A Reduced Element Map Representation and Applications: Map Merging, Path Planning, and Target Interception

by

Jinyoung Park

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 5, 2017

Keywords: mapping, map merging, navigation, path planning, target interception

Approved by

Andrew J. Sinclair, Chair, Associate Professor of Aerospace Engineering
David A. Cicci, Professor of Aerospace Engineering
David M. Bevly, Professor of Mechanical Engineering
John Y. Hung, Professor of Electrical and Computer Engineering
Ryan E. Sherrill, Research Engineer of Air Force Research Laboratory

Abstract

Modern autonomous systems require complex and heavy computations. The complexity can be reduced by eliminating or integrating redundant information. In the case of mobile vehicles, occupancy grid map representations are conventionally adopted for path planning. Based on a grid map, a reduced element map representation named a rectangular map or an R-map has been introduced. The concept of R-map is integration of empty elements of a grid map into fewer elements with maximal sizes. Since an R-map has a reduced number of elements, path planning computations become much faster than conventional maps. Also, because the R-map algorithm focuses only on free space, it is naturally suited for obstacle avoidance.

The R-map can also be applied to map merging problems. Since R-maps represent spaces with varied sizes of rectangles, this feature can be a good source to recognize certain locations on the maps, unlike regular gridded cells. This work accomplishes map merging of local maps with unknown factors in their orientations, accuracy, and scales using the rectangular features from the R-map.

Further, this study extends the concept of the 2D R-map to 3D environments. Since 3D environments have an additional dimension of the z-axis, the process of R-mapping will be slightly different from 2D R-mapping, and the integrated cells will be represented as cuboids (volumes) instead of rectangles (areas). Those maximal empty cuboids (MECs) are obstacle-free spaces, and autonomous vehicles can accomplish obstacle avoidance by moving through a sequence of MECs. As applications, algorithms for path planning on R-maps are provided for stationary- and maneuvering-target interception in cluttered environments. This approach expects to provide a computational efficiency to guidance and navigation problems of autonomous systems.

Acknowledgments

The author would like to acknowledge and sincerely appreciate Dr. Andrew J. Sinclair for his guidance, support, patience, and encouragement shown during his graduate education. The author also would like to thank the faculty of the Aerospace Engineering Department at Auburn University for their help and the financial support, and Dr. Norman O. Speakman for giving him an opportunity to assist for four years.

In addition, the author thanks Dr. Emily A. Doucette and the Munitions Directorate of Air Force Research Laboratory for the opportunity to investigate this work at the AFRL REEF.

Finally, the author thanks the J-team, Jiyeong and Juwon, for their love and support. And thanks the author's beloved family, Hochul Park, Mija Choi, and Jieun Park, for their strong faith, constant support, and encouragement.

Table of Contents

Abstract ii

Acknowledgments iii

List of Figures vi

List of Tables xii

1 Introduction 1

2 Review of 2D R-mapping and Path Planning 4

2.1 R-mapping 4

2.1.1 Step 1: Initialization 5

2.1.2 Step 2: Finding the MER 6

2.1.3 Step 3: Grid Map Update 13

2.1.4 Step 4: Computing Connection 14

2.1.5 Result 17

2.2 Path Planning 18

3 Map Merging 22

3.1 Orthogonal Orientations 22

3.2 Shared Triangles 28

3.3 Scale matching and Merging 32

3.4 Result 34

4 3D R-mapping 36

4.1 Step 1: Initialization 37

4.2 Step 2: Finding the MEC 39

4.3 Step 3: Grid map update 50

4.4 Step 4: Computing connections 51

4.5 Result 54
5 3D Path Planning 57
5.1 Selection of cuboids 57
5.2 Waypoint navigation 60
5.3 Linear interpolation 61
5.4 Spline interpolation 64
5.5 Result 66
6 Target Interception 70
6.1 Traditional Approaches 70
6.1.1 Pursuit Navigation 71
6.1.2 Proportional Navigation 71
6.2 R-map Approaches 76
6.2.1 Pursuit Navigation 76
6.2.2 Interception-Triangle Navigation 78
6.3 Discussion 82
7 Conclusions 86
Bibliography 88
Appendices 92
A More simulations on target interception 93
A.1 Simulation 2 93
A.2 Simulation 3 93

List of Figures

2.1 A flow chart of R-map algorithm. 5

2.2 (a) A (5×5) grid map is acquired from a mapping robot. (b) The grid map can be expressed as a binary matrix G. (c) A (6×5) corresponding matrix cm is derived from G. 6

2.3 Rectangle-1 is found from the first entry in T. 10

2.4 Rectangle-2 is found from the second entry in T. 10

2.5 Rectangle-3 is found from the first entry in T. 12

2.6 Rectangle-4 is found from the second entry in T. 12

2.7 (a) MER_1 is found from step 2. (b) Corresponding elements of MER_1 in G are updated to zeros. Steps 1 through 3 are repeated with the updated G until all elements have been set to zeros. 13

2.8 (a) After iterations, all elements in G have been set to zeros, and 7 MERs are found. (b) Elements of each MER are updated to their rank, and additional elements equal to zero are added around G, thus the dimension becomes (7×7). 14

2.9 (a) G-map has 19 free-elements (empty cells) and 42 connections. (b) R-map has 7 free-elements (MERs) and 12 connections. 16

2.10 The Auburn University campus map in three different map representations. . . 17

2.11 Path planning with the four different weight definitions. The path starts at the start point (blue star) and ends at the finish point (red diamond) through the sequence of MERs (green rectangles). 19

2.12 Path planning with different map representations. 20

3.1 Two local maps of a simple environment. The maps have naturally different accuracy and scales to each other, and intentionally rotated. 23

3.2 Results from the R-map algorithm. Map 1 and map 2 have 103 and 91 rectangles, respectively. 24

3.3 Zoomed in image of the top-left portion of figure 3.2a. A series of the largest neighbors along each edge-direction divides which rectangles are on which side of the environment. 24

3.4 Pairs of parallel lines: one yellow and one pink lines are a pair. The true rotation angles are $\Lambda_0^{(1)} = 45^\circ$ and $\Lambda_0^{(2)} = 225^\circ$ counterclockwise, and estimated angles are $\Lambda^{(1)} = -45^\circ$ and $\Lambda^{(2)} = -45^\circ$ for map 1 and map 2, respectively. 27

3.5 Results from orthogonal-orientations algorithm. Orthogonally oriented maps are acquired through rotating the initial maps by estimated angles: $\Lambda_0^{(1)} + \Lambda^{(1)} = 0^\circ$ and $\Lambda_0^{(2)} + \Lambda^{(2)} = 180^\circ$ for the map 1 and map 2, respectively. 28

3.6 A set of shared-triangles. The feature vector, FV, contains triangular and rectangular features. (a) A triangle is made by the center points of three rectangles, $\{r_i, r_j, r_k\}$, in map 1. This triangle has multiple corresponding triangles in map 2. (b) One of corresponding triangles is made of rectangles, $\{r_e, r_f, r_g\}$, in map 2. 30

3.7 Results of shared-triangles algorithm. A set of the best shared-triangles has $\min(J)$. The Δ_1's and Δ_2's in both maps are the correct shared-rectangles, but Δ_3's are incorrect due to different map scales. 31

3.8 Orientation-matched maps. The orientations are matched through rotating map 1 by $\Lambda_f = 180°$. Δ_3's are still incorrect shared-rectangles ($ratio = 1.1176$). . . . 33

3.9 Scale-matched maps ($ratio = 1$). The scales are matched through re-scaling map 2 by the previous *ratio*. Δ_3's are the correct shared-rectangles. 33

3.10 Merged maps. (a) Merging maps of figure 3.8 makes a worse fit. (b) Merging maps of figure 3.9 makes a better fit. 33

3.11 Actual maps from two ground mapping robots. The three factors, orientations, accuracy and scales, are unknown and may different to each other map. 35

3.12 Merged map of figure 3.11 35

4.1 A 3D environment can be represented as (a) a 3D array or (b) multiple layers of 2D matrices. 37

4.2 cx is obtained by right-to-left x-wise summations of contiguous free-elements. . 38

4.3 cz is obtained by top-to-bottom z-wise summations of contiguous free-elements. 38

4.4 cm is obtained by a summation of cx as real numbers and cz as imaginary numbers. Also, additional elements equal to zero are added to the foremost of cm. . 39

4.5 Each dimension of a cuboid, (depth × length × height), is associated with the y-, x-, and z-axis, respectively. 40

4.6 (a) The bottom area (1×1) has $height = 2$. (d) Cuboid-1 has a volume of $(1 \times 1 \times 2) = 2$. 45

4.7 (a) The bottom area (2×1) has $height = 2$. (d) Cuboid-2 has a volume of $(2 \times 1 \times 2) = 4$. 45

4.8 (a) The bottom area (1×1) has $height = 3$. (d) Cuboid-3 has a volume of $(1 \times 1 \times 3) = 3$. 46

4.9 (a) The bottom area (1×2) has $height = 3$. (d) Cuboid-4 has a volume of $(1 \times 2 \times 3) = 6$. 46

4.10 (a) The bottom area (1×3) has $height = 2$. (d) Cuboid-5 has a volume of $(1 \times 3 \times 2) = 6$. 46

4.11 (a) The bottom area (3×1) has $height = 2$. (d) Cuboid-6 has a volume of $(3 \times 1 \times 2) = 6$. 48

4.12 (a) The bottom area (2×1) has $height = 2$. (d) Cuboid-7 has a volume of $(2 \times 1 \times 2) = 4$. 49

4.13 (a) The bottom area (2×2) has $height = 2$. (d) Cuboid-8 has a volume of $(2 \times 2 \times 2) = 8$. 49

4.14 Corresponding elements of MEC_1 (cuboid-8) in G are updated to zeros. 50

4.15 Updated G has additional elements equal to zeros around it. Connections of MEC_1 is neighboring elements on each side (red elements). 52

4.16 Two different representations of a 3D environment. (a) G-map has 21 free elements and 66 connections. (b) R-map has 9 free elements and 26 connections. . 53

4.17 A 3D environment of New York City. 55

4.18 Three different map representations in $(16 \times 16 \times 16)$ resolutions. 56

4.19 Three different map representations in $(32 \times 32 \times 32)$ resolutions. 56

4.20 Three different map representations in $(64 \times 64 \times 64)$ resolutions. 56

5.1 Dijkstra's algorithm selects the lowest-cost path to a finish point. 59

5.2 The start point is in cuboid-3, and the finish point is in cuboid-9. A series of cuboids that connects cuboids-3 and cuboid-9 in the shortest distance is 3-6-2-9. 61

5.3 The $(n+1)$ waypoints are connected by n line segments. Each line segment can be found by interpolation methods. 62

5.4 Two methods of waypoint navigation. 63

5.5 Path planning with weight by distance, W_D, in $(32 \times 32 \times 32)$ resolutions. . . . 67

5.6 Path planning with weight by volume, W_V, in $(32 \times 32 \times 32)$ resolutions. 69

5.7 Path planning with weight by area of sharing surface, W_S, in $(32 \times 32 \times 32)$ resolutions. 69

5.8 Path planning with weight by number of connections, W_N, in $(32 \times 32 \times 32)$ resolutions. 69

6.1 Pure pursuit navigation: the computed interception path at t_0. 72

6.2 Pure pursuit navigation: the final trajectories at t_I. 72

6.3 Pure proportional navigation: the final trajectories and the LOS at t_I. 76

6.4 Pursuit navigation on R-map: the computed interception path at t_0. 77

6.5 Pursuit navigation on R-map: the final trajectories at t_I. 77

6.6 Interception-triangle navigation on R-map: the computed interception path at t_0. 81

6.7 Interception-triangle navigation on R-map: the final trajectories at t_I. 81

6.8 Simulation 1: four target interception strategies. 84

6.9 Simulation 1: angles of the pursuer's heading. 85

6.10 Simulation 1: change of angles of the pursuer's heading. 85

A.1 Simulation 2: four target interception strategies. 94

A.2 Simulation 2: angles of the pursuer's heading. 95

A.3 Simulation 2: change of angles of the pursuer's heading. 95

A.4 Simulation 3: four target interception strategies. 96

A.5 Simulation 3: angles of the pursuer's heading. 97

A.6 Simulation 3: change of angles of the pursuer's heading. 97

List of Tables

2.1 Comparisons of mapping with different map representations. 17

2.2 Comparisons of mapping and path planning with different representations. . . . 20

3.1 The largest neighbors of rectangle-1 along each edge-direction. 24

4.1 Comparisons of mapping with different map representations. 55

5.1 Waypoints in figure 5.2a. 61

5.2 Comparisons of mapping with different map representations. 67

Chapter 1

Introduction

The guidance and navigation of mobile vehicles in complex environments often depend on a map of the environment. The most popular map representation is the occupancy grid map (G-map), which expresses obstacles with 0 and free spaces with 1. Mapping robots may generate G-maps with different resolutions. Lower-resolution maps consume less data, but the obstacles are distorted and path planning with the maps can produce unreliable paths. Higher-resolution maps that have increased resolutions by a factor of n can produce more reliable paths, but the map of a two-dimensional (2D) environment has n^2 times more cells to consider. Thus, high-resolution G-maps require large memory spaces and large computational times for processing [1, 2]. As a solution for the drawback of G-maps, multiresolution cell decomposition has been suggested in [3–7] to minimize the distortions of obstacles, thus the map has varied sizes of cells. Prazenica and Kurdila adopted multiresolution decomposition for obstacle-location estimation to use in receding horizon control formulation [8, 9]. As another solution, Ahn and Jeon introduced the concept of a rectangular map (R-map) as a hybrid of a G-map and a topological map in [10]. A topological map is a graph-based map, which only shows relationships between nodes by branches such as a subway map [11,12]. The idea of R-maps is integration of empty cells of a grid map into maximal empty rectangles (MERs), thus MERs are the nodes and their relationships are the branches. Therefore, R-maps have a reduced number of cell elements and connections, and they are computationally more efficient and require less memory space than G-maps for their utilization, such as in path planning. Chapter 2 reviews this previously developed 2D R-map. Section 2.1 describes how to construct a 2D R-map with a simple example, and Section 2.2 describes path planning on a 2D R-map of the Auburn University campus.

In Chapter 3, the 2D R-map is applied to map-merging problems. Since R-maps represent spaces with varied sizes of rectangles, this feature can be a good source to recognize certain locations on the maps, unlike regular gridded cells. Using this feature, R-maps allow the estimation of common areas between two local maps to merge them into a global map. In map-merging problems, the computational complexity depends on the sensor information available from the mapping robots [14–17]. When the robots know their compass/heading directions, real-time locations, etc., the complexity will be much reduced. Also, obstacle-detecting sensors such as laser, mono/stereo camera, sonar, and infrared sensors have different noise levels that affect the map accuracy [18]. In the case such information is unknown and only map images are given, the three differences (orientation, accuracy and scale) between the maps make the map merging challenging. Map merging using rectangular features finds the most similar areas between two local maps by minimizing the differences.

Further, this study advances the idea of the 2D R-map to three-dimensional (3D) spaces. Chapter 4 explains how to construct 3D R-maps with detailed steps. The process of 3D R-mapping is similar with the 2D R-mapping as integrating free cells into maximal empty areas to reduce the number of cell elements. However, since 3D environments have an additional dimension of the z-axis, the process of R-mapping will be slightly different and the integrated cells will be represented as cuboids (volumes) instead of rectangles (areas). Those maximal empty cuboids (MECs) are obstacle-free spaces and the algorithm of 3D R-map also provides connections of the MECs, thus UAVs can accomplish obstacle avoidance by moving through a sequence of MECs.

As an application of the 3D R-map, path planning is provided in Chapter 5. Using the information of MECs from R-mapping, Dijkstra's algorithm selects connected cuboids between two points, then two methods of interpolation determine paths to follow passing through the selected cuboids. In this Chapter, the target is assumed to be placed on a stationary point.

For the case of a moving target, Chapter 6 suggests several approaches to intercept the target. To avoid obstacles in a cluttered environment, the guidance laws can be implemented on the R-map in a receding horizon approach, which takes the advantage of the computationally efficient path planning.

The major contributions of this work are a data reduction and a feature extraction from a map, thus this map representation can be adopted for map-utilizing applications such as map merging and path planning.

Chapter 2

Review of 2D R-mapping and Path Planning

The concept of the R-map has introduced by Ahn and Jeon [10]. Later, the detailed algorithm and the demonstration of its efficiency in path planning have been provided by the author [13]. This Chapter reviews the algorithm of 2D R-mapping and path planning as follows. Section 2.1 describes the R-mapping algorithm with four steps. The process is explained with a simple environment, and the result is provided on a representative environment. Section 2.2 shows the use of R-maps in path planning. The paths between two points can be various depending on weight definitions. Also, the computation time for path planning along different resolutions of the maps is given and compared.

