Campus.gov.il – Autonomous Robots Course

Task 2

Autonomous Obstacle Avoiding Drone Simulator

Gilad Barak 23/06/2020

Task Requirements

Write a program that simulates the flight of a drone inside a given maze without crashing into the walls or other obstacles. Drone should work also in manual mode using movement keys.

Considerations

- Maze is represented as a black and white raster image (black = wall). Note – in images with white border the border was also considered as a wall.
- Program operational parameters should be configurable through a configuration file
- Several operational parameters were given – they are listed and documented in the parameters file.

Implementation

Program was written in Python. The source code is heavily documented so detailed description is not provided here except for some highlights

- There is a file named parameters.txt which contains all operational parameters. Each parameter is documented in the file.
    - The same program is used for either manual or auto operation. The mode is set in the configuration file
    - One parameter which was not provided in the instructions but was included in the file is the rate of angular movement of the drone (how fast can the drone yaw). It is called “d_rotate”
    - There is a safety margin parameter called “margin” which defines the minimum range that the drone gets close to the walls. Instead of defining a fixed value, it is defined as the linear distance traveled during x control loops – e.g a margin value of 3 means “the linear distance covered during 3 control loops”. This way the safety margin changes during operation and depends on drone speed.
- The navigation algorithm, which practically is the algorithm which sets the movement heading by instructing the drone to turn, is based on the principle of movement towards the most open place. This is done by averaging the measurement of the distance sensors.

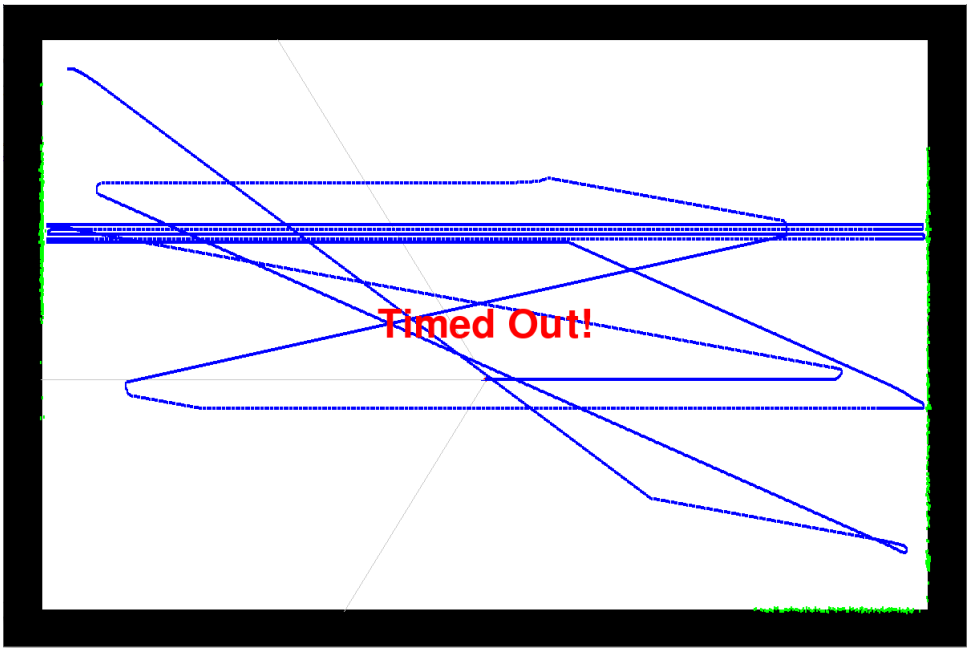
- Forward speed is set to be relative to distance from obstacles. This allows movement as fast as possible but also enables maneuverability near obstacles. This is done in the function “desired_speed”
- Visual aids are provided by showing the drone with its current heading and the distance rays measured by the sensors
- The control loop can be summarized as follows:
    - Measure distances with all sensors
    - Save distances that are within range – this is actually a map of the walls that the drone actually already mapped , and which can be used later. These distance points are drawn in green.
    - Calculate desired speed and the derived actual speed
    - Calculate forward linear movement
    - Calculate desired heading (relative to current heading, actually a rotation instruction)
    - Move according to heading and linear movement

