Mapping for Path Planning using Maximal Empty Rectangles

by

Jinyoung Park

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 3, 2013

Keywords: Mapping, Path Planning, Grid Map,
R-map, Maximal Empty Rectangle

Approved by

Andrew J. Sinclair, Chair, Associate Professor of Aerospace Engineering
John E. Cochran, Jr., Professor and Head of Aerospace Engineering
David A. Cicci, Professor of Aerospace Engineering
Gilbert L. Crouse, Associate Professor of Aerospace Engineering
Andrew B. Shelton, Assistant Professor of Aerospace Engineering

Abstract

This study applies a new map representation, named R-map, to the path-planning problem. The R-map is calculated as a reduced-element representation of a grid map. Grid maps express obstacles and free space with a binary representation for each cell. The concept of the R-map is to integrate free cells into the maximal empty rectangles by surveying the numbers in a grid map. By calculating the R-map, the number of free cells in the grid map is dramatically reduced and this accomplishes data reduction. Since the R-map is a new map representation, it has potential applications in many other fields.

This thesis consists of two major parts, R-map and path planning. The R-map part explains a specific R-map algorithm with a simple example, and demonstrates its advantages comparing examples for indoor and outdoor environments. Also, data reduction by applying R-map is demonstrated. In the path planning part, the path planning using Dijkstra's algorithm is described and paths according to four different weights are illustrated. Moreover, path planning with grid maps and R-maps is compared. R-maps are naturally suited for path planning due to their reduced number of elements and focus on the largest obstacle-free areas.

Acknowledgments

The author would like to sincerely appreciate Dr. Andrew J. Sinclair for his advice, patience and kindness throughout his study at Auburn University. Also the author would like to thank all faculties of the Aerospace Engineering Department at Auburn University. Finally the author would like to express his thanks to his girl friend, Jiyeong Kim, in South Korea for her love and encouragement.

This work is dedicated to the author's grateful parents, Hochul Park and Mija Choi, appreciating their support and devotion.

Table of Contents

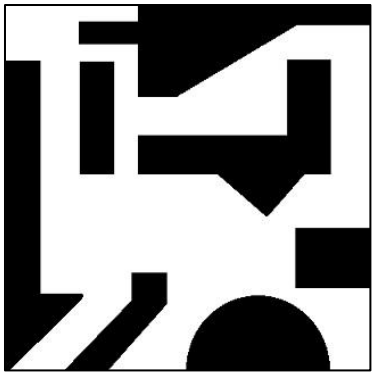
List of Figures

List of Tables

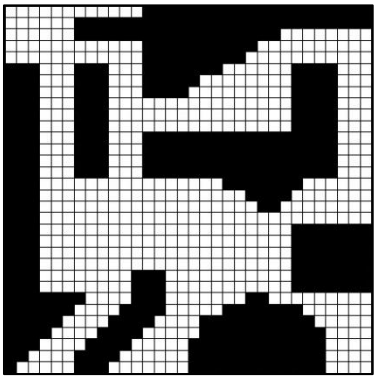
Chapter 1

Introduction

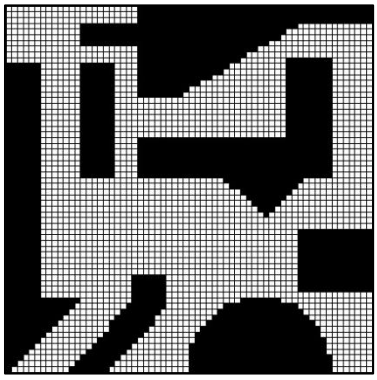
Autonomous mobile robots have been studied for decades. To make the robot move, many things are required. The most essential constituent is a map of its environment. By this map, an optimal path can be calculated, and the robot can move to the destination performing obstacle avoidance. The most popular map representation is a grid map which expresses obstacles and free space with binary numbers, 0 representing an obstacle and 1 representing free space. Also since these numbers are in a matrix form, it is easy to locate obstacles and robots. A grid map, however, has a problem of resolutions. Different-resolution grid maps can be generated from an image of the environment. Figure 1.1 shows two grid maps, (b) and (c), in different resolutions which are generated from the original image (a). They are a 32×32 grid map and a twice higher resolution map, 64×64 grid map. The lower resolution map, the 32×32 grid map, consumes less data but, the obstacles are distorted. Hence, path planning with a lower resolution map may give an unreliable path. The higher resolution map, the 64×64 grid map, represents obstacles close to the image of environment but, this twice bigger map has 2^2 times more cells. Hence, path planning with n times higher resolution map gives more reliable path but, it has n^2 times more cells to consider.



(a) Image of Environment



(b) 32×32 Grid Map



(c) 64×64 Grid Map

Figure 1.1: Two Different-Resolution Grid Maps from the Image

As a solution of this problem, multiresolution cell decomposition has been suggested [1]. The algorithm decomposes cells into smaller cells, in order to minimize the distortions of obstacles. Thus the map has varied sizes of cells. Prazenica and Kurdila adopted multiresolution decomposition for obstacle-location estimation using receding horizon control formulation [2-3]. Another solution for the drawback of grid maps has been suggested, named Rectangular map or R-map. The main idea is integration of empty cells into a maximal empty rectangle. R-maps were invented by Ahn and Jeon in 2010 [4]. They introduce R-map as a hybrid map of the grid and the topological maps. The topological map is a graph-based map which only shows

relationships between nodes by branches such as a subway map. In R-map, each integrated cells as maximal rectangles are the nodes and relationships, connections of the rectangles, are calculated.

This thesis details the computation of R-maps, and implements path planning with R-maps using Dijkstra's algorithm. Furthermore, it includes illustrations of applications of R-map to other fields. Chapter 2 describes the R-map algorithm and result. The algorithm consists of four steps: initialization, finding a maximal empty rectangle, grid map updating and computing connection. Each step is explained with a simple example and Figures as separated subsections. Also, the result of the R-map algorithm for indoor and outdoor maps is illustrated.

Chapter 3 describes the path-planning algorithm and results. Dijkstra's algorithm is applied for path planning, and since paths can be different depending on weights in Dijkstra's algorithm, four different approaches are demonstrated to find an optimal path. The weights are defined based on rectangle area, border length, shortest path, and number of neighbors. Path planning with the indoor and outdoor R-maps from Chapter 2 are illustrated.

Further examples using R-map are suggested in Chapter 4. Because R-map is an image reconstruction, it can be applied to other fields such as image processing. The examples include reduction of data, driving assistance and image recognition.

The purpose of this study is to advance the previous R-map [4] and apply it to path planning. The R-map and path planning are specifically explained with examples and analyzed throughout following Chapters.

Chapter 2

R-map

The main idea of the R-map is the integration of empty cells of the grid map into a maximal empty rectangle. The key issue of R-map algorithm is finding maximal empty rectangles in the form of non-overlapping rectangles. The maximal empty rectangle problem has been studied by mathematicians and computer scientists to mine empty areas [5], and reduce data (image compression) [6]. There are two basic ways to find the rectangles within an image. One is finding the rectangles with geometrical calculations on the image. This way is focused on saving the largest-area rectangular piece in the field such as a piece of fabric or sheet metal with holes [7]. The other way is calculations of values in an image represented with binary numbers. This way is focused on extracting a required or empty data set from the field [8]. The R-map algorithm is the second way as long as the input of the algorithm is a grid map. The flow of the algorithm in this study is referred from Ahn and Jeon [4].

2.1 R-map Algorithm

The input of the R-map algorithm is a binary image, *G*, as shown in Figure 2.1. In step 1, *cx* and *Gx* matrices are defined. *cx* is transformed from *G*, and *Gx* is the same matrix with *G*. *cx* is used in step 2 and *Gx* is used in step 4. In step 2, every element of free space in *cx* is surveyed to compare the areas of possible rectangles and find the largest one. The property of the largest rectangle in each survey is saved as r which includes its coordinate of the left-upper corner, the

width and height. As the survey proceeds, the property of the larger rectangle is updated to the largest one. The survived largest rectangle is the maximal empty rectangle and the property of the rectangle is saved as r_k. In step 3, the corresponding elements of r_k in *G* are updated to zeros to find non-overlapping empty rectangles in the subsequent iterations. The process goes back to step 1 with updated *G*, and steps 1, 2 and 3 are repeated until all of the elements in *G* have been set to zero. In step 4, the maximal empty rectangles are labeled as k in the rank of r_k to find neighbors of kth rectangles and they are saved as $connection_k$. The output of the algorithm is a structure *M* which consists of r_k.

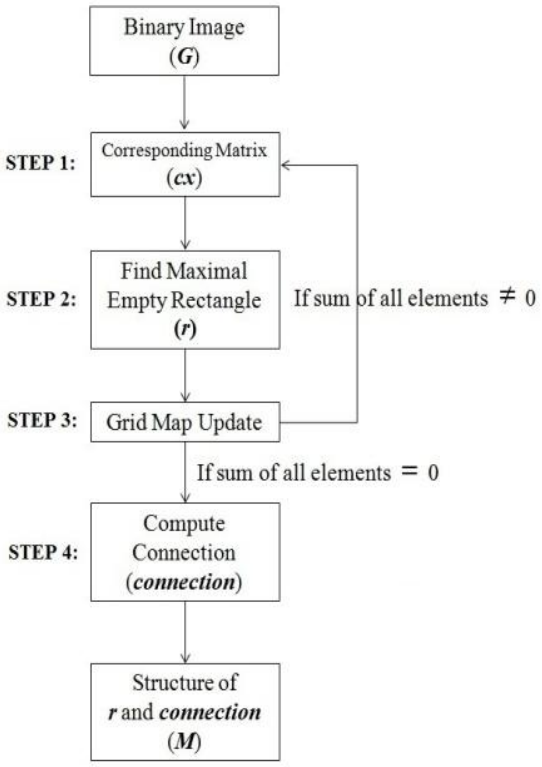


Figure 2.1: Diagram of the R-map Algorithm

2.1.1 Step1: Initialization

A binary image that shows obstacles and free space can be represented as a grid map, and is the starting point for creating the R-map. The grid map is a $n \times m$ matrix *G* whose elements consist of 0 and 1, with 0 representing an obstacle and 1 representing free space. To aid in defining the R-map, *G* is preprocessed to define a $(n+1) \times m$ matrix *cx* which is a right-to-left row-wise summation of contiguous free-space elements in *G*. The elements of $(n+1)$th row of *cx* are set as zeros. Also, *Gx* matrix which is the same as *G* is defined for use in step 4. Figure 2.2 illustrates an example of *G* and *cx*.

	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	1	1	1	1	1	1
2	1	1	0	0	1	1	1	0	0	0
3	1	1	1	0	1	1	1	0	0	0
4	1	1	1	1	1	1	1	1	1	1
5	1	1	0	0	0	0	0	1	1	1
6	1	1	0	0	0	0	0	1	1	1
7	1	1	0	0	0	0	0	1	0	0
8	1	1	1	1	1	1	1	1	0	0
9	0	0	0	0	1	1	1	1	0	0
10	1	1	1	1	1	1	1	1	0	0

(a) *G* Matrix

	1	2	3	4	5	6	7	8	9	10
1	2	1	0	0	6	5	4	3	2	1
2	2	1	0	0	3	2	1	0	0	0
3	3	2	1	0	1	2	1	0	0	0
4	10	9	8	7	6	5	4	3	2	1
5	2	1	0	0	0	0	0	3	2	1
6	2	1	0	0	0	0	0	3	2	1
7	2	1	0	0	0	0	0	1	0	0
8	8	7	6	5	4	3	2	1	0	0
9	0	0	0	0	4	3	2	1	0	0
10	8	7	6	5	4	3	2	1	0	0
11	0	0	0	0	0	0	0	0	0	0

(b) *cx* Matrix

Figure 2.2: Example of *G* and *cx*

2.1.2 Step 2: Find a Maximal Empty Rectangle

The critical step in computing the R-map is finding the maximal empty rectangle in the grid map. The idea of the algorithm used here is referred from Vandevoorde [9]. Step 2 is done by surveying *cx* column-by-column, starting with the first element of the first column. The survey of each column considers the largest rectangles that can be formed with the left edge of

the rectangles in that column. As the survey progresses, the candidate rectangles under current consideration are stored in a stack called *stack*. Each entry in *stack* stores the width of that rectangle and the top row number of that rectangle.

As the survey for each element proceeds down the jth column, the value of the next element $cx(i+1, j)$ is compared to the value of the current element $cx(i, j)$. There are four possibilities of the comparison. In the first possibility, if these values are equal, then no changes are made to *stack*. In the second possibliity, if the $(i+1)$th value is greater than the ith value, then a new rectangle is added to *stack*. The new rectangle has a top row equal to $i+1$ and a width equal to $cx(i+1, j)$. In the third possibility, if the $(i+1)$th value of *cx* is less than the *i*th value, the areas of the rectangles in *stack* are calculated and compared to the area of the largest previously found rectangle. The larger area is updated to the largest area and the property of the rectangle is saved as r. As the areas are calculated, the widths of the rectangles with width greater than $cx(i+1, j)$ in *stack* are updated to $cx(i+1, j)$. Of all the resulting rectangles with width $cx(i+1, j)$ only the one with the lowest row number is retained, and the other rectangles with this width (if any) are removed from *stack*. In the fourth possibility, if the $(i+1)$th value is 0, all rectangles are removed from *stack*. In addition, because the $(i+1)$th value of *cx* is zero, all rectangles are removed at the end of the column. The survey then proceeds to the next column. For the example shown in Figure 2.3, *cx*(1, 1) is 2. This means that there is a rectangle whose top row is 1 and width is 2, then *stack* is [1, 2] and this is the rectangle 1. The second element *cx*(2, 1) is also 2, and no changes are made in *stack*, because the rectangle 1 can be extended to the second row. The third element *cx*(3, 1) is 3 which is greater than the previous value. Therefore, a new rectangle, the rectangle 2, whose top row is 3 and width is 3 is added to *stack*.

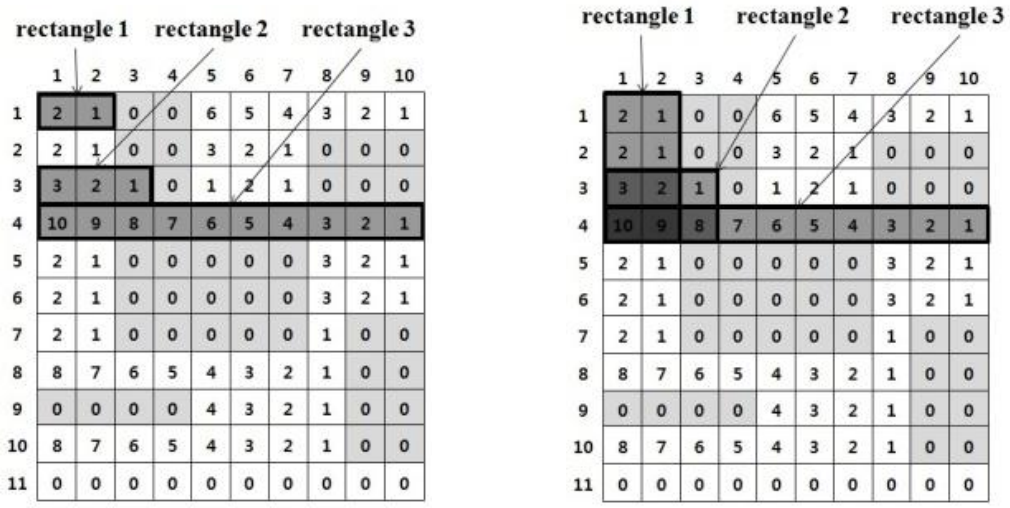


Figure 2.3: Top Rows and Areas of Rectangles 1, 2 and 3

$$\text{At third element:} \quad stack = \begin{bmatrix} 1 & 2 \\ 3 & 3 \end{bmatrix} \tag{2.1}$$

The fourth element *cx*(4, 1) is 10, and it is greater than the third element. Thus another new rectangle with top row 4 and width 10 is added to *stack*, and this is the rectangle 3.

$$\text{At fourth element:} \quad stack = \begin{bmatrix} 1 & 2 \\ 3 & 3 \\ 4 & 10 \end{bmatrix} \tag{2.2}$$

The fifth value *cx*(5, 1) is 2, and that is less than the previous value. Thus the areas of the rectangles in *stack* are calculated and compared. The heights of the rectangles are subtraction of their top row and the $(i+1)$th row, i.e. the height of the *j*th rectangle in *stack* is $(i+1) - stack(j, 1)$.

$$\text{Rectangle 1 : height} = (\text{i} + 1) - stack(1,1) = 5 - 1 = 4$$

$$\text{area} = stack(1,2) \times \text{height} = 2 \times 4 = 8 \quad (2.3)$$

$$\text{Rectangle 2 : height} = (\text{i} + 1) - stack(2,1) = 5 - 3 = 2$$

$$\text{area} = stack(2,2) \times \text{height} = 3 \times 2 = 6 \quad (2.4)$$

$$\text{Rectangle 3 : height} = (\text{i} + 1) - stack(3,1) = 5 - 4 = 1$$

$$\text{area} = stack(3,2) \times \text{height} = 10 \times 1 = 10 \quad (2.5)$$

Rectangle 3 is the largest rectangle so far. The property of the rectangle r which indicates column number, top row, width and height, is $[1, 4, 10, 1]$ Then, the widths of the candidate rectangles with widths greater than the fifth element, 2, are updated to 2 and other rectangles except the lowest row number are removed from *stack* thus the rectangle 1 is retained but, the rectangles 2 and 3 are removed from *stack*.

$$\text{At fifth element:} \quad stack = [1 \quad 2] \quad (2.6)$$

The rectangle in *stack* is the rectangle 1 but, redefine it as the rectangle $1'$ to distinguish from the rectangle 1. The fifth value *cx*(5, 1) and the next values up to seventh *cx*(7, 1) are equal, so *stack* has no changes. The eighth element *cx*(8, 1) is 8, and this is greater than the seventh element. Thus a new rectangle is added to *stack*. The new rectangle is the rectangle 4 in Figure 2.4, which has the top row 8 and the width 8. Now, *stack* has two candidate rectangles, the rectangle $1'$ and 4.

$$\text{At eighth element:} \quad stack = \begin{bmatrix} 1 & 2 \\ 8 & 8 \end{bmatrix} \quad (2.7)$$

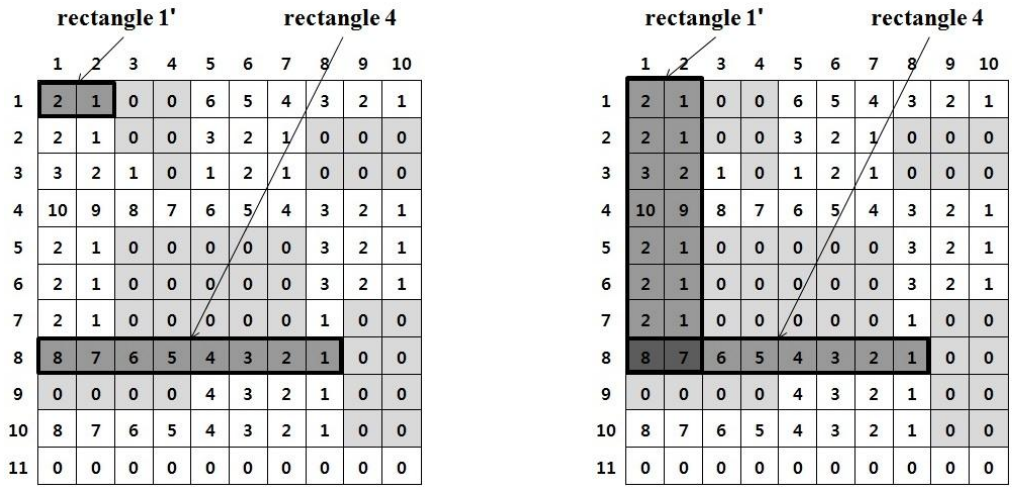


Figure 2.4: Top Rows and Areas of Rectangles $1'$ and 4

The ninth element *cx*(9, 1) is 0, then the area is calculated and compared to the largest rectangle.

$$\text{Rectangle } 1'\text{: height} = (\text{i} + 1) - stack(1,1) = 9 - 1 = 8$$

$$\text{area} = stack(1,2) \times \text{height} = 2 \times 8 = 16 \tag{2.8}$$

$$\text{Rectangle 3: height} = (\text{i} + 1) - stack(2,1) = 9 - 8 = 1$$

$$\text{area} = stack(2,2) \times \text{height} = 8 \times 1 = 8 \tag{2.9}$$

Since the rectangle $1'$ is larger than the previous largest rectangle, the rectangle $1'$ is updated to the largest rectangle. The property of the rectangle $1'$ is $[1, 1, 2, 8]$. In addition, because the ninth element *cx*(9, 1) is 0, all rectangles in *stack* are removed, and the updated *stack* is empty.