2.1 R-mapping

The main idea of the R-map is the integration of empty cells of the G-map into MERs. The key issue of the R-map algorithm is finding MERs in the form of non-overlapping rectangles. A flow chart of the algorithm which consists of four steps is shown in figure 2.1. First, it is assumed that a grid map is acquired from a mapping robot or other source. This map can be expressed as an $(m \times n)$ binary matrix, 0 for obstacles and 1 for free spaces. Here, unknown/unexplored areas can be considered as free spaces. The input to the algorithm is this binary matrix, G. In step 1, a matrix cm is defined based on G. In step 2, surveys to find an MER proceed in cm column-by-column along the vertical direction. The survey compares the current element and the previous element by four possibilities. In step 3, corresponding elements of the MER from step 2 in G are updated to zeros to ensure that the subsequent iterations only search the remaining free space. Then, step 1 through step 3 are repeated until all elements in G have been set to zeros. After the iterations, a

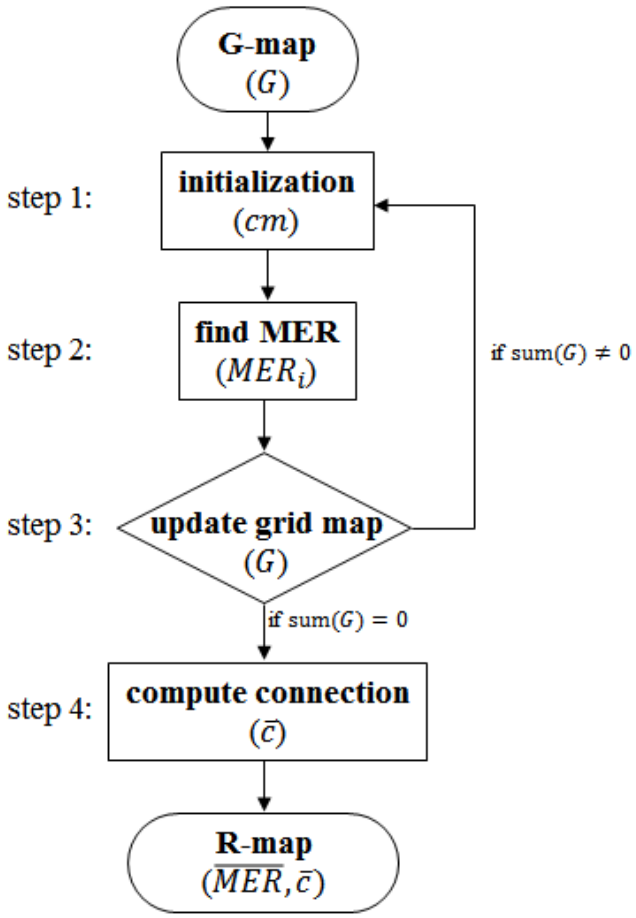
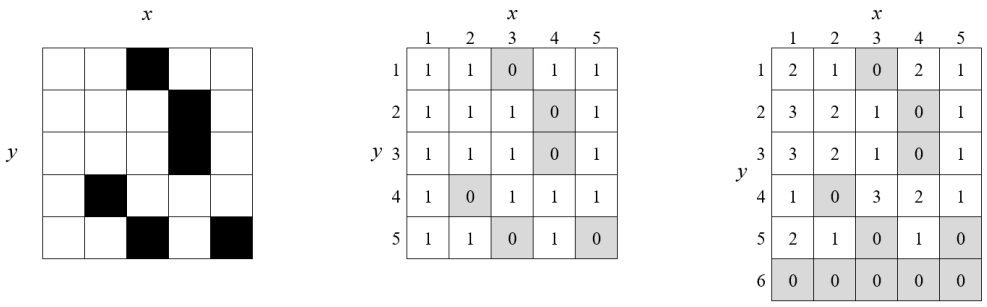


Figure 2.1. A flow chart of R-map algorithm.

list of MERs in the order of the area size is acquired. In step 4, connections of the MERs are computed. Finally, the algorithm returns a list of MERs and their connections. This information will be utilized in applications of the R-map. The following sections describe the four steps in more detail and provide results.

2.1.1 Step 1: Initialization

A G-map is defined as an $(m \times n)$ binary matrix G, where m is the total number of rows and n is the total number of columns. Thus, m and n are associated with the y- and x-axis, respectively, and the coordinates of each element in G are represented as $G(y, x)$. Figure 2.2a shows a G-map and figure 2.2b shows a (5×5) binary matrix G as an example.



(a) G-map (b) G (c) cm

Figure 2.2. (a) A (5×5) grid map is acquired from a mapping robot. (b) The grid map can be expressed as a binary matrix G. (c) A (6×5) corresponding matrix cm is derived from G.

A corresponding $((m+1) \times n)$ matrix cm is derived from this G. Each element of cm, $cm(y, x)$, is computed by right-to-left x-wise summations of contiguous free-space elements in G, thus the computing order of x is from n to 1. Also, an additional row with elements equal to zeros is added to the bottom of cm, which is necessary for step 2.

$$cm(y,x) = \begin{cases} G(y,x) & \text{if } y \le m,\ \ x = n \\ cm(y,x+1)+1 & \text{if } G(y,x) = 1,\ \ y \le m,\ \ x < n \\ 0 & \text{if } G(y,x) = 0,\ \ y \le m,\ \ x < n \\ 0 & \text{if } y = m+1 \end{cases} \tag{2.1}$$

Figure 2.2c shows a (6×5) matrix cm. Each element in cm is the number of free-elements along the x-direction, and indicates the width of a possible rectangle.

2.1.2 Step 2: Finding the MER

This step finds the maximal empty rectangle (MER) by surveying every element column-by-column in cm. The approach in this step is referred from Vandevoorde [19]. Initially, *best-rectangle* is defined as zero. The survey starts from the first row ($y = 1$) of the first column ($x = 1$), and proceeds through each row of a column before proceeding to the next column. At the first row of each column, $cm(1, x)$, the current element initializes a temporary

matrix, T, that contains the current row number and the value of the current element.

$$T = [1, cm(1, x)] \tag{2.2}$$

Proceeding to subsequent elements, the current element, $cm(y, x)$, and the previous element, $cm(y-1, x)$, are compared. If the value of the current element is greater than the value of the previous element, a new entry is added to T.

$$T := \begin{bmatrix} T \\ - - - - - - \\ y, cm(y, x) \end{bmatrix} \tag{2.3}$$

Here, := indicates updating the variable on the left-hand side with the value on the right-hand side. Each entry in T is a candidate MER, where the first element describes the starting row and the second element describes the width.

If the current element of cm is lesser than the previous element, areas of possible rectangles are calculated by each entry in T to find any larger rectangle than *best-rectangle*. The area is calculated by $(height \times width)$, where *height* is the number of elements from the row of the current surveying element (y) to the row of each entry in T (the first column of T) and *width* is the value of the current entry in T (the second column of T).

$$height = y - T(e, 1) \tag{2.4a}$$

$$width = T(e, 2) \tag{2.4b}$$

Here, e is an entry number. As areas are calculated, if there exists any larger rectangle than *best-rectangle*, the larger rectangle replaces *best-rectangle*. To summarize, there are four possibilities in the comparisons.

Case 2.1.1. $cm(y, x) = cm(y-1, x)$:

No changes are made to T.

Case 2.1.2. $cm(y,x) < cm(y-1,x)$:

Calculate areas of possible rectangles with each entry in T, and update *best-rectangle* if necessary. Also, as the areas are calculated, update any widths in T, $T(e,2)$, greater than the current width, $cm(y,x)$, to the current width.

$$\min p \equiv \min \{p | T(p,x) \geq cm(y,x)\} \tag{2.5a}$$

$$T(\min p, 2) := cm(y,x) \tag{2.5b}$$

And this updating leaves redundant entries in T, because rectangles made with any $T(p,q)$ for $p > \min p$ will be within a rectangle made with $T(\min p, q)$, thus any entries $T(p,q)$ for all $p > \min p$ are removed from T.

Case 2.1.3. $cm(y,x) > cm(y-1,x)$:

Add the current element to T.

Case 2.1.4. $cm(y,x) = 0$:

Calculate areas of possible rectangles with each entry in T, and update *best-rectangle* if necessary. After the area calculations, remove all elements from T and set $T = [\,\cdot\,]$.

The detailed process is illustrated here with cm shown in figure 2.2c. First, *best-rectangle* and T are defined.

$$\begin{gathered} \textit{best-rectangle} = 0 \\ T = [\,\cdot\,] \end{gathered} \tag{2.6}$$

The survey begins with the first element in the first column, whose coordinate is $cm(1,1)$. As mentioned, the first element is always added to T.

$$\text{At first element} : T = \begin{bmatrix} 1 & 2 \end{bmatrix} \tag{2.7}$$

The survey proceeds to the second element, $cm(2,1)$. Since the current element is greater than the previous element, it is case 2.1.3 and the information of the current element

is added to T.

$$\text{At second element} : cm(2,1) = 3 > cm(1,1) = 2$$

$$T = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \tag{2.8}$$

The third element, $cm(3,1)$, is the same with the previous element, thus case 2.1.1 is applied and no changes are made in T.

$$\text{At third element} : cm(3,1) = 3 = cm(2,1) = 3$$

$$T = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \tag{2.9}$$

The fourth element, $cm(4,1)$, is lesser than the previous element, and case 2.1.2 is applied. Areas of possible rectangles are calculated with T from equation 2.9 and compared.

$$\text{At fourth element} : cm(4,1) = 1 < cm(3,1) = 3 \tag{2.10}$$

Currently, T from equation 2.9 has two entries ($e = 1, 2$) and the parameter is $y = 4$. Thus, areas are calculated by equation 2.4.

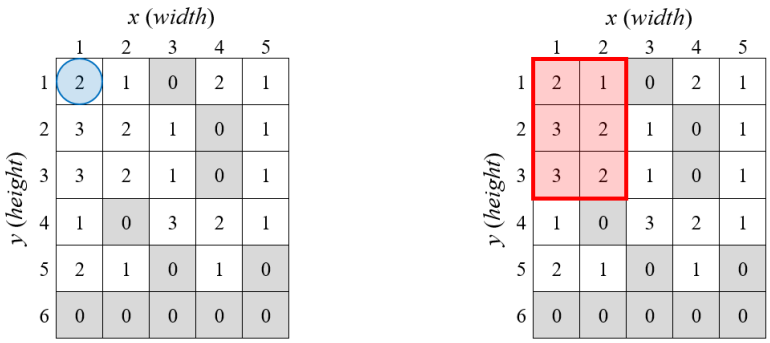
$$\text{for } e = 1,\ height = y - T(e,1) = 4 - 1 = 3 \tag{2.11a}$$

$$width = T(e,2) = 2 \tag{2.11b}$$

$$\text{rectangle-1} = height \times width = 3 \times 2 = 6 \tag{2.11c}$$

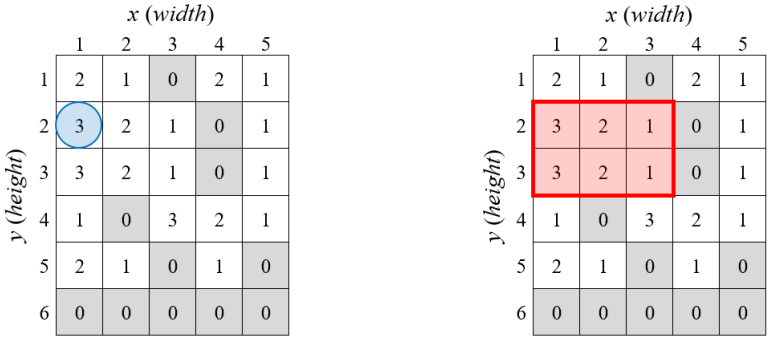
Figure 2.3a shows the first entry in T, and figure 2.3b shows rectangle-1. Rectangle-1, which has a larger area than *best-rectangle* replaces *best-rectangle*.

$$best\text{-}rectangle = \text{rectangle-1 (with area of 6)} \tag{2.12}$$



(a) first entry ($e = 1$) (b) rectangle-1

Figure 2.3. Rectangle-1 is found from the first entry in T.



(a) second entry ($e = 2$) (b) rectangle-2

Figure 2.4. Rectangle-2 is found from the second entry in T.

The area calculation is applied to the second entry in T.

$$\text{for } e = 2,\ \mathit{height} = y - T(e,1) = 4 - 2 = 2 \tag{2.13a}$$

$$\mathit{width} = T(e,2) = 3 \tag{2.13b}$$

$$\text{rectangle-2} = \mathit{height} \times \mathit{width} = 2 \times 3 = 6 \tag{2.13c}$$

Figure 2.4a shows the second entry in T, and figure 2.4b shows rectangle-2. Since rectangle-2 has the same area with rectangle-1, *best-rectangle* keeps rectangle-1. According to case 2.1.2, after area-calculations, any widths in T greater than the current width is updated to the current width. In T from equation 2.9, widths of both entries, the elements in the second column in T, are greater than the width of the current element, $cm(4,1) = 1$, thus $p = 1, 2$

and $\min p = 1$. Then, the elements in the second column are updated to $cm(4,1) = 1$, and the second entry is removed from T.

$$T = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \qquad \rightarrow \qquad T = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{2.14}$$

The survey proceeds to the fifth element, $cm(5,1)$. The current element is greater than the previous element, thus it is case 2.1.3 and the current element is added to T.

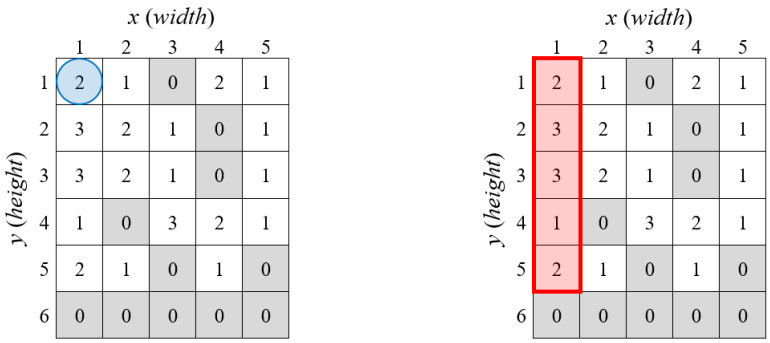
$$\text{At fifth element} : cm(5,1) = 2 > cm(4,1) = 1$$
$$T = \begin{bmatrix} 1 & 1 \\ 5 & 2 \end{bmatrix} \tag{2.15}$$

At the sixth element, $cm(6,1)$, the current element is zero, and it is case 2.1.4. Therefore, areas are calculated with T from equation 2.15. After area calculations, all elements will be removed from T.

$$\text{for } e = 1, \ height = y - T(e,1) = 6 - 1 = 5 \tag{2.16a}$$
$$width = T(e,2) = 1 \tag{2.16b}$$
$$\text{rectangle-3} = height \times width = 5 \times 1 = 5 \tag{2.16c}$$

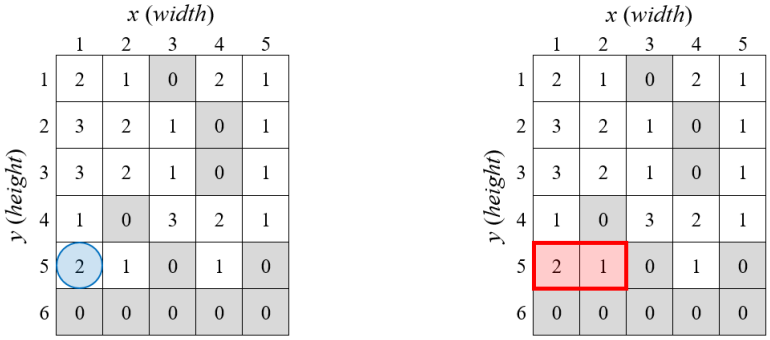
$$\text{for } e = 2, \ height = y - T(e,1) = 6 - 5 = 1 \tag{2.17a}$$
$$width = T(e,2) = 2 \tag{2.17b}$$
$$\text{rectangle-4} = height \times width = 1 \times 2 = 2 \tag{2.17c}$$

Figures 2.5 and 2.6 show each entry in T and rectangles-3 and -4, respectively. Since rectangle-1 is the largest rectangle, equation 2.12 is kept as *best-rectangle*.



(a) first entry ($e = 1$) (b) rectangle-3

Figure 2.5. Rectangle-3 is found from the first entry in T.



(a) second entry ($e = 2$) (b) rectangle-4

Figure 2.6. Rectangle-4 is found from the second entry in T.

