ME5751 Robotic Motion Planning – Autonomous Navigation

Johanna Martinez-Felix
Mechanical Engineering Department
California Polytechnic University,
Pomona
Pomona, US
johannafelix@cpp.edu

***Abstract*— To achieve autonomous navigation methods for kinematic control, potential map generation, path finding, and navigation concepts have been implemented. First, cost maps and buffer zones to generate a potential map will be discussed. Second, implementation of A* search, epsilon value and heuristic methods using the potential map will be covered. Third topic to cover will be navigation point tracking methods. Then, an overview of how Kinematic control is achieved through proportional control for point tracking of a robot described in the inertial frame. Followed by methods description for the application of a two-wheel drive truck. Lastly, a performance analysis and discussion.**

Keywords—autonomous, navigation, kinematics, path finding, potential map, A* search, heuristic, epsilon, navigation, point tracking, proportional control, two-wheel drive

I. INTRODUCTION (*HEADING 1*)

In this project the prompt given is to navigate a vehicle autonomously through a given map of obstacles and free space to reach a goal point. There are three major tasks that need to be completed to accomplish this: map processing, path planning and navigating to the goal. To process a given map, generating a cost map by calculating distance using breadth first search method was studied. For path planning two methods were studied via implementation, however A* search was the only path finding method implemented for the purpose of navigation. Selected due to an intriguing history in artificial intelligence. Often considered the first AI, it was one of the important research topics for methods to implement in autonomous navigation to be outputted from the 1966 Shakey project. For navigation to the goal proportional control of a differential drive robot is implemented.

II. POTENTIAL MAPS

Cost maps are a type of map generated from the original given map for the path finding algorithm for the purpose of better generating a viable path to the goal point. The map depicts color degradation from black/dark gray to white. Where white represents free space with darker shades of gray depicting closer distance to the object as shown in figure 1a and 1b for visualization.

Behind the color scaling of the calculated distances is numerical representations of these colors. To simplify the problem, it can be said to be a recreation of the common island problem. However, here a buffer zone allotting for the diameter of the robot is implemented. On the implemented potential map, free space has the numerical value of "1", obstacles are "0" and the buffer is represented by the number "255" shown in figure 2b. The implemented potential map can be seen in figure 2a.

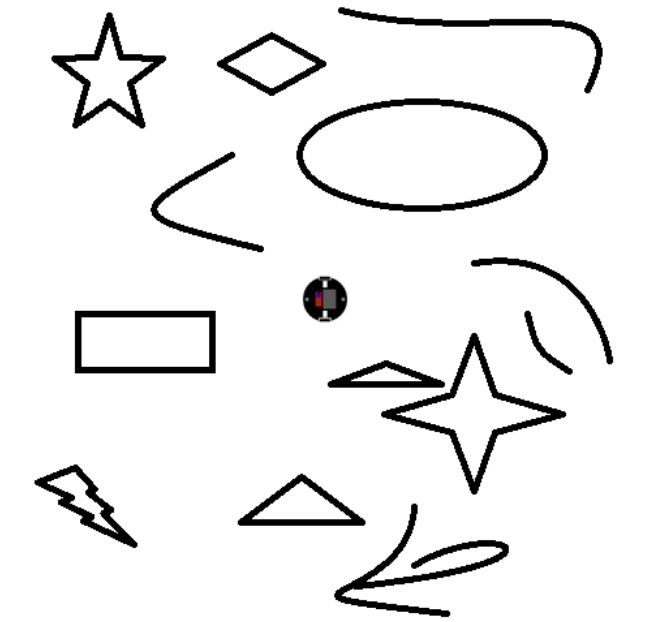

Figure 1a. Original map example – taken to be converted to potential map.