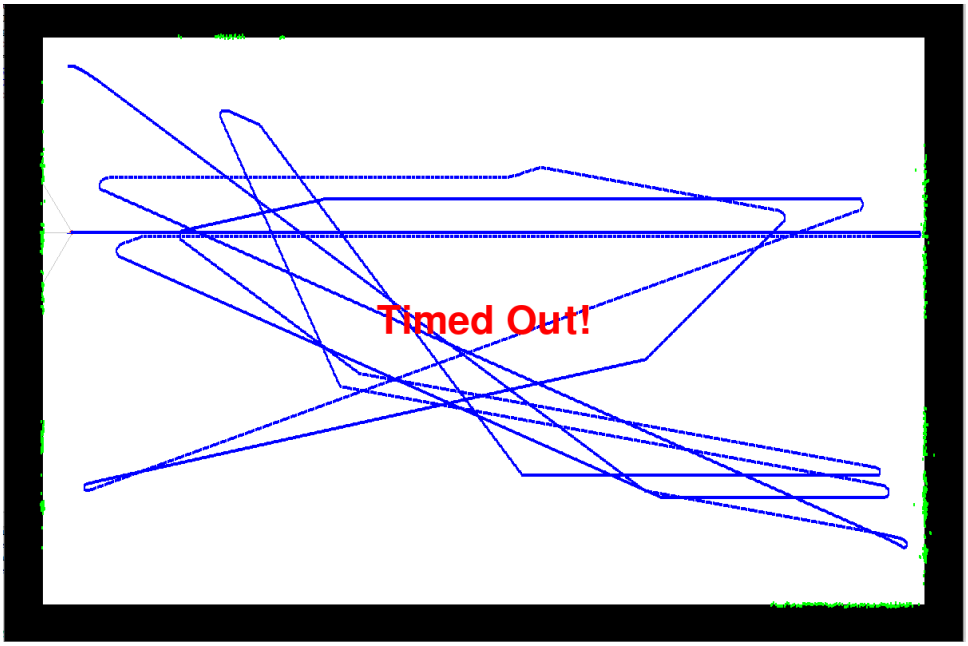
Execution and Results

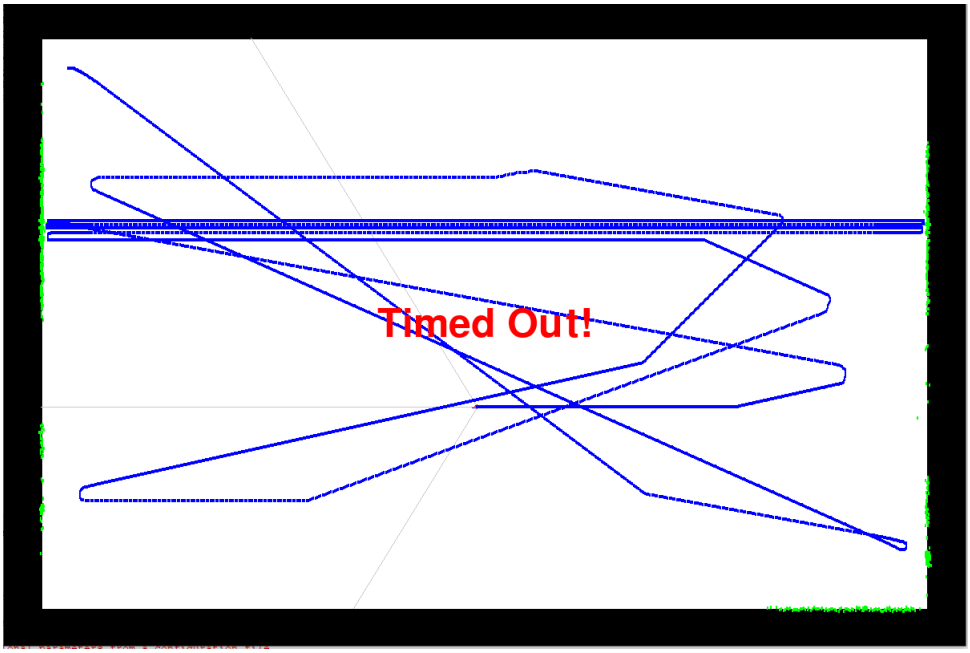
The program was executed on each of the 6 provided mazes. Each maze was run 3 times. Starting point and starting orientation was identical to all runs (x=200,y=200 heading=south/down)

Maze 1

As can be seen, although the maze is simple the mapping coverage (green pixels) is rather low. This is because most of the time the walls are not within the sensors range. Furthermore, once the drone gets into a “heads on” impact with a wall it begins to basically fly back and forth and only a random noisy reading gets it out of this track. The navigation algorithm creates a movement which resembles a ball bouncing from a wall. For such a geometry (actually for every topology that does not have “islands”) a “wall tracking” algorithm is probably better as it would provide better coverage.