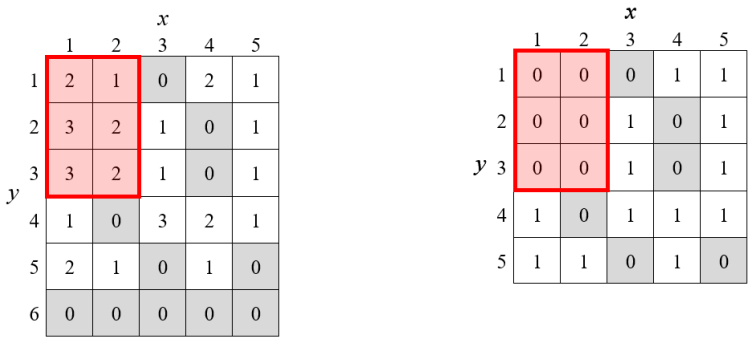
$$best\text{-}rectangle = \text{rectangle-1 (with area of 6)} \tag{2.18}$$

Also, all elements are removed from T.

$$T = [\,\cdot\,] \tag{2.19}$$

The survey proceeds to the second column. After surveying every element in cm in this manner, the final *best-rectangle* becomes an MER, and an MER in the i-th iteration is saved as

$$MER_i = [y_i, x_i, height_i, width_i] \tag{2.20}$$



(a) MER_1 in cm (b) MER_1 in G

Figure 2.7. (a) MER_1 is found from step 2. (b) Corresponding elements of MER_1 in G are updated to zeros. Steps 1 through 3 are repeated with the updated G until all elements have been set to zeros.

Here, y_i and x_i are the coordinates of the upper-left corner of MER_i. In the case of this example, rectangle-1 from equation 2.11 is the final *best-rectangle*, and it becomes an MER in this first iteration.

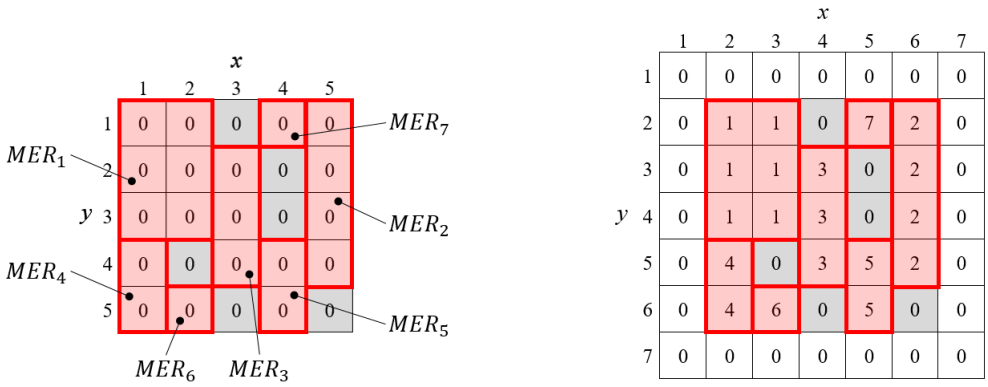
$$MER_1 = [1, 1, 3, 2] \tag{2.21}$$

Equation 2.21 indicates that MER_1 has its upper-left corner at $cm(1, 1)$ (or $G(1, 1)$) and the dimensions of (3×2).

2.1.3 Step 3: Grid Map Update

This step updates elements of the MER in G to zeros to consider the MER as an obstacle. For example, MER_1 from step 2 is shown in figure 2.7a, and its corresponding elements in G are updated to zeros as shown in figure 2.7b. This updating ensures that the subsequent iterations only search the remaining free space.

With the updated G, steps 1 through 3 are repeated in an iterative fashion until all elements in G have been set to zeros or until a desired threshold is reached in terms of certain size or number of MERs, related to the desired R-map resolution. In the case of this example, the iteration returns 7 MERs, and all elements in G have been set to zeros as shown in figure 2.8a. A list of MERs is acquired in the order of sizes, from the biggest to



(a) 7 MERs in G (b) Labeled-G

Figure 2.8. (a) After iterations, all elements in G have been set to zeros, and 7 MERs are found. (b) Elements of each MER are updated to their rank, and additional elements equal to zero are added around G, thus the dimension becomes (7×7).

the smallest.

$$MER_1 = [1, 1, 3, 2] \tag{2.22a}$$

$$MER_2 = [1, 5, 4, 1] \tag{2.22b}$$

$$MER_3 = [2, 3, 3, 1] \tag{2.22c}$$

$$MER_4 = [4, 1, 2, 1] \tag{2.22d}$$

$$MER_5 = [4, 4, 2, 1] \tag{2.22e}$$

$$MER_6 = [5, 2, 1, 1] \tag{2.22f}$$

$$MER_7 = [1, 4, 1, 1] \tag{2.22g}$$

When all elements in G have been set to zeros as shown in figure 2.8a, it proceeds to the next step.

2.1.4 Step 4: Computing Connection

This step finds the connections of MERs. To find what MERs are connected to each MER, elements of each MER will be labeled as their MER's rank in the manner of updating elements of MER_i to i. Also, additional elements equal to zero are added around G, thus

G has a dimension of $((m+2) \times (n+2))$. By doing so, neighboring elements of an MER indicate connecting MERs. As the dimension of G has changed, the coordinates of MERs also should be changed.

$$MER_i^G = [y_i + 1,\, x_i + 1,\, height_i,\, width_i] \tag{2.23a}$$

$$= [Y_i,\, X_i,\, height_i,\, width_i] \tag{2.23b}$$

Here, X_i and Y_i are the coordinates of the upper-left corner of MER_i in the labeled-G as shown in figure 2.8b. From the labeled-G, connections of an MER on each of the four sides are obtained by listing the unique neighboring-values on each side, except zeros.

$$left_i = u[G(Y_i : (Y_i + height_i - 1),\ \ X_i - 1)] \tag{2.24a}$$

$$right_i = u[G(Y_i : (Y_i + height_i - 1),\ \ X_i + width_i)] \tag{2.24b}$$

$$upper_i = u[G(Y_i - 1,\ \ X_i : (X_i + width_i - 1))] \tag{2.24c}$$

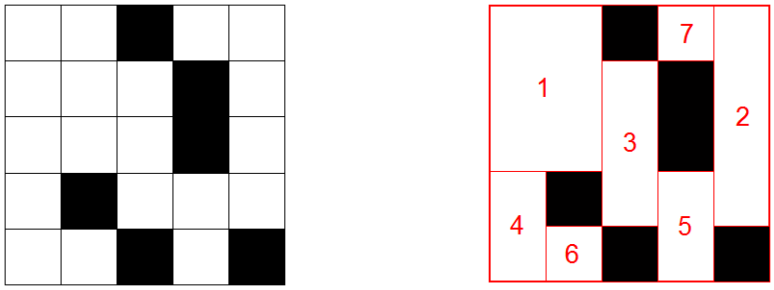
$$lower_i = u[G(Y_i + height_i,\ \ X_i : (X_i + width_i - 1))] \tag{2.24d}$$

Here, u means the unique values except zeros, and $[A : B]$ means a list of continuous numbers from A to B, for example, $[2 : 4] = [2, 3, 4]$. A complete list of connections of MER_i is obtained by applying u to all elements from equation 2.24.

$$c_i = u[left_i, right_i, upper_i, lower_i] \tag{2.25}$$

Continuing the example, MER_1 from equation 2.22a will have new coordinates for the labeled-G.

$$MER_1^G = [2, 2, 3, 2] \tag{2.26}$$



(a) G-map (b) R-map

Figure 2.9. (a) G-map has 19 free-elements (empty cells) and 42 connections. (b) R-map has 7 free-elements (MERs) and 12 connections.

And the connections of MER_1 on each side are computed from figure 2.8b using equation 2.24.

$$left_1 = u[G(2 : (2+3-1),\ \ 2-1)] = u[0,0,0] = [\ \cdot\ ] \tag{2.27a}$$

$$right_1 = u[G(2 : (2+3-1),\ \ 2+2)] = u[0,3,3] = [3] \tag{2.27b}$$

$$upper_1 = u[G(2-1,\ \ 2 : (2+2-1))] = u[0,0] = [\ \cdot\ ] \tag{2.27c}$$

$$lower_1 = u[G(2+3,\ \ 2 : (2+2-1))] = u[4,0] = [4] \tag{2.27d}$$

Finally, a complete list of the connections of MER_1 is obtained by equation 2.25. Also, the connections of the rest MERs are computed in the same manner.

$$c_1 = [3,4] \tag{2.28a}$$

$$c_2 = [5,7] \tag{2.28b}$$

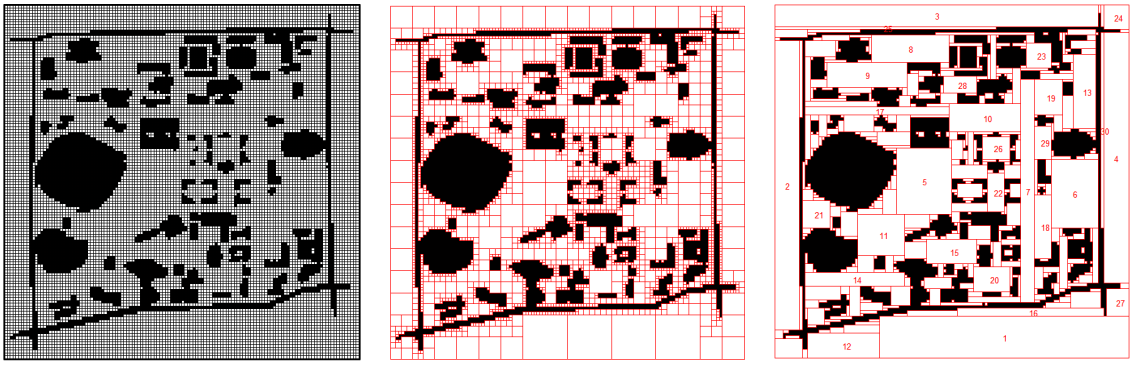
$$c_3 = [1,5] \tag{2.28c}$$

$$c_4 = [1,6] \tag{2.28d}$$

$$c_5 = [2,3] \tag{2.28e}$$

$$c_6 = [4] \tag{2.28f}$$

$$c_7 = [2] \tag{2.28g}$$



(a) G-map (b) Q-map (c) R-map

Figure 2.10. The Auburn University campus map in three different map representations.

Table 2.1. Comparisons of mapping with different map representations.

Resolutions	**128×128**			**256×256**			**512×512**		
Map type	**G**	**Q**	**R**	**G**	**Q**	**R**	**G**	**Q**	**R**
free elements	12,304	2,488	472	49,273	5,752	874	198,368	12,389	1,710
connections	45,640	8,728	1,556	189,712	21,412	3,086	778,384	47,568	6,170
mapping time (sec.)	-	0.84	1.98	-	2.09	10.46	-	5.66	63.80

Therefore, the R-map algorithm returns MERs and their connections such as equations 2.22 and 2.28. Figure 2.9 shows the initial G-map and the final R-map.

2.1.5 Result

As an example of a complex environment, the Auburn University campus map with a resolution of (128×128) is shown in figure 2.10 in three different map representations. First, as mentioned, a G-map is a map from a mapping robot, and it has equal-sized elements. Second, a Q-map (quadtree-decomposition map) is derived from a G-map, and it subdivides a space into four quadrants if the space needs higher resolutions [38]. Thus, the cell elements in Q-maps are in squares but can be in different sizes. Third, an R-map is similar with a Q-map as reducing the number of cell elements, however while Q-maps subdivide a space, R-maps integrate empty cells into maximal-sized rectangles. Thus, R-maps can have elements in various sizes and shapes of rectangles instead of squares, and much fewer number of elements than G-map and Q-map. Table 2.1 compares the three maps with three different resolutions. R-map has a dramatically reduced number of elements and connections, and

the difference with the other two maps gets bigger as the resolution gets higher. However, the R-map requires an investment of computational time to construct. Notice that the Matlab code for the Q-map uses a Matlab built-in function, *qtdecomp*. Also, the code for the R-map is not optimized, and the R-mapping time would be different depending on the implementation of the algorithm in the code.

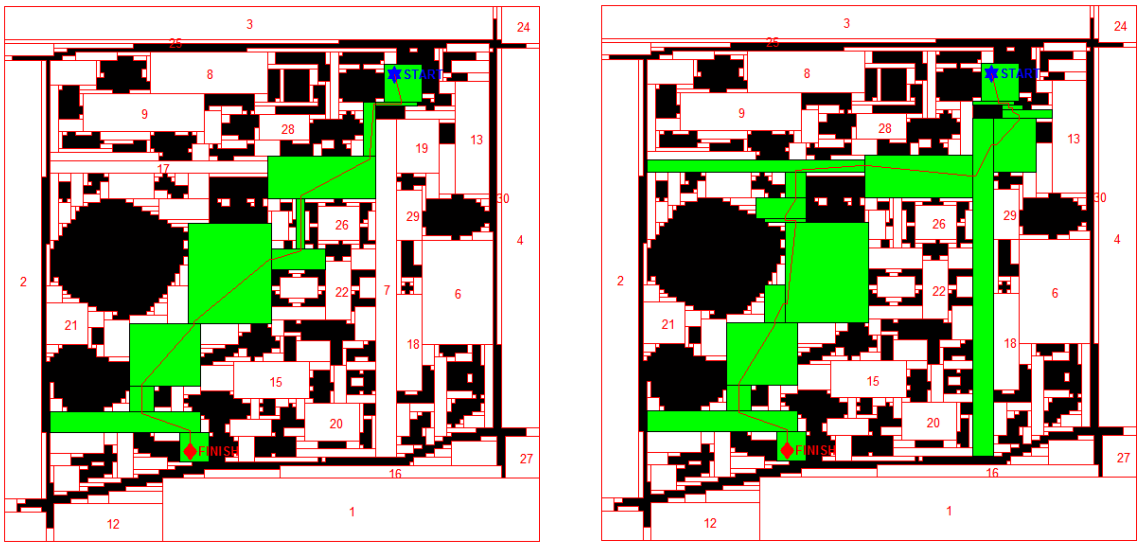
2.2 Path Planning

The path planning problem for a mobile robot avoiding obstacles has been studied for years [20–22]. This path planning on the R-map can be another approach for obstacle avoidance. Because R-maps provide a reduced-element map representation focused on maximal-sized obstacle-free spaces, it appears to be naturally suited for path-planning problems. For path planning with R-maps, Dijkstra's algorithm [23] is applied. This cost-based algorithm takes nodes and edges to calculate the optimal path between any two points on a map. When the algorithm is applied to an R-map, MERs are considered as nodes and their connections are considered as edges. Then the algorithm will select the lowest cost path computing weights between two points. In this study, four different weight definitions are considered. In each definition, $W(j,k)$ is a cost to travel from MER_j to MER_k.

Definition 2.2.1. $W_A(j,k) = k$. Weight by area. Since the R-map algorithm produces a ranked list from the largest rectangle to the smallest rectangle, the list of MERs is in the order of area. Thus, by simply giving a lower weight to a higher-ranked rectangle, their weights are set by area and the algorithm selects a path of larger area.

Definition 2.2.2. $W_L(j,k) =$ (border length between MER_j and MER_k). Weight by border length. By giving a lower weight to an MER which has a longer connected-border length, the algorithm selects a wider path.

Definition 2.2.3. $W_D(j,k) =$ (distance between the center points of MER_j and MER_k). Weight by distance. By giving a lower weight to closer center points of two rectangles,



(a) Weight by area, W_A

(b) Weight by border length, W_L

(c) Weight by distance, W_D

(d) Weight by number of connections, W_N

Figure 2.11. Path planning with the four different weight definitions. The path starts at the start point (blue star) and ends at the finish point (red diamond) through the sequence of MERs (green rectangles).

the algorithm selects a shorter path to the destination, a classic application of Dijkstra's algorithm.

Definition 2.2.4. $W_N(j,k) = 1/(\text{the dimension of } c_k)$. Weight by number of connections. To give a lower weight to a rectangle which has more potential paths to the next rectangle,

(a) G-map (b) Q-map (c) R-map

Figure 2.12. Path planning with different map representations.

Table 2.2. Comparisons of mapping and path planning with different representations.

Resolutions	**128×128**			**256×256**			**512×512**		
Map type	**G**	**Q**	**R**	**G**	**Q**	**R**	**G**	**Q**	**R**
free elements	12,304	2,488	472	49,273	5,752	874	198,368	12,389	1,710
connections	45,640	8,728	1,556	189,712	21,412	3,086	778,384	47,568	6,170
mapping time (sec.)	-	0.84	1.98	-	2.09	10.46	-	5.66	63.80
path planning time (sec.)	1.53	1.17	0.28	17.64	2.72	0.49	284.99	6.09	0.91

find the number of connections of each rectangle. Thus, the result is a path that has diverse alternative paths.

Note that $W_L(j,k) = W_L(k,j)$ and $W_D(j,k) = W_D(k,j)$, but $W_A(j,k) \neq W_A(k,j)$ and $W_N(j,k) \neq W_N(k,j)$ (cost going is not the same as cost returning). Through Dijkstra's algorithm, a sequence of MERs is selected between start and finish points. To define a precise path on the map, particular points within each selected MER are defined as waypoints. Each MER has two waypoints: entering and exiting. Both waypoints are selected to cross the center of connected-border of the previous and subsequent MERs. Examples of path planning with the four different definitions on the (128 × 128) Auburn University campus map are demonstrated in figure 2.11. The four paths are different depending on their weight parameters, but some of the paths are similar, because rectangles with larger area also tend to have longer border length and more connections.