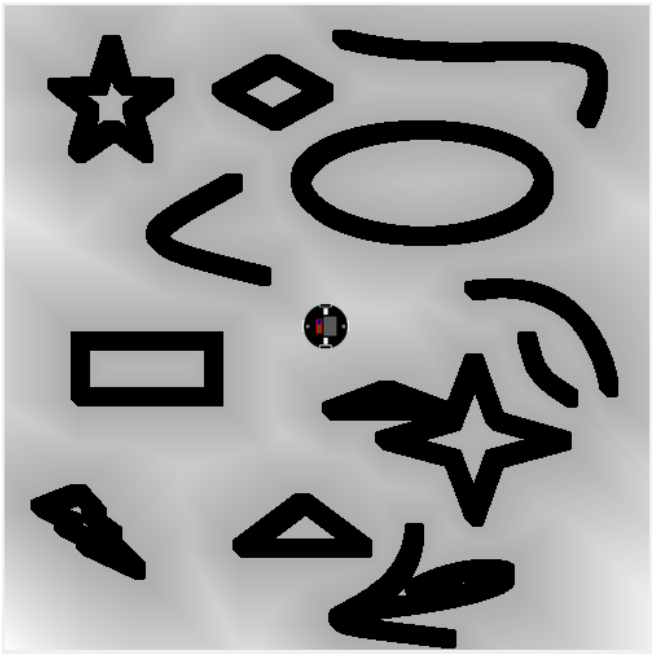

Figure 1b. Cost map with color scaling depicting distance from object.

Figure 2a Implemented map Summary: Island problem with buffer.

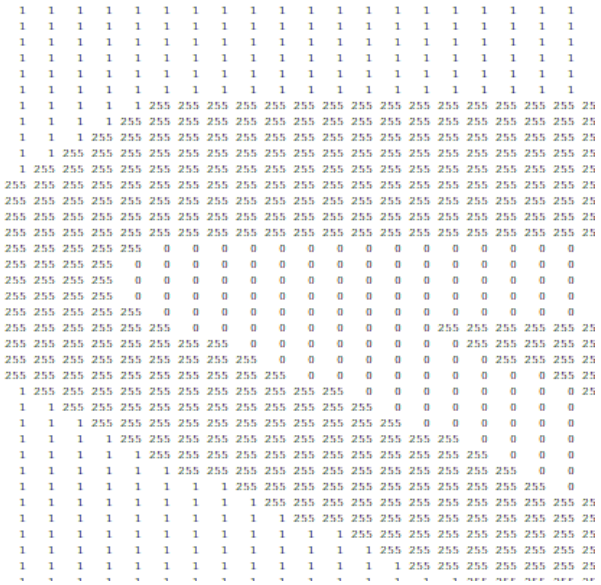

Figure 2b Closer look into implemented map. Summary: Island problem with buffer.

III. Path finding – A*

The amount of time it takes for the algorithm to generate a path depends on the heuristic method used, the number of neighbors searched and the epsilon or μ value selected (between 0 and 1).

The length of the path generated to a given point is reliant upon the same factor mentioned to affect the time it takes to generate a path. More specifically these factors control the number of times (iterations) the search algorithm runs though neighbor searches. Higher number of iterations means a longer wait time to find a path and a longer wait time to find a path meant a longer path. So low number of iterations would be beneficial. Lower number of iterations would mean finding a path sooner which also means finding a shorter path.

Figures 3, 4a and 4b depict the outcome of the path found and number of iterations taken using different heuristic method used, the number of neighbors searched and the epsilon or μ value selected

There are three common methods for solving for heuristic value. We will take a brief look at Manhattan, Diagonal and Euclidean.

1. **Manhattan** distance heuristic application equation:

$$h = abs\,(current\_cell.x - goal.x) + abs\,(current\_cell.y - goal.y)$$

The down fall of this method it that it can only move in 4 directions. (North, South, West, East)

2. **Diagonal** distance heuristic equation:

$$h = D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)$$

Where,

$$dx = abs(current_{cell}.x - goal.x)$$
$$dy = abs(current_{cell}.y - goal.y)$$

and D is length of each node(~1) and D2 is diagonal distance between each node (~ sqrt(2)). This method is strictly for moving in eight directions

3. **Euclidean** Method equation:

$$h = sqrt\,(\,(current_{cell}.x - goal.x)^2 + (current_{cell}.y - goal.y)^2\,)$$

Here, the number of neighbors we can move to does not matter.