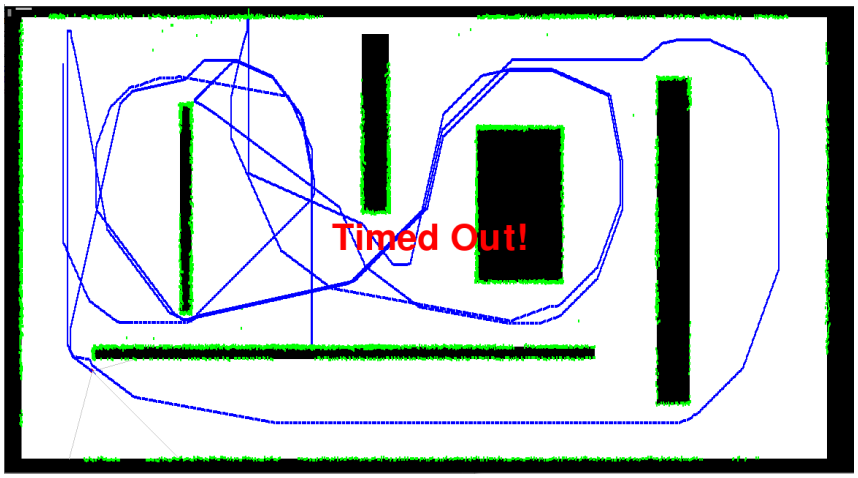


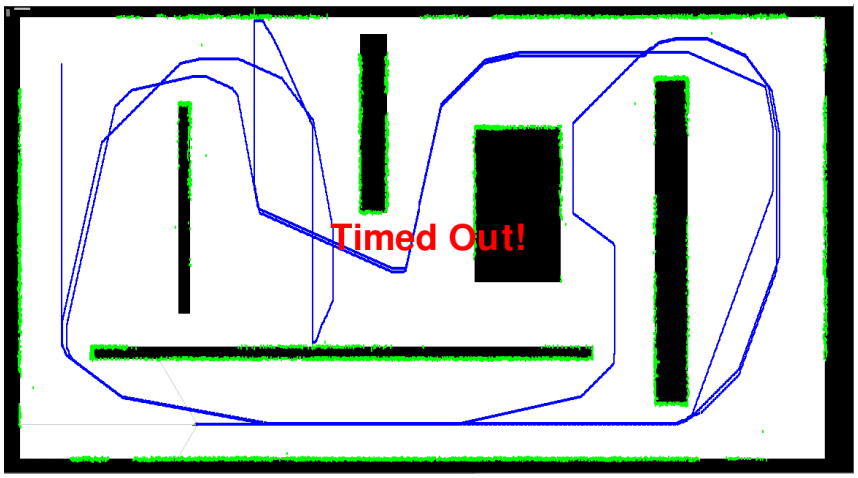

Timed Out!

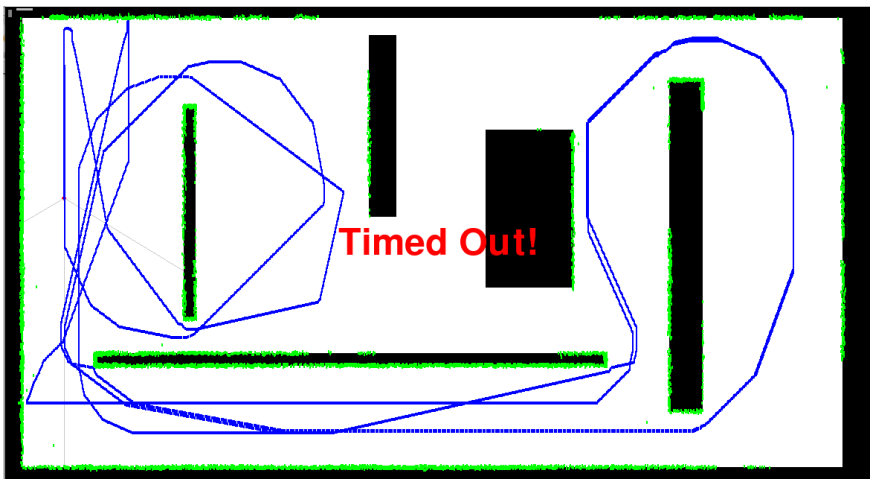

Timed Out!