For comparisons, the path planning is also performed on G-map and Q-map with the weight by distance as shown in figure 2.12. The G-map takes weights of 1 for any two free

cells that share a side, and the Q-map and the R-map take W_D from definition 5.1.3. Table 2.2 specifies values for three different map representations. From the table, the R-map has much fewer elements and connections to compute a path than the other maps. Consequently, the computation time for path planning with the R-map becomes faster than with the other maps as the resolution rises. Furthermore, paths from Q-maps or R-maps can be shorter (not always) than G-maps, because paths from G-maps are always horizontal and vertical segments and this staircase-path can be longer than diagonal paths from Q-maps or R-maps. However, considering the total time of mapping and path planning, R-map is proper and efficient in the case of multiple path planning on a complete map such as repeated path planning missions in an environment that has permanent obstacles.

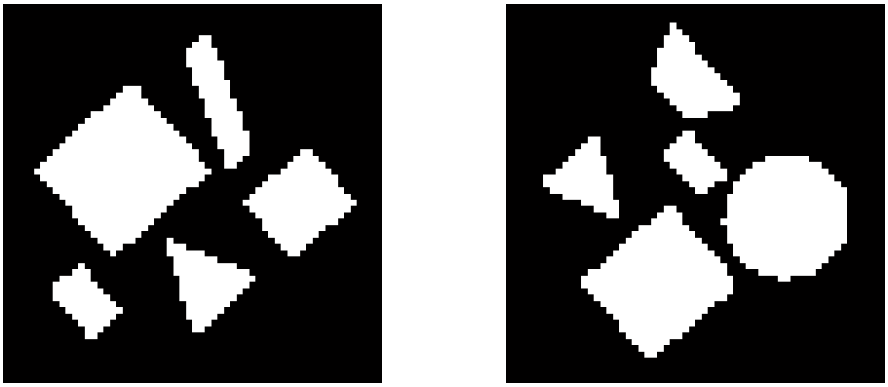
Chapter 3

Map Merging

This chapter presents a new approach to map merging as another application of the R-map. Map merging requires matching of features in the two maps. In this study, the maps have three unknown factors: orientation, accuracy, and scale. Due to the possible variations in those factors across the maps, it is desirable for the features to be scale and orientation invariant. In the case of such problems, the rectangular features from R-maps can be useful to find the best-shared rectangles from both maps. However, to achieve scale and orientation invariance, matching of individual rectangles is not possible. Therefore, the approach used here will match sets of three rectangles from each R-map. A set of three rectangles implicitly defines a triangle. Based on recognizing matched features in the two maps, a translation, rotation, and scaling transformation between the maps can be computed. In the following section, a method is described to reduce the distortions in the R-maps generated by variations in orientation. This method also has the potential to remove the requirement for orientation invariant features. Then, the approach for matching triangles is described in detail. The process is explained with simple maps as shown in figure 3.1, and the result with practical maps from a mapping robot is presented in section 3.4.

3.1 Orthogonal Orientations

Human made environments tend to have features aligned along orthogonal angles. R-maps and G-maps are also defined along orthogonal directions. Therefore, it is desirable to use an R-map where the orthogonal angles that may exist within the environment have been aligned with the orthogonal axes of the G-map. This allows areas in the environment to be better consolidated within rectangles of the R-map, and improves the possible matches

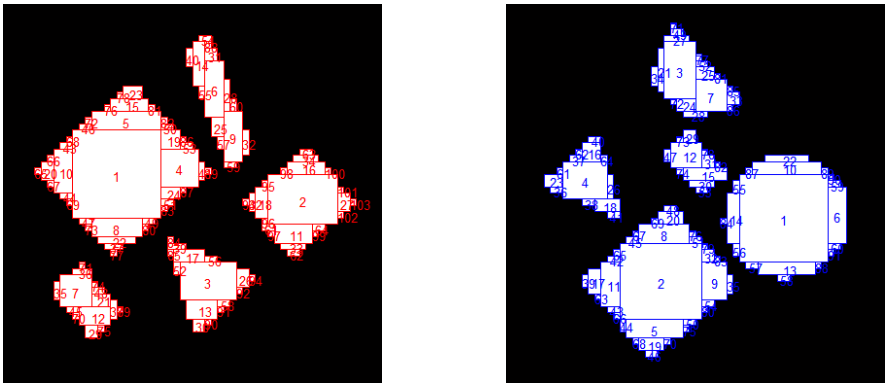


(a) map 1 (b) map 2

Figure 3.1. Two local maps of a simple environment. The maps have naturally different accuracy and scales to each other, and intentionally rotated.

between R-maps. Performing this alignment also removes the need to solve for the orientation transformation between R-maps. Instead, only four possible 90 degree orientation transformations exist, and each possibility can be directly investigated.

The R-map algorithm is applied to figure 3.1 to extract rectangular features (MERs). Figure 3.2 shows the result of R-mapping. Here, map 1 has 103 rectangles, and map 2 has 91 rectangles. Also, their connections on each side are acquired. Those rectangles of a given area can appear quite different depending on the orientation of the maps. For example, in figure 3.2, rectangle-1 in map 1 corresponds to some of the same area as rectangle-2 in map 2. But, the rectangles are significantly different size due to the different orientations of the two maps. In this sense, R-maps have preferred directions: the orthogonal directions along which they search for the height and width of the rectangles. This has two advantages: first, empty spaces can be more efficiently filled with fewer rectangles and second, solving for the relative orientation between the maps has been reduced to four possibilities which are rotations of multiples of 90 degrees. Aligning the environment with the map grids proceeds with two steps. First, points on the edges of the empty spaces are found by sorting the neighboring rectangles. Second, the angles of those edges are found by random sample consensus (RANSAC) [24].



(a) map 1 (b) map 2

Figure 3.2. Results from the R-map algorithm. Map 1 and map 2 have 103 and 91 rectangles, respectively.

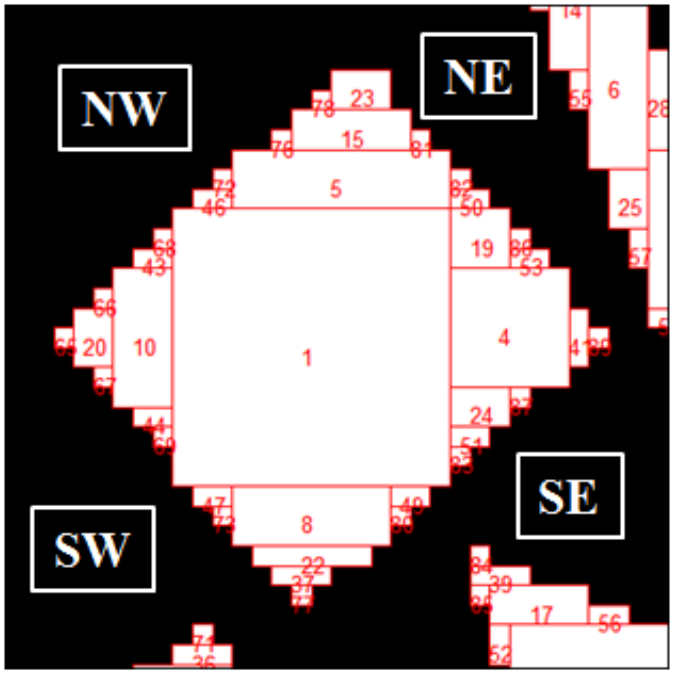


Figure 3.3. Zoomed in image of the top-left portion of figure 3.2a. A series of the largest neighbors along each edge-direction divides which rectangles are on which side of the environment.

direction	1st	2nd	3rd	4th
upper	5	15	23	·
lower	8	22	37	77
left	10	20	65	·
right	4	41	89	·

Table 3.1. The largest neighbors of rectangle-1 along each edge-direction.

The points on each edge are sorted by recognizing the largest neighboring rectangle along each edge of a starting rectangle. A series of the largest neighbors along each edge-direction divides which rectangles are on which side of the environment. Figure 3.3 is the

top-left portion of the environment of figure 3.2a. Table 3.1 shows the series of the largest neighbors to rectangle-1 in each edge-direction. Searching a series of the largest neighbors stops when there are no further connections on any direction. For example, the largest rectangles in the upper-direction are 5, 15 and 23. Those rectangles divide their neighbors: their left neighbors are on NW-edge, and their right neighbors are on NE-edge, and those neighbors are denoted as upper-left and upper-right connections of rectangle-1. Since the series of the largest neighbors to upper-direction ends at the third connection, the series to the other directions also take up to their third connections. The lists of edge-touching rectangles on the eight directions from rectangle-1 are shown below.

$$c_{1UL} = \{5, 15, 23, 46, 72, 76, 78\} \tag{3.1a}$$

$$c_{1UR} = \{5, 15, 23, 50, 82, 81\} \tag{3.1b}$$

$$c_{1LU} = \{10, 20, 43, 65, 66, 68\} \tag{3.1c}$$

$$c_{1LD} = \{10, 20, 44, 65, 67, 69\} \tag{3.1d}$$

$$c_{1RU} = \{4, 19, 41, 50, 53, 82, 86, 89\} \tag{3.1e}$$

$$c_{1RD} = \{4, 24, 41, 51, 83, 87, 89\} \tag{3.1f}$$

$$c_{1DL} = \{8, 22, 37, 47, 73, 77\} \tag{3.1g}$$

$$c_{1DR} = \{8, 22, 37, 49, 77, 80\} \tag{3.1h}$$

Here, c_1 is the connections of rectangle-1, and the subscripts U, L, R, and D are the directions to upper-, left-, right-, and lower-side, respectively, thus c_{1UL}, for example, means the connections on the upper-left side of rectangle-1. As it can be seen in figure 3.3, rectangles on two directions cover one edge of the environment: c_{1UL} and c_{1LU} for NW-edge, c_{1UR} and c_{1RU} for NE-edge, c_{1DL} and c_{1LD} for SW-edge, and c_{1DR} and c_{1RD} for SE-edge. Also, the largest rectangle in the environment, rectangle-1, belongs to the edge-touching rectangles for each edge. Thus in general, the list of rectangles on each edge of an empty space having the

largest rectangle i can be written as

$$NW_i = \{i, c_{iUL}, c_{iLU}\} \tag{3.2a}$$

$$NE_i = \{i, c_{iUR}, c_{iRU}\} \tag{3.2b}$$

$$SW_i = \{i, c_{iDL}, c_{iLD}\} \tag{3.2c}$$

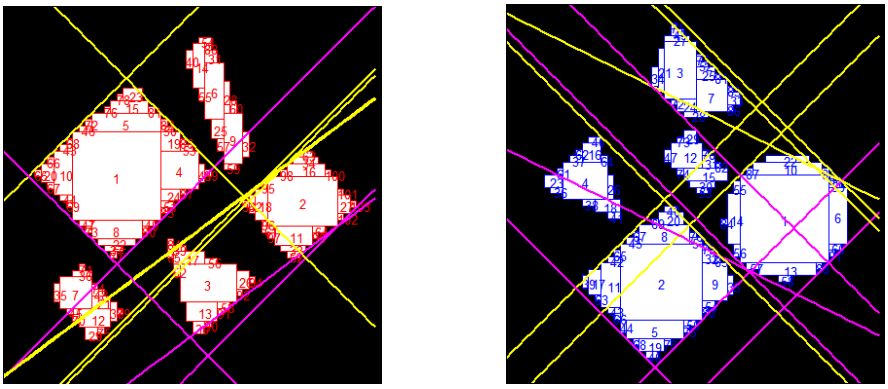
$$SE_i = \{i, c_{iDR}, c_{iRD}\} \tag{3.2d}$$

Because the rectangles in an R-map are of maximal sizes, larger rectangles are located closer to the center of an empty space. Thus, equation 3.2 should be applied to larger rectangles, $i = 1, \cdots, N$, to extract rectangles on the edges of empty spaces, where N is a user-selected variable.

Next, angles of the edges are computed, with the goal of aligning the edges with the R-map axes. Two rectangles from each edge can make a line segment. Note that the coordinates of an edge-touching vertex of a rectangle depend on the four edges in equation 3.2, and are easily obtained using the properties of the rectangles: (y, x) for NW-edge, $(y, x + w)$ for NE-edge, $(y + h, x)$ for SW-edge and $(y + h, x + w)$ for SE-edge, where y, x are the upper-left coordinates and h, w are the height and width of a rectangle. Combinations of two rectangles from each given set of equation 3.2 make all possible line segments. The number of two-combinations is denoted as

$${}_{n_{edge}}\mathrm{C}_2 = \binom{n_{edge}}{2} = \frac{n_{edge}!}{2!\,(n_{edge} - 2)!} \tag{3.3}$$

where, n_{edge} is the number of elements in a corresponding edge in equation 3.2. For example, according to equation 3.2a, NW_1 has 13 elements ($n_{NW} = 13$), and equation 3.3 indicates that there are 78 line segments on NW-edge. With those lines, to count the number of line segments with similar slopes, the idea of RANSAC is applied here. Each line segment can make a region by defining a tolerance, $\pm\tau$, in the y-axis. A data value within the region is

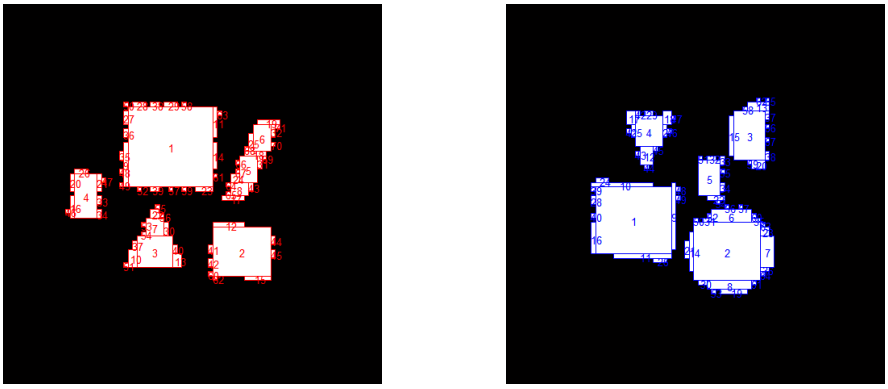


(a) map 1 (b) map 2

Figure 3.4. Pairs of parallel lines: one yellow and one pink lines are a pair. The true rotation angles are $\Lambda_0^{(1)} = 45°$ and $\Lambda_0^{(2)} = 225°$ counterclockwise, and estimated angles are $\Lambda^{(1)} = -45°$ and $\Lambda^{(2)} = -45°$ for map 1 and map 2, respectively.

noted as inlier. This process is applied to the four edges. For each line segment, if there is a line segment with the same slope on the facing-edges (NW-SE and NE-SW), the two parallel lines are selected as a valid pair. The example in figure 3.4 illustrates the valid lines: one yellow and one pink lines are a pair of parallel lines. Finally, the slope of a line segment with the most inliers becomes the rotation angle, Λ, to place a map in the orthogonal orientation. The true initial rotation angles of figure 3.4a and 3.4b are $\Lambda_0^{(1)} = 45°$ and $\Lambda_0^{(2)} = 225°$ counterclockwise, and the algorithm returns calculated rotation angles of $\Lambda^{(1)} = -45°$ and $\Lambda^{(2)} = -45°$, respectively. Therefore, maps with the obstacle edges aligned with the R-map axes are acquired by rotating the initial maps by the estimated angles as shown in figure 3.5: $\Lambda_0^{(1)} + \Lambda^{(1)} = 0°$ and $\Lambda_0^{(2)} + \Lambda^{(2)} = 180°$ for the map 1 and map 2, respectively.

When the edges of empty spaces are not aligned with the map grids, the R-map algorithm returns smaller rectangles to fill out areas near edges. However, when the spaces are in the orthogonal orientations so their edges are aligned with the map grids, the spaces can be fitted with fewer and larger rectangles (the spaces may have micro-rectangles around larger rectangles due to distortional uneven edges). As shown in figure 3.5, once the empty spaces are in the orthogonal orientations, rectangles can be kept in the same dimensions while rotating if the rotation angles are 90°-angles (this is an advantage of using the maximal size of rectangles). Since both maps in figure 3.5 are in the orthogonal orientations



(a) map 1 (b) map 2

Figure 3.5. Results from orthogonal-orientations algorithm. Orthogonally oriented maps are acquired through rotating the initial maps by estimated angles: $\Lambda_0^{(1)} + \Lambda^{(1)} = 0°$ and $\Lambda_0^{(2)} + \Lambda^{(2)} = 180°$ for the map 1 and map 2, respectively.

but the orientation-matching is unknown, an effort is required to select one from the four of 90°-rotation angles to match their orientations.

3.2 Shared Triangles

To perform the map merging, features in each map need to be matched. To perform the matching, feature vectors from each map are defined. Also, a norm of the difference between a pair of feature vectors is defined as a cost. Then, a match is defined as the pair of vectors that provide the lowest cost. In order to match maps that vary in scale and orientation, the feature vector must be invariant to these transformations. The method described in the previous section, however, aligns the map to within four possible 90 degree orientation transformations, and this eliminates the need to solve for the orientation transformation from matched features.