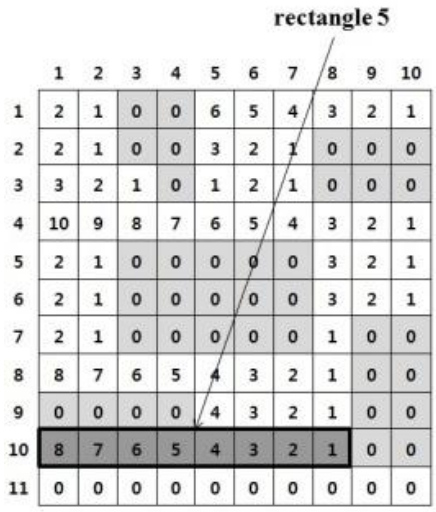


Figure 2.5: Top Row of Rectangle 5

As shown in Figure 2.5, the tenth element *cx*(10, 1) is 8. Thus a new rectangle, the rectangle 5, which has top row 10 and width 8 is added to *stack*.

$$\text{At tenth element:} \quad stack = [10 \quad 8] \tag{2.10}$$

Because the last element *cx*(11, 1) is 0, the area of the rectangle in *stack* is calculated, compared and removed.

$$\text{Rectangle 5}: \quad \text{height} = (\text{i} + 1) - stack(1,1) = 11 - 10 = 1$$
$$\text{area} = stack(1,2) \times \text{height} = 8 \times 1 = 8 \tag{2.11}$$

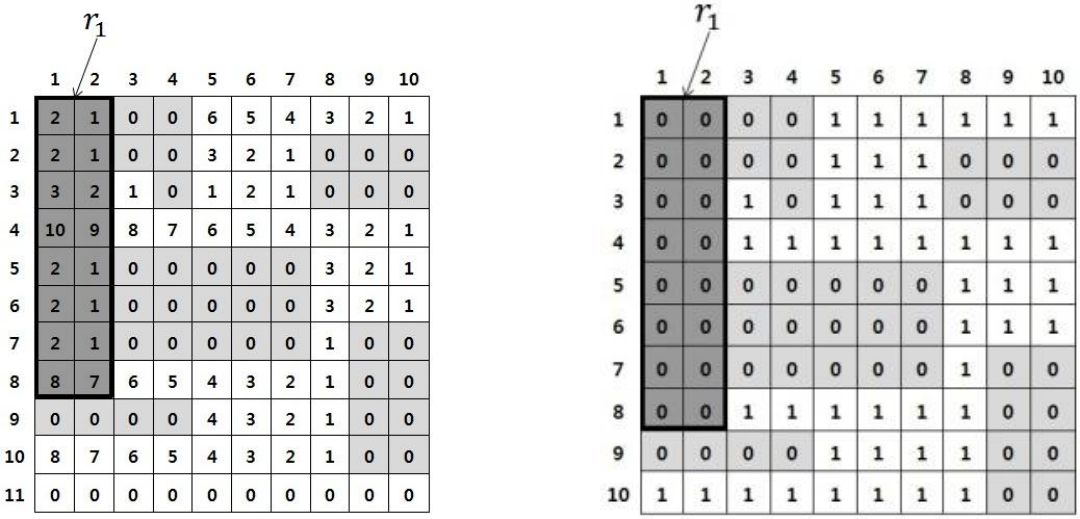
The largest area is still larger than the area of rectangle 5 therefore, the rectangle $1'$ is kept as the largest rectangle.

The survey proceeds to the second column, and is continued to the last column. As the

survey proceeds, the largest rectangle is replaced by a larger area. After the survey of the last column, the largest rectangle is the maximal empty rectangle and the property of the rectangle is saved as r_k in the structure *M*. After the survey of this example, the rectangle $1'$ is in fact the maximal empty rectangle and saved as r_1 in *M*.

2.1.3 Step 3: Grid Map Update

For the elements of the maximal empty rectangle calculated in the previous step, the corresponding elements of *G* are updated to zeros. For example, the maximal empty rectangle in *cx* found in step 2 is r_1, then the corresponding elements of *G* are updated to zeros as in Figure 2.6. This updating ensures that the subsequent iteration only searches the remaining free space.



(a) Original *cx* Matrix (b) Updated *G* Matrix

Figure 2.6: (a) Maximal Empty Rectangle 1 in the Original *cx* Matrix (b) Corresponding Elements in the Updated *G* Matrix are Updated to Zeros

2.1.4 Iteration of Steps 1, 2 and 3

Steps 1, 2, and 3 are repeated with the updated *G* in an interative fashion until all of the elements in *G* have been set to zero or until a desired threshold is reached in terms of the number or size of the maximal rectangles, related to the desired R-map resolution. In this example, after all iterations with *G*, properties of 10 maximal empty rectangles are saved as r_1 up to r_{10} in *M*. Figure 2.7 shows *G* with elements are all updated to zeros and the 10 rectangles after the iteration.

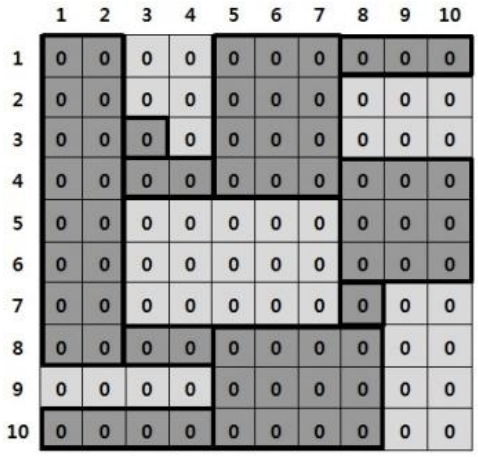


Figure 2.7: 10 Rectangles after Iteration

2.1.5 Step 4: Compute Connection

The purpose of this step is finding neighbors of the maximal empty rectangles using *Gx* from step 1. To find what neighbors are connected to the rectangles, the rectangles need to be labeled. To label the rectangles, the elements of r_k are updated to k. Also, additional columns and rows are added on both sides, top and bottom of *Gx*, with elements equal to zeros. Thus, *Gx* is updated to $(n+2) \times (m+2)$ demension. As the size of *Gx* has changed, the locations of the rectangles are changed and they can be derived from the properties of the rectangles, r_k which

contains column number, top row, width and height. Because of the additional column and row on the left and top of *Gx*, every element of *Gx* is moved one cell right and one cell down. Therefore, the left-upper corner of a rectagle should have an additional 1. The right-lower corner of a rectangle is ((the column number) + (the width), (the top row) + (the height)). Knowledge of the corner positions define the entire boundary of the r_k rectangle, and the labeling of neighboring elements in *Gx* therefore allows determination of all connected rectangles.

Continuing the example, the elements of r_1 are updated to 1, and the elements of r_2 are updated to 2. In the same manner, the elements of r_k are updated to *k*, where *k* is up to 10. Figure 2.8 shows the labeled rectangles in updated *Gx* as a 12×12 matrix. The property of the maximal empty rectangle 1 in *G* is r_1, and the location of the rectangle 1 in *Gx* can be derived from r_1. The location of the maximal empty rectangle 1 in updated *Gx* is represented as L_1.

$$r_1 = [1, 1, 2, 8] \tag{2.12}$$

$$\text{Left-Upper Corner of } r_1 = (r_1(1) + 1, r_1(2) + 1) = (2, 2) \tag{2.13}$$

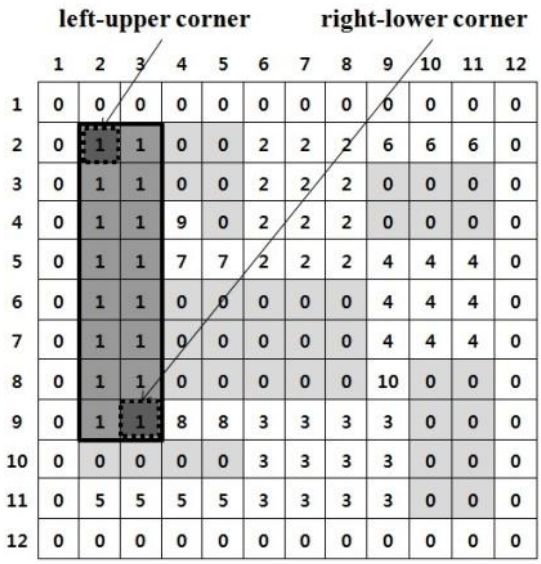
$$\text{Right-Lower Corner of } r_1 = \big(r_1(1) + r_1(3), r_1(2) + r_1(4)\big) = (3, 9) \tag{2.14}$$

$$L_1 = [2, 2, 3, 9] \tag{2.15}$$

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	2	2	2	6	6	6	0
3	0	1	1	0	0	2	2	2	0	0	0	0
4	0	1	1	9	0	2	2	2	0	0	0	0
5	0	1	1	7	7	2	2	2	4	4	4	0
6	0	1	1	0	0	0	0	0	4	4	4	0
7	0	1	1	0	0	0	0	0	4	4	4	0
8	0	1	1	0	0	0	0	0	10	0	0	0
9	0	1	1	8	8	3	3	3	3	0	0	0
10	0	0	0	0	0	3	3	3	3	0	0	0
11	0	5	5	5	5	3	3	3	3	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2.8: Labeled Rectangles in *Gx*

Figure 2.9 depicts the left-upper and right-lower corners of the maximal empty rectangle 1 which are (2, 2) and (3, 9) respectively. Because the rectangle 1 is known and other rectangles are labeled, the neighbors of rectangle 1 can be found. The left connections of the rectangle 1 are the values of the left column of the left-upper corner and the right connections are the values of the right column of the right-lower corner as many cells as the height of the rectangle 1. The upper connections are the values of the upper row of the left-upper corner and the lower connections are the values of the lower row of the right-lower corner as many cells as the width of rectangle 1.



	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	2	2	2	6	6	6	0
3	0	1	1	0	0	2	2	2	0	0	0	0
4	0	1	1	9	0	2	2	2	0	0	0	0
5	0	1	1	7	7	2	2	2	4	4	4	0
6	0	1	1	0	0	0	0	0	4	4	4	0
7	0	1	1	0	0	0	0	0	4	4	4	0
8	0	1	1	0	0	0	0	0	10	0	0	0
9	0	1	1	8	8	3	3	3	3	0	0	0
10	0	0	0	0	0	3	3	3	3	0	0	0
11	0	5	5	5	5	3	3	3	3	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2.9: Location of the Corners in *Gx*

$$L_1 = [2, 2, 3, 9]$$

$$left\_connection_1 = [\text{from } L_1(2) \text{ to } L_1(4), L_1(1) - 1] = [0, 0, 0, 0, 0, 0, 0, 0] \quad (2.16)$$

$$right\_connection_1 = [\text{from } L_1(2) \text{ to } L_1(4), L_1(3) + 1] = [0, 0, 9, 7, 0, 0, 0, 8] \quad (2.17)$$

$$upper\_connection_1 = [L_1(2) - 1, \text{from } L_1(1) \text{ to } L_1(3)] = [0, 0] \quad (2.18)$$

$$lower\_connection_1 = [L_1(4) + 1, \text{from } L_1(1) \text{ to } L_1(3)] = [0, 0] \quad (2.19)$$

The unique numbers of the connections except zero indicate the connected rectangles of the maximal empty rectangle 1.

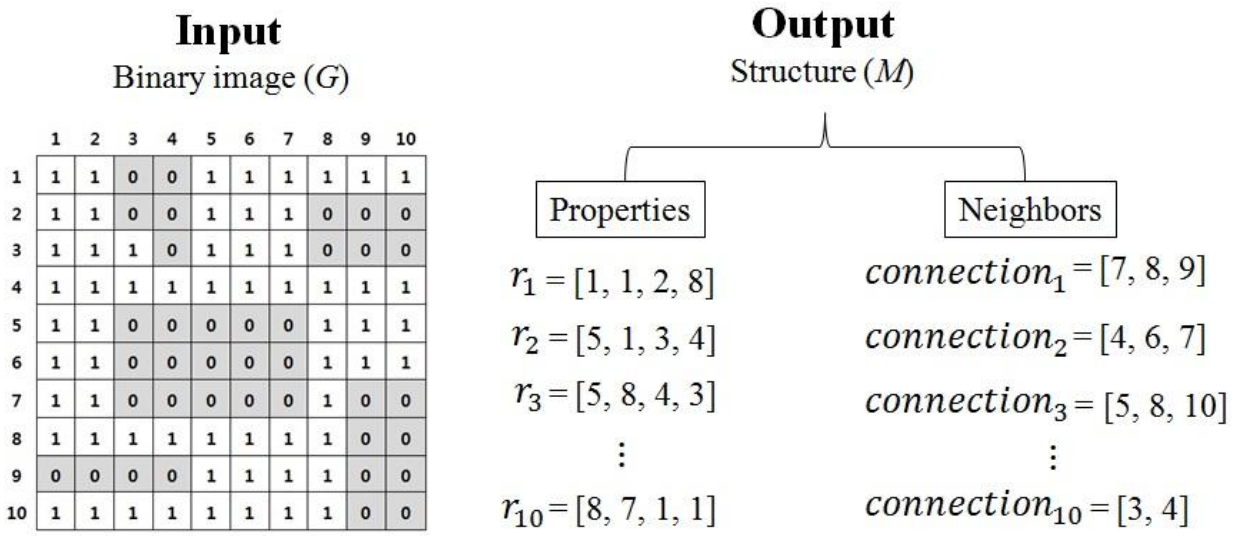
$$connection_1 = [7, 8, 9] \quad (2.20)$$

The connections up to $connection_{10}$ are saved in a structure *M*. In this manner,

$connection_k$ is saved in *M*. The output of the R-map algorithm is *M* which consists of r_k and $connection_k$.

2.2 Result of R-map Algorithm

The input to the R-map algorithm is a binary image *G*, and the output is a structure *M*. *G* is an $n \times m$ matrix, and *M* is a structure which consist of properties and neighbors of maximal empty rectangles. The properties are from *cx* in step 1 and the neighbors are from *Gx* from step4. Since properties and neighbors are information of each rectangle, *M* has as many elements as the number of maximal empty rectangles in *G*. Figure 2.10 shows the input and output of the example. Because the input *G* has 10 maximal empty rectangles, there are 10 elements in the output *M*.



	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	1	1	1	1	1	1
2	1	1	0	0	1	1	1	0	0	0
3	1	1	1	0	1	1	1	0	0	0
4	1	1	1	1	1	1	1	1	1	1
5	1	1	0	0	0	0	0	1	1	1
6	1	1	0	0	0	0	0	1	1	1
7	1	1	0	0	0	0	0	1	0	0
8	1	1	1	1	1	1	1	1	0	0
9	0	0	0	0	1	1	1	1	0	0
10	1	1	1	1	1	1	1	1	0	0

Figure 2.10: Input and Output of R-map Algorithm

2.3 R-map Examples

To demonstrate how the R-map represents a higher resolution map with fewer cells, different resolution R-maps are compared. Obstacles in a map image can have various shapes of surfaces with vertical, horizotal, diagonal and curved lines, but the grid map represents these obstacles with a gridded approximation. The grid-map resolution can be increased to improve this approximation. Therefore, the map image can produce different resolution grid maps. As different resolution grid maps have different number of cells, their numbers of free cells which need to be surveyed to find maximal empty rectangles are different. Thus, depending on the resolution of the grid map, the computation time to generate the R-map varies.

An example indoor environment is shown in Figure 2.11 to Figure 2.15, and an example outdoor environment is shown in Figure 2.17 to Figure 2.21. For the indoor case, the indoor map image of Figure 2.11 is reproduced in four different resolutions, 16×16, 32×32, 64×64 and 128×128. Table 2.1 compares the number of free cells in the grid maps and the number of rectangles in the R-maps of the indoor map. For the outdoor map, the Auburn University campus map of Figure 2.17 is also reproduced in four different resolution: 20×20, 40×40, 80×80 and 160×160. Table 2.2 compares the number of free cells in the grid maps and the number of rectangles in the R-maps of the outdoor map. In both tables, as the level of the resolution raises by factor of two, the number of free cells in the grid map increases by approximately four. However, the number of R-map rectangles increases by only a factor of approximately two. Therefore, R-map accomplishes a higher resolution map with fewer elements. Additionally, Tables 2.1 and 2.2 show the computational expense of calculating the R-map from each grid map using MATLAB implementation on a laptop computer. Whereas higher resolution maps require more computation time, a trend for the scaling of the computation time is not immediately

obvious. It is also noteworthy that these computation times are for the complete R-map. Some reduction in computation time could be achieved by setting thresholds for the number or size of the rectangles in the R-map.