Maze 2

In this maze the coverage was very good. Except for the last run, all but one of the "corridors" were covered in at least one run. However, in all cases most of the coverage was done within the first 2 minutes and time was lost on going through almost the same tracks 2-3 times. Improvement for such a situation may be implemented by adding "mapping intelligence" – i.e remembering the paths already covered and trying to avoid them on subsequent passes (i.e – turn left/right if finding that you are at an old point at the same heading), or by marking "junctions" and trying another direction if getting to them again.

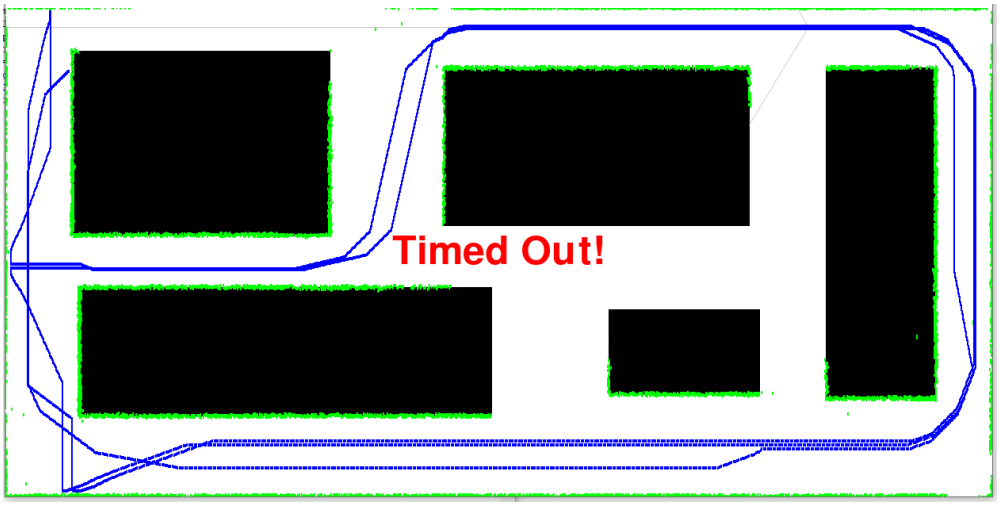


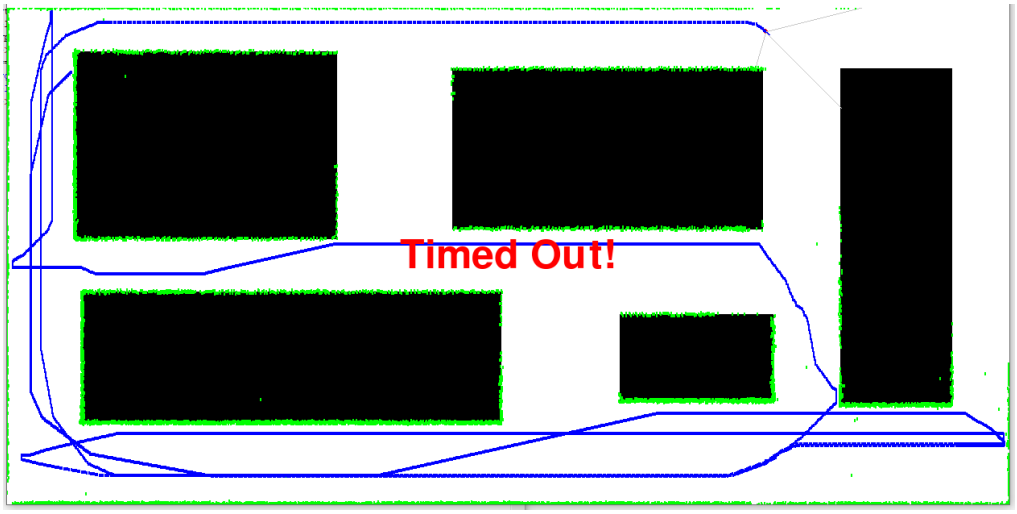


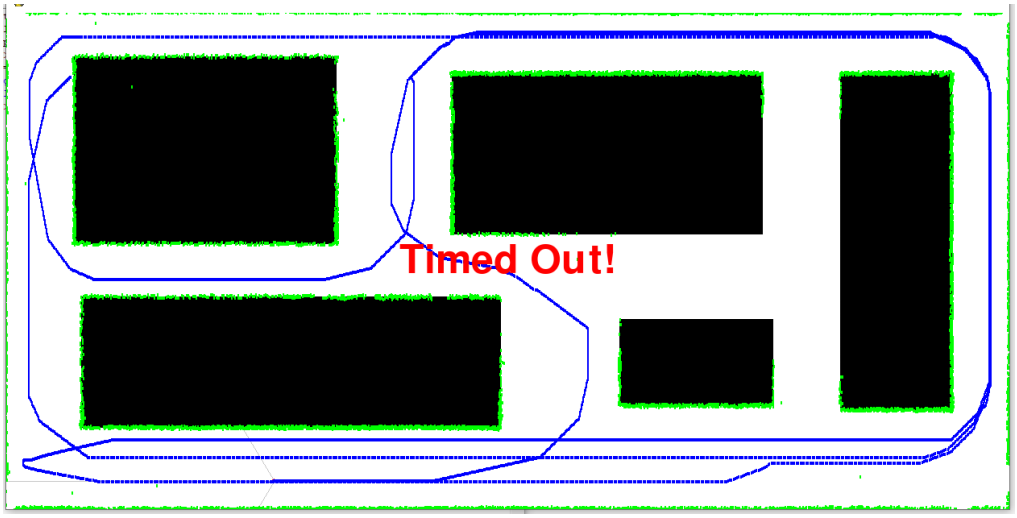


Maze 3

This maze results are very similar to those of maze 2, for the same reasons mentioned above. In this maze the issue of covering the same path were enhanced due to the fact that there is a wide external “corridor” and more dense maze internals with narrower entries so the drone is less “tempted” to go inside, nevertheless the random noise generated some signals that caused it to go into the maze at least once in each run.

