Program-Based Algorithms for City Map Streets Recognition and Pathfinder

Ahmed Ismaiel[a)]

Ural Federal University, Yekaterinburg, Russia

[a)]*Corresponding author: en.ismaiel@gmail.com*

Abstract. This article presents a street recognition and pathfinder algorithm. In addition, It explains the programming sequence of this algorithm by using Python and Processing programming languages. The algorithm deals with modified grid maps described by free and occupied cells. It recognizes streets inside the map and categorizes them into horizontal and vertical streets. The algorithm also has a search function to find the path between any two streets.

INTRODUCTION

Pathfinder and street recognition is mostly used in navigation applications not only in the transportation field but also in mobile robot fields. Pathfinder is a part of path planning approaches in which an algorithm tries to find a valid path between a start and destination location in the robot environment. This path should be collision free and satisfying optimization conditions. Robot environment may have static obstacles or moving obstacles, that's why there are types of pathfinder algorithms that deal with static environments and other types to deal with dynamic environments [1]. In this research we deal with a static obstacle environment. Pathfinder algorithms that deal with static environments depend on graphical methods like visibility graph [2].Another common pathfinder algorithm that relies on graphical methods are using grid maps like in [3]. Grid map is a map where the environment is discretized into squares of certain resolution [4]. In general, decomposing environment into grid map is a successful base for many pathfinder algorithms as Dijikstra [5] and A* [6] algorithms. These algorithms depend on the grid map searching approach to investigate the cells from the start point to the goal point in order to find the shortest path. However, the novelty of this research is that the algorithm is dealing with a city map and converting it into a grid map with free cells representing only the valid streets. Arranging the resulting streets into a graph representation in order to find the path. There are some algorithms that use the same approach like Depth First Search [7] algorithm but in this research the developed algorithm is able to find more than one valid path. This paper is organized as the following: at the beginning is the introduction, an overview about the research and programming tools used to program the algorithm. Then, explanation of the algorithms sequence, street recognition algorithm and pathfinder algorithm. At the end of the paper we show the results and write our comments on the discussion.

RESEARCH OVERVIEW

In this research we develop city map streets recognition and path finder algorithms. Algorithm workflow is shown in figure 1. At the beginning the algorithm prepares the city map shot and converts it into a grid map matrix. Then the algorithm starts to check free cells in the grid map trying to understand and categorize them into horizontal and vertical streets. The resulted streets are being filtered and combined in order to prepare the streets cross matrix. The cross matrix is used by the path finder algorithm to find a path between any two streets with the lowest search cost. This paper is a part of a research project to develop an autonomous differential drive mobile robot for smart city application. A collision avoidance algorithm is developed in [8] as a part of this project, and this paper presents the first part of a navigation algorithm for the mentioned robot.

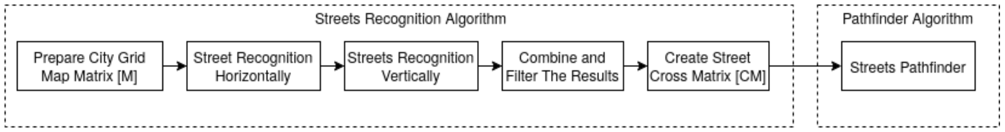
Streets Recognition Algorithm
Pathfinder Algorithm
Prepare City Grid Map Matrix [M]
Street Recognition Horizontally
Streets Recognition Vertically
Combine and Filter The Results
Create Street Cross Matrix [CM]
Streets Pathfinder

FIGURE 1. Algorithms sequence block diagram

PROGRAMMING TOOLS

Two programming languages are used for algorithms implementation: Processing and Python. Processing is an open-source programming language and Integrated Development Environment (IDE). Processing language is based on Java and provides a variety of libraries and classes to deal with media arts and graphics [9]. Processing is used to import and prepare the city grid map. For streets recognition and path finder algorithms Python programming language is used. Python is an open-source general purpose programming language with simple readable syntaxes. It has powerful libraries for algorithms modeling and mathematical representation [10]. Numpy library [11] is used for algorithm implementation and matplotlib library [12] is used for plotting the results.