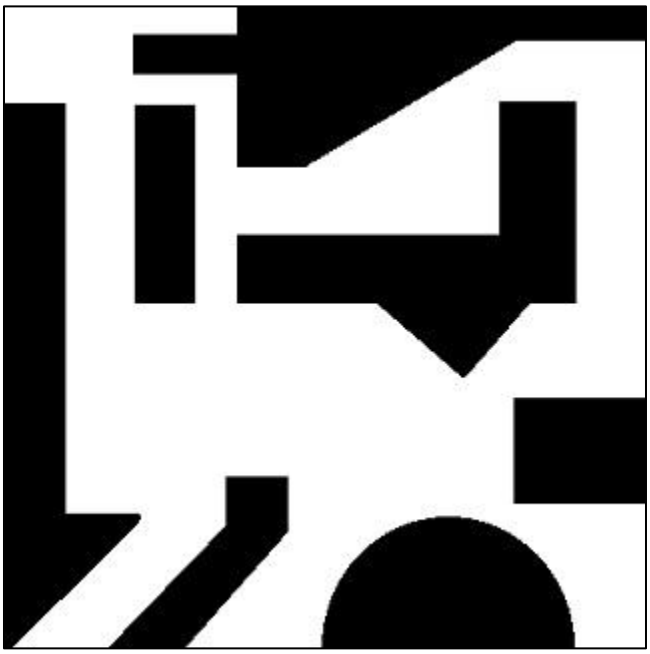


Figure 2.11: Indoor Map Image

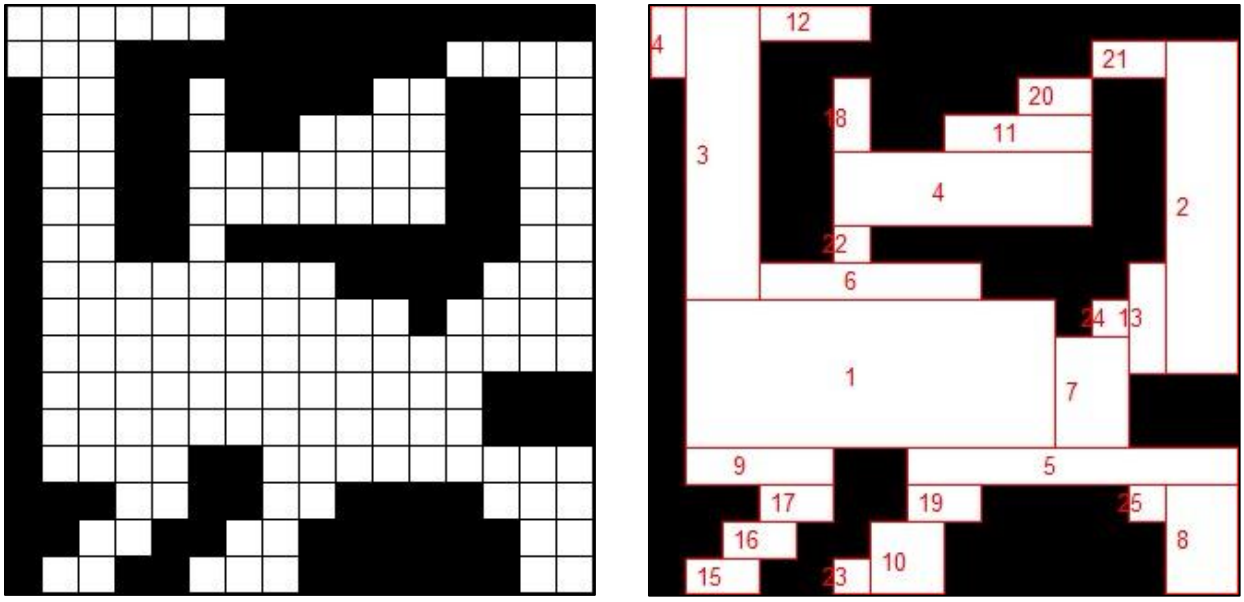


Figure 2.12: 16×16 Grid Map and Its R-map

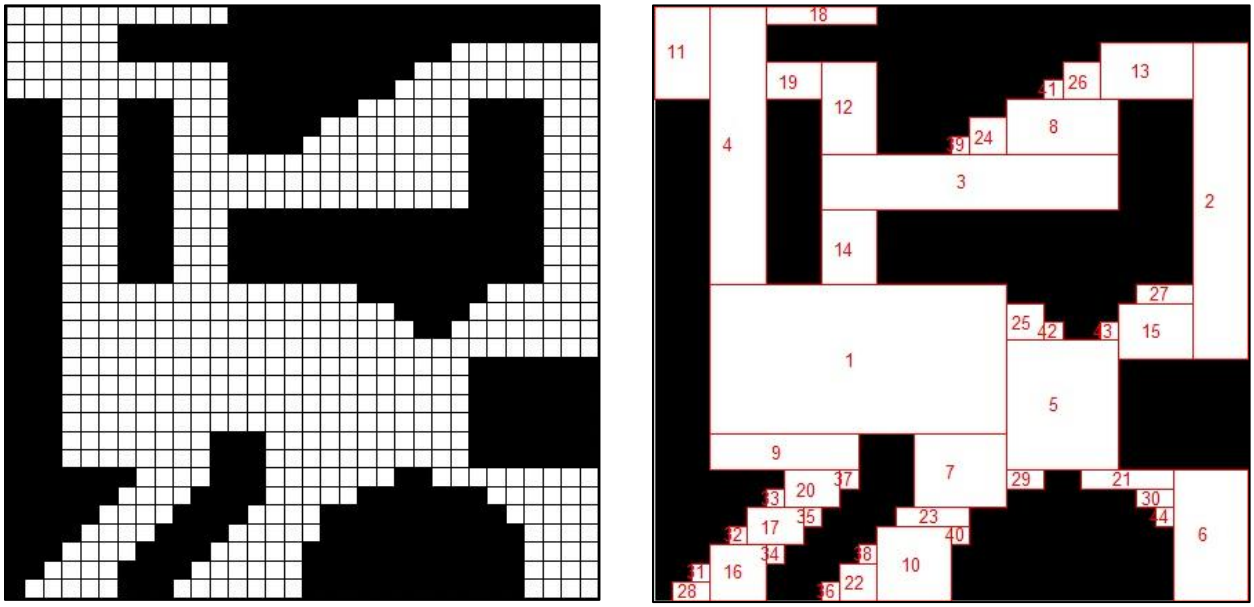


Figure 2.13: 32×32 Grid Map and Its R-map

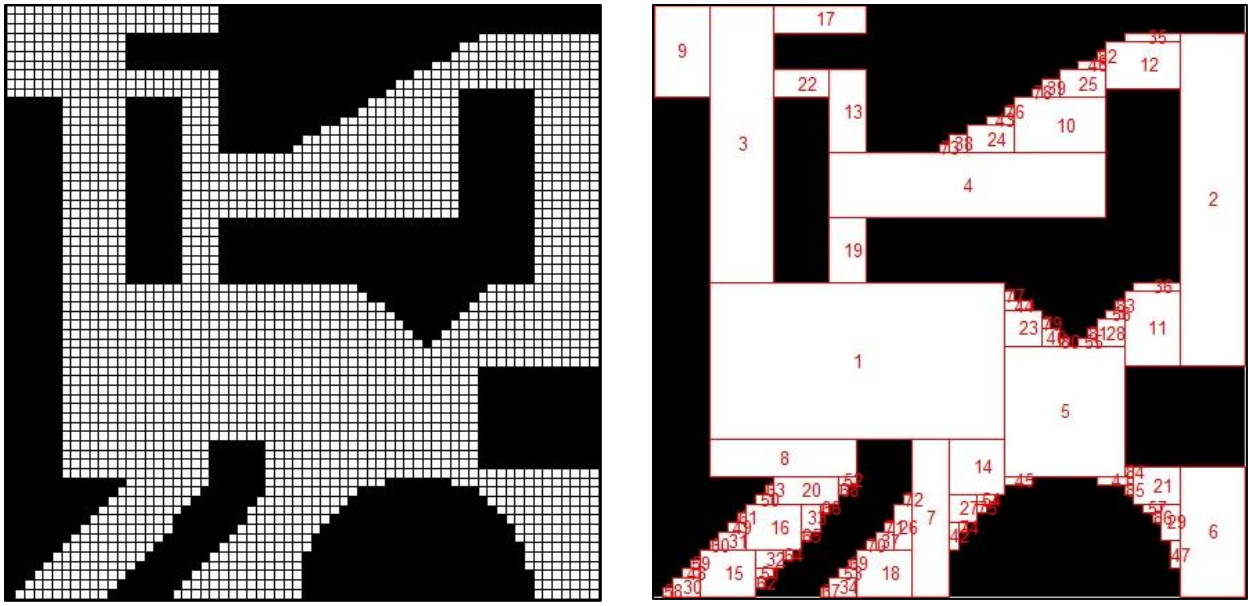


Figure 2.14: 64×64 Grid Map and Its R-map

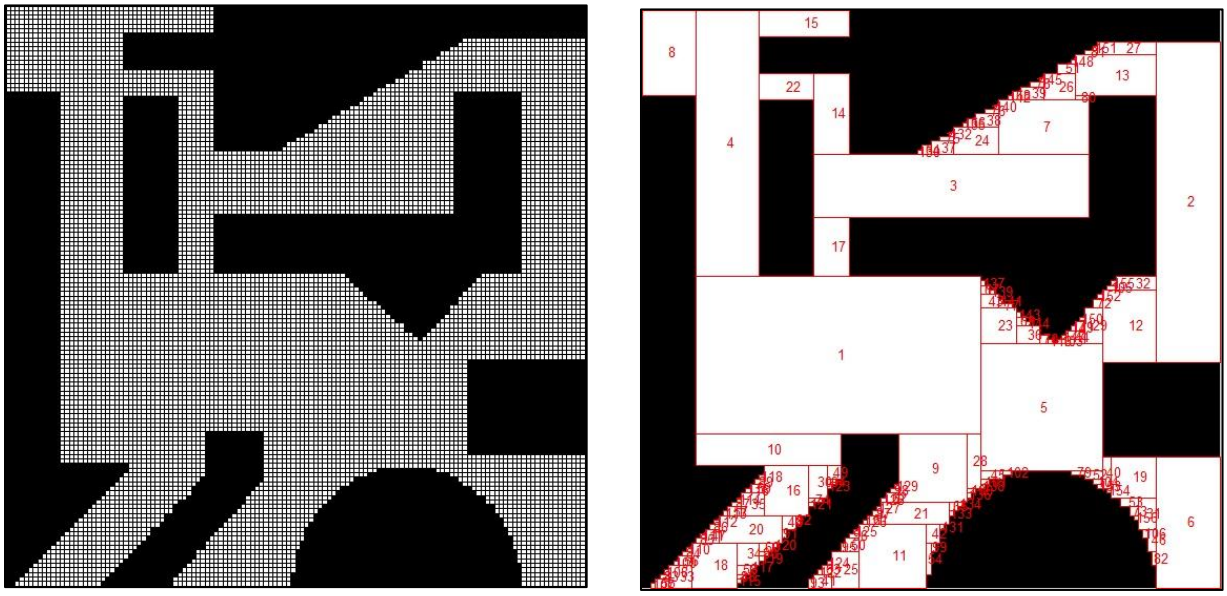


Figure 2.15: 128×128 Grid Map and Its R-map

Table 2.1: Indoor R-mapping by Different Resolutions

Resolutions	16×16	32×32	64×64	128×128
Computation Time of R-mapping (sec)	0.041	0.061	0.146	0.549
Free cells in Grid Map	153	562	2,325	9,331
Rectangles in R-map	25	44	86	156