As the specific process, first, corresponding rectangles across the maps are collected. Since it is assumed that two maps have some common spaces, it is also assumed that each rectangle in map 1 has a corresponding rectangle in map 2, which has the same dimension. However, because the maps have distortions and different scales, a tolerance, δ, is allowed in defining corresponding rectangles. Due to the tolerance, each rectangle in map 1 will

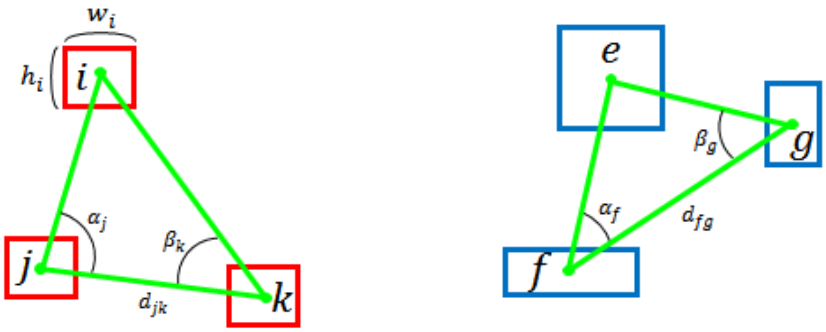
have multiple corresponding rectangles in map 2 that have the same dimension within the tolerance.

$$r_i^{(1)} = \left\{ r^{(2)} \middle| \text{areas between } ((w_i^{(1)} - \delta) \times (h_i^{(1)} - \delta)) \text{ and } ((w_i^{(1)} + \delta) \times (h_i^{(1)} + \delta)) \right\} \quad (3.4)$$

Equation 3.4 means that the i-th rectangle in map 1, $r_i^{(1)}$, with a dimension of $(w_i^{(1)} \times h_i^{(1)})$ corresponds to every rectangles in map 2 smaller or larger than $r_i^{(1)}$ by within the tolerance, δ. Accordingly, to make a triangle having vertices of the center points of three rectangles, equation 3.4 is evaluated for three rectangles in map 1. Therefore, one triangle in map 1 can be compared to multiple corresponding triangles in map 2 made by combinations of vertices from each list. The total number of possible triangles in map 1 is the number of combinations of rectangles in map 1.

$$_{n_r^{(1)}}\mathrm{C}_3 = \binom{n_r^{(1)}}{3} = \frac{n_r^{(1)}!}{3!\,(n_r^{(1)} - 3)!} \quad (3.5)$$

Here, $n_r^{(1)}$ is the total number of rectangles in map 1. For example, figure 3.5a has 71 rectangles, and according to equation 3.5, there are $57,155$ possible triangles in map 1. However, because micro-rectangles near the edges of the environment have non-remarkable features, the computational burden can be reduced by eliminating small rectangles. If rectangles with area less than 10 are eliminated from the list of rectangles in map 1, the number of possible triangles will be reduced to 56, also the number of corresponding triangles in map 2 will be reduced. One triangle in map 1 and one of its corresponding triangles in map 2 are a set of shared-triangles. The rectangular and triangular features are extracted from the set and compared to find the most similar set. Figure 3.6 illustrates a set of triangles made of $\{r_i, r_j, r_k\}$ and $\{r_e, r_f, r_g\}$. To find the most similar rectangles not only in their dimensions but also in their formations, the area of each rectangle, $(w \times h)$, and two vertex-angles, α and β, are adopted as the features. Also, the areas are normalized for comparisons by the



(a) map 1 (b) map 2

Figure 3.6. A set of shared-triangles. The feature vector, FV, contains triangular and rectangular features. (a) A triangle is made by the center points of three rectangles, $\{r_i, r_j, r_k\}$, in map 1. This triangle has multiple corresponding triangles in map 2. (b) One of corresponding triangles is made of rectangles, $\{r_e, r_f, r_g\}$, in map 2.

distance between the second and the third rectangles, d. Each triangle from map 1 and map 2 has a feature vector, FV.

$$FV^{(1)} = \begin{bmatrix} (w_i \times h_i)/d_{jk} \\ (w_j \times h_j)/d_{jk} \\ (w_k \times h_k)/d_{jk} \\ \alpha_j \\ \beta_k \end{bmatrix}, \qquad FV^{(2)} = \begin{bmatrix} (w_e \times h_e)/d_{fg} \\ (w_f \times h_f)/d_{fg} \\ (w_g \times h_g)/d_{fg} \\ \alpha_f \\ \beta_g \end{bmatrix} \tag{3.6}$$

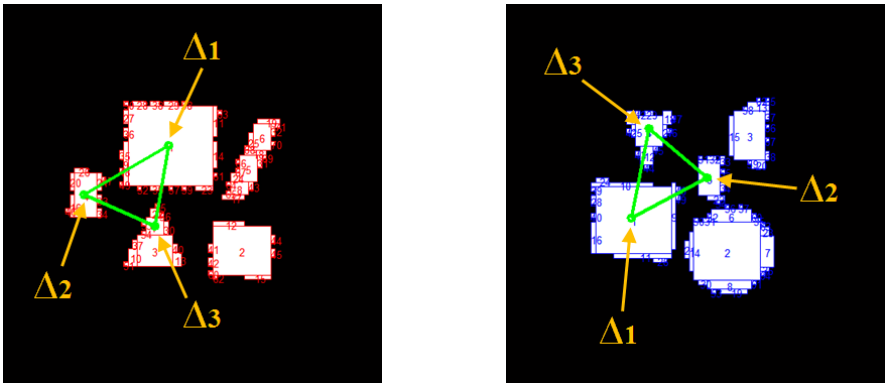
A cost function, J, with the feature vectors is defined.

$$J = \left|FV^{(1)} - FV^{(2)}\right|^T \cdot W \cdot \left|FV^{(1)} - FV^{(2)}\right| \tag{3.7}$$

Here, W is a weight matrix to give weights to desired features. In this example, an identity matrix is applied. The minimum cost refers a set of the best shared-triangles.

$$\Delta^{(1)} = \{r_i, r_j, r_k\}_{\min(J)} \tag{3.8a}$$

$$\Delta^{(2)} = \{r_e, r_f, r_g\}_{\min(J)} \tag{3.8b}$$



(a) map 1 (b) map 2

Figure 3.7. Results of shared-triangles algorithm. A set of the best shared-triangles has $\min(J)$. The Δ_1's and Δ_2's in both maps are the correct shared-rectangles, but Δ_3's are incorrect due to different map scales.

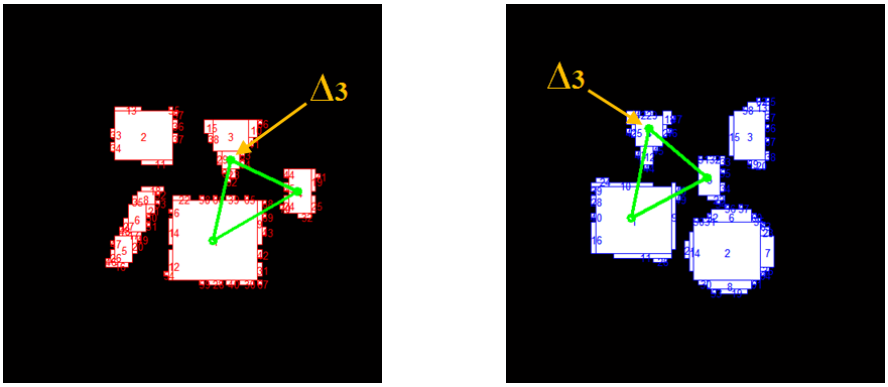
Here, $\Delta^{(1)}$ and $\Delta^{(2)}$ denote the best shared-triangles in map 1 and map 2, respectively, and contain rectangles in the order of size. Also, each element of Δ is denoted as a subscript, for example, $\Delta_1^{(1)}$ means the first (the largest) rectangle of a triangle in map 1. Figure 3.7 shows $\Delta^{(1)}$ and $\Delta^{(2)}$ with the green triangles. In this figure, the first and the second shared-rectangles are selected correctly ($\Delta_1^{(1)} = \Delta_1^{(2)}$ and $\Delta_2^{(1)} = \Delta_2^{(2)}$), but the last shared-rectangles in the triangular environment in both maps are not the same ($\Delta_3^{(1)} \neq \Delta_3^{(2)}$). It occurs because of the difference in map scales. Even though maps are in different scales, it has been observed that the algorithm usually selects the correct Δ_1's and Δ_2's in both maps, but not for the last rectangles, Δ_3's. This is because Δ_1's and Δ_2's are larger ones even among all rectangles across map environments, and usually they are separated to each other with some distance (even two neighboring larger rectangles have some distance between their center points due to their dimensions). However, Δ_3's are smaller ones, and similar rectangles can exist across the map. Therefore, $\Delta^{(1)}$ and $\Delta^{(2)}$ can have Δ_3's of similar dimensions but at slightly different locations. The location error is limited because FV contains features for the triangular-shape matching. The Δ_3's in map 1 and map 2 will be matched better as the map scale is matched in the next section and this process is repeated with a re-scaled map.

3.3 Scale matching and Merging

This step finally matches the orientations and scales between two maps using Δ's. Because Δ's are the common triangles in both maps, matching them yields the same orientation. The final rotation angle, Λ_f, is obtained by matching bisector-vectors in both maps. The bisector-vector is the vector from Δ_1 to the midpoint between Δ_2 and Δ_3 in each map. Because the maps are already in the orthogonal orientations, the final rotation angle should be one of the 90°-angles. From figure 3.7, $\Lambda_f = 180°$. Rotating map 1 by Λ_f matches the orientations of map 1 and map 2 as shown in figure 3.8. However, Δ_3's in both maps are still incorrect shared-rectangles because of the different scales. For the scale matching, as discussed in the previous section, Δ_1's and Δ_2's are mostly the correct shared-rectangles, thus matching a dimension of Δ_1's yields the same scale. A scale ratio is defined as

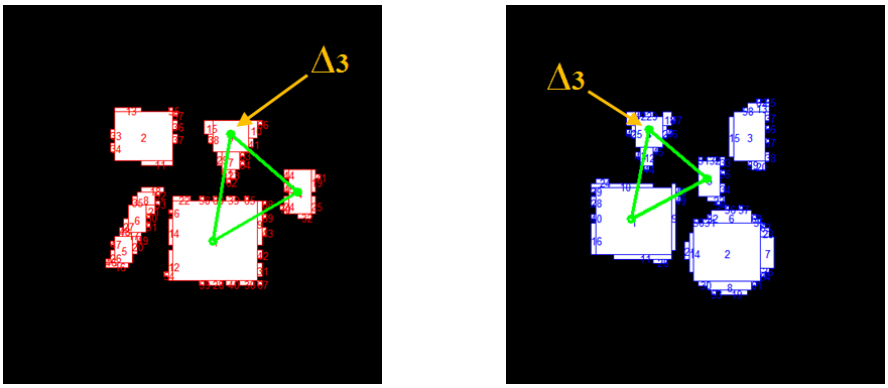
$$ratio = w_{\Delta 1}^{(1)} / w_{\Delta 1}^{(2)} \tag{3.9}$$

where, $w_{\Delta 1}^{(1)}$ and $w_{\Delta 1}^{(2)}$ are the widths of $\Delta_1^{(1)}$ and $\Delta_1^{(2)}$, respectively. Note that the heights of them can be taken instead of the widths. By simply re-scaling map 2 by *ratio*, both maps will finally become in the same scale. From the example of figure 3.8, *ratio* is 1.1176, and this implies that map 1 is 1.1176 times larger than map 2, thus map 2 is re-scaled by 1.1176. Since map 2 has been re-scaled, entire process from R-mapping should be repeated with the re-scaled maps. Thus, the next iteration will return better estimations. The iteration stops when the maps are in the same scale ($ratio = 1$). The maps in figure 3.9 are in the same scale after 2 iterations, and the shared-triangles in both maps are the correct sets. Finally, a merged map is accomplished by overlapping the center points of the shared-triangles. Figure 3.10a illustrates a merged map without re-scaling process (no iterations), thus the maps have a worse fit. However, in figure 3.10b, the merged map has a better fit after iterations.



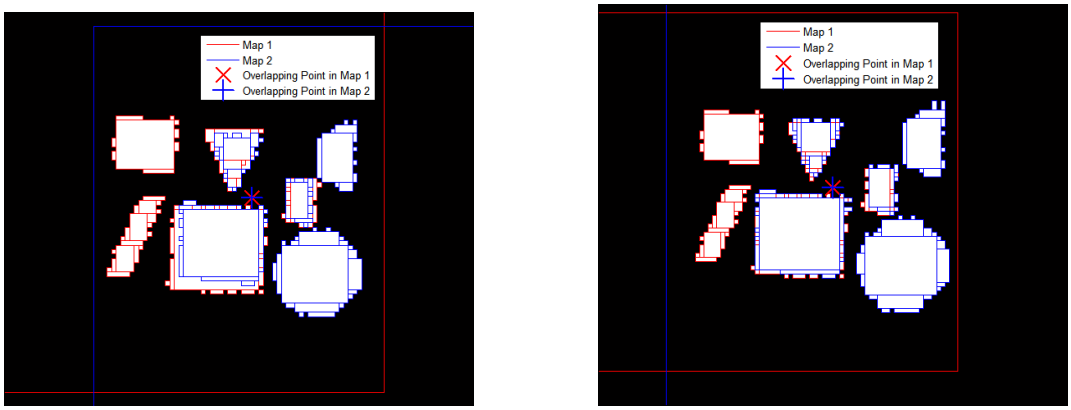
(a) map 1 (b) map 2

Figure 3.8. Orientation-matched maps. The orientations are matched through rotating map 1 by $\Lambda_f = 180°$. Δ_3's are still incorrect shared-rectangles ($ratio = 1.1176$).



(a) map 1 (b) map 2

Figure 3.9. Scale-matched maps ($ratio = 1$). The scales are matched through re-scaling map 2 by the previous *ratio*. Δ_3's are the correct shared-rectangles.



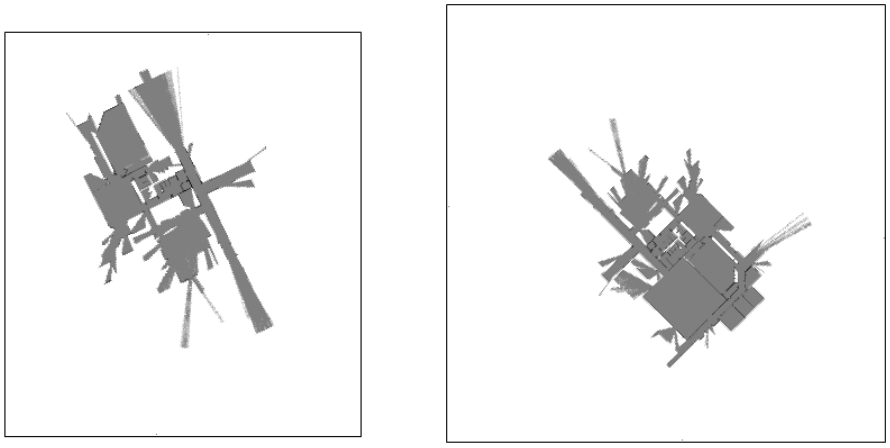
(a) without re-scaling. (b) with re-scaling.

Figure 3.10. Merged maps. (a) Merging maps of figure 3.8 makes a worse fit. (b) Merging maps of figure 3.9 makes a better fit.

3.4 Result

The practical maps of figure 3.11 is acquired from two identical ground robots. The maps naturally have corruptions and different scales. The orientations of the maps are intentionally rotated by $\Lambda_0^{(1)} = 27^\circ$ and $\Lambda_0^{(2)} = 225^\circ$ for map 1 and map 2, respectively. The orthogonal-orientations algorithm estimates the rotation angles $\Lambda^{(1)} = 63.4349^\circ$ and $\Lambda^{(2)} = 45^\circ$ for each, thus both maps are in the orthogonal orientations through rotating the maps by $\Lambda_0^{(1)} + \Lambda^{(1)} = 90.4349^\circ$ and $\Lambda_0^{(2)} + \Lambda^{(2)} = 270^\circ$, respectively. With the orthogonal oriented maps, the shared-triangles algorithm returns $\Delta^{(1)} = \{r_3, r_{10}, r_{11}\}$ and $\Delta^{(2)} = \{r_2, r_6, r_{14}\}$ with $\min(J) = 68.0174$. By comparing the bisector-vectors from both maps, the final rotation angle is $\Lambda_f = 180^\circ$, thus the maps are in the same orientation through rotating map 1 by 180°. The scale ratio between the maps is $ratio = 1.0313$, and map 2 is re-scaled by 1.0313. As map 2 has been re-scaled, the entire process is repeated with the re-scaled maps. After the second iteration, $ratio = 1.1515$, and finally $ratio = 1$ after the third iteration. Figure 3.12 shows the final merged map, and it has a good fit of two local maps.