STREETS RECOGNITION ALGORITHM

Preparing City Grid map

The Grid map divides the map into small square cells. It can be characterized by map width (Mw), map length (Ml) and map resolution (Mr) which represents the side length of the square cell. Figure 2 shows a sample of a grid map and indicates its parameters. It also shows the cell indexing, free cells and occupied cells. We are going to use a map shot of Yekaterinburg city center, Russian Federation as an example to implement the algorithms. The center of the shot has the coordinates 5650’07.9"N 6036’45.0"E and is shown in figure 3. Processing is used to configure the grid

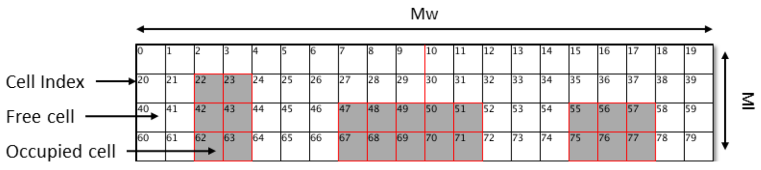

FIGURE 2. Grid Map Sample

map and to convert the map shot into image. Then it shows the grid map and map shot in the same canvas. For the grid map we configure the parameters as the following: Mw = 1000 pixels, Ml = 640 pixels and Mr = 20 pixels. These parameters are being estimated and configured according to the details of the map shot. However, the higher (Mr), the more complicated calculation and therefore, more processing time is consumed by the algorithm. City grid map is exported as a matrix M = [NR x NC], as (NR) represents total number of rows and (NC) represents total number of columns (NC = Mw/Mr) and (NR = Ml/Mr). Then, the map is being filtered to keep white pixels as it and convert any other pixels with color level below the white into black. Then, cells with white contents are converted into occupied cells. Such conversion is conducted because the white color represents the streets in the map shot. However, this is not an accurate practice as there are other details that are white colored, for example in figure 4 we notice that pixels representing river “Iset” is white colored. That’s why we have to manually correct these details till we get the exact list of occupied cells. City grid map is represented by a matrix (M) with zeros and ones elements. Occupied cells are represented as ones and free cells as zeros. Figure 3, 4 and 5 are showing the three steps to convert the city map shot from a simple image to a gird map.

Streets Recognition Horizontally

The algorithm starts by checking cells in each row in a horizontal direction. For each row, it collects the portions that contain cascading free cells and stores each horizontal portion (HP) as an item paired with the following elements: portion index (PI), portion first cell index (FC) and portion last cell index (LC). The algorithm repeats the previous check for each row in the grid map and stores all (HP) items in an array. Figure 6 shows flowchart of algorithm searching sequence to extract all (HP). I1 and I2 are an iteration variable over each row and column. After collecting all the (HP) , the algorithm collects the portions that have the same (FC) and (LC) and located in cascading rows, and stores them as a Horizontal Street Item (HSI). For each HSI item, Number of cells from (FC) to (LC) represents the

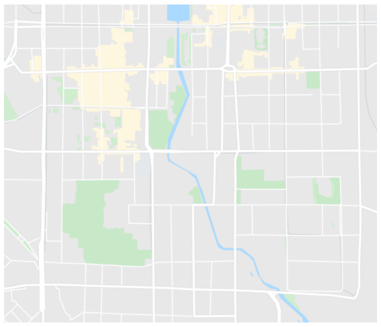

FIGURE 3. City map shot

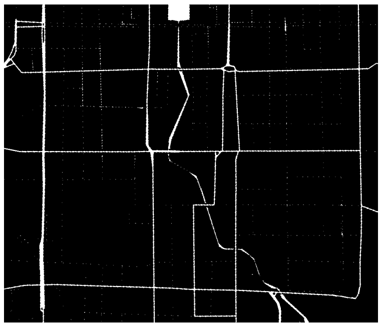

FIGURE 4. City map shot with filter

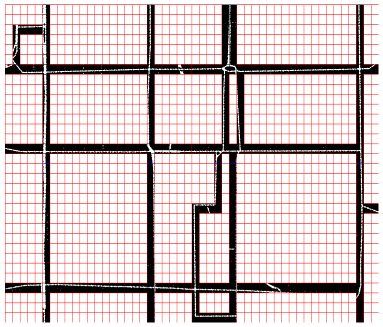

FIGURE 5. Grid city map shot

length of the (HSI), and the number of cascading rows represents the width of (HSI). By comparing the length and width of (HSI), the algorithm eliminates the (HSI) elements whose width value is larger than its length one. Finally, each (HSI) item is stored paired with all cell indices included in the (HP) items corresponding to each (HSI).