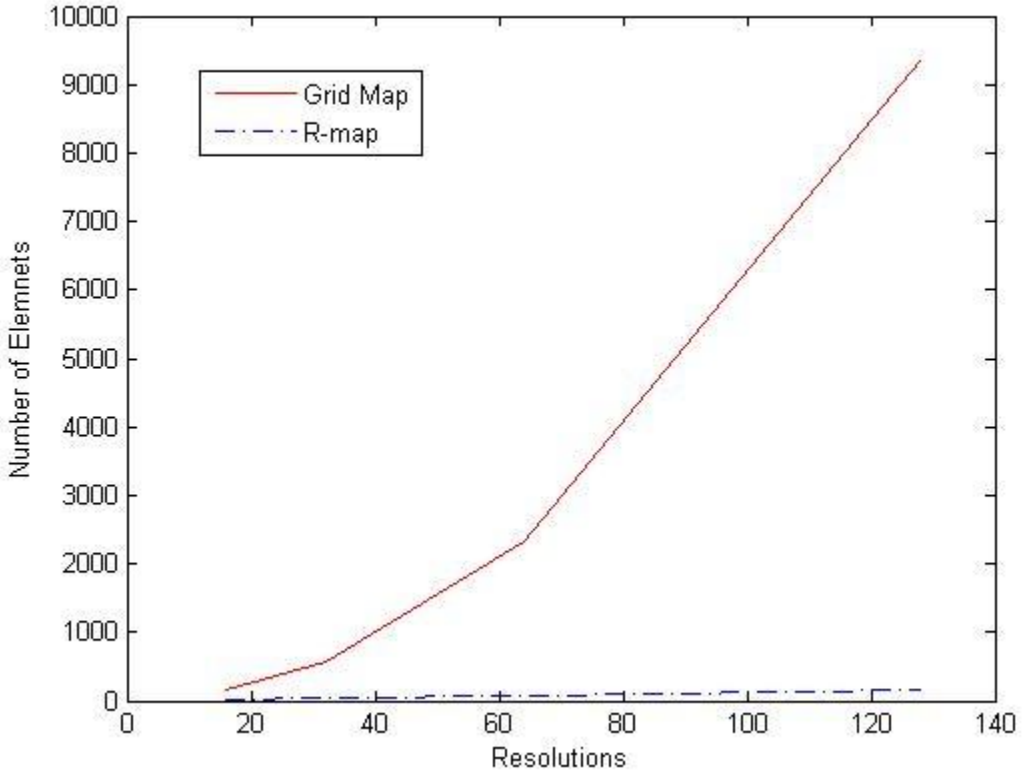


Figure 2.16: Comparison of Indoor Maps by Number of Elemnets

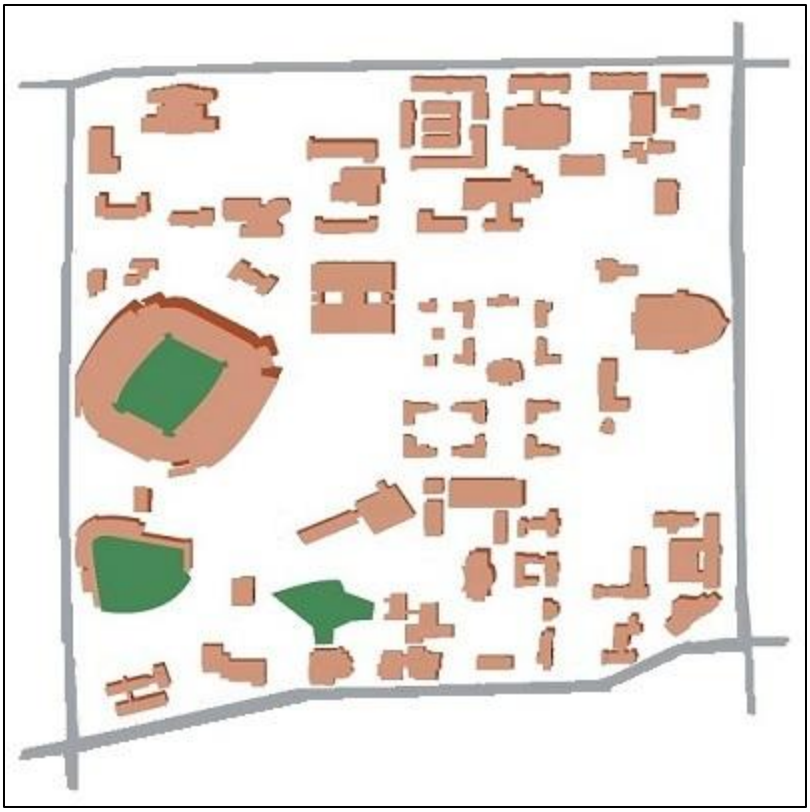


Figure 2.17: Auburn University Campus

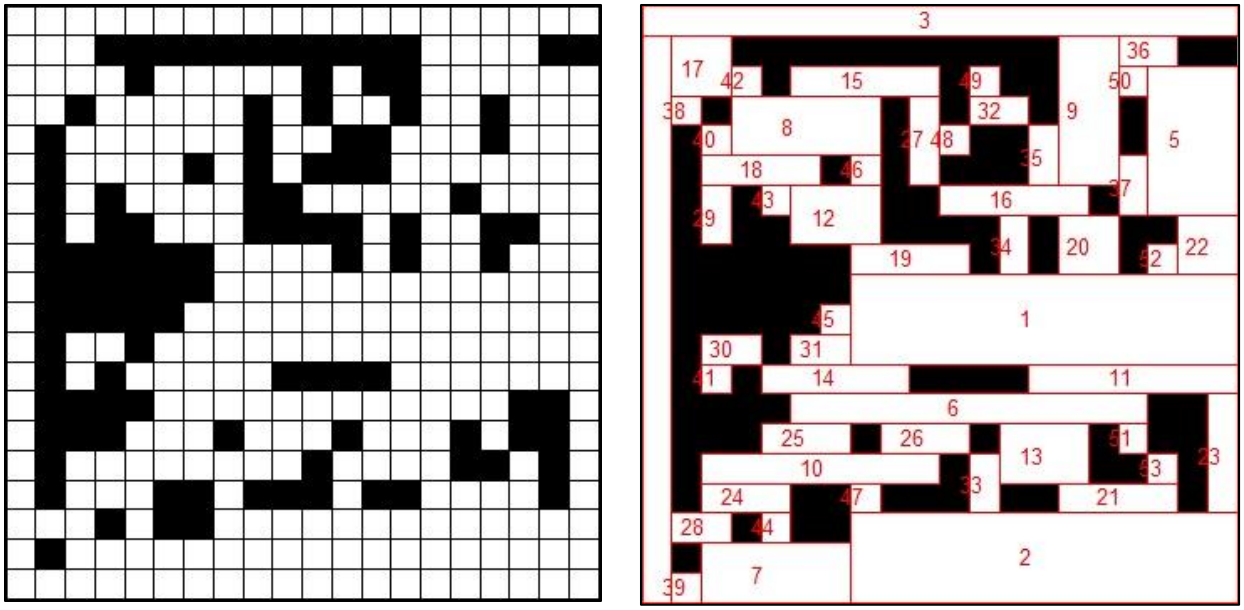


Figure 2.18: 20×20 Grid Map and Its R-map

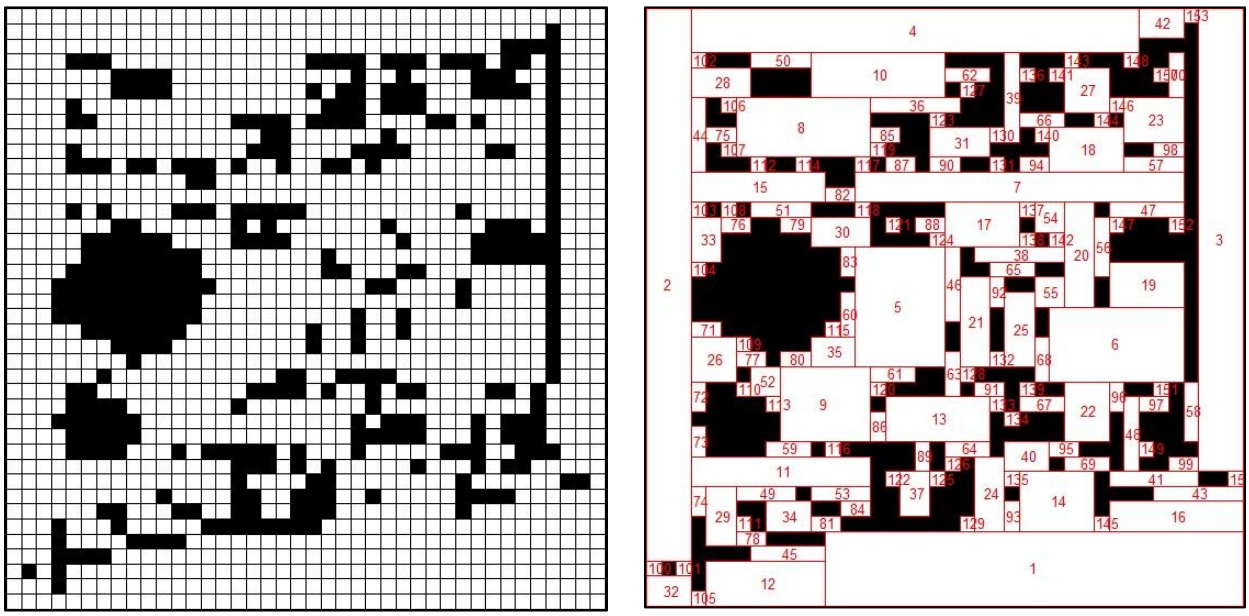


Figure 2.19: 40×40 Grid Map and Its R-map

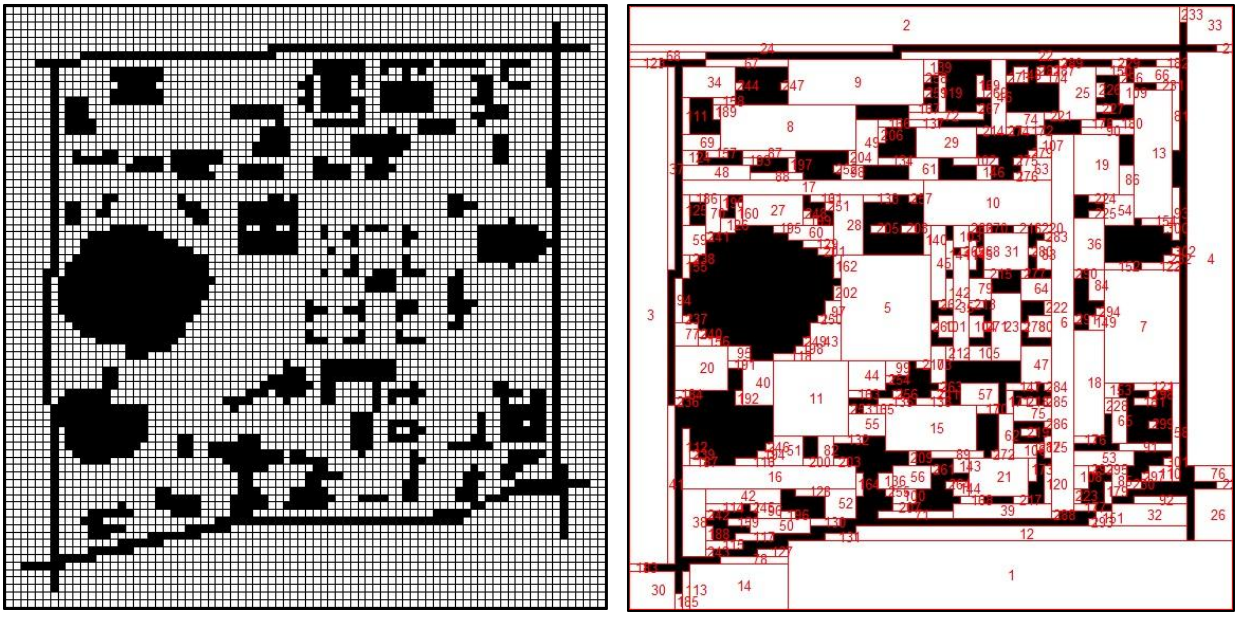


Figure 2.20: 80×80 Grid Map and Its R-map

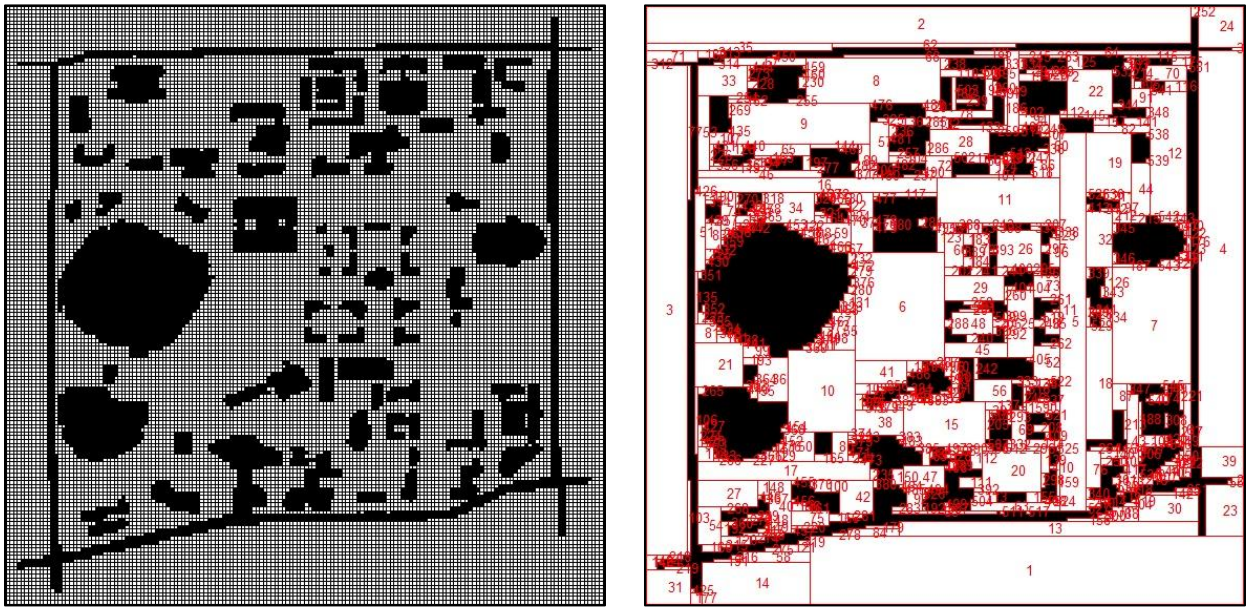


Figure 2.21: 160×160 Grid Map and Its R-map

Table 2.2: Outdoor R-mapping by Different Resolutions

Resolutions	**20×20**	**40×40**	**80×80**	**160×160**
Computation Time of R-mapping (sec)	0.057	0.189	0.747	3.283
Free cells in Grid Map	292	1,240	4,859	19,397
Rectangles in R-map	53	154	302	552

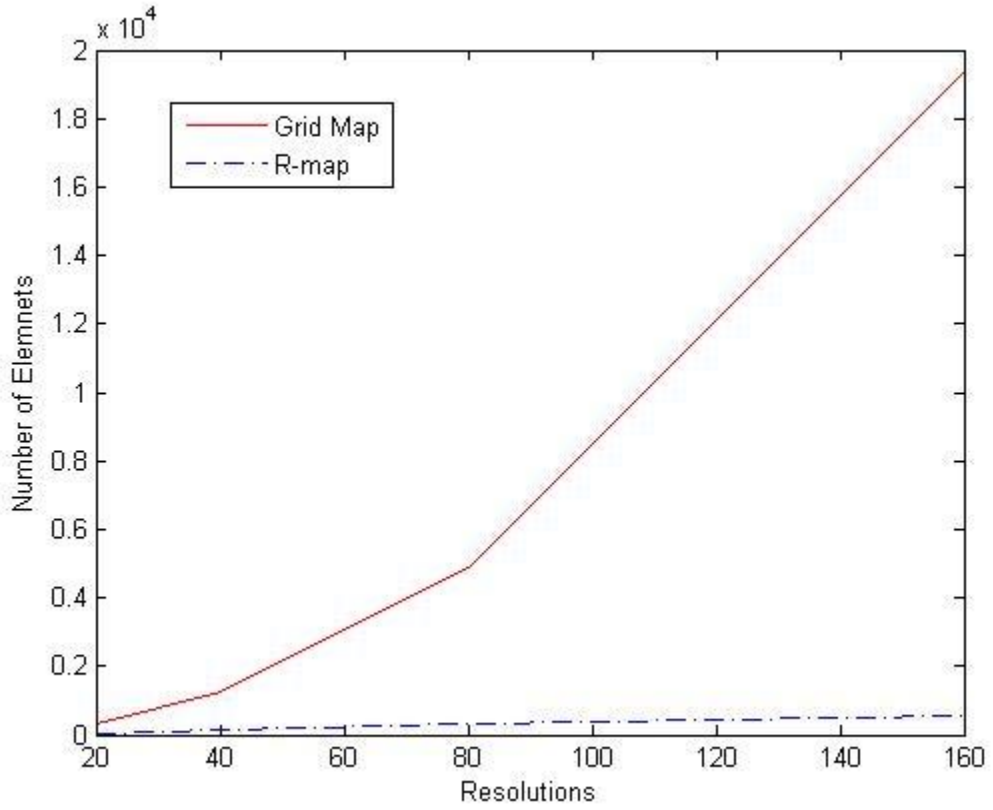


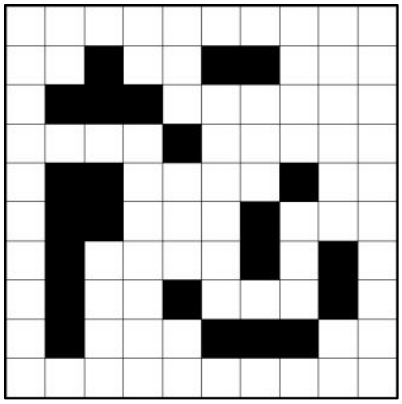
Figure 2.22: Comparison of Outdoor Maps by Number of Elemnets

2.4 Comparison of Data Storage

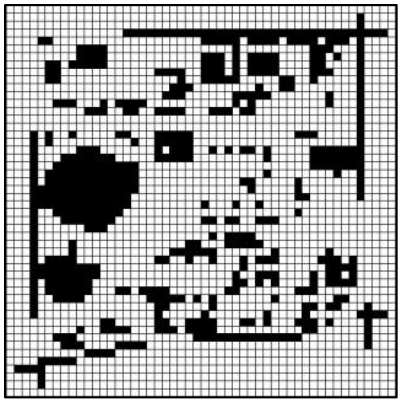
In this section, data storage of an R-map is discussed. A number is saved in a computer memory as binary numbers. One binary digit occupies one bit, and 8 bits are bound as one byte. This one byte can express $2^8 = 256$ different numbers, and the numbers in this study are considered as integers. To save a map, there three approaches are considered here: saving entire grid map as binary numbers, saving locations of only empty cells in a grid map and saving locations of empty rectangles in an R-map. For the first case, saving entire grid map as binary numbers, (width)×(length) bits are required to save them. For the second case, the locations of empty cells such as (1,1), (1,2), (2,3),⋯, (x,y) are the numbers to be stored. Each location (x,y) has two numbers to be stored in a memory. If the dimension of the map is less than 256, then two bytes are required to save one location. The last case is with an R-map. In this case, locations of empty rectangles which are the properties from the R-map algorithm are considered. The properties from Figure 2.10 are [1, 1, 2, 8], [5, 1, 3, 4], ⋯, [8, 7, 1, 1] where [x, y, width, length], and each rectangle has four numbers. Thus for map dimensions less than 256, one rectangle needs four bytes to be saved.

Four different resolution maps are shown in Figure 2.23, and Table 2.3 shows required spaces to save maps using each cases. In Table 2.3, the numbers of the first case are intuitively calculated, (width)×(length) bits. The numbers of the cases 2 and 3 are actual numbers from MATLAB using class of variables integer. Note that the first three maps whose dimensions are less than 256 are saved as integer 8 in MATLAB, but since the last 1250×1250 map has dimensions more than 256, it is saved as integer 16. For the first two 10×10 and 50×50 maps, case 1 needs less space than case 3. Therefore, to store the first two maps, saving the entire map as binary numbers is more efficient. However, the obstacles in their maps are distorted, and

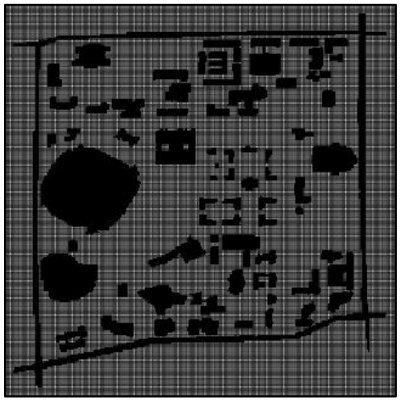
a reliable path cannot be generated from these maps. On the other hand, for the last two maps, 250×250 and 1250×1250 resolutions, case 3 occupies less space than case 1. This means that if a map has enough resolution to generate a reliable path, saving locations of empty rectangles is more efficient than saving the entire map.



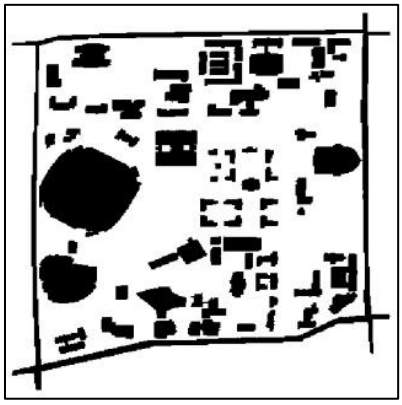
(a) 10×10



(b) 50×50



(c) 250×250



(d) 1250×1250
(grid removed)

Figure 2.23: Grid Maps in Different Resolutions

Table 2.3: Comparison of Data Storage

Resolutions	**10×10**	**50×50**	**250×250**	**1250×1250**
1. Entire Map (bytes)	13	313	7,813	195,313
2. Empty Cells (bytes)	154	3,838	94,776	2,365,330
3. Empty Rectangles (bytes)	80	764	3,128	32,936

Chapter 3

Path Planning

Because the R-map provides a reduced-element map representation focused on the largest obstacle-free spaces, it appears to be naturally suited for path-planning problems. For path planning with R-maps, Dijkstra's algorithm will here be used. This cost-based algorithm calculates the optimal path between any two points on a map.

This chapter investigates several methods for defining costs (i.e. weights) based on the properties of the R-map and the desired path. The following sections review Dijkstra's algorithm, introduce several weight definitions, and present path-planning examples.

3.1 Dijkstra's Algorithm

Dijkstra's algorithm was introduced by his paper in 1959 [10]. The algorithm computes optimal path based on a cost function. The algorithm applies to graphs with vertices (nodes) connected by edges (branches) which have nonnegative weights (costs) [11]. Thus the graph expresses the relationships between the vertices, and shows possible ways to a destination vertex. To travel vertex to vertex, a cost is incurred. The weighted graph is $G(V, E)$ where V is the set of vertices and E is the set of edges. The algorithm starts with the start node as the "current" node. All connected nodes become "tentative" and a tentative cost is assigned to each. The current node then becomes "settled", and the tentative node with the lowest tentative cost becomes the new current node. All nodes connected to the new current node are then evaluated.

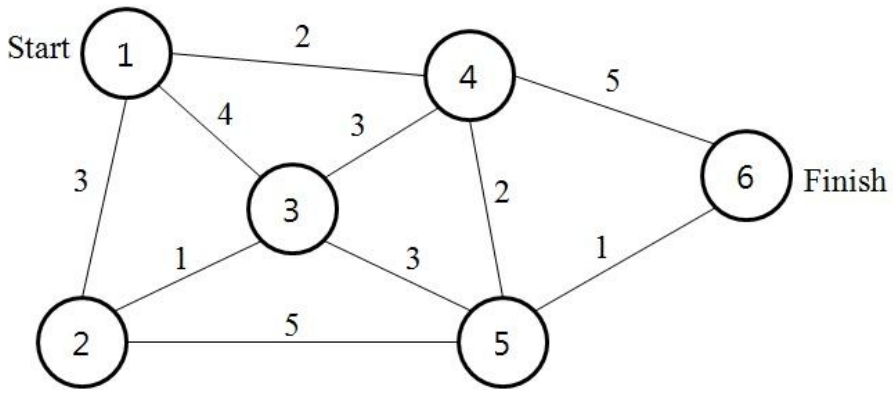


Figure 3.1: Example of Dijkstra's Algorithm

Among those connected nodes, previously unvisited nodes become tentative and an accumulated cost is assigned, and connected nodes that are already tentative potentially have their accumulated cost updated. The steps are repeated until the lowest-cost path to the destination node is found. The algorithm is in $O(|V|^2)$, because every node is current once, and the possible connections or settled nature of every other node must be considered. Figure 3.1 shows an example of undirected graph with start node 1 and destination node 6. In step 1, the tentative nodes are 2, 3 and 4, and their tentative costs are 3, 4 and 2 respectively. Because node 4 has the lowest cost, it becomes the next current node. In step 2, nodes 5 and 6 become tentative in addition to 2 and 3, and their accumulated costs are 4 and 7. For node 3, a new tentative cost of 5 is evaluated, but because the tentative cost of this node is already lower, it is not updated. These steps and the remaining steps are summarized in Table 3.1. The final costs for each node and optimal path from node 1 to node 6 are indicated in Figure 3.2.

To apply Dijkstra's algorithm to an R-map, each maximal empty rectangle, r_k, is considered as a vertex, and they are connected as $connection_k$ from the R-map algorithm, with the vertex weights defined in next section. Depending on the desired properties of the resulting

Table 3.1: Tentative Costs in Each Step

Step	Current Node	Cost of Current Node	Tentative Nodes	Tentative Costs
1	1	0	2 3 4	3 4 2
2	4	2	2 3 5 6	3 4 4 7
3	2	3	3 5 6	4 4 7
4	3	4	5 6	4 7
5	5	4	6	5

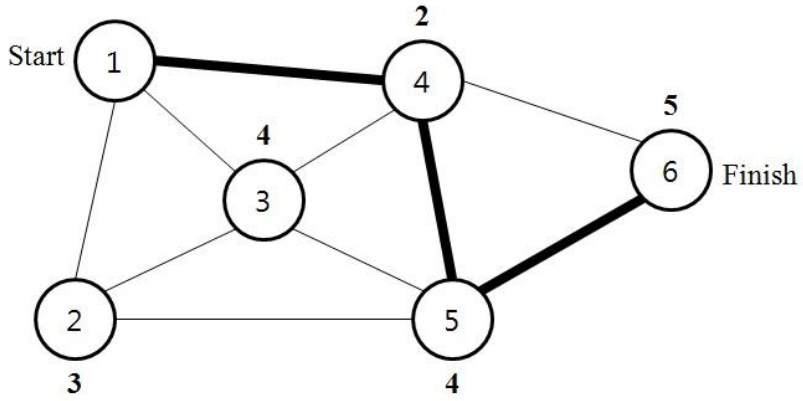


Figure 3.2: The Final Costs and the Lowest-Cost Path

path, a number of different parameters can be considered in assigning weights to the edges, such as the area, border length, distance and connections of the maximal empty rectangles.These definitions will result in symmetric directed graphs such that the cost to traverse from r_k to r_j may be different than the cost from r_j to r_k.

3.2 Weight Definitions

The weights in this study are defined with four parameters. The first approach is weight by area. Since the R-map algorithm produces a ranked list from the largest to smallest rectangle, the list of r_k in the structure *M* is in order of area. Thus, by simply giving lower weight to the higher ranked rectangle, their weights are set by area and the algorithm selects the path of larger area. The second approach is weight by border length of the rectangles. By giving lower weight to a rectangle which has longer border length, the algorithm selects a wider path. The third approach is weight by distance between center points of two rectangles. By giving lower weight to closer points, the path planning algorithm selects the shortest path to a destination, a classic application of Dijkstra's algorithm. The last possible approach is weight by number of connections. To give lower weight to the rectangle which has more potential paths to the next rectangle, find the number of connections of each rectangle. Thus, the result is a path who has diverse alternative path.

3.2.1 Weight by Area

Since the list of the property r_k in the structure *M* is in order from the largest to smallest rectangle, the weight to a rectangle j is simply the value itself, j.

$$W_{k,j} = j \tag{3.1}$$

where $W_{k,j}$ is a cost weight to travel from k to j. Table 3.2 shows the weight by area, W_{Area} of the example in Chapter 2.

3.2.2 Weight by Border Length

To let the algorithm select a wider path, a rectangle with longer connected-border length should have a lower weight. The connected-border length of two rectangles is the number of connected elements In step 4 of the R-map algorithm, neighboring elements of each maximal rectangle in *Gx* have been found to find the neighbors of the maximal rectangles. The neighboring elements of rectangle k (N_k) is the elements from equations (2.16)-(2.19).

$$N_k = [left\_connection_k, right\_connection_k, upper\_connection_k, lower\_connection_k] \quad (3.2)$$

where N_k indicates what rectangle is connected to the rectangle k and how many cells are connected. For example, from the equations (2.16)-(2.19), $N_1 = [7, 8, 9]$. Thus, rectangle 1 and rectangles 7, 8, 9 are connected with one cell, and since the path from rectangle 1 to the others is the narrowest, it should cost the most weight. Therefore, the weight is set to the reciprocal of the number of elements.

$$W_{k,j} = \frac{1}{b_{k,j}} \quad (3.3)$$

where $b_{k,j}$ is the number of connected element beween rectangles k and j. Table 3.2 shows the weight by border length, W_{Border} of the example; however, in the example, all borders have a length of one cell.

