CSE473 Final Project: Vision-based Micromouse Maze Wall Detection

Scott Will, Mack Ward
University at Buffalo
Amherst, NY 14261
scottwil@buffalo.edu, mward4@buffalo.edu

Abstract

Since the late 1970s, students and professional engineers alike have competed in Micromouse, an event in which teams of participants construct wheeled robots that attempt to autonomously traverse and solve planar mazes in as little time as possible. A common strategy adopted by competitors is a two-stage process: first, a mouse explores the maze and incrementally generates a virtual representation of the maze layout in its internal memory, then computes the optimal path from the starting position to the center of the maze, and finally returns to the starting position and moves along the optimal path. Although this approach is effective and reliable, most of the total time spent in transit by the mouse tends to be devoted to the exploration phase. To reduce this time, we propose a much different strategy: using a camera mounted on the mouse at sufficient height and techniques from computer vision and image processing, generate an image of the maze from the perspective of a viewer looking down from above, extract the locations of the maze walls programmatically, and proceed to solve the maze by directly moving to its center.

1. Introduction

1.1. Competition rules

Typical Micromouse events follow standard rules and maze-guidelines. We reproduce them here for thoroughness (adapted from the official rules from the IEEE Region 1 Micromouse Competition) [1]:

1.1.1 Rules for the micromouse

1. A Micromouse shall be self-contained (no remote controls). A Micromouse shall not use an energy source employing a combustion process.

2. A Micromouse shall not leave any part of its body behind while negotiating the maze.

3. A Micromouse shall not jump over, fly over, climb, scratch, cut, burn, mark, damage, or destroy the walls of the maze.

4. A Micromouse shall not be larger either in length or in width, than 25 centimeters. The dimensions of a Micromouse that changes its geometry during a run shall not be greater than 25 cm $\times$ 25 cm. There are no restrictions on the height of a Micromouse.

1.1.2 Rules for the maze

1. The maze is composed of multiples of an 18 cm $\times$ 18 cm unit square. The maze comprises 16×16 unit squares. The walls of the maze are 5 cm high and 1.2 cm thick (assume 5% tolerance for mazes). The outside wall encloses the entire maze.

2. The sides of the maze walls are white, the tops of the walls are red, and the floor is black. The maze is made of wood, finished with non-gloss paint.

3. The start of the maze is located at one of the four corners. The start square is bounded on three sides by walls. The start line is located between the first and second squares. That is, as the mouse exits the corner square, the time starts. The destination goal is the four cells at the center of the maze. At the center of this

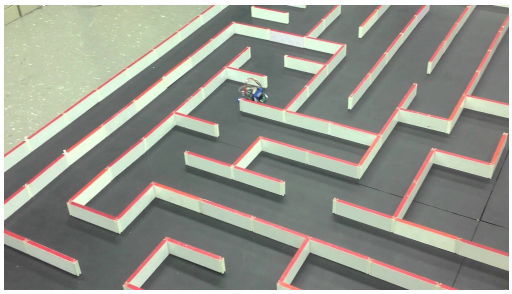

Figure 1. A typical Micromouse and maze

zone is a post, 20 cm high and each side 2.5 cm. (This post may be removed if requested.) The destination square has only one entrance.

4. Small square zones (posts), each 1.2 cm $\times$ 1.2 cm, at the four corners of each unit square are called lattice points. The maze is so constituted that there is at least one wall at each lattice point.

5. Multiple paths to the destination square are allowed and are to be expected. The destination square will be positioned so that a wall-hugging mouse will NOT be able to find it.

1.2. Common solution methods

As briefly described above, a great majority of teams produce robots that follow relatively the same high-level steps for reaching the center of a maze that meets the above specifications:

1. Fully traverse the maze (including all dead-end routes) and generate an internal representation of the maze topology

2. Use one or more searching algorithms such as bread-first or depth-first search, to find the optimal path from the starting point to the maze center

3. Move along the optimal path to reach the maze center

A less-popular method is the so-called "wall-follower" approach, which seeks to eventually reach the center of the maze by simply moving along a given wall for its entire length. This is not guaranteed to be effective for all possible maze structures, and is officially discouraged by the competition organizers (see §1.1.2).