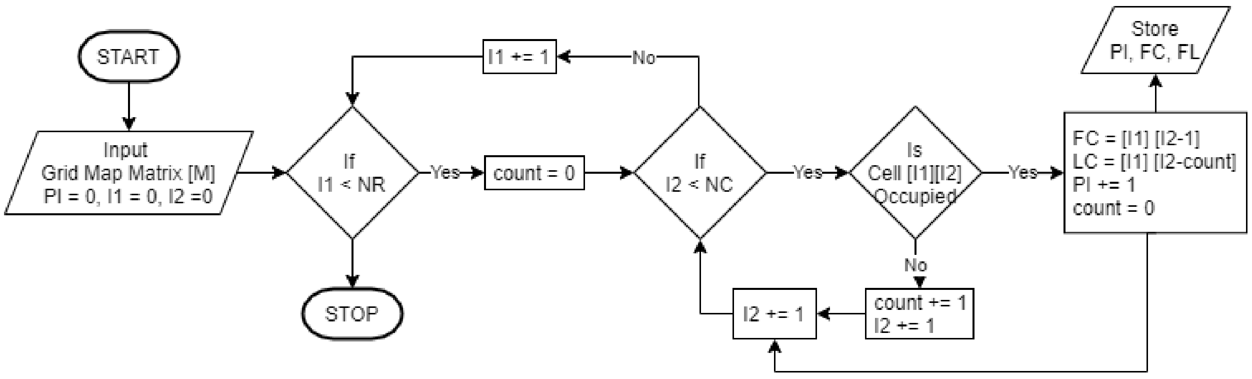

FIGURE 6. Flowchart of horizontal streets recognition process

Streets Recognition Vertically

Algorithm follows the same process as in street recognition horizontally except that the algorithm checks cells in each column in a vertical direction. Then it collects vertical portions (VP) and stores Vertical Streets Items (VSI). Another difference is that in vertical recognition the algorithm eliminates all (VSI) that has a width value larger than or equal its length. In this case, the algorithm avoids repeating the streets with width values equal to its length in both (HSI) and (VSI). As in horizontal recognition the algorithm eliminates all (HSI) that only has width value larger than length

Combine and Filter the Results

Algorithm combines all (VSI) and (HSI) in one array-list (SL) with length (SN). Length of the array-list (SN) is the sum of (VSI) and (HSI). There is a case where a recognized street is a portion of another bigger street. To eliminate these streets, the algorithm compares cells of each (HSI) or (VSI) with cells of each item in (SL). At this point, the algorithm has created a list of all horizontal and vertical streets in the city grid map. Each item in this list represents a unique street in the city map and contains the following information: Street Index (from 1 to SN), Street Type (HSI or VSI) and Street Cells (all the cells included in this street).