3.2.3 Weight by Distance

R-maps have a diverse size of cells, unlike the grid maps. Although two rectangles are connected, their area could be larger than that of several smaller rectangles. For that reason, the distance between two center points of two rectangles impacts whether that is a shorter path or not. To calculate the distances, the properties of rectangles r_k are recalled. As mentioned, the property contains location of the upper-left corner (x, y), width and height of a rectangle (w, h). From the property, the horizontal and vertical center points are calculated. Note that the locations of the points are $x + w/2$ and $y + h/2$. For example, r_1 and r_7 from the example of Chapter 2 are $r_1 = [1, 1, 2, 8]$ and $r_7 = [3, 4, 2, 1]$. The center points (C.P.) are calculated as below.

$$\text{Horizontal C. P. of Rectangle } 1 = r_1(1) + \frac{r_1(3)}{2} = 1 + \frac{2}{2} = 2 \tag{3.4}$$

$$\text{Vertical C. P. of Rectangle } 1 = r_1(2) + \frac{r_1(4)}{2} = 1 + \frac{8}{2} = 5 \tag{3.5}$$

$$\text{C. P. of Rectangle } 1 = (2, 5) \tag{3.6}$$

$$\text{Horizontal C. P. of Rectangle } 7 = r_7(1) + \frac{r_7(3)}{2} = 3 + \frac{2}{2} = 4 \tag{3.7}$$

$$\text{Vertical C. P. of Rectangle } 7 = r_7(2) + \frac{r_7(4)}{2} = 4 + \frac{1}{2} = 4.5 \tag{3.8}$$

$$\text{C. P. of Rectangle } 7 = (4, 4.5) \tag{3.9}$$

$$\text{Distance of two points} = \sqrt{(2-4)^2 + (5-4.5)^2} = 2.0616 \tag{3.10}$$

The distance value is the cost weight to travel from rectangle 1 to 7. In this manner, the weights can be designated to each connection.

$$W_{k,j} = \sqrt{(x_k - x_j)^2 + (y_k - y_j)^2} \tag{3.10}$$

where (x_k, y_k) is the center point of rectangle k and (x_j, y_j) is the center point of rectangle j. Table 3.2 shows the weight by distance, $W_{Distance}$.

3.2.4 Weight by Number of Connections

To define the weight by number of the connections of rectangles, recall $connection_k$ from the structure M. To provide for a diversity of connections in the event of re-planning rectangles with more neighbors should have lower weight. Therefore the weight of each rectangle is designated as the reciprocal of the number of connections.

$$W_{k,j} = \frac{1}{c_j} \tag{3.11}$$

where c_j is the number of elements of $connection_j$. For example, rectangle 2 in the example is connected to rectangles 4, 6 and 7. Rectangle 4 has 2 connections as in $connection_4 = [2, 10]$, rectangle 6 has 1 connection as in $connection_6 = [2]$ and rectangle 7 has 3 connections as in $connection_7 = [1, 2, 9]$. Thus, their weights are $weight_4 = 1/2$, $weight_6 = 1/2$ and $weight_7 = 1/3$. The weight by number of connections $W_{Connection}$ is shown in Table 3.2.

Table 3.2: Comaparison of Weights using Different Parameters

k	Neighbor	W_{Area}	W_{Border}	$W_{Distance}$	$W_{Connection}$
1	7	7	1	2.0616	0.3333
1	8	8	1	4.0311	0.5000
1	9	9	1	2.1213	0.5000
2	4	4	1	3.9051	0.5000
2	6	6	1	3.3541	1
2	7	7	1	2.9155	0.3333
3	5	5	1	4.1231	1
3	8	8	1	3.1623	0.5000
3	10	10	1	2.5000	0.5000
4	2	2	1	3.9051	0.3333
4	10	10	1	2.2361	0.5000
5	3	3	1	4.1231	0.3333
6	2	2	1	3.3541	0.3333
7	1	1	1	2.0616	0.3333
7	2	2	1	2.9155	0.3333
7	9	9	1	1.1180	0.5000
8	1	1	1	4.0311	0.3333
8	3	3	1	3.1623	0.3333
9	1	1	1	2.1213	0.3333
9	7	7	1	1.1180	0.3333
10	3	3	1	2.5000	0.3333
10	4	4	1	2.2361	0.5000

3.3 Selected Rectangles

In Table 3.2, a start rectangle of a path is among *k* and a finish rectangle is among the neighbors. For example, using the weight by area W_{Area}, if a path starts at rectangle 1 and the destination is rectangle 3, the algorithm selects a next rectangle among the neighbors of $k = 1$, there are three possible paths as [7, 8, 9], and the chosen neighbor becomes *k*. If rectangle 7 is chosen as the next rectangle, 7 becomes *k* and it has three possible paths as [1, 2, 9]. However, note that because the path should have the lowest total cost, the algorithm does not choose the lowest weight at each decision. In Figure 3.3, for example, there are two paths from rectangle 1

to 3. The path 1 is rectangles $1 \rightarrow 7 \rightarrow 2 \rightarrow 4 \rightarrow 10 \rightarrow 3$ and the path 2 is rectangles $1 \rightarrow 8 \rightarrow 3$. Among the three neighbors of rectangle 1, rectangle 7 has the lowest weight. However, selecting rectangle 7 makes the path go around to get to rectangle 3 and the total cost goes up as shown in Figure 3.3 (a). Selecting rectangle 8 as the path 2 costs more than selecting rectangle 7, but because the destination shows up right after rectangle 8, the path 2 has the lower total cost than the path 1. Therefore, the path planning algorithm selects the path 2 as the lowest cost path, Figure 3.3 (b).

	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	2	2	2	6	6	6
2	1	1	0	0	2	2	2	0	0	0
3	1	1	9	0	2	2	2	0	0	0
4	1	1	7	7	2	2	2	4	4	4
5	1	1	0	0	0	0	0	4	4	4
6	1	1	0	0	0	0	0	4	4	4
7	1	1	0	0	0	0	0	10	0	0
8	1	1	8	8	3	3	3	3	0	0
9	0	0	0	0	3	3	3	3	0	0
10	5	5	5	5	3	3	3	3	0	0

(a) Path 1

	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	2	2	2	6	6	6
2	1	1	0	0	2	2	2	0	0	0
3	1	1	9	0	2	2	2	0	0	0
4	1	1	7	7	2	2	2	4	4	4
5	1	1	0	0	0	0	0	4	4	4
6	1	1	0	0	0	0	0	4	4	4
7	1	1	0	0	0	0	0	10	0	0
8	1	1	8	8	3	3	3	3	0	0
9	0	0	0	0	3	3	3	3	0	0
10	5	5	5	5	3	3	3	3	0	0

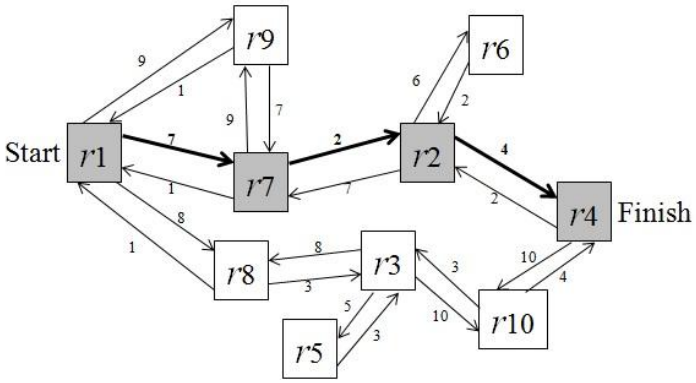
(b) Path 2

Figure 3.3: Comparison of Two Possible Paths

Using the weight by area W_{Area}, the selected rectangles as the path and the graph are shown in Figure 3.4, The start rectangle is 1 and the destination is rectangle 4. The algorithm considers all of the possible paths from rectangle 1 to 4, and compare their cost weights to select the lowest one. Figure 3.5 illustrates the results of path planning with $W_{Distance}$.

	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	2	2	2	6	6	6
2	1	1	0	0	2	2	2	0	0	0
3	1	1	9	0	2	2	2	0	0	0
4	1	1	7	7	2	2	2	4	4	4
5	1	1	0	0	0	0	0	4	4	4
6	1	1	0	0	0	0	0	4	4	4
7	1	1	0	0	0	0	0	10	0	0
8	1	1	8	8	3	3	3	3	0	0
9	0	0	0	0	3	3	3	3	0	0
10	5	5	5	5	3	3	3	3	0	0

(a)



(b)

Figure 3.4: (a) Result of Path Planning (b) Selected Path in Graph

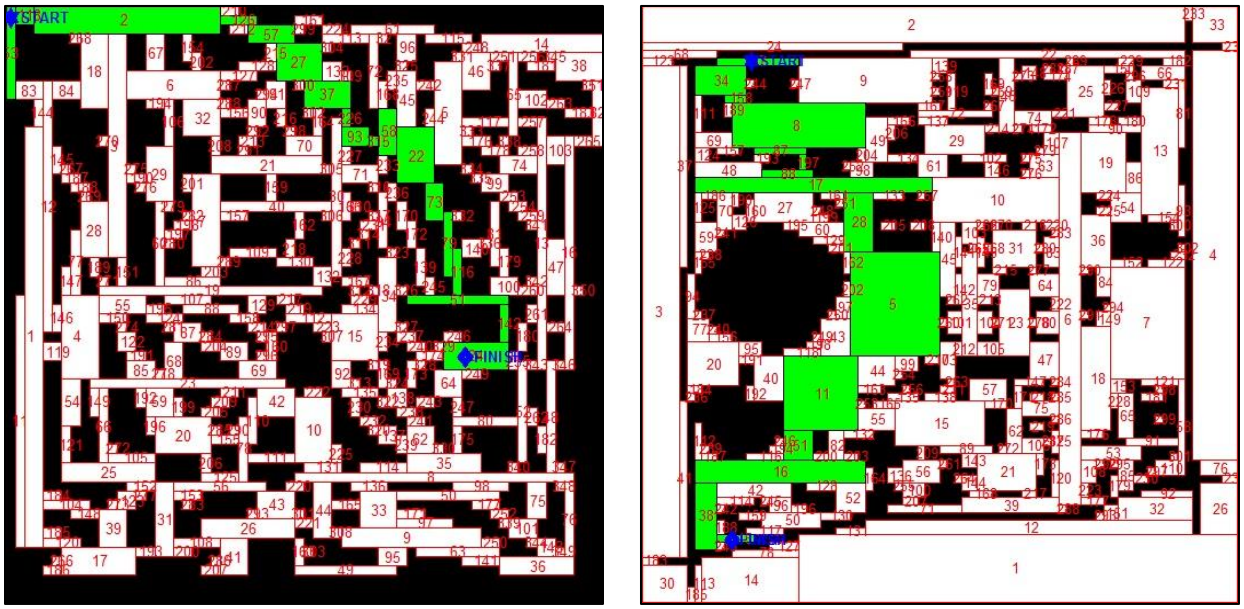


Figure 3.5: Results of Path Planning with Maps

3.4 Way Point Navigation

Path planning with grid maps give exact points to move on, but path planning with R-maps give only areas (rectangles). Although a vehicle will get to its destination after following

selected rectangles, it needs exact way points to define a precise path. To select way points, there are several approaches. A vehicle may pass the center points of each rectangle or the center point of the boundary of two rectangles. In this study, the latter appoach is adopted. Thus, way points consist of outward and inward points in each rectangle. To pick a center point of the boundary of two rectangles, two things are required. First, in which direction the next rectangle is connected. Depending on which side is the way to the next rectangle, its outward point will be different. Second, how many elements are connected between the next and current rectangles. By knowing this, the coordinates of a center point can be calculated.

	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	2	2	2	6	6	6
2	1	1	0	0	2	2	2	0	0	0
3	1	1	9	0	2	2	2	0	0	0
4	1	1	7	7	2	2	2	4	4	4
5	1	1	0	0	0	0	0	4	4	4
6	1	1	0	0	0	0	0	4	4	4
7	1	1	0	0	0	0	0	10	0	0
8	1	1	8	8	3	3	3	3	0	0
9	0	0	0	0	3	3	3	3	0	0
10	5	5	5	5	3	3	3	3	0	0

Figure 3.6: Selected Rectangles as a Path from 1 to 4 in Matrix *G*.

Figure 3.6 shows the selected rectangles from rectangle 1 to 4 and the path is 1 → 7 →2 → 4. At this point in time, the current rectangle is 1 and next rectangle is 7. The required information is that which side is rectangle 7 on rectangle 1, and with how many elements they are connected. For initialization, zeros are added around *G*. From step 4 of the R-map algorithm,

elements around rectangle 1 are acquired and they are from equations (2.16)-(2.19).

$$left\_connection_1 = [\text{from } L_1(2) \text{ to } L_1(4), L_1(1) - 1] = [0,0,0,0,0,0,0,0] \quad (2.16)$$

$$right\_connection_1 = [\text{from } L_1(2) \text{ to } L_1(4), L_1(3) + 1] = [0,0,9,7,0,0,0,8] \quad (2.17)$$

$$upper\_connection_1 = [L_1(2) - 1, \text{from } L_1(1) \text{ to } L_1(3)] = [0,0] \quad (2.18)$$

$$lower\_connection_1 = [L_1(4) + 1, \text{from } L_1(1) \text{ to } L_1(3)] = [0,0] \quad (2.19)$$

The next rectangle is 7 and it is in $right\_connection_1$. Thus, it is obvious that rectangle 7 is on the right side of rectangle 1. To find with how many elements they are connected, two columns are considered, the column on the rightmost of rectangle 1 (*column* 1) and its next column on right side (*column* 2). In Figure 3.6, *column* 1 is the second column and *column* 2 is the third column. Note that *G* in Figure 3.6 has ten elements in each column but, *column* 1 and 2 have twelve elements each since zeros are added around *G*.

$$column\ 1 = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 5\ 0]^{\mathrm{T}} \quad (3.12)$$

$$column\ 2 = [0\ 0\ 0\ 9\ 7\ 0\ 0\ 0\ 8\ 0\ 5\ 0]^{\mathrm{T}} \quad (3.13)$$

Elements of rectangle 1 in *column* 1 and elements of rectangle 7 in *column* 2 can be found as true or false, 1 or 0. The locations of their intersection indicate row numbers of connected elements. In this example, they are connected with one element.

$$column\ 1 = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0]^{\mathrm{T}} \quad (3.14)$$

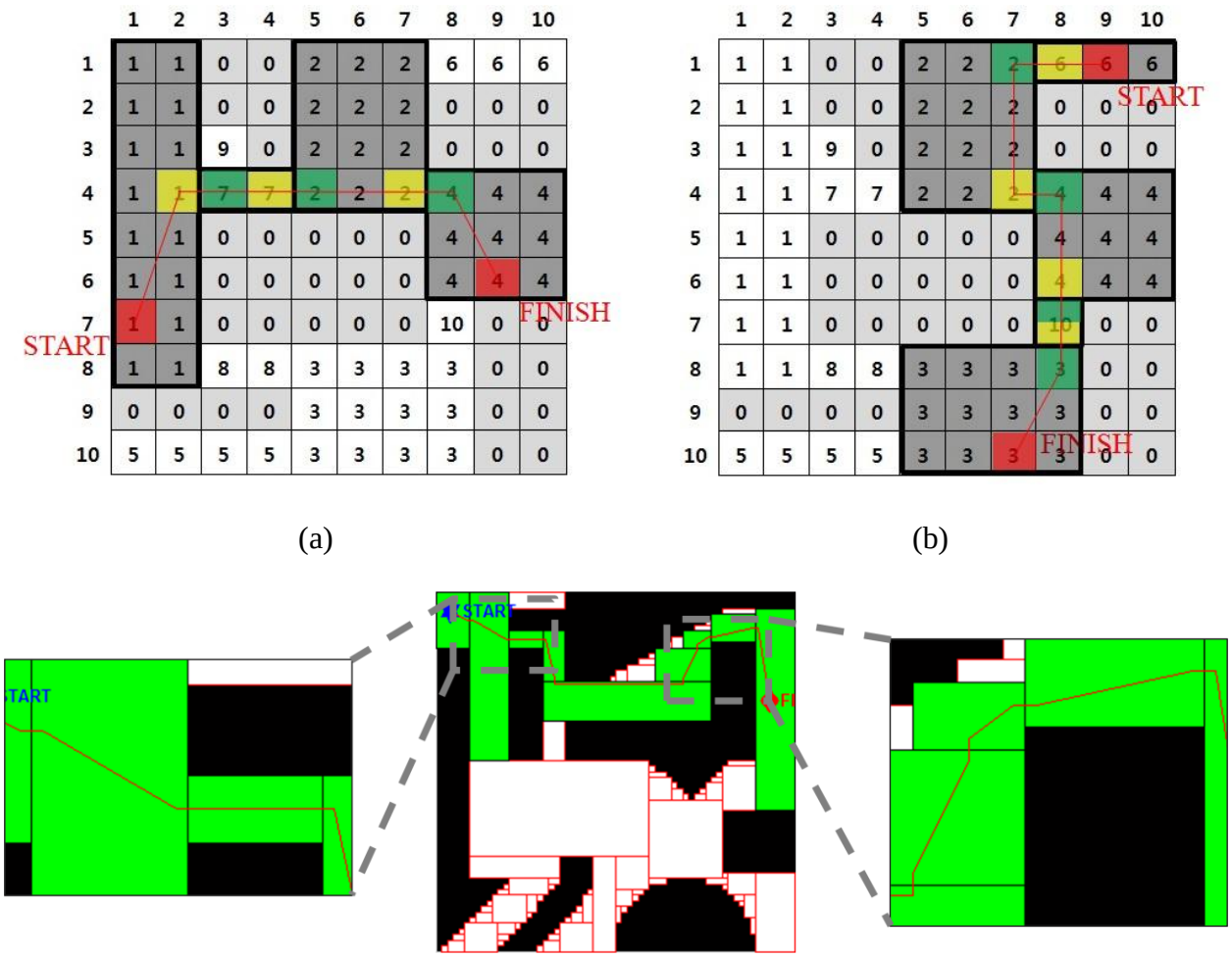
$$column\ 2 = [0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^{\mathrm{T}} \quad (3.15)$$

The two *columns* are connected with fifth element. To calculate the center point of their connection, the first and last locations of the intersected elements are considered.

$$\text{row number of center point} = \frac{(\text{first location}) + (\text{last location})}{2} \tag{3.16}$$

The center point is located on fifth row and the next column of the current rectangle. In this example, the row number is 5 and column number is 4. Since zeros are added around *G*, the location should be adjusted to the original *G*. The adjusted location is (3,4) where (x,y), and it will be the inward point to rectangle 7. The outward point from rectangle 1 is the next element on left side of the inward point, (2,4). Therefore, the way points from rectangle 1 to 7 are (2,4) and (3,4) as shown in Figure 3.7 (a), and next way points can be found in this manner. If the next rectangle is on the upper or lower sides, elements of two rows are extreacted from *G* instead of columns, and equation (3.16) is to find column number of the center point.

Examples of the result of way point navigation are shown in Figure 3.7 (a) and (b) with red for start and finish points, yellow for outward points and green for inward points. Figure 3.7 (c) shows details of the result from MATLAB.



