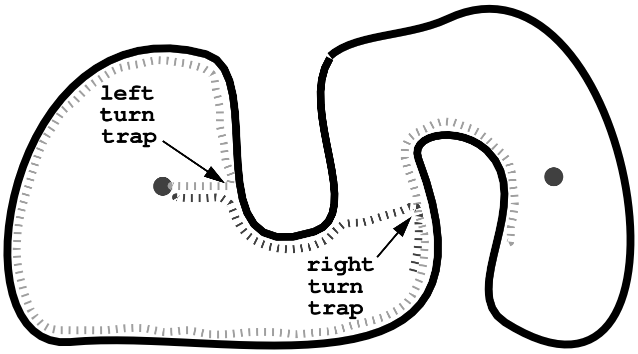

Figure 2: Left-handed and Right-handed Maze

Right-handed Angulus $\propto$ Angulus

Let $\rho(x,y)$ be the set of cartesian coordinates making up the obstacle environment encountered by the point automaton(within visual range). Take the original normalized vector $\vec{ST}$ and make that the cartesian line of reflection. Then reflect $\rho(x,y)$ about $\vec{ST}$. This may be performed using quaternions[3] quite simply. For this transformation, designate S as the origin ($\rho \rightarrow \rho'$) and compute the zero-scalar quaternions for $\vec{S\rho'}$ and $\vec{ST}$ denoted $\dot{r}$ and $\dot{q}$, respectively. The reflection is computed using a double angle rotation $\dot{r'} = \dot{q}\dot{r}\dot{q^*}$ where $\dot{q^*}$ is the quaternion conjugate of $\dot{q}$. This rotation preserves the zero-scalar quaternion and thus the resulting vector, $\vec{r'}$, may be extracted directly. The reflected point, r, may then be recalled from the the vector $\vec{r'}$ and the point S. For the planar case this transformation is shown in Equation 1.

$$r(x,y) \quad = \quad < x + 2\vec{ST}_y(\vec{ST}_x y - \vec{ST}_y x), y - 2\vec{ST}_x(\vec{ST}_x y - \vec{ST}_y x) > \tag{1}$$

Keep in mind translation and rotations must be reflected in this domain. Thus every right-hand turn becomes a left-handed turn. Angles are also reflected then. $(0, 2\pi) \rightarrow (-2\pi, 0)$ and $[2\pi, 4\pi) \rightarrow (-4\pi, -2\pi]$ are reflected as such. Notice that the Right-handed Angulus Algorithm reduces identically to the Angulus Algorithm in the reflected domain. Thus any input for the Right-handed Angulus Algorithm may be converted to input for the Angulus Algorithm, solved and converted back by the same reflection mapping. Therefore Right-handed Angulus α Angulus.

5.1 Inclusion of Mazes with Holes

This algorithm is proven to always work for mazes without holes like Angulus Algorithm. With the addition of *Step* 5, it can be used on mazes with holes. This algorithm still does not solve all mazes with holes, but it will solve a different proper subset of the general case (some mazes with holes that Angulus cannot solve are solvable by this algorithm). The maze in Figure 1 can be solved by this algorithm. That particular maze is classified as a right-handed maze since it cannot be solved by the left-handed Angulus Algorithm. If the maze was reflected about the vector $\vec{ST}$, then it could be solved by the Angulus Algorithm. The resulting maze, however, cannot be solved by the Right-handed Angulus Algorithm. This reflective property has led to the development of an algorithm combining left-handed and right-handed robots. The need for these is evident in the harder cases such as the maze shown in Figure 2. This maze cannot be solved by either of the two algorithms developed so far.

6 The Left-Right Angulus Algorithm

- To be completed at a later date, if the material is sufficient for a paper.
- This still may not prove to cover all types of mazes. The inclusion of holes forces a higher complexity, quite obviously.

7 Discussion

The work that was performed first was the implementation of the Angulus Algorithm. Then the algorithm was studied and cases were deduced that could not be solved (those with "holes" in). Then the Right-handed Angulus Algorithm was developed to handle another set of cases that couldn't be solved by the Angulus Algorithm. There were still cases that could not be solved. This led to the development of the Left-Right Angulus Algorithm. The details of each of these are laid out in the following sections.

7.1 Implementation of the Angulus Algorithm

The language C++ was chosen for the implementation since it allows for protection of variables through an abstract data type called classes. This means that data types can be made for the robot and cartesian-based points that have protected access. Since it is desired that the robot not see the cartesian coordinates of the destination, this information can be made private in the target point data type. This assures that the robot only uses directional input such as from a data type for the compass information.

Another implementation issue came up in the decision of the form of the obstacle environment. The obstacles were defined as simple, piecewise smooth closed curves. The easiest most general means for representing these obstacles is with a bitmap. A bitmap could be used to represent the free space (black pixels) and the obstacles (colored pixels), and provides for quick collision checking. Since the software is designed to work on an SGI workstation, the standard image format (SGI) was chosen. This format can be easily converted from nearly all other formats via conversion routines available on the SGI workstations. The bitmap, once read by the software, is converted to a sequence of hit or miss values. This reduces the memory storage of the bitmap and is all that is required for collision detection.

7.2 The Winding Properties

As the result of discovery of a flaw in the algorithm or its defined conditions for boundaries, I have developed a couple of properties. These have been discussed with the original author(s)[2] and are named the Winding Properties. These are relevant to the two cumulative angles described in the original paper and are important for pinpointing the flaw. The Winding Principles are defined as:

1. If a cumulative angle to the likeness of the β angle in your paper is measured and a reference point is taken on an obstacle, upon each perimeter traversal it will increase or decrease by $2 * \pi$.
2. If a cumulative angle to the likeness of the α angle in your paper is measured and a reference point is taken on an obstacle, upon each traversal on a "hole-less"—shall we call it—object

[2] Sanjay Tiwari and Vladimir Lumelsky, University of Wisconsin—Madison

the angle measure remains the same (unwinds). If the obstacle has a hole, then it reduces to the likeness of the β angle condition above.

Given these properties stated in this manner, *Step 5* in the original algorithm cannot be reached unless the start point, s, is in the set-theoretic exterior of the target point, t. And as stated in Section 4, this is redundant since this definition of a maze is not always solvable by the Angulus Algorithm.

If the maze in Figure 1 is used, with s in the interior of t, then β increases positively only and α follows suit after an initial non-terminating negative swing. This leads to "loop ad infinitum" of the type where the robot never leaves the boundary of the obstacle.

The software may be obtained by emailing Greg Schmidt at:

- gschmidtcs.tamu.edu

8 Conclusion

As far as the results from the software, the algorithm works great with the addition of the corrections to the original flawed algorithm. The mazes that are not solvable by the regular Angulus Algorithm and are in the class of mazes that are solvable by the Right-handed Angulus Algorithm, may be solved by the software by reflecting the image. This, in effect, illustrates how the mathematical reflection proof works. More conclusions will be added if this report is extended into a paper.

9 Acknowledgements

This work was inspired by the graduate class, Robotics Programming (CPSC 643), in the Computer Science Department at Texas A&M University. This class is taught by Dr. Jeff Trinkle who is involved with the Robotics Laboratory.

References

[1] V. Lumelsky and S. Tiwari, "An Algorithm for Maze Searching with Azimuth Input," In *IEEE International Conference on Robotics and Automation*, San Diego, CA, May 1994.

[2] J. Chen, "Computational Geometry," Computer Science Department, Texas A&M University, August 1990.

[3] B. K. P. Horn, "Relative orientation revisited," *Journal of Optical Society of America*, October 1991.