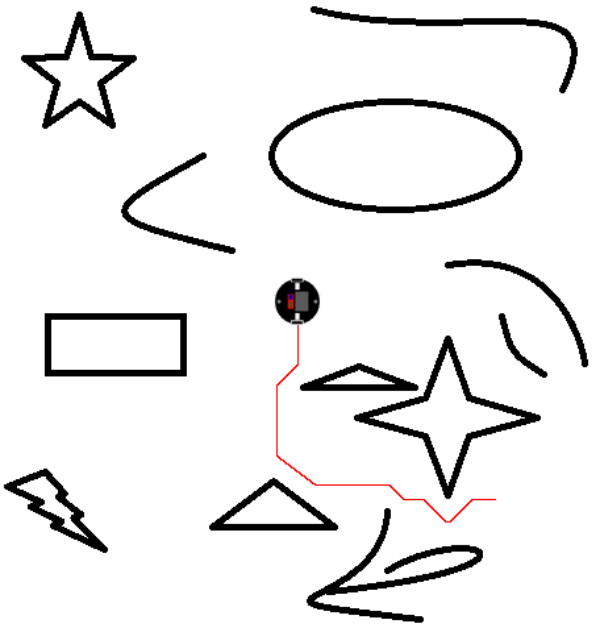

Figure 3 Manhattan, 4 neighbors, [150 -150], μ = .5, count(63382)

Figure 4a. Euclidean with 4 neighbors with coordinates [150 -150] , μ = 1, count(68603)

There are counter intuitive actions the algorithm depicts. For example, figure 4b uses the Euclidean, searches 8 surrounding neighbors and has an epsilon value of zero; if the epsilon value is changed to 1 the path found would go right through the objects in the way of a direct straight path to the goal.

Figure 4b. Euclidean, 8 neighbors, [150, -150], μ = 0, count(114829)

IV. Navigation to the Goal

To accomplish navigation, the points found from the A* search algorithm needs to be translated into a series of readable points for the robot to move towards. The path is discretized into a few points rather than using every point found from the A* path as a goal.

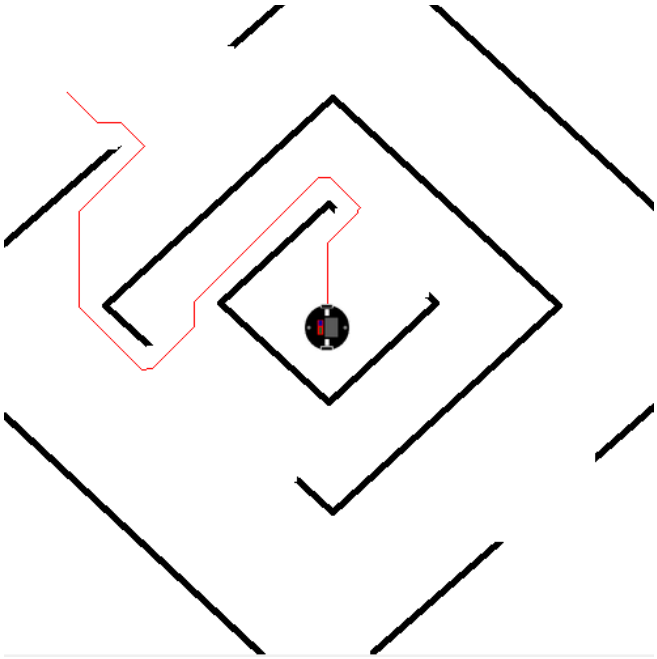

Figure 5. A * path found has 510 points creating the path.

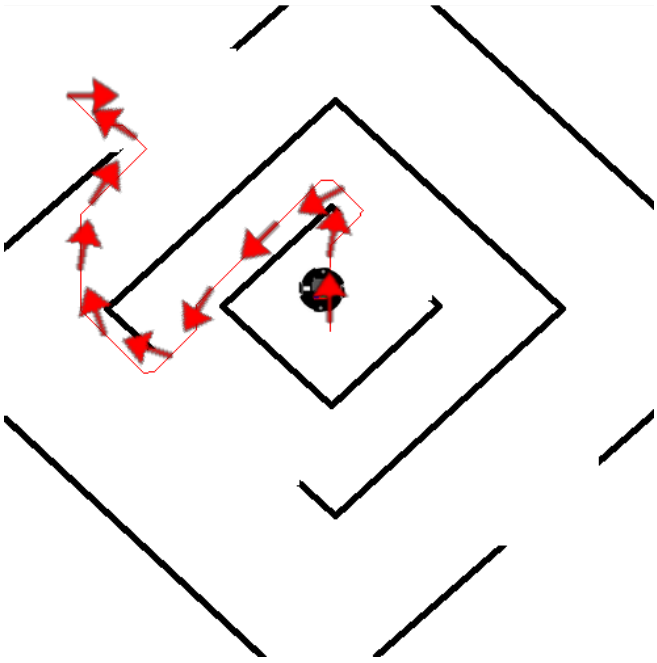

Figure 6a. Discretized path is every 50 points in the A* path . Turns 510 points from A* path to 11 goal points for the robot to go to.