(a) (b)

(c)

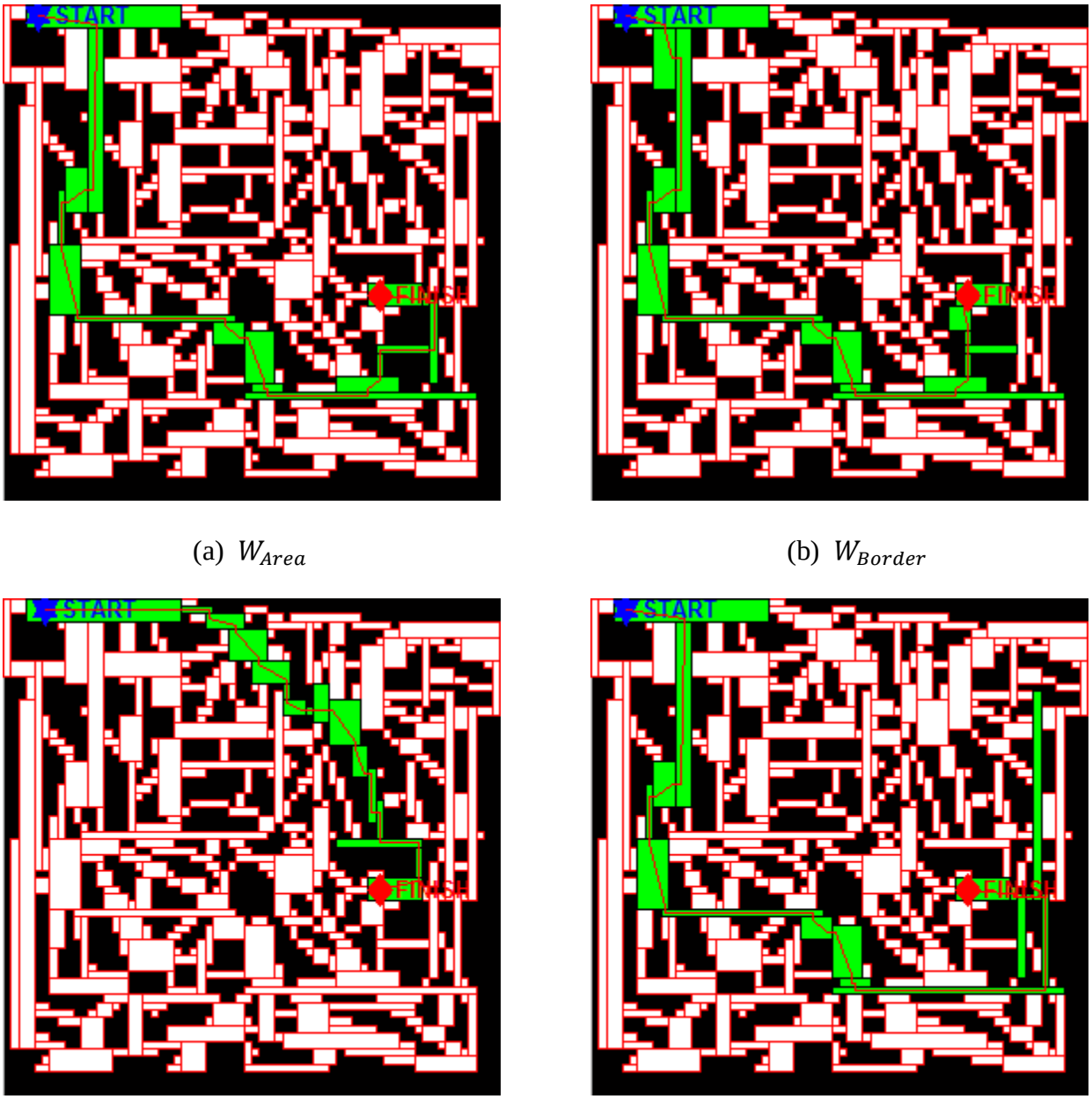
Figure 3.7: Results of Way Point Navigation

3.5 Path Planning Examples

In this section, examples of path planning with the four different weights are demonstrated. Three maps (complex, simple and AU campus map) are used for the path planning to make the difference of the weights clear, Figures 3.8-3.9 are path plannings with a complex map, Figures 3.10-3.11 are with a simple map and Figures 3.12-3.13 are with the AU campus map. The four paths with the complex map are all different depending on their weight parameters. Figure 3.8 (a) illustrates the path with the weight by area, W_{Area}. The list of the properties, r_k,

is used to designate lower weight to larger area. The algorithm selects the path of larger area. Figure 3.8 (b) shows the path with the weight by border length, W_{Border}. By calculating number of connected cells and designate lower weight to wider path, the algorithm selects the path of wider passageway. Figure 3.8 (c) shows the path with the weight by distance of center points, $W_{Distance}$. The location of center points of every rectangle is calculated and distances for them are computed to find a shorter path to the finish point. Figure 3.8 (d) illustrates the path with the weight by number of connections, $W_{Connection}$. To get the number of connections of each rectangle, $connection_k$ is used and the algorithm selects the rectangle which has more potential ways.

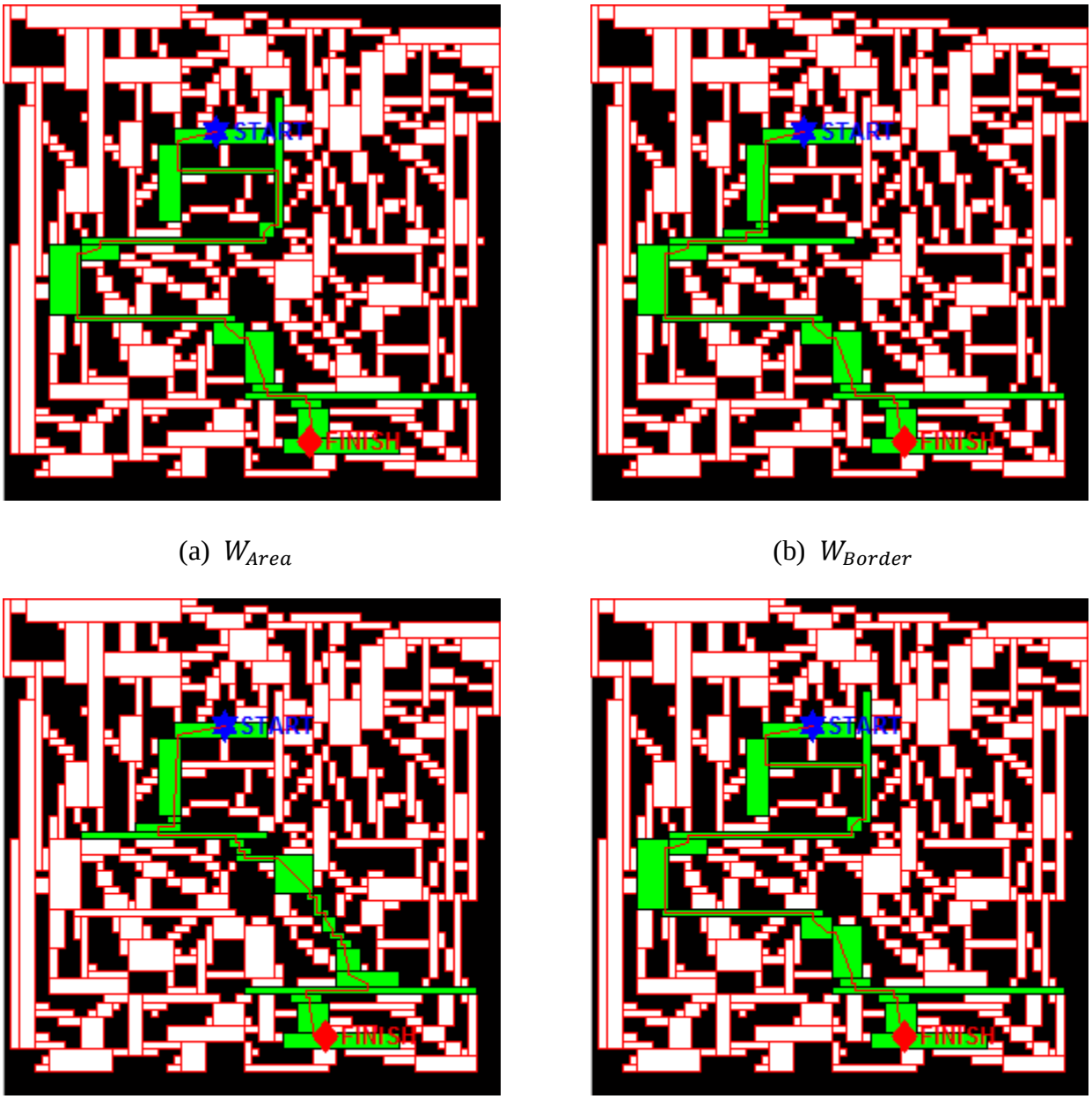
However, if a map does not have diverse rectangles and connections, the optimal paths with different weight definitions may be similar to each other. To demonstrate that the weight parameters do not always have a big effect on the path planning, paths with different start and end points are shown in Figure 3.10-3.11, a simple map, and Figure 3.12-3.13, the AU campus map. In Figure 3.10, the three paths are same out of four. This similarity is because of the simplicity of the map and lack of diverse connections. Also, basically paths with three weight definitions, W_{Area}, W_{Border} and $W_{Connection}$, are similar because larger area has longer border length and more connections. Figure 3.11 also shows the similarity in path planning. Therefore, the importance of the definition of the weight parameters are marked as the complexity of a map increases.



(a) W_{Area} (b) W_{Border}

(c) $W_{Distance}$ (d) $W_{Connection}$

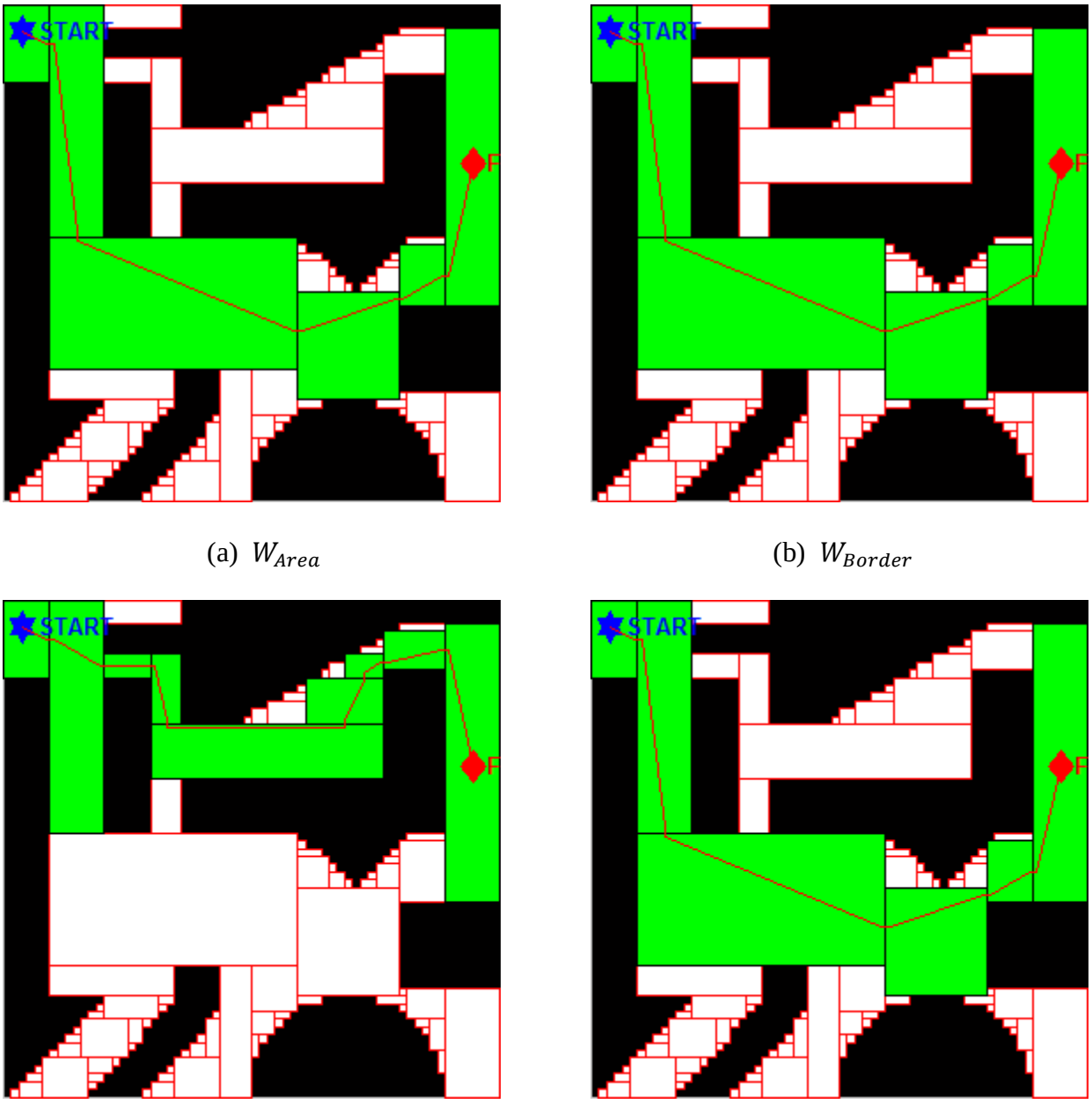
Figure 3.8: Path Planning 1 with a Complex Map



(a) W_{Area} (b) W_{Border}

(c) $W_{Distance}$ (d) $W_{Connection}$

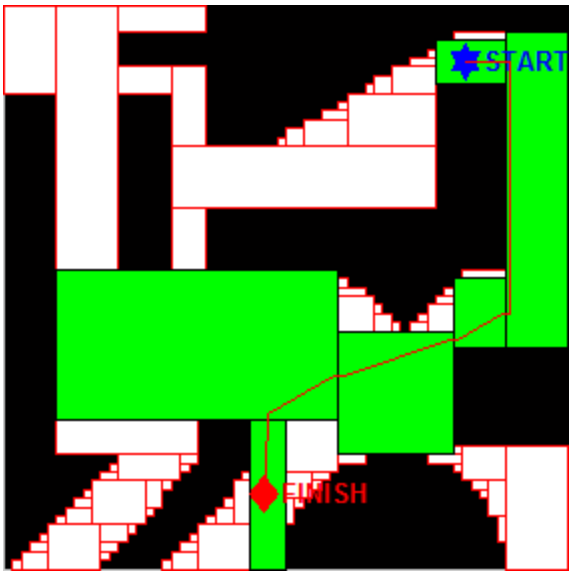
Figure 3.9: Path Planning 2 with a Complex Map



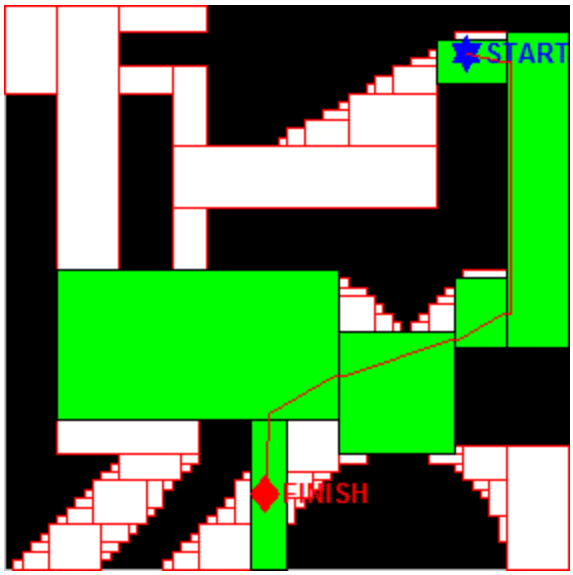
(a) W_{Area} (b) W_{Border}

(c) $W_{Distance}$ (d) $W_{Connection}$

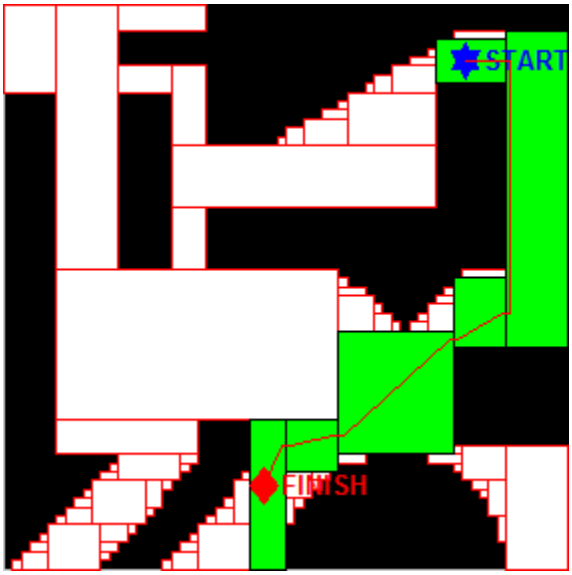
Figure 3.10: Path Planning 1 with a Simple Map