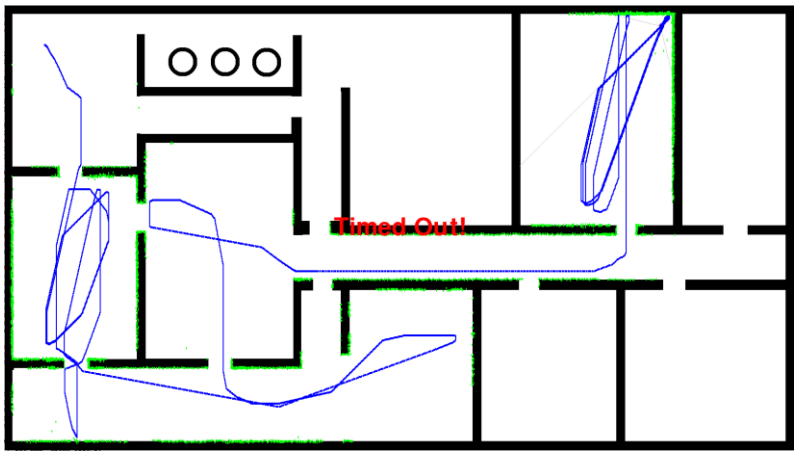


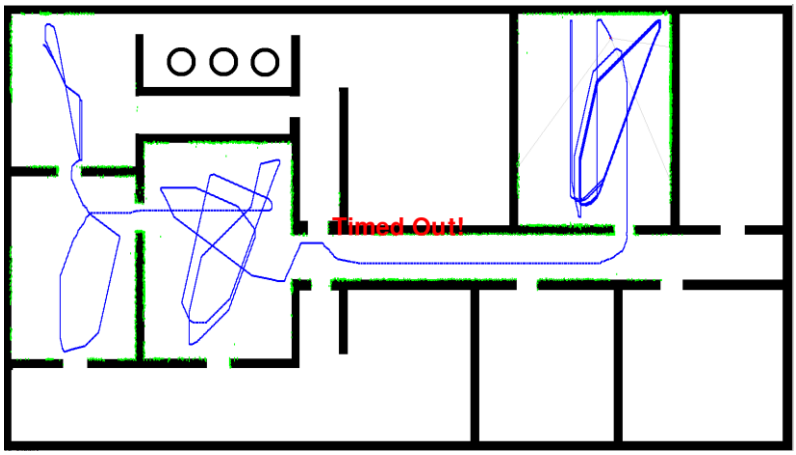


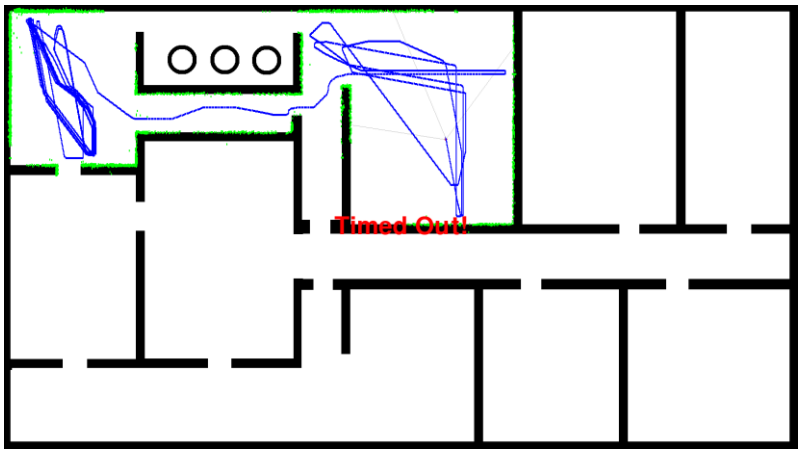


Maze 4

This maze is probably the most complicated for the algorithm that was implemented. The fact that it has many rooms with narrow entrances leads to the drone flying a long time inside a room that is entered before it is able to “escape” if at all. This caused very poor coverage of the whole maze. The third run practically got stuck in the first room before being able to move to the second room where it was stuck again. This type of maze definitely requires a “junctions” list to be able to take a different path when reaching them again. For a yet unknow reason the distance safety margin had to be increased substantially to prevent collision with the walls.