In the form of G-maps, map 1 and map 2 in figure 3.11 have resolutions (361×317) and (393×393), respectively. The simulation was processed in Matlab on a normal laptop PC, and the computation time is about 80 seconds through the three iterations. Simpler environments in lower resolutions will take shorter computation time for map merging.



(a) map 1 (b) map 2

Figure 3.11. Actual maps from two ground mapping robots. The three factors, orientations, accuracy and scales, are unknown and may different to each other map.

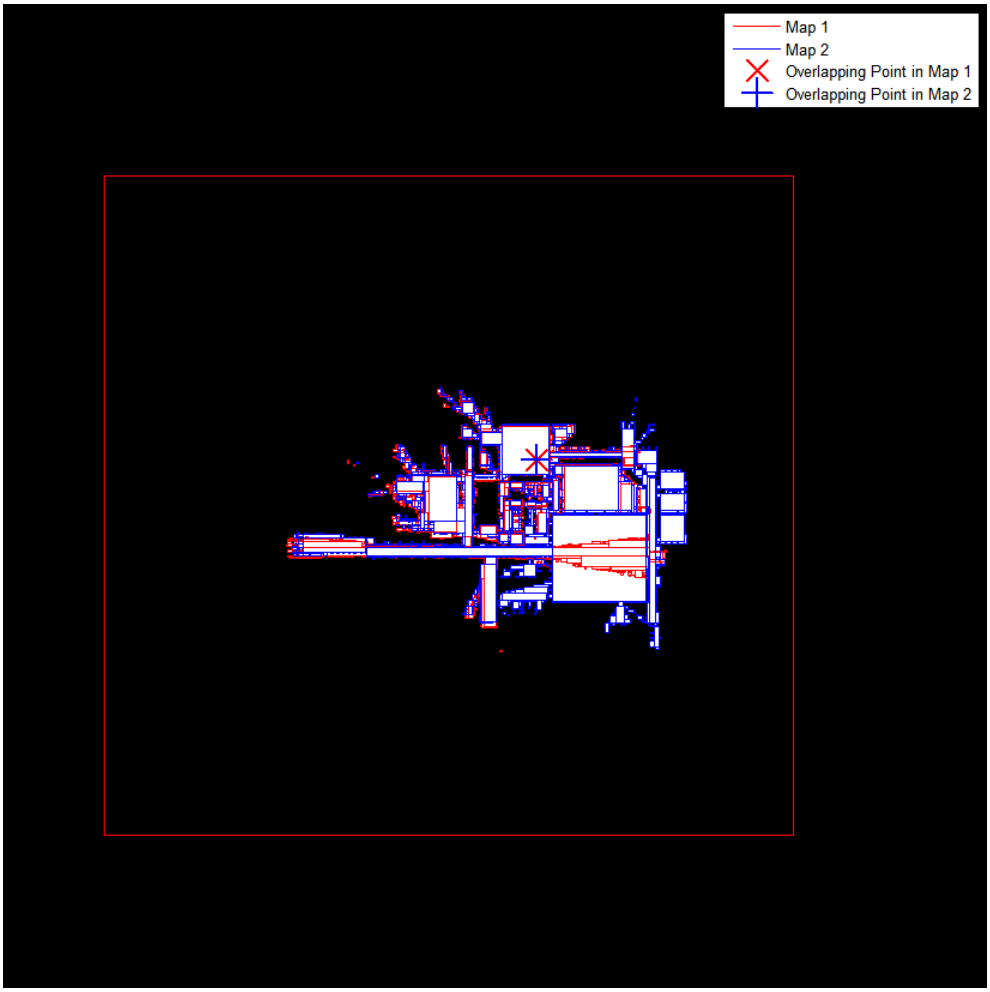


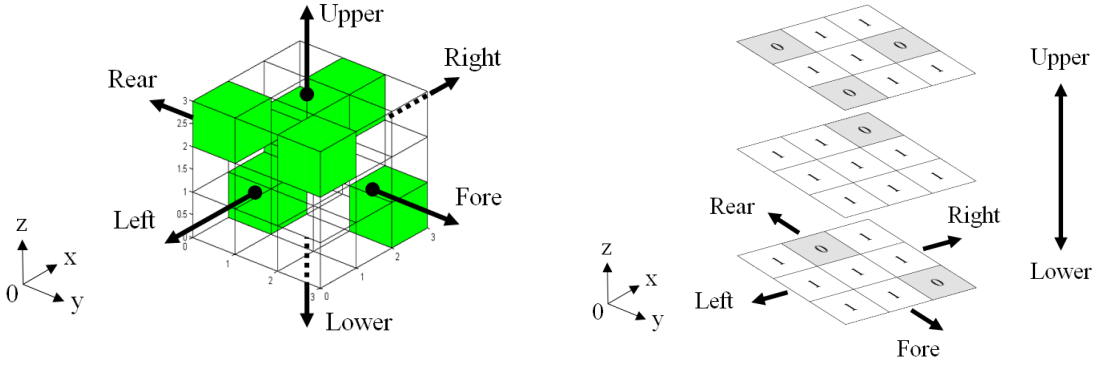
Figure 3.12. Merged map of figure 3.11

Chapter 4

3D R-mapping

This Chapter describes an enabling process to accomplish computational efficiency in obstacle avoidance and target interception. The purpose of this process is the same as 2D R-mapping, i.e. integrating free cells into maximal empty areas to reduce the number of cell elements by the four steps. However, since 3D environments have an additional dimension of the z-axis, the process of R-mapping will be slightly different and the integrated cells will be represented as cuboids (volumes) instead of rectangles (areas). Those maximal empty cuboids (MECs) are obstacle-free spaces and the algorithm of 3D R-map also provides connections of the MECs, thus UAVs can accomplish obstacle avoidance by moving through a sequence of MECs.

A 3D environment can be represented as an $(m \times n \times l)$ array or l layers of $(m \times n)$ matrices as illustrated in figure 4.1. Here, y, x, and z are the axis on each direction, and m, n, and l are the dimensions on each axis, respectively. For example, the coordinates of each element in an $(m \times n \times l)$ array are represented as (y, x, z), where (row, column, layer). Also, obstacles and free spaces are represented as 0 and 1 in the array, respectively. This binary array is defined as G, and four steps of 3D R-mapping begin with G. In step 1, a corresponding array, cm, is derived from G. This cm is made by summations of contiguous free-elements in both x and z directions. In step 2, the survey to find an MEC proceeds element-by-element (y-direction) and column-by-column (x-direction) then layer-by-layer (z-direction) in cm. The survey compares the current element and the previous element by four possibilities. In step 3, corresponding elements of the MEC from step 2 in G are updated to zeros to ensure that the subsequent iterations only search the remaining free space. Then, step 1 through step 3 are repeated until all elements in G have been set to zeros. After the



(a) $(m \times n \times l)$ array

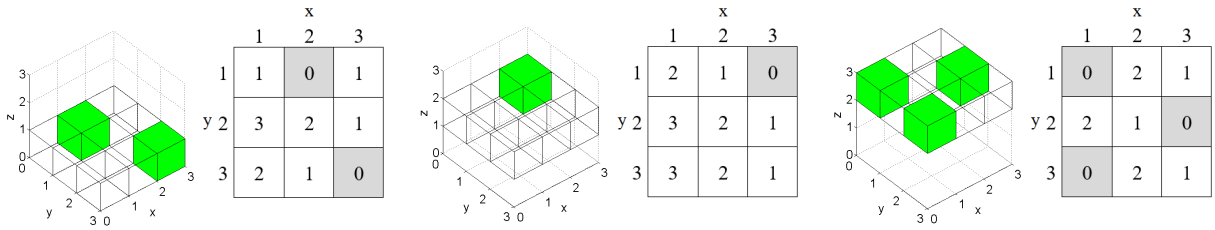
(b) l layers of $(m \times n)$ matrices

Figure 4.1. A 3D environment can be represented as (a) a 3D array or (b) multiple layers of 2D matrices.

iterations, a list of MECs in the order of volume size is acquired. In step 4, connections of the MECs are computed. Finally, the algorithm returns a list of MECs and their connections. This information will be utilized in applications of R-maps. The following sections describe the four steps in more detail and provide an example environment.

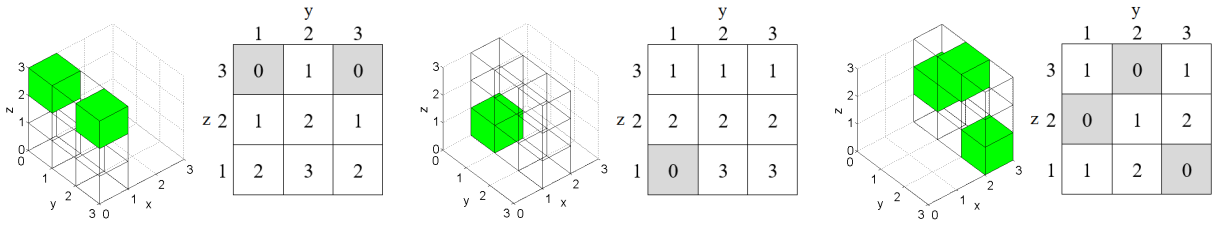
4.1 Step 1: Initialization

This step introduces an array, cm. First, a 3D grid environment with the dimensions of $(m \times n \times l)$ is defined as G, which can be also expressed as l layers of $(m \times n)$ matrices. Figure 4.1b shows a $(3 \times 3 \times 3)$ array as an example, where 0 and 1 indicate obstacles and free spaces, respectively. Here, two matrices, cx and cz, are derived from the binary numbers. In cx, each of whose elements is computed by right-to-left x-wise summations of contiguous free-elements as shown in figure 4.2. Because the summation is from right to left in x-wise, the computing-order of x in cx is from n to 1 and it is assumed that $cx(y, n+1, z) = 0$. In cz, each of whose elements is computed by upper-to-lower z-wise summations of contiguous free-elements as shown in figure 4.3. Also, because the summation is from upper to lower in z-wise, the computing-order of z in cz is from l to 1, and it is assumed that $cz(y, x, l+1) = 0$.



(a) 1st layer ($z = 1$) (b) 2nd layer ($z = 2$) (c) 3rd layer ($z = 3$)

Figure 4.2. cx is obtained by right-to-left x-wise summations of contiguous free-elements.



(a) 1st layer ($x = 1$) (b) 2nd layer ($x = 2$) (c) 3rd layer ($x = 3$)

Figure 4.3. cz is obtained by top-to-bottom z-wise summations of contiguous free-elements.

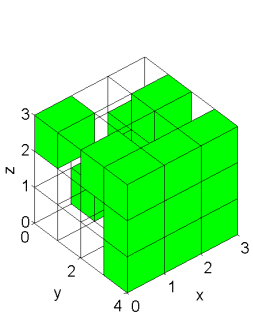
$$cx(y, x, z) = \begin{cases} cx(y, x+1, z) + 1 & \text{if } G(y, x, z) = 1 \\ 0 & \text{if } G(y, x, z) = 0 \end{cases} \tag{4.1}$$

$$cz(y, x, z) = \begin{cases} cz(y, x, z+1) + 1 & \text{if } G(y, x, z) = 1 \\ 0 & \text{if } G(y, x, z) = 0 \end{cases} \tag{4.2}$$

Thus, each value in cx indicates number of free-cells on the right-side, and each value in cz indicates number of free-cells on the upper-side. Finally, a corresponding array, cm, is obtained by taking the values of cx as a real part and the values of cz as an imaginary part. Also, additional elements equal to zero are added to the foremost of cm.

$$cm = cx + cz \cdot j \tag{4.3}$$

$$cm(m+1, x, z) = 0 \tag{4.4}$$



(a) 1st layer

x

y	1	2	3
1	1+2j	0+0j	1+1j
2	3+3j	2+3j	1+2j
3	2+2j	1+3j	0+0j
4	0+0j	0+0j	0+0j

(b) 1st layer

x

y	1	2	3
1	2+1j	1+2j	0+0j
2	3+2j	2+2j	1+1j
3	3+1j	2+2j	1+2j
4	0+0j	0+0j	0+0j

(c) 2nd layer

x

y	1	2	3
1	0+0j	2+1j	1+1j
2	2+1j	1+1j	0+0j
3	0+0j	2+1j	1+1j
4	0+0j	0+0j	0+0j

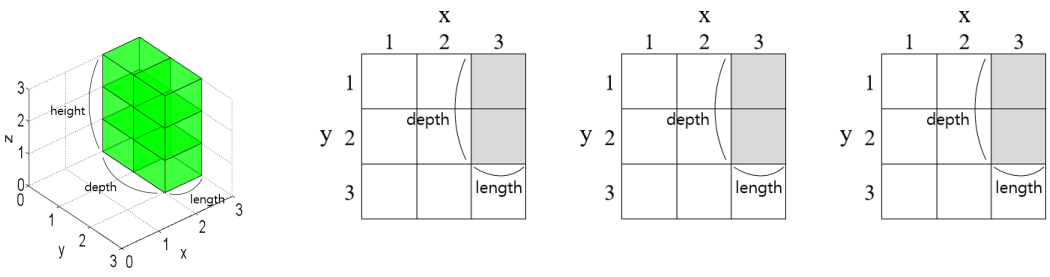
(d) 3rd layer

Figure 4.4. cm is obtained by a summation of cx as real numbers and cz as imaginary numbers. Also, additional elements equal to zero are added to the foremost of cm.

Therefore, the dimensions of cm become $(m+1, n, l)$, and each element contains values that indicate the number of free-cells in the x- and z-directions. Figure 4.4 shows cm derived from G, and it has the dimensions of $(4 \times 3 \times 3)$.

4.2 Step 2: Finding the MEC

This step produces the maximal empty cuboid (MEC) from cm by surveying every element. Note that a cuboid has the dimensions of (depth $\times$ length $\times$ height), and each of these dimensions is associated with the y-, x-, and z-axis of an array, respectively, as shown in figure 4.5. In cm, each element consists of two distinct parts $(\mathbb{R}+\mathbb{I}j)$ as shown in figure 4.4: the real part ($\mathbb{R}$: x-wise summations) and the imaginary part ($\mathbb{I}$: z-wise summations), which determine the length (x-direction) and the height (z-direction) of potential cuboids, respectively. Beginning at the first element, $cm(1,1,1)$, the survey proceeds element-by-element in the y-direction comparing the real numbers of the current element and the previous element, $\mathbb{R}(cm(y,x,z))$ and $\mathbb{R}(cm(y-1,x,z))$. Surveying in the y-direction and comparing the real parts of cm find bottom rectangles of candidate MECs. As the survey proceeds, candidate bottom rectangles are stored in a temporary matrix, T. At the first row of each column, $cm(1,x,z)$, the current element initializes T, that contains the current row number and the



(a) a cuboid (b) 1st layer (c) 2nd layer (d) 3rd layer

Figure 4.5. Each dimension of a cuboid, (depth × length × height), is associated with the y-, x-, and z-axis, respectively.

value of the current element.

$$T = [1,\ \ \mathbb{R}(cm(1, x, z))] \tag{4.5}$$

As proceeding to subsequent elements, a new entry is added to T if the value of the current element is greater than the value of the previous element.

$$T := \begin{bmatrix} T \\ \text{- - - - - - - - -} \\ y,\ \ \mathbb{R}(cm(y, x, z)) \end{bmatrix} \tag{4.6}$$

Here, := indicates updating the variable on the left-hand side with the value on the right-hand side. Thus, T will have multiple entries, e, as the survey proceeds. Additionally, a variable D is defined to avoid redundant searching in T.

There are four possibilities in the comparisons of the current and the previous elements.

Case 4.2.1. $\mathbb{R}(cm(y, x, z)) = \mathbb{R}(cm(y-1, x, z))$:

No changes are made to T.

Case 4.2.2. $\mathbb{R}(cm((y, x, z)) < \mathbb{R}(cm(y-1, x, z))$:

Calculate volumes of cuboids with each entry in T, and update *largest–cuboid* if necessary. Also, as the volumes are calculated, update any length, $T(e, 2)$, greater than the current length, $\mathbb{R}(cm(y, x, z))$, to the current length, and update D if necessary.

Case 4.2.3. $\mathbb{R}(cm(y, x, z)) > \mathbb{R}(cm(y-1, x, z))$:

Add a new element to T.

Case 4.2.4. $\mathbb{R}(cm(y, x, z)) = 0$:

Calculate volumes of cuboids with each entry in T, and update *largest–cuboid* if necessary. After the volume calculations, remove all temporary memories and set $D = 0$ and $T = [\,\cdot\,]$.