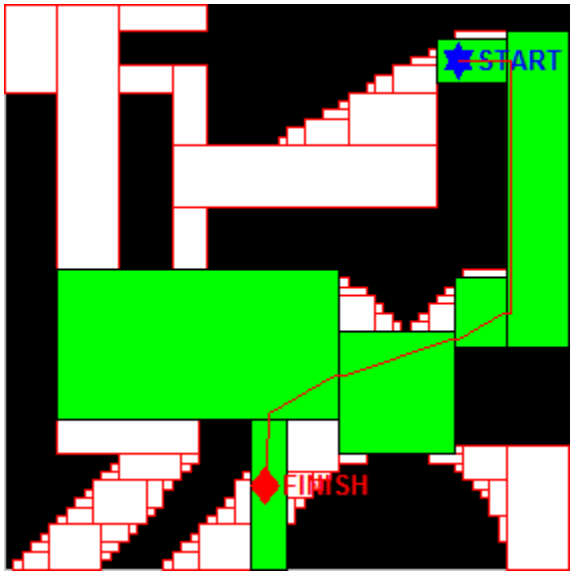
(a) W_{Area}



(b) W_{Border}

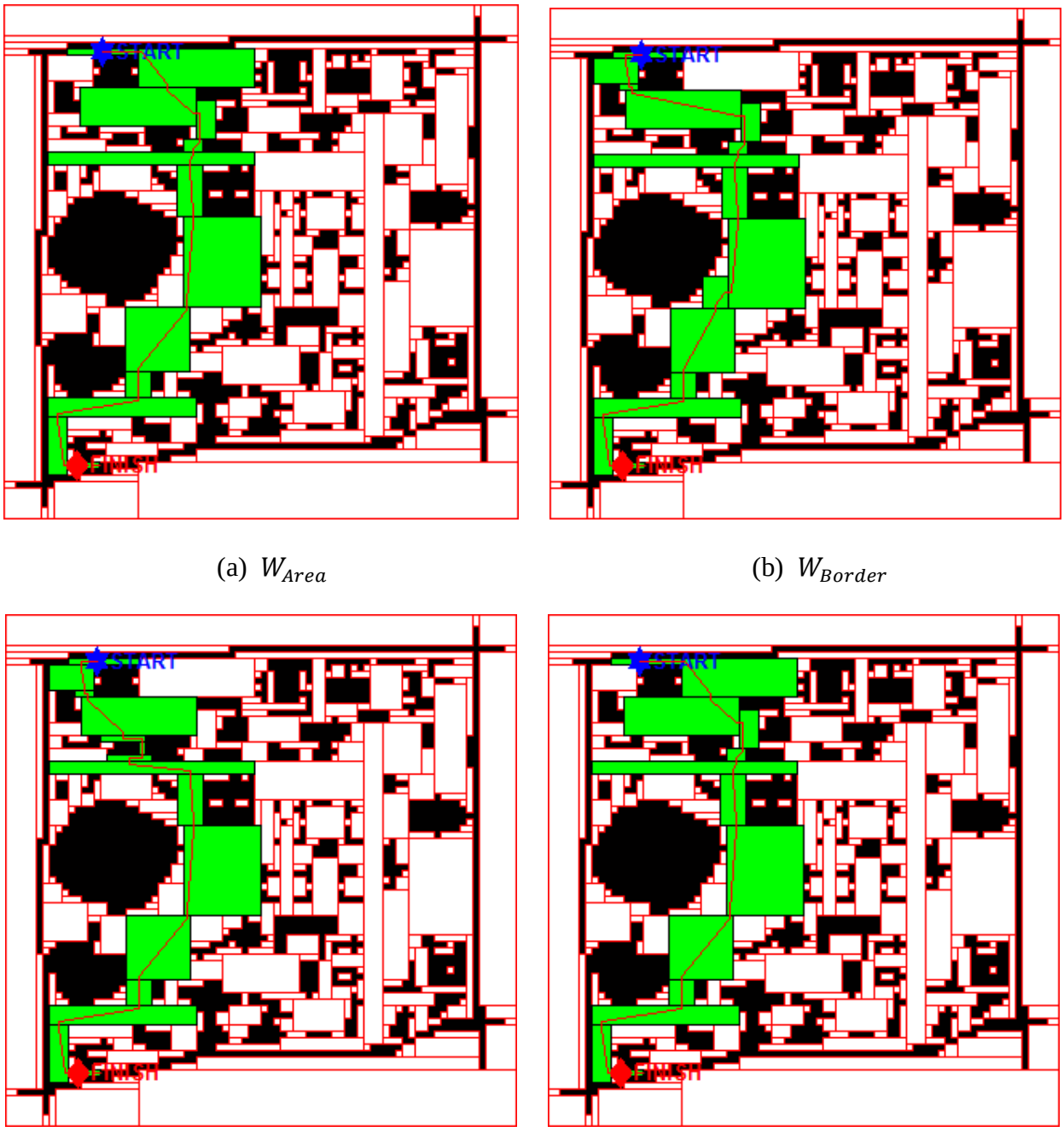


(c) $W_{Distance}$



(d) $W_{Connection}$

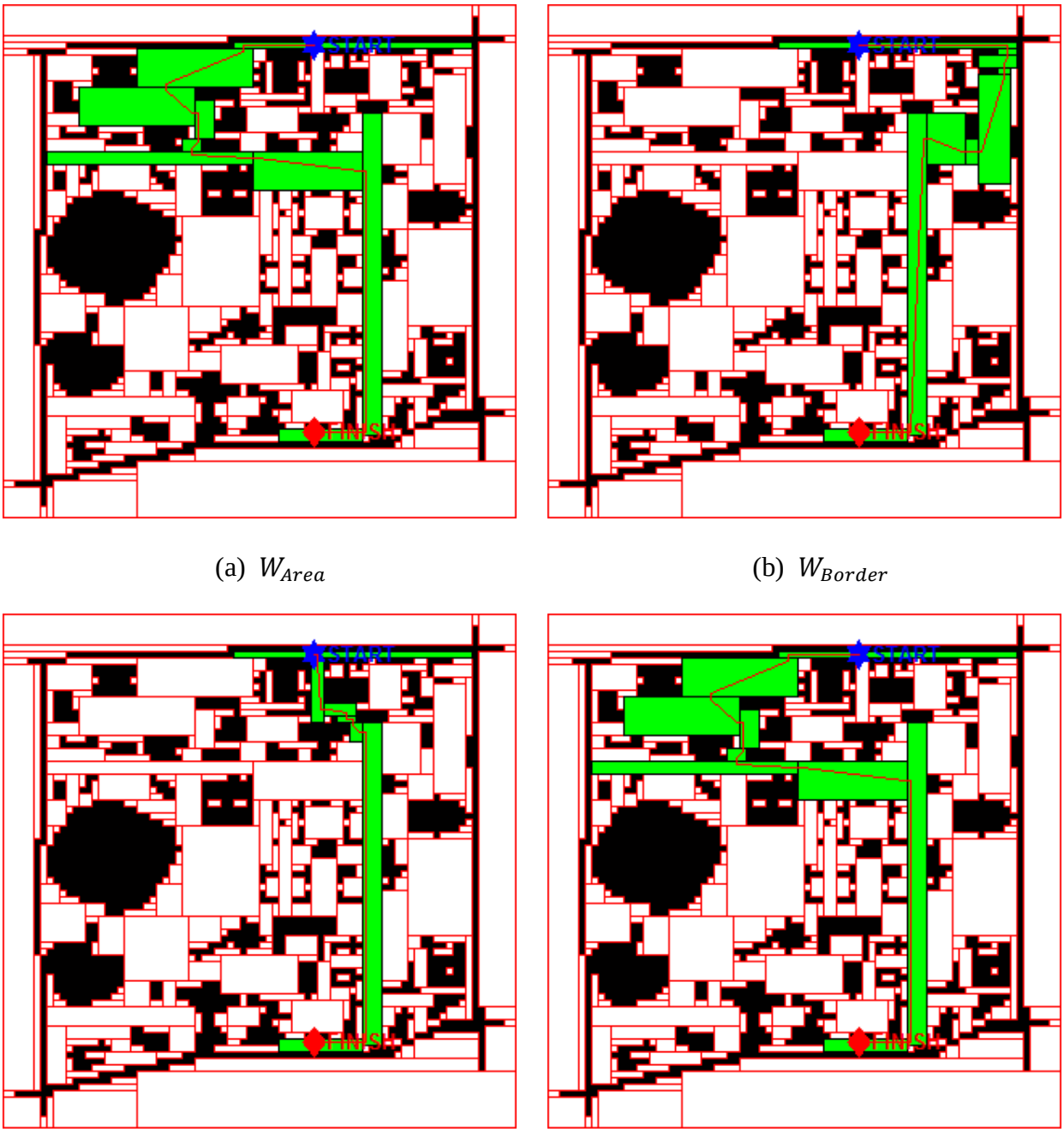
Figure 3.11: Path Planning 2 with a Simple Map



(a) W_{Area} (b) W_{Border}

(c) $W_{Distance}$ (d) $W_{Connection}$

Figure 3.12: Path Planning 1 with the AU Campus Map



(a) W_{Area} (b) W_{Border}

(c) $W_{Distance}$ (d) $W_{Connection}$

Figure 3.13: Path Planning 2 with the AU Campus Map

3.6 Comparison of Path Planning with Grid maps and R-maps

Path planning with R-maps is completed by considering Dijkstra's algorithm, four different weight definitions and way point navigation. In this section, to demonstrate efficiency of path planning with R-maps, paths with grid maps and with R-maps are compared. In both path planning, weight parameter is distance. For path planning with grid maps, Dijkstra's algorithm is applied as well, and each cell has weight one. Thus, the algorithm selects the shortest path between a start point and a destination. In Figure 3.14, 64×64 maps are used for path planning. Figure 3.15 shows path planning with four times higher resolution maps. Table 3.3 has comparisons of numbers of elements, connections, computation time of path planning and total distance. The computation time for R-maps includes weight and way point calculations. For both resolutions, path planning with an R-map takes less time than with a grid map. This is because R-maps have less elements and connections to compute a path with Dijkstra's algorithm. Total distance is the distance between start and finish points in each map. Since every cell in grid maps has weight one, path planning with grid maps should generate the shortest path. However, because paths in grid maps are always horizontal and vertical lines, these staircase paths can be longer than diagonal paths from R-maps as shown in Figure 3.14. Total distance in Table 3.3 demonstrates that path planning with R-maps can generate shorter paths than paths from grid maps.

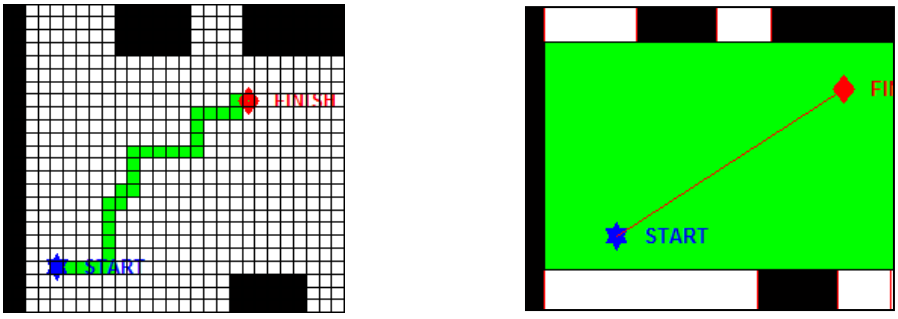
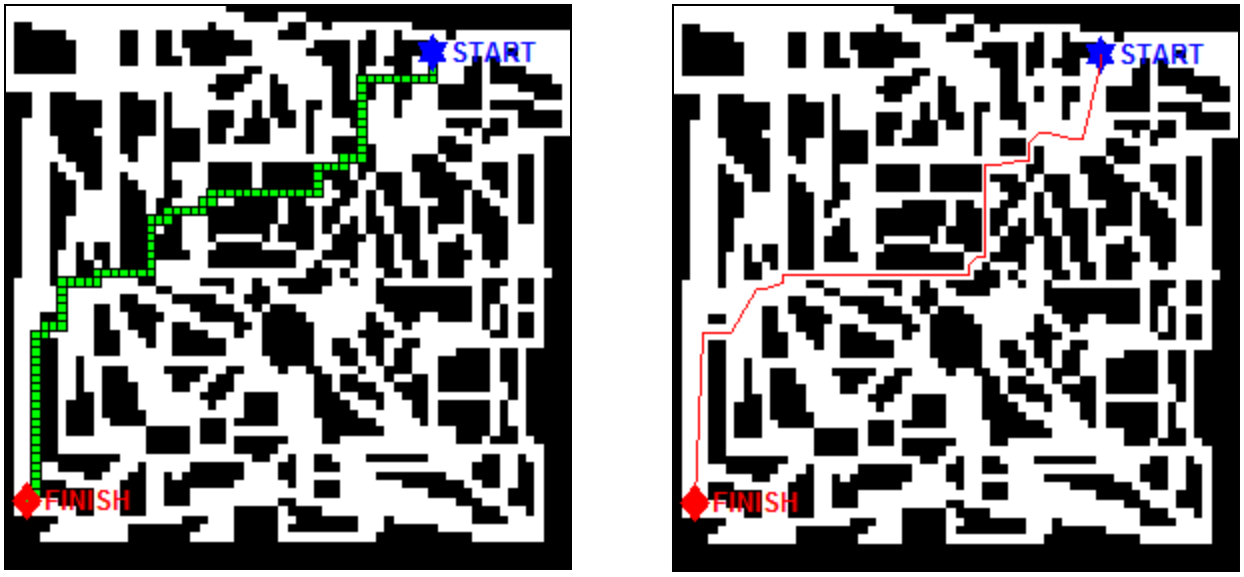


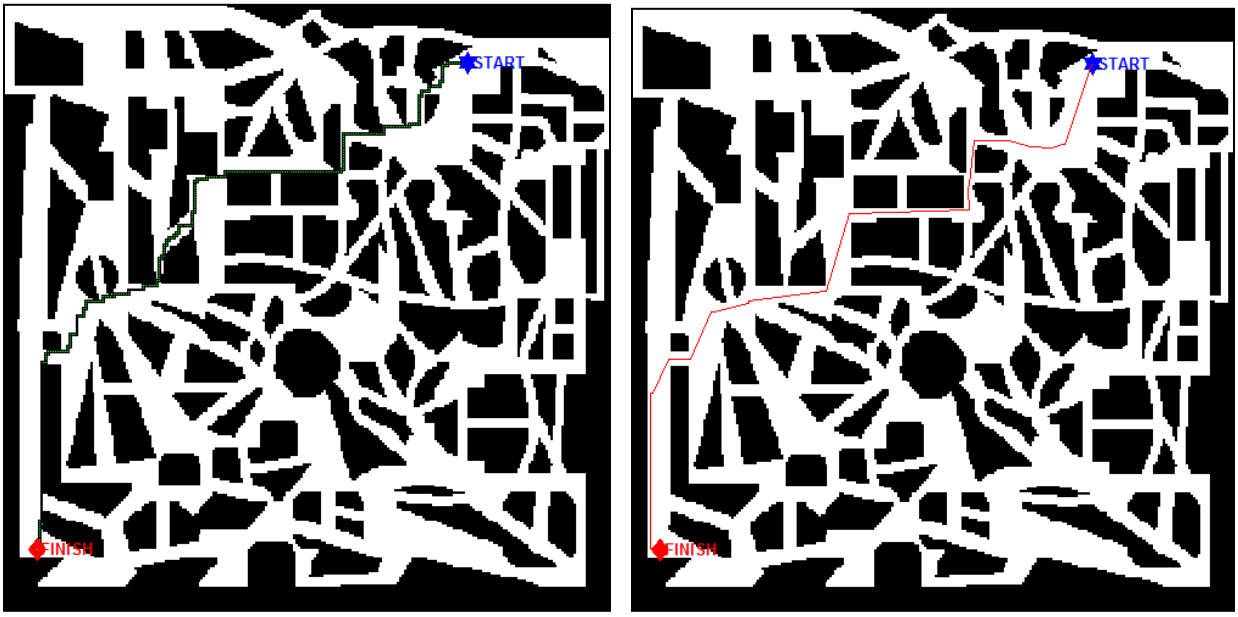
Figure 3.14: Total Distance in Each Map



(a) Path Planning with Grid Map (grid removed)

(b) Path Planning with R-map (rectangles removed)

Figure 3.15: Path Planning with 64×64 Resolution Maps



(a) Path Planning with Grid Map (grid removed)

(b) Path Planning with R-map (rectangles removed)

Figure 3.16: Path Planning with 256×256 Resolution Maps

Table 3.3: Comparison of Path Planning

Resolutions	**64×64**		**256×256**	
Map Type	**Grid map**	**R-map**	**Grid map**	**R-map**
Elements (nodes)	2,084	351	33,138	1,482
Connections (edges)	6,220	988	123,502	5,456
Computation Time of Path Planning (sec)	0.184	0.213	7.666	0.714
Total Distance (cells)	97	91.6	387	355.9

Chapter 4

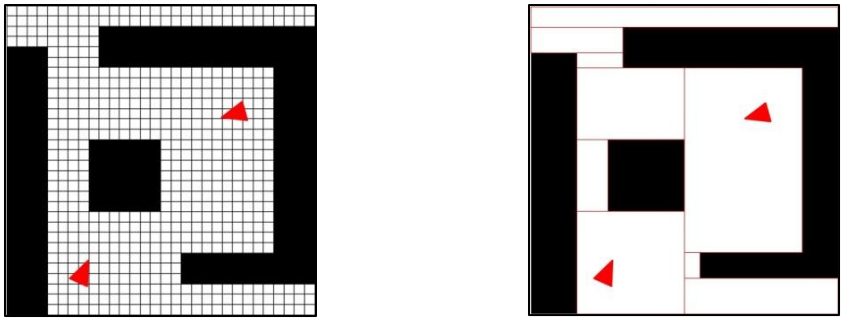
Further Applications and Conclusions

Because the input of the R-map algorithm is an image, R-map can be applied to other fields such as image processing. In addition, R-map reduces the amount of data by integration of redundant data. In this chapter, several further examples of R-map are suggested and conclusions of this work are described.

4.1 Further Applications

4.1.1 Data Reduction in Communication between Cooperative Agents

In the situation of sharing information between agents, their ability to send data can be limited for some reasons such as hardware limitation, computing speed or disturbance from outside. In Figure 4.1 (a), the two red agents are flying, doing collision avoidance using a grid map in a region. Assuming that they are on the same plane (same altitude), they gather information of obstacles through onboard sensors. Each agent sensing different area communicates their information about obstacles. In this scenario, the amount of data can be reduced by R-mapping. The three different approaches for data storage in section 2.4 can be applied for this scenario. As mentioned in section 2.4, if a map has resolution high enough to avoid distortion of obstacles, saving locations of empty rectangles from an R-map is more efficient. In the same sense, sharing locations of empty rectangles requires fewer amounts of data. Therefore, by using R-maps, the agents may not need high performance hardware.

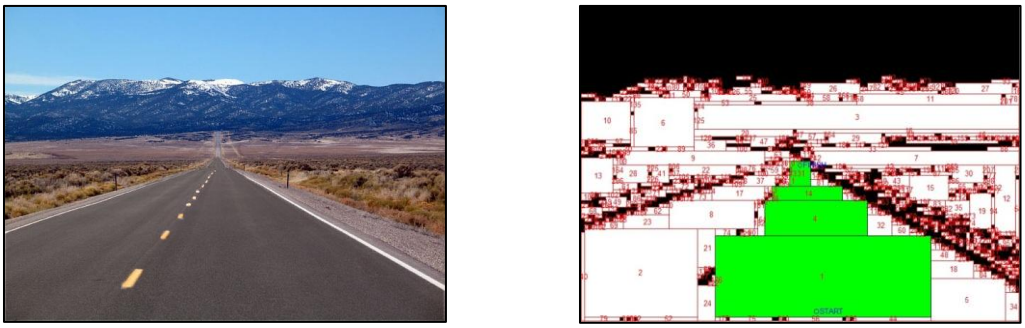


(a) Grid Map (b) R-map

Figure 4.1: Example of the Data Reduction

4.1.2 Driving Assistance

Not only a map image but also a landscape image can be used in the R-map algorithm. In a landscape image, bright parts are considered as free space and dark parts are considered as obstacles. The landscape image of Figure 4.2 (a) can be acquired from a visual camera mounted on a car. To make the road empty space and the lines on the road obstacles, the color should be reversed. Then the car can run through the empty space, and cannot cross the obstacles or the lines. Therefore, the car can keep in the lane not crossing the lines. This system can Figure out the area that a car can pass through, and give notice to the driver when the car gets close to the lines. However, the system has a weakness of lights and signs on the surface of the road.



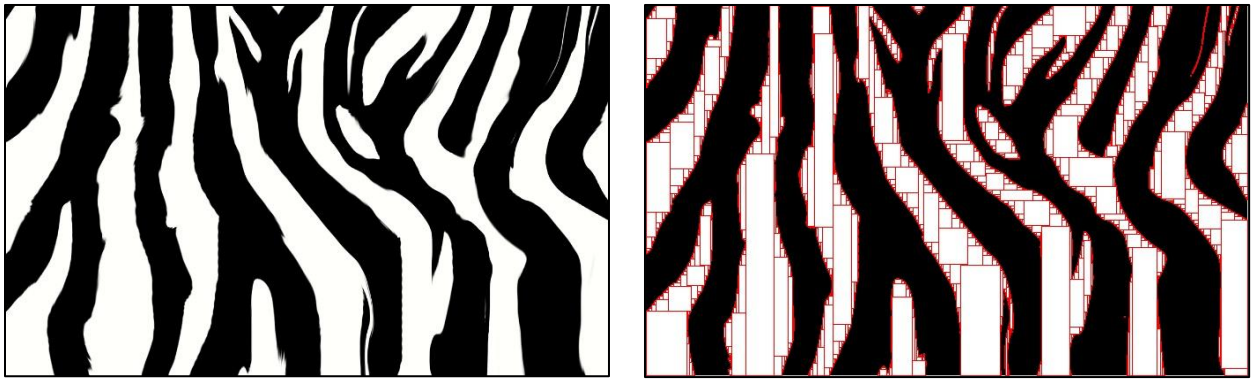
(a) Image from Visual Camera (b) Path Planning for Driving

Figure 4.2: Example of the Driving Assistance

4.1.3 Image Recognition

The next applications are of the image processing using r_k or $connection_k$ from the R-map algorithm. A pattern image, Figure 4.3 (a), is R-mapped to find connections of the rectangles. By knowing the relationships of the rectangles, Figure 4.3 (b), similar relationships can be found from a library and that is a similar pattern.