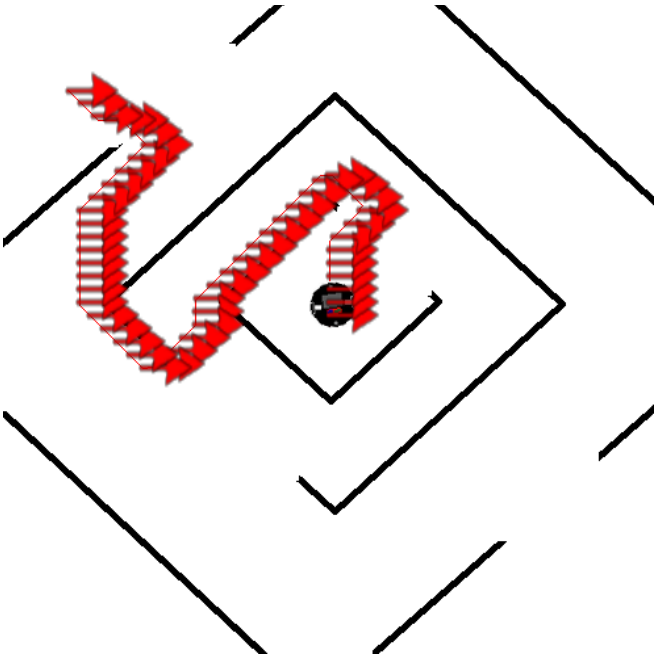

Figure 6b. Discretized path is every 10 points in the A* path. Turns 510 points from A* path to 51 goal points for the robot to go to.

The angular trajectory needs to be corrected so the robot can move straight to the next pose. If we recall from kinematic applications for movement of the robot, angular trajectory can be controlled with geometric equations. Used here in equation (1). The correction of angular trajectory can be seen between figure 6b and figure7.

$$\alpha = atan2\,(\Delta y, \Delta x) \qquad (1)$$

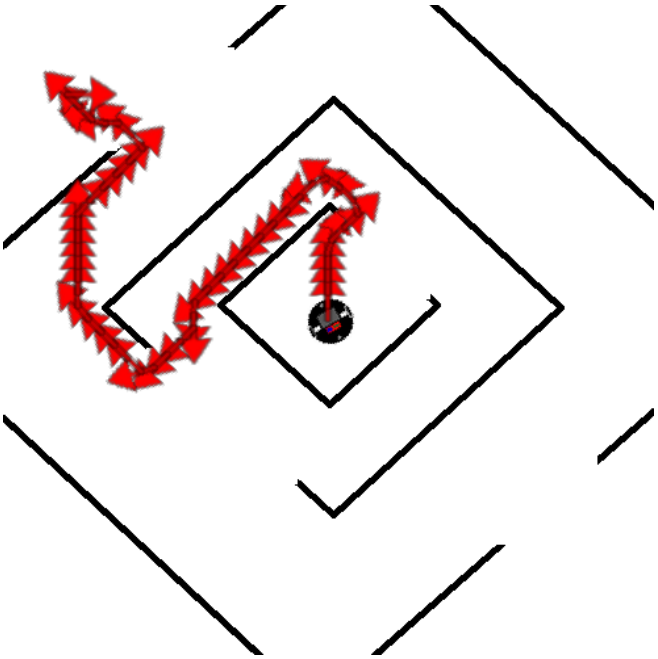

Figure 7 Angular trajectories of the discretized path is corrected. Equation (1) implemented.

The density of the discretized points can be modified. Figure 6a and 7 show the difference between different levels of discretization density. Density of goal points on the path correlates to the level of accuracy in execution the robot moves in to go to the next goal point compared to the original A* path. The more discretized goal point there are the longer it will take for the robot to get to the goal.