In the comparisons, when the current element is smaller than the previous element or equal to zero, the dimensions of possible cuboids are computed from T to find any larger volume. Possible depth is the number of elements in the y-axis from the row of the current surveying element, y, to the row of the current entry in T, $T(e, 1)$, which is $y - T(e, 1)$. Possible length is simply the real part of the current entry in T, which is $T(e, 2)$. This computation returns only the largest value for each depth and length, thus they determine one bottom rectangle, (depth $\times$ length), of a potential cuboid. Although knowing only the largest values for them was sufficient for 2D rectangles, in 3D cuboids, all possible rectangles within the largest bottom rectangle should be considered because a smaller bottom rectangle may have a taller height and a larger volume. For example, a narrow-and-tall cuboid can have a larger volume than a wide-and-short cuboid. For this reason, multiple possibilities should be considered for each depth and length from 1 up to $y - T(e, 1)$ and 1 up to $T(e, 2)$, respectively. Accordingly, combinations of the multiple values from the two dimensions produce multiple bottom rectangles within the largest possible bottom rectangle. Then, the height of a potential cuboid is the number of empty cells in the z-axis up to where the bottom dimensions can extend, which is the smallest imaginary number in the bottom dimensions, min[$\mathbb{I}$(bottom-rectangle)].

$$depth = [1 : (y - T(e, 1))] \tag{4.7a}$$

$$length = [1 : T(e, 2)] \tag{4.7b}$$

$$height = \min[\mathbb{I}(cm(T(e, 1) : (T(e, 1) + depth - 1),\ \ x : (x + length - 1),\ \ z))] \tag{4.7c}$$

Here, $[A : B]$ means a list of continuous numbers from A to B, for example, $[1 : 3] = [1, 2, 3]$. Combinations of the elements from equations 4.7b and 4.7a make bottom rectangles, and each of the bottom rectangle has its corresponding height depending on its dimensions as shown in equation 4.7c. Therefore, these three dimensions make one possible cuboid with a volume of $(depth \times length \times height)$ having the entry element in cm, $cm(T(e, 1), x, z)$, as the bottom-left-rear corner of the cuboid.

However, multiple entries in T can produce some identical cuboids with the cuboids previously found. Since these repetitive calculations can delay R-mapping, elements that produce such repetitive cuboids need to be removed by defining a deletion parameter, D, which is an optional process for faster R-mapping. This D is defined initially as zero, and it takes the largest number among the values of depths in each volume calculation. A list of depths that produces repetitive cuboids is the depths that the survey has been considered in the previous volume calculations. The list of depths to be removed is calculated by D and e, if the condition is satisfied.

$$eliminate\text{–}elements = [1 : (D - e + 1)] \qquad \text{if } (D - e + 1) > 0 \tag{4.8}$$

By removing *eliminate–elements* from *depth* in equation 4.7a, the algorithm can avoid redundant calculations.

Also, *largest–cuboid* is defined initially as zero, and as the volume calculations proceed, any cuboid with a larger volume than the current *largest–cuboid* replaces *largest–cuboid*. After surveying all elements of cm, the final *largest–cuboid* becomes an MEC of the i-th iteration.

$$MEC_i = [y_i, x_i, z_i, depth_i, length_i, height_i] \tag{4.9}$$

Here, the first three values indicate the coordinates of the bottom-left-rear corner of MEC_i, and the last three values are the dimensions of MEC_i.

An example process is given here with cm shown in figure 4.4. First, $largest\text{–}cuboid$ and D are defined.

$$largest\text{–}cuboid = 0 \tag{4.10}$$

$$D = 0 \tag{4.11}$$

The survey begins with the first element in the first column of the first layer, which is the top-left element in figure 4.4b, and whose coordinate is $cm(1,1,1)$. As assumed, the first element is added to T.

$$\text{At first element: } T = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{4.12}$$

The survey proceeds to the second element, $cm(2,1,1)$. The comparison of the real parts of the current and the previous elements is case 4.2.3, thus the current element is added to T. Now, T has two entries.

$$\begin{gathered} \text{At second element: } \mathbb{R}(cm(2,1,1)) = 3 > \mathbb{R}(cm(1,1,1)) = 1 \\ T = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} \end{gathered} \tag{4.13}$$

As the survey proceeds to the third element, $cm(3,1,1)$, case 4.2.2 is applied.

$$\text{At third element: } \mathbb{R}(cm(3,1,1)) = 2 < \mathbb{R}(cm(2,1,1)) = 3 \tag{4.14}$$

Without any changes to T of equation 4.13, volumes of possible cuboids are calculated and compared to find any larger cuboid than $largest\text{–}cuboid$. The current parameters are $y = 3$ (third element) and $e = 1, 2$ (T has two entries), and consider each entry for the volume

calculations.

$$\text{for } e = 1,\ \mathit{depth} = [1 : (y - T(e,1))] = [1 : (3-1)] = [1,2]^{*} \tag{4.15a}$$

$$^{*}(D - e + 1) = (0 - 1 + 1) = 0 \not> 0 \quad \text{(false)} \tag{4.15b}$$

$$\rightarrow \text{no } \mathit{eliminate\text{–}elements} \text{ in } \mathit{depth} \tag{4.15c}$$

$$\mathit{length} = [1 : T(e,2)] = [1] \tag{4.15d}$$

$$\mathit{height}_1 = \min[\mathbb{I}(cm(1:1,\ \ 1:1,\ \ 1))] = \min[2] = 2 \tag{4.15e}$$

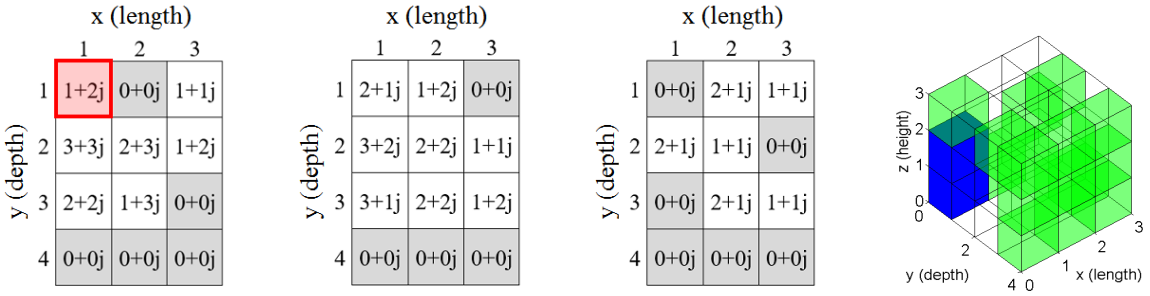
$$\mathit{height}_2 = \min[\mathbb{I}(cm(1:2,\ \ 1:1,\ \ 1))] = \min[2,3] = 2 \tag{4.15f}$$

$$\text{cuboid-1} = \mathit{depth}(1) \times \mathit{length}(1) \times \mathit{height}_1 = 1 \times 1 \times 2 = 2 \tag{4.15g}$$

$$\text{cuboid-2} = \mathit{depth}(2) \times \mathit{length}(1) \times \mathit{height}_2 = 2 \times 1 \times 2 = 4 \tag{4.15h}$$

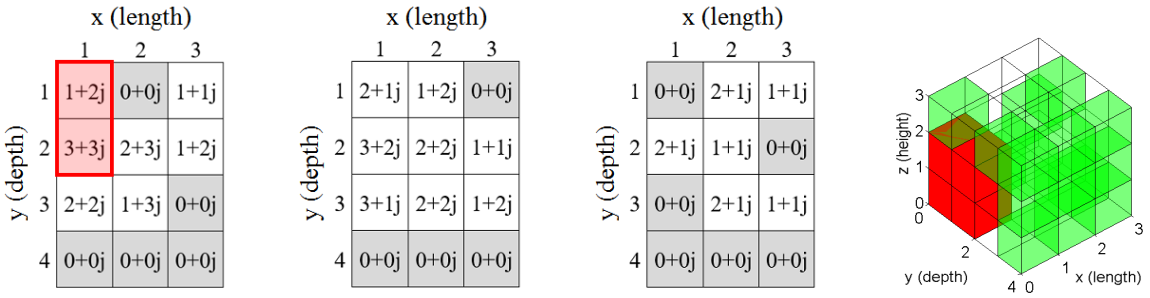
From the first entry ($e = 1$) in T, the *depth* and *length* combinations produce two bottom rectangles of (1×1) and (2×1) as shown in figures 4.6a and 4.7a. In this calculations, because this is early in the survey, there are no repetitive cuboids yet and the elimination condition is false as shown in equation 4.15b. Each bottom rectangle has its corresponding height as shown in equations 4.15e and 4.15f. Therefore, two cuboids are found from the first entry in T as shown in figures 4.6d and 4.7d, and they are compared. Currently, *largest–cuboid* is zero, and as cuboid-2 from equation 4.15h is the largest volume, it replaces *largest–cuboid*.

$$\mathit{largest\text{–}cuboid} = \text{cuboid-2 (with volume of 4)} \tag{4.16}$$



(a) 1st layer (b) 2nd layer (c) 3rd layer (d) cuboid-1

Figure 4.6. (a) The bottom area (1×1) has $height = 2$. (d) Cuboid-1 has a volume of $(1 \times 1 \times 2) = 2$.



(a) 1st layer (b) 2nd layer (c) 3rd layer (d) cuboid-2

Figure 4.7. (a) The bottom area (2×1) has $height = 2$. (d) Cuboid-2 has a volume of $(2 \times 1 \times 2) = 4$.

The same process is applied to the second entry $(e = 2)$ in T.

$$\text{for } e = 2,\ depth = [1 : (y - T(e,1))] = [1 : (3-2)] = [1]^* \tag{4.17a}$$

$$^*(D - e + 1) = (0 - 2 + 1) = -1 \not> 0 \quad \text{(false)} \tag{4.17b}$$

$$\rightarrow \text{no } eliminate\text{–}elements \text{ in } depth \tag{4.17c}$$

$$length = [1 : T(e,2)] = [1,2,3] \tag{4.17d}$$

$$height_1 = \min[\mathbb{I}(cm(2:2,\ \ 1:1,\ \ 1))] = \min[3] = 3 \tag{4.17e}$$

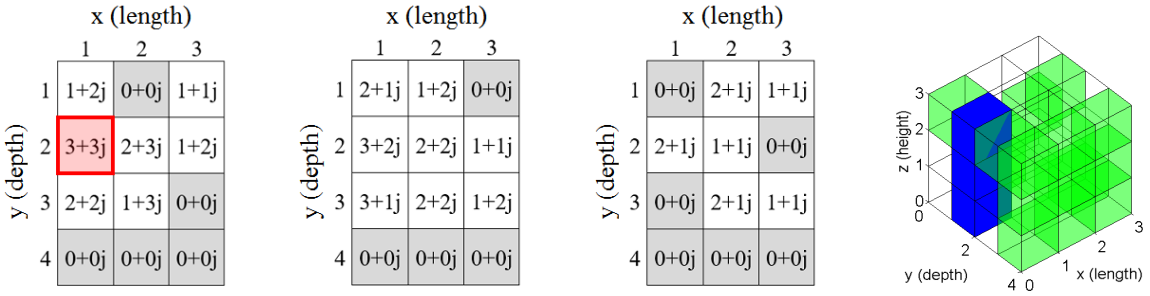
$$height_2 = \min[\mathbb{I}(cm(2:2,\ \ 1:2,\ \ 1))] = \min[3,3] = 3 \tag{4.17f}$$

$$height_3 = \min[\mathbb{I}(cm(2:2,\ \ 1:3,\ \ 1))] = \min[3,3,2] = 2 \tag{4.17g}$$

$$\text{cuboid-3} = depth(1) \times length(1) \times height_1 = 1 \times 1 \times 3 = 3 \tag{4.17h}$$

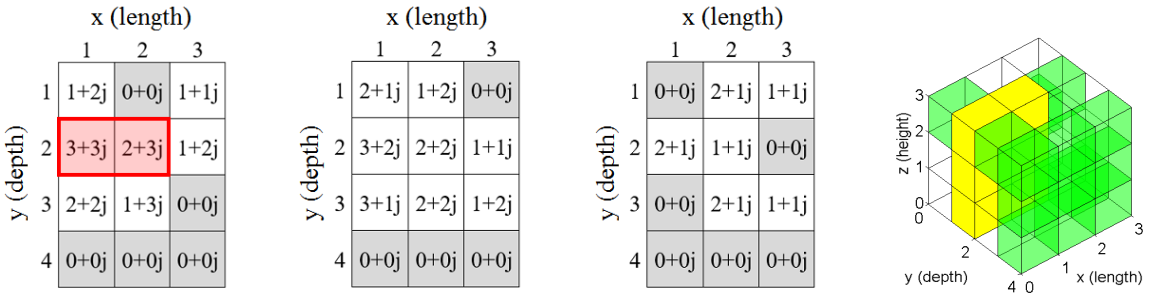
$$\text{cuboid-4} = depth(1) \times length(2) \times height_2 = 1 \times 2 \times 3 = 6 \tag{4.17i}$$

$$\text{cuboid-5} = depth(1) \times length(3) \times height_3 = 1 \times 3 \times 2 = 6 \tag{4.17j}$$



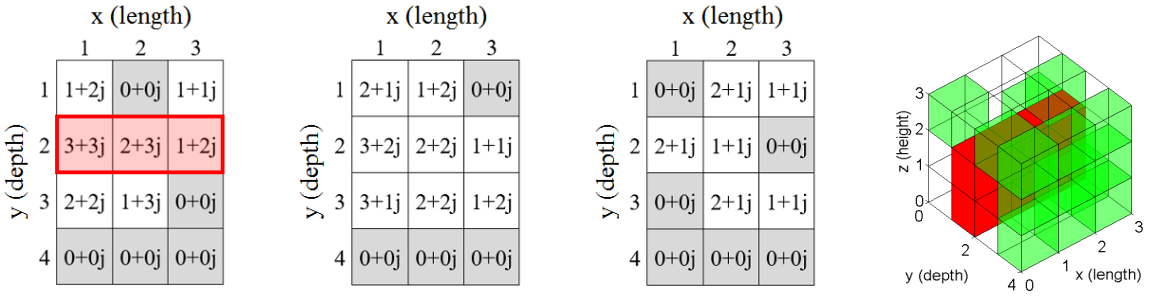
(a) 1st layer (b) 2nd layer (c) 3rd layer (d) cuboid-3

Figure 4.8. (a) The bottom area (1×1) has $height = 3$. (d) Cuboid-3 has a volume of $(1 \times 1 \times 3) = 3$.



(a) 1st layer (b) 2nd layer (c) 3rd layer (d) cuboid-4

Figure 4.9. (a) The bottom area (1×2) has $height = 3$. (d) Cuboid-4 has a volume of $(1 \times 2 \times 3) = 6$.



(a) 1st layer (b) 2nd layer (c) 3rd layer (d) cuboid-5

Figure 4.10. (a) The bottom area (1×3) has $height = 2$. (d) Cuboid-5 has a volume of $(1 \times 3 \times 2) = 6$.

Figures 4.8, 4.9, and 4.10 illustrate cuboids-3, -4, and -5, respectively. Cuboid-4 and cuboid-5 have larger volumes than *largest–cuboid*, however because cuboid-4 has found earlier than cuboid-5, it replaces *largest–cuboid*.

$$largest\text{–}cuboid = \text{cuboid-4 (with volume of 6)} \tag{4.18}$$

Also, D takes the largest value among *depth* of equations 4.15a and 4.17a.

$$D = 2 \tag{4.19}$$

After the volume calculations, according to case 4.2.2, any length in T greater than the current length are updated to the current length. The real part of the current element is the current length, $\mathbb{R}(cm(3,1,1)) = 2$, thus the length of the second entry in T, $T(2,2) = 3$, is updated to 2.

$$T = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} \qquad \rightarrow \qquad T = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \tag{4.20}$$

This updating helps to find one or more new bottom rectangles with dimensions narrower in the x-axis and longer in the y-axis that fit to the current length ($\mathbb{R}(cm(3,1,1)) = 2$).

The survey proceeds to the fourth element, $\mathbb{R}(cm(4,1,1)) = 0$, and it is case 4.2.4. With the updated T from equation 4.20, volumes of possible cuboids are calculated and compared. First, volume calculations with the first entry in T.

$$\text{for } e = 1,\ \mathit{depth} = [1 : (y - T(e,1))] = [1 : (4-1)] = [1,2,3]^{*} \tag{4.21a}$$

$$^{*}(D - e + 1) = (2 - 1 + 1) = 2 > 0 \quad \text{(true)} \tag{4.21b}$$

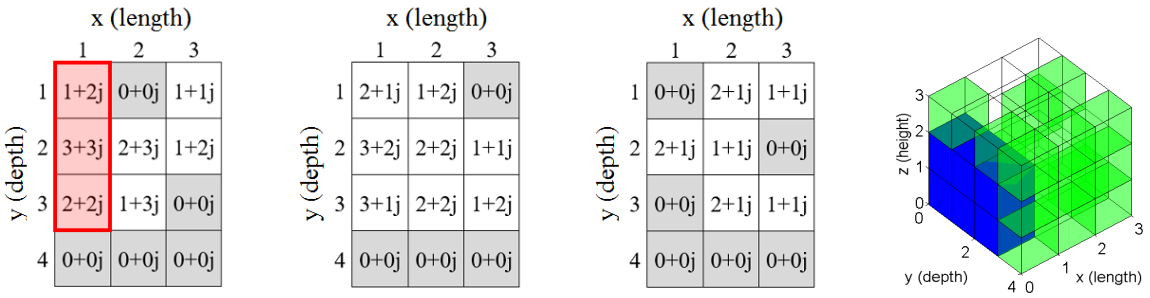
$$\rightarrow \mathit{eliminate\text{–}elements} = [1 : (D - e + 1)] = [1 : 2] = [1, 2] \tag{4.21c}$$

$$\mathit{length} = [1 : T(e,2)] = [1] \tag{4.21d}$$

$$\mathit{height}_1 = \min[\mathbb{I}(cm(1:3,\ \ 1:1,\ \ 1))] = \min[2,3,2] = 2 \tag{4.21e}$$

$$\text{cuboid-6} = \mathit{depth}(1) \times \mathit{length}(1) \times \mathit{height}_1 = 3 \times 1 \times 2 = 6 \tag{4.21f}$$

In this calculation, the elimination condition becomes true as shown in equation 4.21b, and elements that produce repetitive cuboids are found in equation 4.21c. Those elements are removed from *depth* as shown in equation 4.21a. If they are not removed, $\mathit{depth} = [1,2]$ and $\mathit{length} = [1]$ will produce cuboid-1 and cuboid-2 again. Figure 4.11 illustrates cuboid-6.