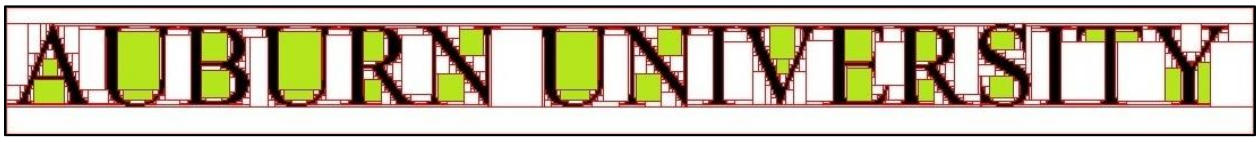
To apply R-map to the letter recongnition, two selected rectangles are compared after R-mapping as in Figure 4.4 (b). Then, a rule of the rectangles may be found. For example, in letter A, the two rectangles are selected. If the upper one is smaller than the lower one, it might be A. If the upper one is larger than the lower one and they are touching, it might be U. Hence, by scanning or taking a picture of a document, the letters are recognizable.



(a) Pattern (b) R-map of the Pattern

Figure 4.3: Examples of the Pattern Recognition

(a) Letter Image



(b) R-map of the Letter Image

Figure 4.4: Example of the Letter Recognition

4.2 Summary

Figure 4.5 illustrates the flows to implement path planning with the two mapping representations. Through this study, grid mapping is always faster than R-mapping, because grid mapping is only converting an RGB image to a binary image. On the other hand, R-mapping includes four steps to find maximal empty rectangles. However, for path planning with a grid map takes much longer than with an R-map. This is because path planning with a grid map considers every single cell to compute a path, while path planning with an R-map considers dramatically reduced number of rectangles. Furthermore, the reduced number of rectangles yields reduced space requirement to save a map.

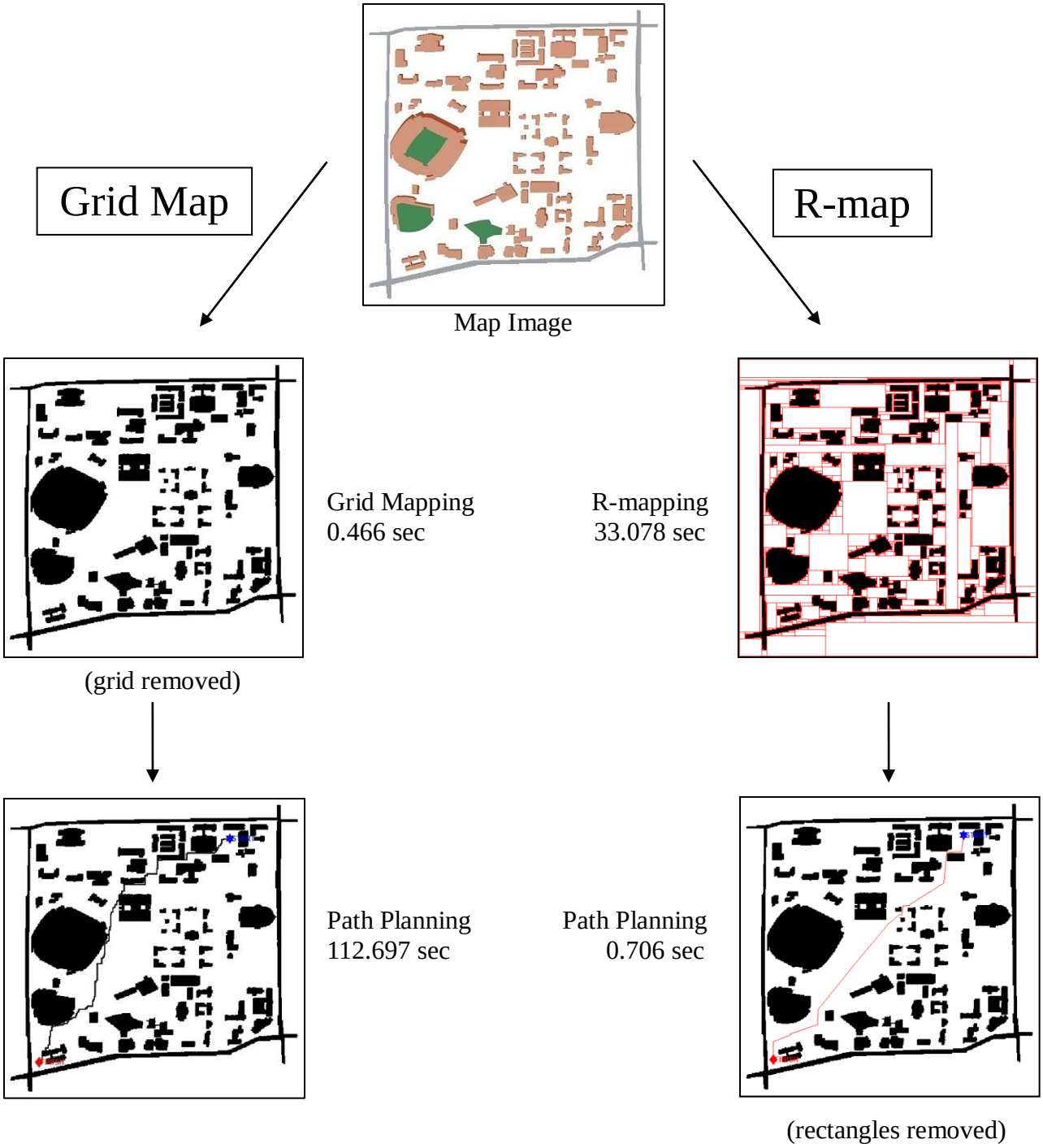


Figure 4.5: Comparison of Entire Process of Two Maps

4.3 Conclusions

In this thesis, the efficiency of R-maps is demonstrated comparing cell numbers of actual examples: the indoor and outdoor maps. In R-map chapter, specific steps of the algorithm, examples of R-mapping and efficiency in data storage were described. In path planning chapter, Dijkstra's algorithm, four different weight definitions, way point navigation, examples of path planning and efficiency in path planning were described to highlight the utility of R-maps in path planning.

R-map helps to build a higher resolution map with fewer elements by applying the maximal empty rectangle problem. R-map representation has several advantages. First, by focusing on the maximal empty rectangles, it is naturally suited to obstacle avoidance by focusing on the largest possible obstacle-free space. Second, by using a reduced number of elements, R-map is more efficient for path-planning algorithms such as Dijkstra's algorithm. Last, by integration of empty cells into a larger rectangle, R-map reduces redundant data thus it is efficient for data saving.

Although, the R-map has been introduced three years ago, this study advances the R-map for path planning, and demonstrates its efficiency in data storage and path planning. This new map representation, R-map, could be beneficial not only for path planning but also image processing. Since the R-map algorithm starts with an image, it can be applied to other fields related to image reconstruction.

Bibliography

[1] Cowlagi, R.V. and Tsiotras, P., "Multiresolution Path Planning with Wavelets: A Local Replanning Approach," 2008 American Control Conference, June 11-13, 2008.

[2] Prazenica, R.J., Kurdila, A.J. and Sharpley, R.C., "Receding Horizon Control for MAVs with Vision-Based State and Obstacle Estimation," AIAA Guidance, Navigation and Control Conference and Exhibit, August 2007.

[3] Kurdila, A., Nechyba, M., Prazenica, R., Dahmen, W., Binev, P., DeVore, R. and Sharpley, R., "Vision-Based Control of Micro-Air-Vehicles: Progress and Problems In Estimation," 43rd IEEE Conference on Decision and Control, Paradise Island, Bahamas, December 2004.

[4] Ahn, J.G. and Jeon, H.S., "R-map : A Hybrid Map Created by Maximal Rectangles," International Conference on Control, Automation and Systems 2010, Gyeonggi-do, Korea, pp. 1336-1339, October 2010.

[5] Dehne, F., "Computing the Largest Empty Rectangle on One- and Two-Dimensional Processor Arrays," *Journal of Parallel and Distributed Computing*, vol. 9, no. 1, pp. 63-68, May 1990.

[6] Cheng, Y., Iyengar, S.S. and Kashyap, R.L., "A New Method of Image Compression Using Irreducible Covers of Maximal Rectangles," *IEEE Transactions on Software Engineering*, vol. 14, no. 5, pp. 651-658, May 1988.

[7] Chazelle, B., Drysdale, R.L. and Lee, D.T., "Computing the Largest Empty Rectangle," *Society for Industrial and Applied Mathematics Journal on Computing*, vol. 15, no. 1, pp. 300-315, February 1986.

[8] Edmonds, J., Gryz, J., Liang, D. and Miller, R.J., "Mining for Empty Spaces in Large Data Sets," *Theoretical Computer Science*, vol. 296, no. 3, pp. 435-452, March 2003.

[9] Vandevoorde, D., "The Maximal Rectangle Problem," *Dr. Dobb's Journal*, vol. 23, no. 4, pp. 28-32, April 1998.

[10] Dijstra, E.W., "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.

[11] Barbehenn, M., "A Note on the Complexity of Dijkstra's Algorithm for Graphs with Weighted Vertices," *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 263, Feb. 1998.

[12] Crauser, A., Mehlhorn, K., Meyer, U. and Sanders, P., "A Parallelization of Dijkstra's Shortest Path Algorithm," 23rd International Symposium, MFCS'98, vol. 1450, pp. 722-731, Brno, Czech Republic, August 24-28, 1998.

Appendices

Appendix A

Matlab Codes Used for R-mapping and Path Planning

```
clear all
close all
clc

% ================================================================================
% ================================ R-mapping ================================
% ================================================================================


% ========================= Image Read and Plotting =========================

RGB = imread('complex.jpg');

I = rgb2gray(RGB);
threshold = graythresh(I);
bw = im2bw(I,threshold);
r1 = imresize(bw,0.2,'nearest');
r2 = imresize(r1,5,'nearest');

[G M] = Rmap(r2);

Ga = -G;
[m n] =  size(Ga);
Ga = [zeros(m,1) Ga zeros(m,1)];
Ga = [zeros(1,n+2) ; Ga ; zeros(1,n+2)];

imshow(r2);
hold on;
for i=1:length(M)
    r = M(i).r;
    rectangle('Position',[r(1:2)-0.5 r(3:4)],'EdgeColor','r')
end


% ============================ Main Algorithm ============================

function [Gx M] = Rmap(G)

Gx = double(G);
a = sum(sum(G));
M(a,1) =
struct('r',[],'connection',[],'ele',[],'left',[],'right',[],'upper',[],'lower
',[]);
```

```
for i=1:a
    r = FindMaximalEmptyRectangle(G);
    M(i).r = r;
    G = GridMapUpdate(G,r);
    if ~sum(sum(G))
        break;
    end
end
M = M(1:i,1);
[Gx M] = ComputeConnection(Gx,M);


% ======================== Step 1: Initialization ==========================
%                                    and
% ================= Step 2: Finding Maximal Empty Rectangle =================

function [best_r] = FindMaximalEmptyRectangle(G)

[M N] = size(G);
cx = zeros(M,N);
cx(:,N) = G(:,N);
for i = N-1:-1:1
    cx(:,i) = cx(:,i+1)+G(:,i);
    cx(G(:,i)==0,i)=0;
end
width = 0;
push = zeros(M,2);
largest_area = 0;
k = 1;
for x=1:N
    c = [cx(:,x);0];
    for y=1:M+1
        if c(y)>width
            k = k+1;
            width = c(y);
            push(k,:) = [y width];
        end
        if c(y)<width
            while (1)
                y0 = push(k,1);
                w0 = push(k,2);
                area = w0*(y-y0);
                if  area > largest_area
                    best_r =  [x y0 w0 y-y0];
                    largest_area = area;
                end
                k = k-1;
                if c(y)>=push(k,2)
                    break;
                end
            end
            width = c(y);
            if width ~= 0
                k = k+1;
                push(k,:) = [y0 width];
            end
```

```
        end
    end
end



% ========================= Step 3: Grid Map Update =========================

function G = GridMapUpdate(G,r,k)

if nargin<3
    k=0;
end
G(r(2):r(2)+r(4)-1,r(1):r(1)+r(3)-1)= -k;


% ======================= Step 4: Compute Connection ========================

function [Gx M] = ComputeConnection(Gx,M)
[m n] =  size(Gx);
for i=1:size(M,1)
    Gx = GridMapUpdate(Gx,M(i).r,i);
end
Gx = [zeros(m,1) Gx zeros(m,1)];
Gx = [zeros(1,n+2) ; Gx ; zeros(1,n+2)];
for i=1:size(M,1)
    r = M(i).r;
    x0 = r(1)+1;
    y0 = r(2)+1;
    x = r(1)+r(3);
    y = r(2)+r(4);

    left_conn  = Gx(y0:y,x0-1)';
    right_conn = Gx(y0:y,x+1)';
    upper_conn = Gx(y0-1,x0:x);
    lower_conn = Gx(y+1,x0:x);

    M(i).left = -unique(left_conn);
    M(i).right = -unique(right_conn);
    M(i).upper = -unique(upper_conn);
    M(i).lower = -unique(lower_conn);

    CONN = -[left_conn right_conn upper_conn lower_conn];
    M(i).ele = CONN;
    conn = unique(CONN);
    conn(conn==0)=[];
    M(i).connection = conn;
end
Gx = Gx(2:end-1,2:end-1);
```

```
% ================================================================================
% ============================== Path Planning ===================================
% ================================================================================


% ==================== Start and Finish Points Selection ====================


MAP=G;

% Select START point
pause(1);
h=msgbox('Please Select the START point using the Left Mouse button');
uiwait(h,5);
if ishandle(h) == 1
    delete(h);
end
xlabel('Please Select the START point ','Color','black');
but=0;
while (but ~= 1)
    [xstart,ystart,but]=ginput(1);
end
xstart=floor(xstart);
ystart=floor(ystart);
STARTISIN=-MAP(ystart,xstart);
MAP(ystart,xstart)=11111111;
plot(xstart,ystart,'bo');
text(xstart+5,ystart,'START','color','b')

% Select FINISH point
pause(1);
h=msgbox('Please Select the FINISH point using the Left Mouse button');
uiwait(h,5);
if ishandle(h) == 1
    delete(h);
end
xlabel('Please Select the FINISH point using the Left Mouse
button','Color','black');
but=0;
while (but ~= 1)
    [xfinish,yfinish,but]=ginput(1);
end
xfinish=floor(xfinish);
yfinish=floor(yfinish);
FINISHISIN=-MAP(yfinish,xfinish);
MAP(yfinish,xfinish)=99999999;
plot(xfinish,yfinish,'bd');
text(xfinish+5,yfinish,'FINISH','color','b')


% ============== Weight Calculation and Dijkstra's Algorithm ==============

% Calculate number of all connections
NUMBEROFCONNECTION=zeros(length(M),1);
for i=1:length(M)
```

```
    A=size(M(i).connection);
    NUMBEROFCONNECTION(i)=A(2);
end
LENGTH=sum(NUMBEROFCONNECTION);

% Rearrange the connections into CONNECTION
CONNECTION=zeros(LENGTH,3);
t = 0;
for j=1:length(M)
    CONNECTION(t+1:t+size(M(j).connection,2),1)= j;
      for k=1:size(M(j).connection,2)
          CONNECTION(k+t,2)= M(j).connection(k);
      end
    t= t+size(M(j).connection,2);
end

%% Calculate weights

% =====Weight by area=====
W = CONNECTION(:,2);

% =====Weights by border length=====
for i=1:length(CONNECTION)
    rect=CONNECTION(i,1);
    neigh=CONNECTION(i,2);
    s=sum(M(rect).ele==neigh);
    W(i)=1/s;
end

% =====Weight by distance=====
center_point=zeros(length(M),2);
for i=1:length(M)
    center_point(i,1)=M(i).r(1)+M(i).r(3)/2;
    center_point(i,2)=M(i).r(2)+M(i).r(4)/2;
end
for j=1:length(CONNECTION)
    rectangles=[...
        center_point(CONNECTION(j,1),1),center_point(CONNECTION(j,1),2);...
        center_point(CONNECTION(j,2),1),center_point(CONNECTION(j,2),2)];
    W(j)=pdist(rectangles,'euclidean');
end
W=W';

% =====Weight by number of connection=====
for i=1:length(CONNECTION)
    j = CONNECTION(i,2);
    W(i) = 1/numel(M(j).connection);
end
W = W';

% Calculate path
DG=sparse(CONNECTION(:,1),CONNECTION(:,2),W);
h=view(biograph(DG,[],'ShowWeights','on'))
[dist,path,pred]=graphshortestpath(DG,STARTISIN,FINISHISIN);
set(h.Nodes(path),'Color',[1 0.4 0.4])
```

```
edges = getedgesbynodeid(h,get(h.Nodes(path),'ID'));
set(edges,'LineColor',[1 0 0])
set(edges,'LineWidth',1.5)
%% Path Color
for i=path
    r = M(i).r;
    rectangle('Position',[r(1:2)-0.5 r(3:4)],'FaceColor','g')
end

%% Labeling on the rectangles
% Find center point of rectangle
center=zeros(length(M),2);
for i=1:length(M)
    center(i,1)=floor(M(i).r(3)/2+M(i).r(1))-4;
    center(i,2)=floor(M(i).r(4)/2+M(i).r(2));
end

% Labeling
for i=1:length(M)
    text(center(i,1),center(i,2),['',num2str(i)],'color','r')
end
plot(xstart,ystart,'bo')
text(xstart+5,ystart,'START','color','b')
plot(xfinish,yfinish,'bd')
text(xfinish+5,yfinish,'FINISH','color','b')

%% Waypoint navigation
location = zeros(2*(length(path)-2)+2,2);
k = 1;

for i = 1:length(path)-1
    j = path(i);
    x1 = M(j).r(1)+1;
    y1 = M(j).r(2)+1;

    if sum(M(j).left == path(i+1))>=1
        left1 = (Ga(:,x1)==path(i));
        left2 = (Ga(:,x1-1)==path(i+1));
        y = find(left1&left2);
        s = size(y);
        location(k,2) = (y(1)-1+y(s(1))-1)/2;
        location(k,1) = M(j).r(1);
        location(k+1,2) = location(k,2);
        location(k+1,1) = location(k,1)-1;
    end
    if sum(M(j).right == path(i+1))>=1
        right1 = (Ga(:,x1+M(j).r(3)-1)==path(i));
        right2 = (Ga(:,x1+M(j).r(3))==path(i+1));
        y = find(right1&right2);
        s = size(y);
        location(k,2) = (y(1)-1+y(s(1))-1)/2;
        location(k,1) = M(j).r(1)+M(j).r(3)-1;
        location(k+1,2) = location(k,2);
        location(k+1,1) = location(k,1)+1;
    end
```

```
    if sum(M(j).upper == path(i+1))>=1
        upper1 = (Ga(y1,:)==path(i));
        upper2 = (Ga(y1-1,:)==path(i+1));
        x = find(upper1&upper2);
        s = size(x);
        location(k,1) = (x(1)-1+x(s(2))-1)/2;
        location(k,2) = M(j).r(2);
        location(k+1,1) = location(k,1);
        location(k+1,2) = location(k,2)-1;
    end
    if sum(M(j).lower == path(i+1))>=1
        lower1 = (Ga(y1+M(j).r(4)-1,:)==path(i));
        lower2 = (Ga(y1+M(j).r(4),:)==path(i+1));
        x = find(lower1&lower2);
        s = size(x);
        location(k,1) = (x(1)-1+x(s(2))-1)/2;
        location(k,2) = M(j).r(2)+M(j).r(4)-1;
        location(k+1,1) = location(k,1);
        location(k+1,2) = location(k,2)+1;
    end
    k = k+2;
end
location(:,1) = location(:,1)-0.5;
location(:,2) = location(:,2)+0.5;
location2 = [xstart,ystart; location; xfinish yfinish];
y = location2(:,1)';
x = location2(:,2)';
line(y,x,'Color','r','LineWidth',1)
```