V. Dynamics of a Truck

In proportional control of a differential drive robot two wheels of the robot had individual motors. Which means that the speed of each wheel was independently controlled. If the robot was to turn; one wheel had to slow down and the second wheel had to speed up to go in the direction of the slow wheel. However, in the dynamics of a truck, both left and right wheel move cohesively. Movement becomes simplified to a single motor control problem. The single motor controls the speed of both left and the right wheel, the speeds of the wheels are equal. For the vehicle to achieve turns both wheels move by β, which controlling angular trajectory from the current position to the next.

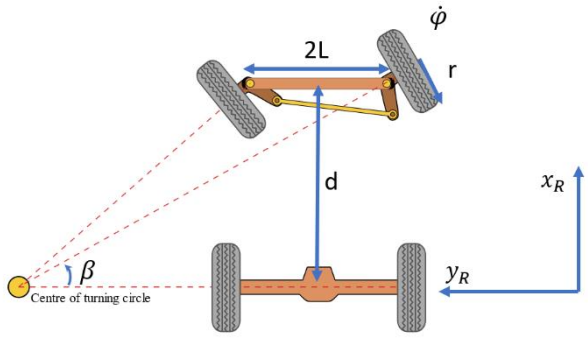

Figure 8 Dynamics of a truck

Car width $2L = 16$, car axle distance $d = 20$, wheel diameter $r = 3$

Front-right wheel steering angle $\beta < 40°$, no wheel speed should be above 20rad/s (in your program, leave these parameters as constants)

Figure 9 truck variable descriptions. Note that beta and speed are constants so they can be inputted manually.

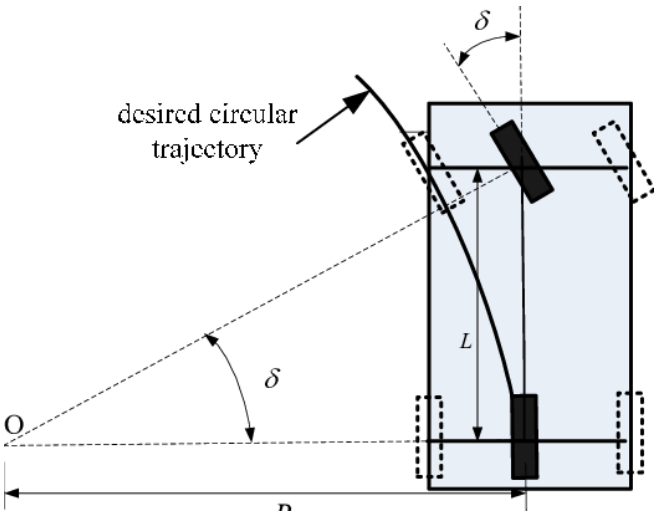

Figure 10 Simplified Ackerman. Bicycle dynamics

VI. Performance Analysis

A* algorithm always finds a path to the goal point when possible, however, sometimes the path will run right through the obstacles. This happens depending on the location of the goal point, the map given, and/or the A* search setting of heuristics, neighbors and μ. We are given a specific map and goal point that cannot be changed. So, the expected action to take in order fix the error of generating a path through the obstacles is to modify the A* search settings.

For example, Figure 11 and 12 uses the same heuristics method and number of neighbors. The difference between then is μ. Which makes a big difference in correcting that error of planning a path right through obstacles.

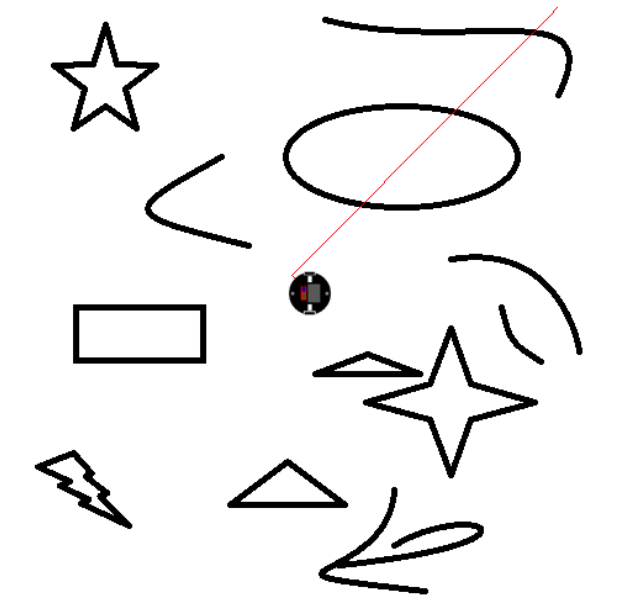

Figure 11 Euclidean, 8 neighbors, μ = 1

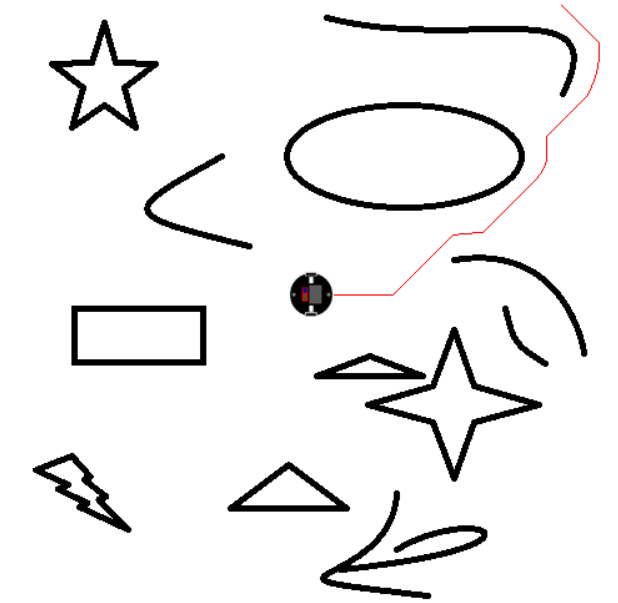

Figure 12 Euclidean 8 neighbors, μ = 0

Robot wheel speed and wheel angles control execution efficiency (smoothness of differential drive car traveling through the path). Like the A* algorithm p-control also has a lot of modifications in order to adjust the quality of the robot movement. For P-control the factors that need modifying are the k-rho, k-beta, and k-alpha. When referring to differential drive robot proportional gain Kp refers to local stability. Local stability is typically an issue, however, if the following the conditions are applied the robot control system sees better stabilization:

$$kp > 0 \; ; \; k\beta < 0 \; ; \; k\alpha - k\rho > 0 \qquad (11)$$

or the strong stability conditions

$$kp > 0 \; ; \; k\beta < 0 \; ; \; k\alpha + 5\,3\,k\beta - 2\,\pi\,k\rho > 0 \qquad (12)$$

for heightened stability of positional control.

The changes need to be made depending on the path. If the path had a lot of turns and the angular trajectories change dramatically often the robot is expected to make a lot of turns. If the path is more straight than bendy is easier to travel to. Different paths may cause the need to tune robot navigation angles to change.

VII. Discussion and Conclusion

Iterations and paths can be manipulated using heuristics, number of neighbors searched (4 or 8) and μ. Different heuristics, neighbor and μ value combinations have different path output, different iteration numbers and different times generation.

Usually, higher number of iterations means a longer wait time for the path to be generated and the number of iterations is controlled by the distance the path needs to be to get to the goal point.

Depending on the map, goal, heuristics, neighbor and μ combo the path will go thru a wall. The issue is easily fixed by adjusting μ or changing heuristics.

The car moves slowly depending on the controller settings; fine tuning is required depending on the path.

There are six variables for the user to control, k-rho, k-alpha, k-beta, heuristic method, neighbors, and μ value. Autonomous navigation performs better or worse (path, time, speed, movement) depending on the set up of all the previously mentioned.

Acknowledgment

Thank you to Blob gamed for providing in depth explanation and examples of path finding methods. As well as Dr. Chang for helpful office hours.

References

[1] R. Siegwart, I. Nourbakhsh, D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed., The MIT Press : Cambridge, 2011. pp 57 – 98.

[2] Y. Chang, Robotics Motion Planning , Lecture Note Set #1-13. CPP: ME5751, 2022.

[3] J. Martinez-Felix, ME 5751 - Module 1: Proportional Control for Point Tracking, CPP: ME 5751, 2022

[4] J. Martinez-Felix, Module 3 : Path planning with A* search, CPP: ME 5721, 2022

[5] https://stackoverflow.com/questions/40205223/priority-queue-with-tuples-and-dicts

[6] https://www.geeksforgeeks.org/python-dictionary/

[7] https://www.w3schools.com/python/python_dictionaries.asp

[8] https://www.geeksforgeeks.org/priority-queue-set-1-introduction/

[9] https://www.redblobgames.com/pathfinding/a-star/implementation.html#optimize-graph