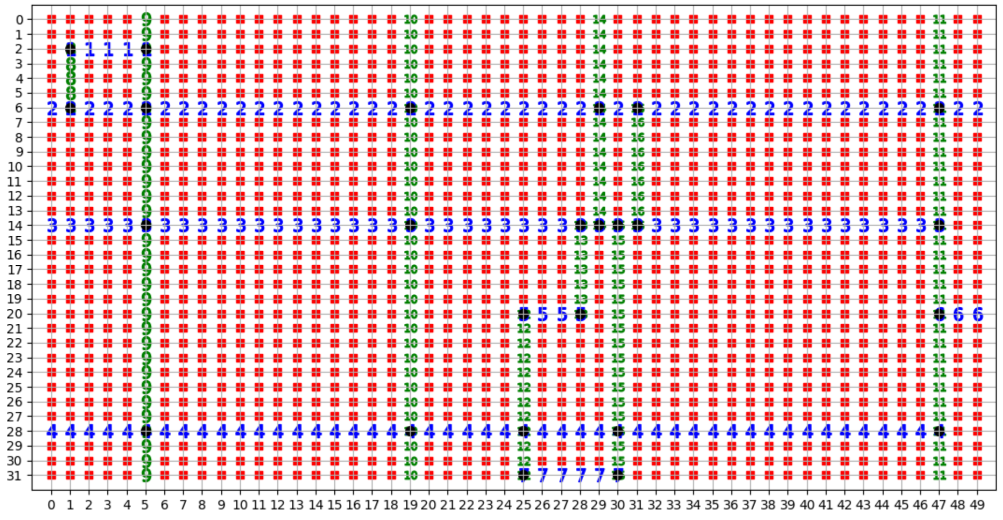

FIGURE 7. Grid City Map Results from Streets Recognition Algorithm

Create Streets Cross Matrix

The last stage in the streets recognition algorithm is to create streets cross matrix (CM). This is a zeros and ones matrix with size CM = [SN x SN]. In this matrix, rows and columns represent all street items in (SL) array-list. The algorithm finds street items with common cells by checking cells of each street item in (SL) and comparing them with cells of all the other street items in (SL). In (CM) matrix street items with common cells have one value and other streets have zero values.

no.	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
01	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
02	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0	1
03	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1
04	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0
05	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
06	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
07	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
08	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
09	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
10	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
11	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
12	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
13	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
14	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0
16	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE I. Streets Cross Matrix (CM)

PATH FINDER ALGORITHM

Pathfinder algorithm gets (CM) matrix, start street and destination street as input data. The algorithm searches a path between start and end streets in levels. In each level, it uses the (CM) matrix to get all the streets crossing the streets stored in the previous level. For instance, in search level one the algorithm gets all the streets that cross with the start street and stores them in an array. Then, the algorithm populates search level two by getting all the streets that cross each street item in search level one. If there is at least one valid path between the start and destination streets, the algorithm repeats the search process till it reaches the destination street. However, if there is no valid path between the start and destination streets, the algorithm repeats the search process (SN) times and then stops. The algorithm finds the paths with the lowest search level. It returns the resulting paths as an array items and each item contains the sequence of streets from the start street till the destination street. In case if there is no path between the start and destination streets, the algorithm returns False Boolean value. Figure 8 shows the flowchart of the pathfinder

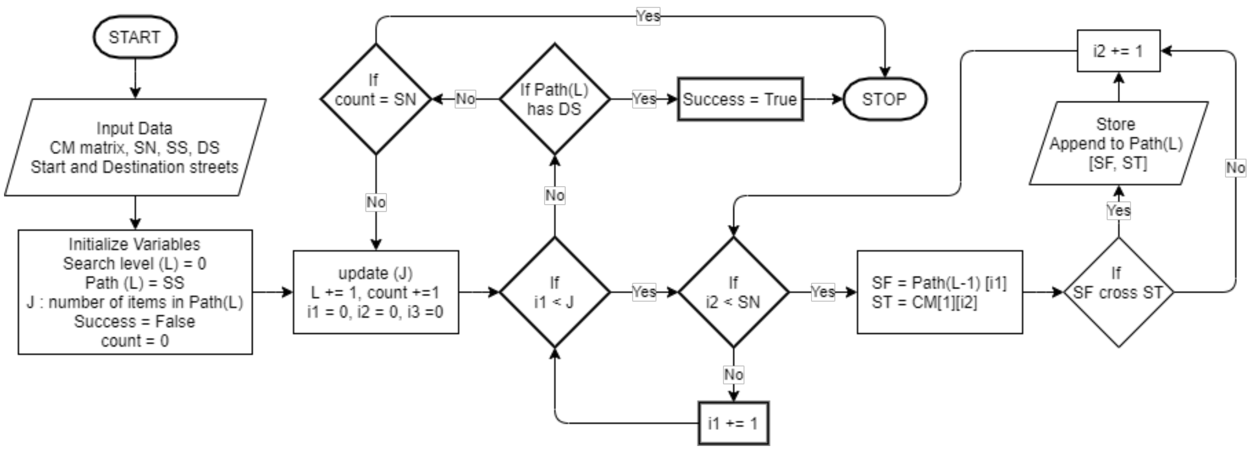

FIGURE 8. Flowchart of the pathfinder algotirhm

search algorithm. At the beginning, the algorithm gets streets cross matrix (CM) , total number of streets (SN), start street (SS) and destination street (DS) as input data. Algorithm search stores results in a path array that contains the following items; Search level, street from (SF) and street to (ST). (SF) is the index of the street at the previous search level and (ST) is the index of the street in the current search level that crosses (SF). In this way the algorithm can keep track of the whole path from (SS) to (DS).