(a) 1st layer (b) 2nd layer (c) 3rd layer (d) cuboid-6

Figure 4.11. (a) The bottom area (3×1) has $height = 2$. (d) Cuboid-6 has a volume of $(3 \times 1 \times 2) = 6$.

This cuboid-6 has the same volume with the current *largest–cuboid* as 6, thus cuboid-4 from equation 4.18 is kept as *largest–cuboid*. The same process is applied to the second entry in T.

$$\text{for } e = 2,\ depth = [1 : (y - T(e,1))] = [1 : (4-2)] = [1, 2]^{*} \tag{4.22a}$$

$$^{*}(D - e + 1) = (2 - 2 + 1) = 1 > 0 \quad \text{(true)} \tag{4.22b}$$

$$\rightarrow eliminate\text{–}elements = [1 : (D - e + 1)] = [1 : 1] = [1] \tag{4.22c}$$

$$length = [1 : T(e,2)] = [1, 2] \tag{4.22d}$$

$$height_1 = \min[\mathbb{I}(cm(2:3,\ \ 1:1,\ \ 1))] = \min[3, 2] = 2 \tag{4.22e}$$

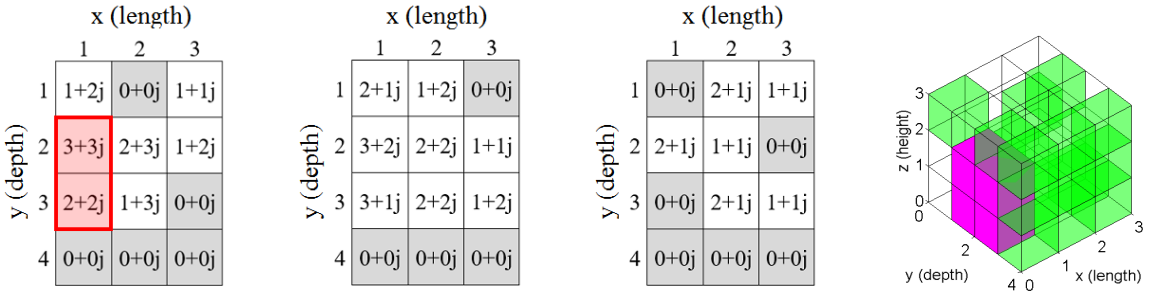
$$height_2 = \min[\mathbb{I}(cm(2:3,\ \ 1:2,\ \ 1))] = \min[3, 3, 2, 3] = 2 \tag{4.22f}$$

$$\text{cuboid-7} = depth(1) \times length(1) \times height_1 = 2 \times 1 \times 2 = 4 \tag{4.22g}$$

$$\text{cuboid-8} = depth(1) \times length(2) \times height_2 = 2 \times 2 \times 2 = 8 \tag{4.22h}$$

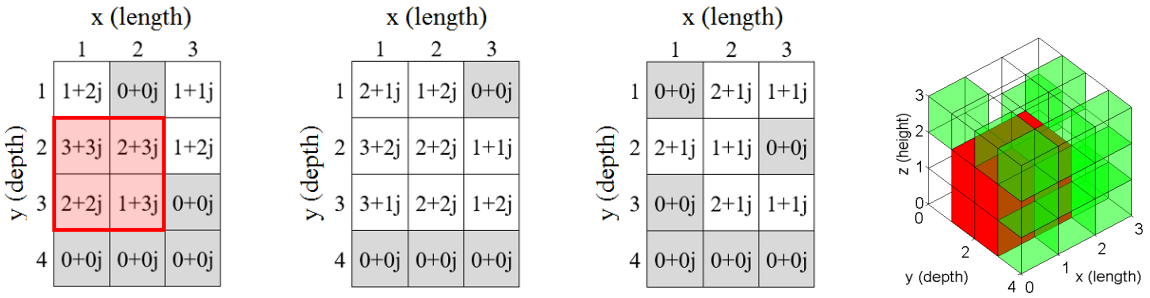
Figures 4.12 and 4.13 illustrate cuboid-7 and cuboid-8, respectively. Cuboid-8 has a larger volume than *largest–cuboid*, and it replaces *largest–cuboid*. Also, according to case 4.2.4, all temporary memories are removed, and the survey proceeds to the next column, $cm(y, 2, z)$.

$$largest\text{–}cuboid = \text{cuboid-8 (with volume of 8)} \tag{4.23}$$



(a) 1st layer (b) 2nd layer (c) 3rd layer (d) cuboid-7

Figure 4.12. (a) The bottom area (2×1) has $height = 2$. (d) Cuboid-7 has a volume of $(2 \times 1 \times 2) = 4$.



(a) 1st layer (b) 2nd layer (c) 3rd layer (d) cuboid-8

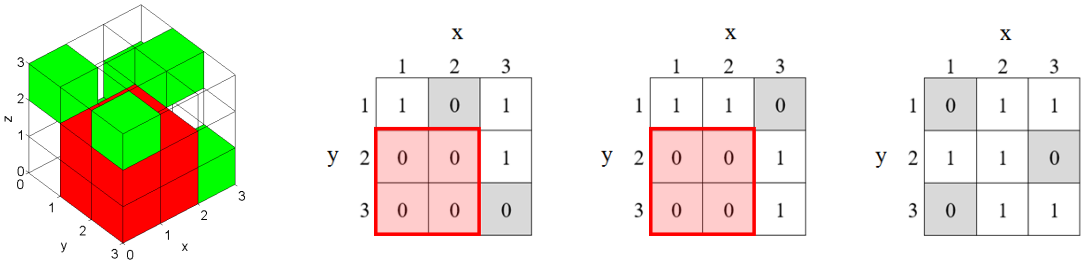
Figure 4.13. (a) The bottom area (2×2) has $height = 2$. (d) Cuboid-8 has a volume of $(2 \times 2 \times 2) = 8$.

$$T = [\,\cdot\,] \quad \text{and} \quad D = 0 \tag{4.24}$$

After surveying every element in cm in this manner, the final $largest\text{–}cuboid$ will be an MEC. In this first iteration, for example, cuboid-8 is the final $largest\text{–}cuboid$, and the information of cuboid-8 is saved as MEC_1.

$$MEC_1 = [2, 1, 1, 2, 2, 2] \tag{4.25}$$

Generally, MEC_1 is the largest cuboid in an environment, and MECs from the next iterations will be smaller than this, thus a list of MECs will be in the order of volume size from the largest to the smallest.



(a) MEC_1 (b) 1st layer (c) 2nd layer (d) 3rd layer

Figure 4.14. Corresponding elements of MEC_1 (cuboid-8) in G are updated to zeros.

4.3 Step 3: Grid map update

This step updates corresponding elements of MEC_i in G to zeros to consider MEC_i as an obstacle. For example, the corresponding elements of MEC_1 in G are updated to zeros as shown in figure 4.14. This updating ensures that the subsequent iterations only search the remaining free space.

With the updated G, steps 1 though 3 are repeated in an iterative fashion until all elements in G have been set to zeros. In the case of this example, the iterations occurred nine times, and nine MECs are acquired.

$$MEC_1 = [2, 1, 1, 2, 2, 2] \tag{4.26a}$$

$$MEC_2 = [1, 2, 3, 1, 3, 1] \tag{4.26b}$$

$$MEC_3 = [1, 1, 1, 1, 1, 2] \tag{4.26c}$$

$$MEC_4 = [1, 3, 1, 1, 2, 1] \tag{4.26d}$$

$$MEC_5 = [2, 3, 2, 1, 2, 1] \tag{4.26e}$$

$$MEC_6 = [1, 2, 2, 1, 1, 1] \tag{4.26f}$$

$$MEC_7 = [2, 1, 3, 1, 1, 1] \tag{4.26g}$$

$$MEC_8 = [1, 3, 3, 1, 1, 1] \tag{4.26h}$$

$$MEC_9 = [3, 3, 3, 1, 1, 1] \tag{4.26i}$$

4.4 Step 4: Computing connections

This step finds the connections of MECs. First, MECs are labeled by means of updating the elements of each MEC_i to their rank, i. Also, additional elements equal to zero are added around G, thus the dimensions of G become $((m+2)\times(n+2)\times(l+2))$. Accordingly, the coordinates of MEC_i should be changed for the updated G.

$$MEC_i^G = [y_i+1, x_i+1, z_i+1, depth_i, length_i, height_i] \tag{4.27a}$$

$$= [Y_i, X_i, Z_i, depth_i, length_i, height_i] \tag{4.27b}$$

Here, MEC_i^G means the corresponding MEC_i in the updated G. The connections of each MEC can be recognized by looking at the neighboring elements on each side of MEC_i^G. A list of connections on each side is obtained by taking the unique neighboring elements except zeros, u[neighboring-elements], where u means the unique numbers except zeros.

$$lower_i = u[G(Y_i : (Y_i + depth_i - 1),\ \ X_i : (X_i + length_i - 1),\ \ Z_i - 1)] \tag{4.28a}$$

$$upper_i = u[G(Y_i : (Y_i + depth_i - 1),\ \ X_i : (X_i + length_i - 1),\ \ (Z_i + height_i))] \tag{4.28b}$$

$$left_i = u[G(Y_i : (Y_i + depth_i - 1),\ \ (X_i - 1),\ \ Z_i : (Z_i + height - 1))] \tag{4.28c}$$

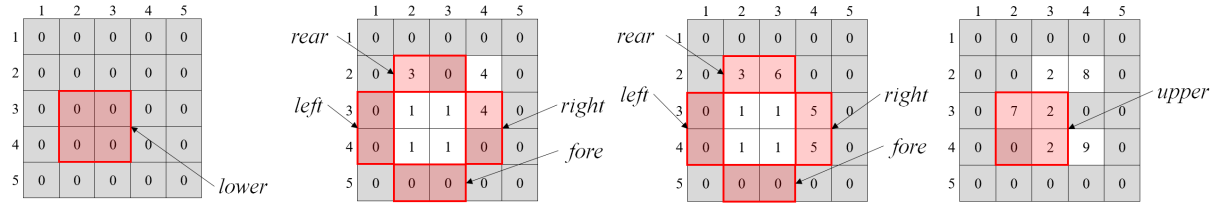
$$right_i = u[G(Y_i : (Y_i + depth_i - 1),\ \ (X_i + length_i),\ \ Z_i : (Z_i + height - 1))] \tag{4.28d}$$

$$fore_i = u[G((Y_i + depth_i),\ \ X_i : (X_i + length_i - 1),\ \ Z_i : (Z_i + height - 1))] \tag{4.28e}$$

$$rear_i = u[G((Y_i - 1),\ \ X_i : (X_i + length_i - 1),\ \ Z_i : (Z_i + height - 1))] \tag{4.28f}$$

Equation 4.28 shows lists of connections on the six sides of MEC_i. Finally, a list of complete connections of MEC_i is the unique numbers among the connections on all sides.

$$c_i = u[lower_i, upper_i, left_i, right_i, fore_i, rear_i] \tag{4.29}$$



(a) 1st layer (b) 2nd layer (c) 3rd layer (d) 4th layer

Figure 4.15. Updated G has additional elements equal to zeros around it. Connections of MEC_1 is neighboring elements on each side (red elements).

For example, G is updated to a $(5 \times 5 \times 5)$ array, and each element is updated to its rank as shown in figure 4.15. Also, the coordinates of MEC_1 from equation 4.26a is changed for the updated G.

$$MEC_1^G = [3, 2, 2, 2, 2, 2] \tag{4.30}$$

The connections of MEC_1 on each side are calculated by equation 4.28.

$$lower_1 = u[G(3:4,\ \ 2:3,\ \ 1)] = [\ \cdot\ ] \tag{4.31a}$$

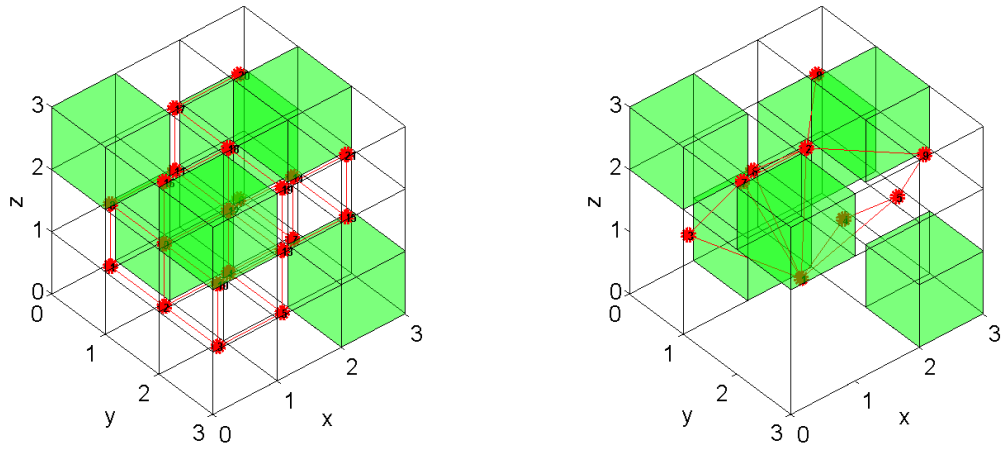
$$upper_1 = u[G(3:4,\ \ 2:3,\ \ 4)] = [2, 7] \tag{4.31b}$$

$$left_1 = u[G(3:4,\ \ 1,\ \ 2:3)] = [\ \cdot\ ] \tag{4.31c}$$

$$right_1 = u[G(3:4,\ \ 4,\ \ 2:3)] = [4, 5] \tag{4.31d}$$

$$fore_1 = u[G(5,\ \ 2:3,\ \ 2:3)] = [\ \cdot\ ] \tag{4.31e}$$

$$rear_1 = u[G(2,\ \ 2:3,\ \ 2:3)] = [3, 6] \tag{4.31f}$$



(a) G-map (b) R-map

Figure 4.16. Two different representations of a 3D environment. (a) G-map has 21 free elements and 66 connections. (b) R-map has 9 free elements and 26 connections.

Lists of complete connections of MEC_1 and the rest MECs are calculated by equation 4.29.

$$c_1 = [2, 3, 4, 5, 6, 7] \tag{4.32a}$$

$$c_2 = [1, 6, 7, 8, 9] \tag{4.32b}$$

$$c_3 = [1, 6] \tag{4.32c}$$

$$c_4 = [1, 5] \tag{4.32d}$$

$$c_5 = [1, 4, 9] \tag{4.32e}$$

$$c_6 = [1, 2, 3] \tag{4.32f}$$

$$c_7 = [1, 2] \tag{4.32g}$$

$$c_8 = [2] \tag{4.32h}$$

$$c_9 = [2, 5] \tag{4.32i}$$

For example, equation 4.32a means that MEC_1 is connected with MEC_2, MEC_3, MEC_4, MEC_5, MEC_6 and MEC_7, and equation 4.31 specifies their directions.

Therefore, the output of the R-map algorithm is a list of MECs and their connections. Figure 4.16 shows the initial G-map and the final R-map. Here, the G-map has 21 free elements (grid cells) and 66 connections, and the R-map has 9 free elements (MECs) and 26 connections.

4.5 Result

As a complex example, a 3D environment of New York City as shown in figure 4.17 is imported from Google Earth to Matlab. The results are shown in figures 4.18, 4.19, and 4.20 with three different map representations in different resolutions. Here, the three different map representations are G-map, O-map, and R-map. First, a G-map is a map from a mapping robot, and it has cube-elements in the even size. Second, an O-map (octree-decomposition map) is derived from a G-map, and it subdivides a space into eight octants if the space needs higher resolutions [39]. Thus, the cell elements in O-maps are cubes, but they can be in different sizes. Third, an R-map is similar with O-maps as reducing the number of cell elements, however while O-maps subdivide a space, R-maps integrate empty cells into maximal-sized cuboids. