INTRODUCTION

During this independent study, the main objective was to track boundaries of regions in an image. A boundary tracking algorithm [3] was developed by mathematics graduate student Zhong Hu, inspired by the robotic vehicle tracking algorithm developed by Jin and Bertozzi [1]. While the current algorithm provided excellent results, we would like to extend the algorithm to (1) simultaneously track multiple regions, (2) be more robust to noise, and (3) allow for parallel computing. We thus extend the algorithm to multiple “robot” boundary trackers. By doing so, running time of boundary tracking would also be decreased. The objective of this independent study was to design and implement an algorithm that would include multiple imaginary “robots” that would simultaneously track the boundaries of different regions in an image. We also explored automating the selection of initial points for the robots.

ALGORITHM:

Stopping Criterion:

An important component of the algorithm is deciding when we stop each robot. The answer is simple: we stop a robot when it has crossed the path of any other robot. More specifically, we stop a robot when the distance between its current boundary point and the path another robot is within a specified amount of pixels. To illustrate, we have the following situation:
We can examine the two robot case in Figure 1. Assume that robot1 is coming behind robot2. In this case, we can pre-determine an a pre-determined distance in order for robot1 to stop tracking, since it has reached the path of robot2 (here d1 can be less than d2, and vice versa). Let us call this distance R, since it will be the maximum possible radius of the circle circumscribing the polygon that robot1 is sweeping out. Our goal is to find the maximum value that R can take on. If we recall a little physics, we obtain a maximum trajectory range when the initial angle is at 45°. With that said, we have the following depiction:

![Figure 2. Max Trajectory](image)

Before starting our boundary tracking on the image, we first run the tracking algorithm based on tracking and angular velocity, using one robot, and starting at (0,0). We stop the robot when its angle reaches -45°. This will output D, the maximum distance that the robots can cover when sweeping in and out of the boundary. The pre-determined distance that we are looking for (R) is simply D/2. Looking again at Figure 1, either d1 or d2 will be less than R.

**Multiple “Robot” Boundary Tracking Algorithm:**

**Input:** velocity v, angular velocity ω, number of robots n

1. Automate n initial starting points for robots
2. Determine value of d using CUSUM filters (d = 1 or d = -1)
3. θ = 0
4. Calculate R based on v and ω
5. while robots have not intersected paths
   For each robot,
   (a) θi = θi + di * ω
   (b) xi = xi + cos(θi), yi = yi + sin(θi)
   (c) increase v if we have made a full circle

![Figure 3. Depiction of](image)
(d) determine value of \( d \) using CUSUM filters
(e) if we have crossed the boundary
   store boundary point
   set \( d = -d \)
   update \( \theta \) using angle correction (Jin-Bertozzi 2007)
(f) if robot intersects boundary path of any other robot (including itself),
   then stop robot

The algorithm developed provides a method of tracking boundaries in any given image. The tracking velocity \( v \) determines how fast (how many pixels in each step) the robot moves forward. The angular velocity \( \omega \) controls the turning radius of the curve. A small \( \omega \) will of course result in faster tracking but less accuracy. To determine whether the current robot is inside or outside the boundary, we use the CUSUM filters [2]. The CUSUM filters provide a method of determining when we have crossed a boundary, even in the presence of noise. We set \( d = 1 \) if inside the boundary and \( d = -1 \) if outside the boundary.

During each iteration, we loop through all the robots. If the current robot is outside the boundary, \( \theta \) will be decreased by \( \omega \). If inside the boundary, \( \theta \) will be increased. Next, the robot will move forward \( v \) pixels. This will cause the robot to move to a new position \((x, y)\), where \( x_i = x_i + \cos(\theta_i) \) and \( y_i = y_i + \sin(\theta_i) \). We will thus obtain a path that sweeps in and out of a boundary in a circular fashion.

If the initial point is more than \( v \) pixels away from the boundary, it will fail to detect it and will therefore cause the robot to infinitely trace a circle. To fix this, we simply increase \( v \) if a robot has made a full circle and continue to do so until it finds the boundary. Once it does, \( v \) is reset to its original value.

Figure 4. Increase of \( v \) in order to detect
When the value of $d$ changes sign, as determined by the CUSUM filters, we have obtained a boundary point. It will be the midpoint of the current point in the robot's path and the previous point. Lastly, when we detect a boundary point, we update $\theta$ using the angle correction formula [1]. It is used to flatten the angle when crossing a boundary, which is more efficient than crossing the boundary perpendicularly.

**RESULTS**

Initial experiments were run by having the user choose the $n$ initial points. The results are shown below.
As shown, initial experiments were successful. Each robot was able to stop tracking once it reached another robot’s path. Furthermore, the CUSUM filters helped the tracking be more robust to noise.

**Automation of starting points:**

During primary experiments, starting points were chosen by the user. However, a major component of this independent study was to devise ways of automating the starting points. The current algorithm first randomly samples the image until it finds \( n \) starting points that are inside the region(s) the robots will track.

The results are shown below:
Results shown in Figure 8 show excellent results. The random starting points were able to start in all areas of the image that we wanted to track. However, we can see that some robots do more work than others, since some robots hit another. Figure 9. Results using random starting points (Gaussian noise variance = 0.05).

When having many more regions in the image, making sure that all regions are initially hit is less likely, as shown in figures 8 and 9. We see that in both experiments, a region was left untracked. This is due to the fact that the points are random, and so it is not guaranteed that all regions will be tracked.
POSSIBLE ENHANCEMENTS

There are definitely much more efficient methods of automating starting points. We can perhaps keep the same idea, but instead make sure that every pair of robots is separated by a distance proportional to the size of the image. Another possibility would be to divide the image into equal-sized cells and make sure that we have robots tracking in each cell.

Furthermore, the current algorithm does not involve cooperative boundary tracking. We can extend the algorithm so that when robots are tracking, there is actually communication between the robots rather than just checking if they have crossed paths. Overall, the multiple robot algorithm provided great results.

REFERENCES


http://www.math.ucla.edu/~wittman/research.html