RESULTS

The results of streets recognition algorithm are streets cross matrix (CM) and streets list (SL). Table I shows the resulting (CM) related to the map shot of Yekaterinburg city center and figure 7 shows the resulting (SL) related to the same map shot. For The map shot of Yekaterinburg city center, the algorithm has recognized a total of sixteen streets, seven horizontal streets and nine vertical streets. Table II shows the pathfinder algorithm resulting outputs for random different start/destination streets inputs.

DISCUSSION

In this research we proposed two algorithms. The first algorithm is to convert the free cells in the grid map into horizontal and vertical streets and the second one to find a path between any two streets depending on the street cross matrix created by the first algorithm. The limitation of the first algorithm is that sloping streets will be recognized as a combination of horizontal and vertical streets, that's why it's more efficient to simplify sloping streets into horizontal or vertical street. Regarding the pathfinder algorithm, although it does not pay attention to the length of the path, it pays attention to the search level and finds a path with the lowest number of streets to the destination street. The main advantage of the pathfinder algorithm is that it finds all the possible solutions that have the lowest search level. There are two points currently in the developing stage for the proposed algorithms: adding the sloping streets recognition to the streets recognition algorithm and adding the length of streets to the pathfinder algorithm to find the optimal shortest path. However, in this stage the algorithms have proved successful results that can be used in map recognition and planning in geographical and mobile robots applications.

Input Streets		output
Start	Destination	Path
8	5	[8, 1, 9, 3, 13, 5]
		[8, 2, 9, 3, 13, 5]
		[8, 2, 10, 3, 13, 5]
		[8, 2, 11, 3, 13, 5]
		[8, 2, 14, 3, 13, 5]
		[8, 2, 16, 3, 13, 5]
		[8, 2, 11, 4, 12, 5]
		[8, 2, 10, 4, 12, 5]
		[8, 1, 9, 4, 12, 5]
		[8, 2, 9, 4, 12, 5]
1	6	[1, 8, 2, 11, 6]
		[1, 9, 2, 11, 6]
		[1, 9, 3, 11, 6]
		[1, 9, 4, 11, 6]
4	2	[4, 9, 2]
		[4, 10, 2]
		[4, 11, 2]

TABLE II. Pathfinder algorithm results for random inputs

REFERENCES

1. M. Ganeshmurthy and G. Suresh, "Path planning algorithm for autonomous mobile robot in dynamic environment," in *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)* (IEEE, 2015) pp. 1–6.
2. C. Ó'Dúnlaing and C. K. Yap, "A 'retraction' method for planning the motion of a disc," Journal of Algorithms **6**, 104–111 (1985).
3. S. K. A. Nair, S. Joladarashi, and N. Ganesh, "Evaluation of ultrasonic sensor in robot mapping," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)* (IEEE, 2019) pp. 638–641.
4. M. Weigl, B. Siemiatkowska, K. A. Sikorski, and A. Borkowski, "Grid-based mapping for autonomous mobile robot," Robotics and Autonomous Systems **11**, 13–21 (1993).
5. M. Sniedovich, "Dijkstra's algorithm revisited: the dynamic programming connexion," Control and cybernetics **35**, 599–620 (2006).
6. R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of a," Journal of the ACM (JACM) **32**, 505–536 (1985).
7. S. Even, *Graph algorithms* (Cambridge University Press, 2011).
8. M. Filimonov and I. Ahmed, "Collision avoidance algorithm with performance optimization and speed control for multi-robot autonomous system," (CEUR-WS, 2017) pp. 115–122.
9. C. Reas and B. Fry, "Processing: programming for the media arts," AI & SOCIETY **20**, 526–538 (2006).
10. G. Van Rossum *et al.*, "Python programming language." in *USENIX annual technical conference*, Vol. 41 (2007) p. 36.
11. T. E. Oliphant, *A guide to NumPy*, Vol. 1 (Trelgol Publishing USA, 2006).
12. S. Tosi, *Matplotlib for Python developers* (Packt Publishing Ltd, 2009).