1.3. Our solution

As members of the IEEE student chapter at UB, we have competed twice in the annual IEEE Region 1 Micromouse Competition, which is open to student teams from universities across the northeastern United States. We conceived the idea to engineer a vision-based Micromouse implementation after observing the weaknesses of the "traditional" style of implementation, which, in our opinion, spent too much unnecessary time determining maze layouts even before attempting solutions. We realized that because of the regularity of competition mazes, which were composed of equally-sized unit squares, uniform-height walls, and so on, that it might be possible to extract all of the necessary information to produce an accurate representation of maze layout using only visual data obtainable from the Micromouse's perspective.

2. Related work

We are not the first to suggest using tools from computer vision to improve the capabilities of Micromouse implementations. In 1997, Ning Chen suggested to revise the Micromouse competition to encourage solutions more relevant to real-world engineering challenges, by introducing uneven mazes resembling real terrain [2]. However, he suggested using information from cameras for low-level control, steering, and navigation, essentially using cameras as sensors instead of infrared and ultrasound, two common sensor variations seen in competitive mice. In the 2011-12 academic year, a student team from University of California, San Diego adopted this approach for their implementation [3]. We stress that our implementation differs fundamentally from that in [2] and [3], in that our approach employs more "global" information to avoid the costly exploration phase of maze solving.

3. Algorithm

In this section, we present the proposed method for obtaining maze solutions using information obtained from the perspective of the Micromouse. We begin by obtaining a panoramic series of images taken from the starting position using a camera mounted at height on top of the Micromouse. To generate the data to test our method, we used a cell phone camera to obtain several image sets taken approximately 12 inches vertically from the maze surface, at the position of one of the corners. Then, we stitch pairs of images together using a stitching script– for example, for a 6-image set, create 3 images by stitching the first and second images, the third and fourth images, and the fifth and sixth images. We used a publicly available Github project for this purpose, which can be found at `https://github.com/cbuntain/stitcher`.

After obtaining a set of stitched images, we begin performing image transformations. Because the end goal is to produce an orthographic image of the maze from the point of view of an observer looking down from directly above the maze, we use homographies to transform the stitched source images. The high-level steps necessary for doing so are reproduced below.

For each stitched image:

1. Obtain a "skeletonized" binary representation of the image by thresholding color values to obtain only red regions, and then reducing the region to single-pixel width by iteratively performing morphological dilation and eroding operations

2. Find all lines in the skeletonized image using the Hough line transform- these lines usually lie along, or near, maze walls.

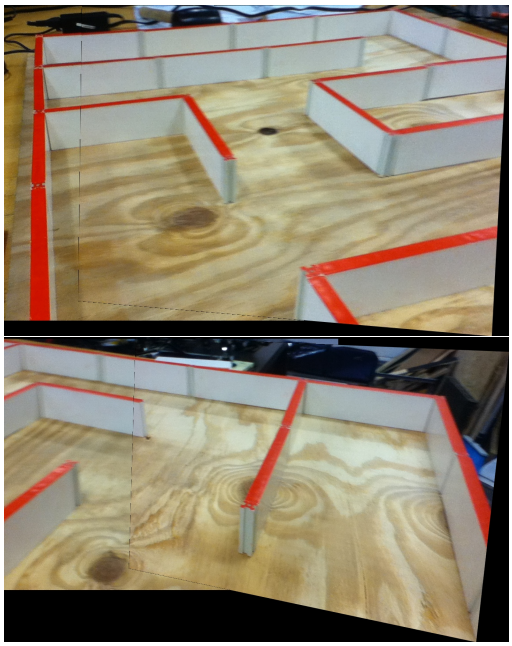

Figure 2. Example stitched maze images from the mouse's perspective.

3. Since skeletonized walls are not perfectly straight, the Hough transform will find multiple lines with similar slope and intercept for each wall. Divide the lines into groups according to slope, and further subdivide by intercept.

4. Obtain a single line for each wall by removing outliers in every cluster and then averaging slope and intercept.

5. Compute all points in the image where the averaged Hough lines intersect.

6. Treating the intersection points as polygon vertices, find all closed polygons in the image.

7. Identify the polygon enclosing the smallest area, since this is likely to be the perspective view of a square in the maze.

8. Compute the homographic transformation between the stitched image and a top-down view of the same scene by using the corners of the aforementioned polygon and the corners of a square centered in a 2500×2500-pixel image as keypoints. The large size of the target image is chosen so that the source image, after being transformed by the homography (which stretches some regions) does not experience boundary clipping.

9. Crop the transformed image by finding the smallest rectangular region that bounds the image information (most of the pixels of this image will be zero, and the transformed, stitched image will occupy a small quadrangle near the center).

In essence, this set of steps automatically finds the maze walls under perspective projection (as they appear in the images from the mouse's point of view), and then attempts to transform each image using the prior knowledge that each polygon enclosed by the maze walls is, in reality, a square, and would be seen as such from a top-down point of view.

In the final stage of the algorithm, all of the previously-stitched and transformed images, which now appear as pieces of a top-down maze view, are stitched together, and the same linefinding technique as before is used to identify the walls of the maze. The output of the algorithm is a textual encoding of the locations of all maze walls, which can directly be fed to a Micromouse to be solved and executed. Therefore, after all transformed images have been obtained:

1. Stitch all transformed images together to produce a single, top-down view of the maze.

2. Skeletonize the stitched image as in the previous stage

3. Perform the Hough line transform again to approximate the grid on which all maze unit cells lie.

4. Find the intersection points of the grid in the image.

5. Extract a digital representation of the wall locations by taking a small window of pixels between pairs of neighboring vertices, determining the total number of red pixels within the window, and then recording the existence of a wall connecting the given vertices if the total exceeds a predefined threshold.

After the wall locations have been determined, the optimal maze-solving path can be determined using standard graph-theoretic methods, and the mouse can consequently solve the maze.

4. Results

We implemented our algorithm using the Python language along with the OpenCV, NumPy, and matplotlib packages for computer vision and image processing, and numerical computation, respectively. We chose this platform over MATLAB and C++ (which also has support for OpenCV) to reduce implementation time, in order to be able to devote more effort to high-level algorithmic development. Also, the flexibility of Python in comparison to MATLAB for performing "non-numerical" tasks such as file input/output and project management made it a natural choice.

4.1. Output images

In this section, we present several images representative of the output of our algorithm at various stages of execution.

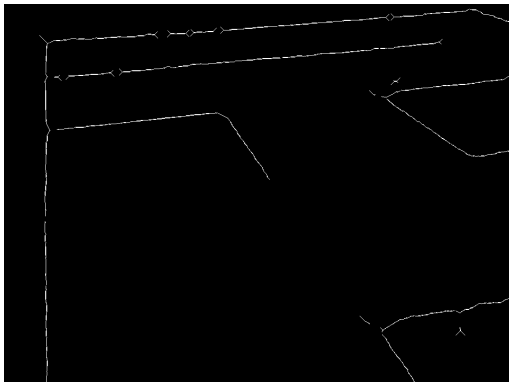

Figure 3. Single-pixel wall representations obtained from a skeletonizing process on red regions of the image

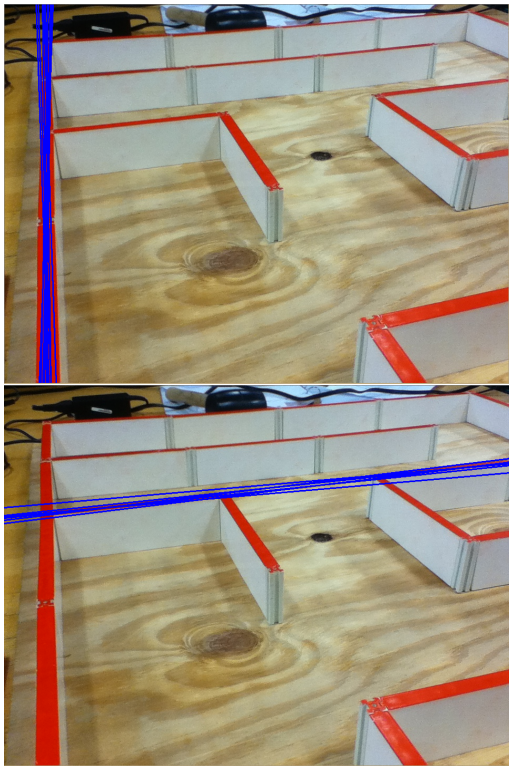

Figure 4. Groups of lines (blue) found by the Hough transform, each corresponding to a single wall

4.2. Performance Analysis

5. Discussion

5.1. Maze size

In order to demonstrate the effectiveness of our algorithm on a small test set, we constructed a 4×6-cell maze, which is smaller than standard 16×16-cell Micromouse mazes. Because of the nature of perspective, the red tops

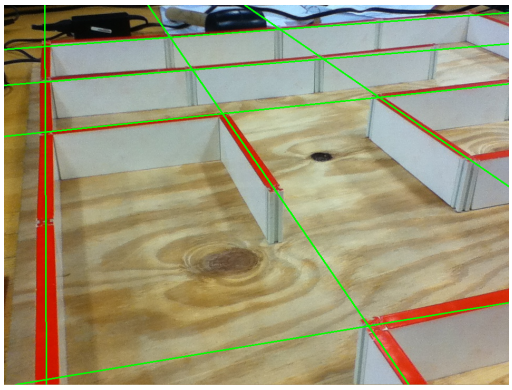

Figure 5. Final, averaged set of Hough lines (green) found for this stitched, perspective maze image

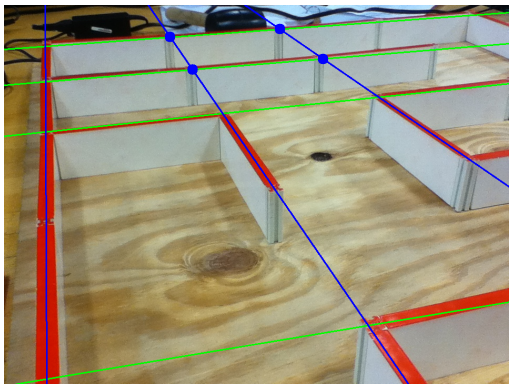

Figure 6. Perspective image with smallest detected square shown (blue dots)

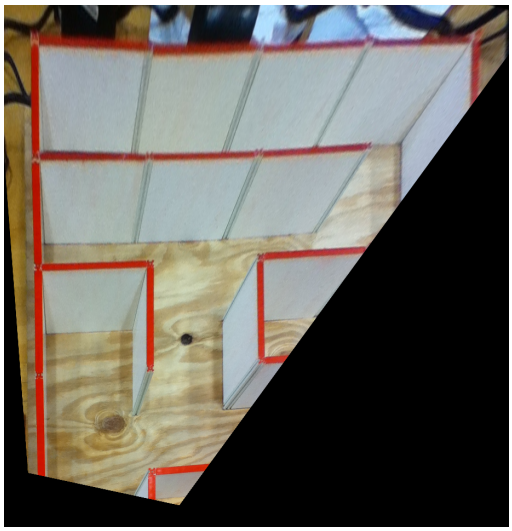

Figure 7. Image after being transformed by homography that transforms the shape found in Figure 6 to a square

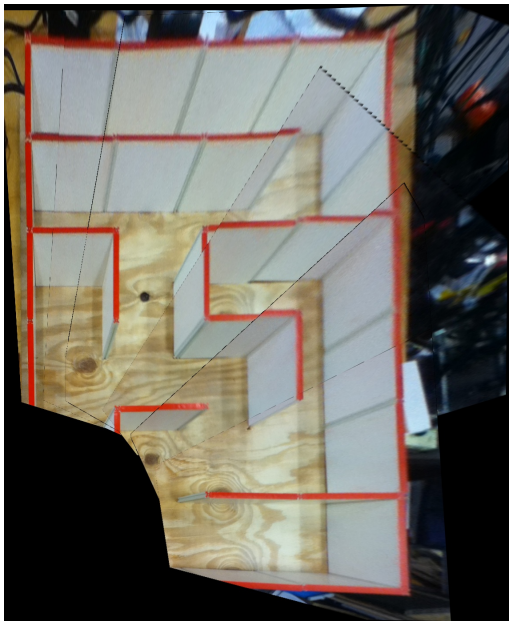

Figure 8. Composite image produced by stitching together all homography-transformerd, previously-stitched images

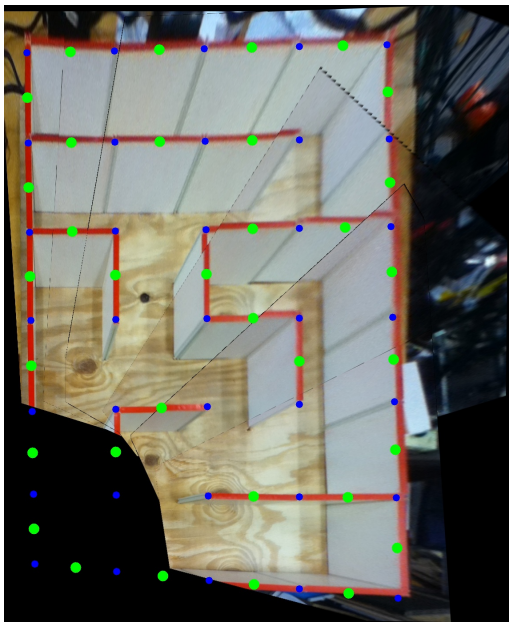

Figure 9. Extracted wall locations (green) and locations of maze gridpoints (blue), both computed by our algorithm

of maze walls at distance appeared much thinner than those close to the camera at the starting position, meaning that our homography-based technique results in distortion that increases approximately with radial distance from the camera position. Fortunately, Micromouse competition rules impose no restrictions on the height of a competing robot, meaning that successfully solve full-size mazes (which are significantly larger than our test cases) would require a camera mounted at a higher height.

Simulation	Brute Force	FloodFill	Our Method
1	714	206	114
2	750	236	128
3	858	236	144
4	700	199	82
5	630	190	74
6	666	191	96
7	744	546	90
8	721	273	65
9	755	191	37
10	944	220	160
11	776	240	162

Table 1. Total discrete steps required to solve a maze using naïve method, a popular algorithm in Micromouse competitions, and our algorithm, measured

5.2. Earlier solution attempts

Our first attempt at a solution to this problem consisted of using trigonometric information to obtain an orthographic projection. The principle is this: for a given line segment with fixed length in space along the maze floor, the size occupied in the image under perspective projection will decrease as the segment moves away from the camera. By deriving a closed-form expression for the relationship between apparent size in an image and distance from camera, we were able to implement a function that "stretched" an image in the vertical direction so that all pixels in that direction represented approximately the same distance in space. Stretching was accomplished by interpolating images to a higher resolution and then increasing the number of rows occupied by various regions of the image appropriately.

This approach was fairly successful for one-dimensional image rectification, i.e. cases in which an image was visibly "distorted" by perspective along only one axis, but it soon became clear that implementing a similar transformation for more general cases would be a formidable mathematical and practical challenge.

We also began by attempting to implement our own SIFT-based image stitching routine. We had originally reasoned that given the regularity of shapes in the image and well-defined corners at wall junctions, SIFT would be highly effective for feature matching between sequential images. In fact, the lack of available wall texture made such an undertaking nearly impossible, and in the end, we adopted the stitching routine (coincidentally, also SIFT-based) referenced in §3.

5.3. Difficulties encountered

5.3.1 Occlusion of region around starting point

As seen in Figure 9, the final image produced by our algorithm is not able to fully retrieve information about wall locations within a short distance of the camera, due to the angle and location of the camera. However, this would be easily rectified by using short-range proximity sensors mounted on the front and sides of the robot, coupled with the prior information that the starting location in any maze is always a corner (i.e. has two walls at a right angle).

5.3.2 Homography implementation

To calculate and perform the homographic transformations used to generate the orthographic image projections seen in Figures 8 and 9, we used the OpenCV function `getPerspectiveTransform()`, which accepts as input two sets of pixel locations describing the locations of the corners of two arbitary quadrangles in the source and destination images, respectively (in our case, the second quadrangle was a square). In order to accurately describe a homography, though, it was necessary that the order of the corners be the same in both input lists. Ensuring that this requirement would be satisfied deterministically in the face of varying image perspectives and maze shapes through the image set proved to be a challenging implementation hurdle to overcome.

Our eventual solution was, given the corners of a quadrangle in the perspective image, to calculate the mean x- and y-coordinates of the points to find the center of the quadrangle, and then determine a clockwise ordering by calculating the signed (i.e. in the range $\theta = [0, 2\pi]$ instead of $\theta = [0, \frac{\pi}{2}]$) angle between the horizontal line passing through the center and the the line connecting the center and a given corner point.

5.4. Future work

This project was a demonstration of the ability to recover maze information using visual data. We did not implement our algorithm on an embedded processor or directly on a functioning Micromouse. Also, our test dataset was somewhat limited in scope. Therefore, there is a possibility, if not a great likelihood, that the parameters of our implementation were adjusted in such a manner that allows it to successfully solve the given test data without being able to solve mazes with more varied topologies. With this in mind, along with the fact that we will again be competing in the IEEE Region 1 Micromouse competition, our future direction of work will be primarily focused on ensuring that our algorithm is both robust and efficient enough to function in all realistic scenarios and on embedded processors with Python support.

References

[1] "http://sites.ieee.org/r1/files/2013/03/2013-Region-1-Micromouse-Competition-Rules.pdf."

[2] N. Chen, "A vision-guided autonomous vehicle: An alternative micromouse competition," *IEEE Transactions On Education*, vol. 40, no. 4, pp. 253–258, 1997.

[3] "http://www.jacobsschool.ucsd.edu/student/student_involved/involved_org/matching_funds/docs/sample_proposal2_11-12.pdf."