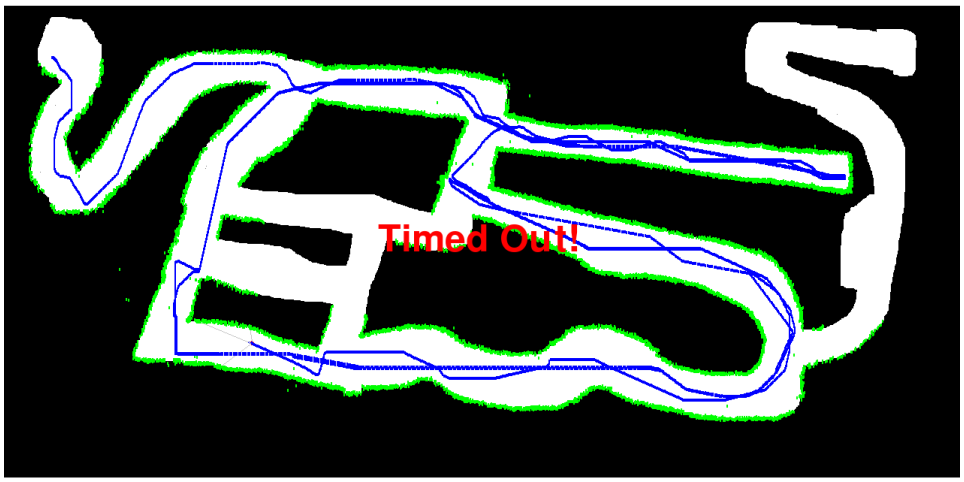


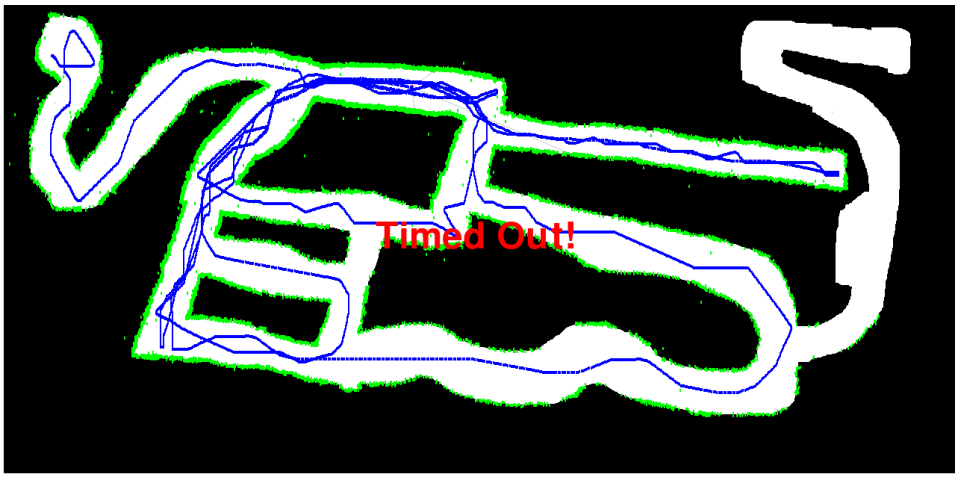


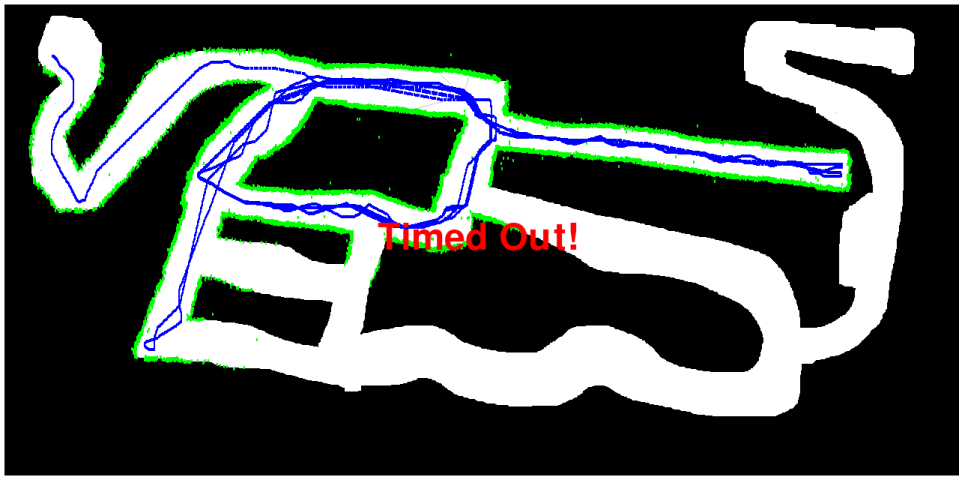


Maze 5

Results for this maze were mixed. Although in average about 75% of the maze was covered, as can be seen the performance variance was big. The limitation of the turning algorithm, which tends to follow gentle curves, prevented entry to places where an abrupt turn had to be taken to get into them. Turns that were taken were mostly due to arbitrary readings. Most of the coverage was done within half the time and the other half was wasted going through the same paths. For this maze the safety margin had to be slightly increased.

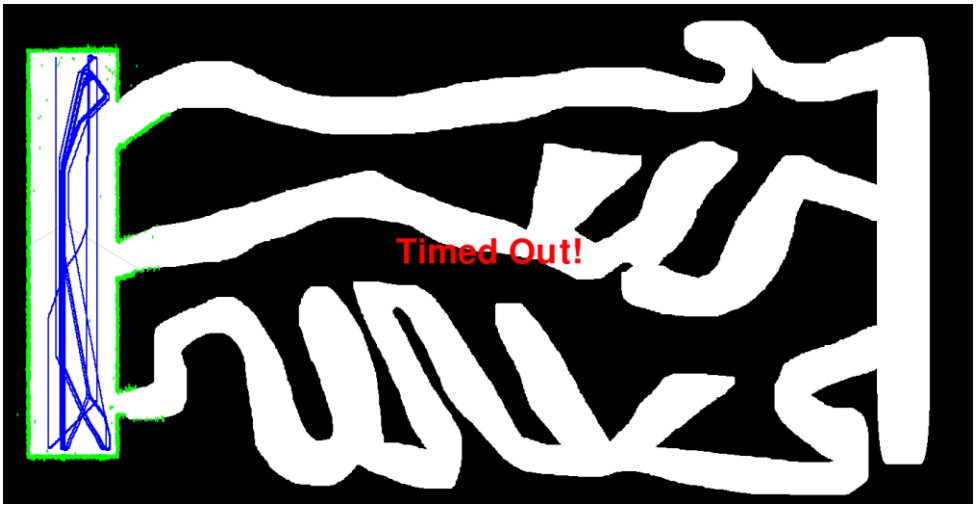


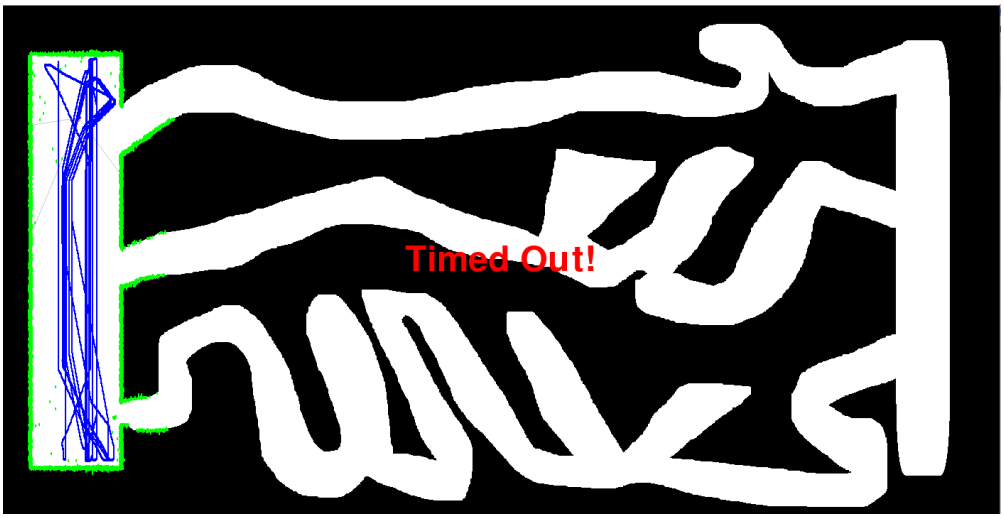


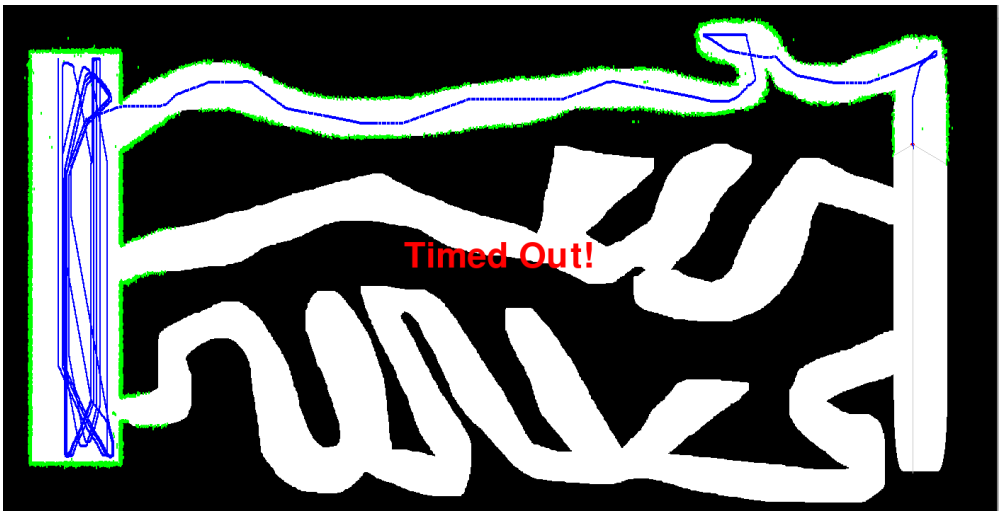


Maze 6

This maze was a failure for the navigation algorithm. The drone got stuck at the starting point room, except for the last run where at the end of the run it managed to escape into one of the corridors. Here, certainly, a “junction” list is required to be able to break out quickly.







Conclusion

For maze mapping an obstacle avoidance algorithm is not sufficient.

Obstacle avoidance algorithms should be used, as the name implies, to avoid obstacles and prevent collisions, and a different algorithm should be used to intelligently travel through the maze.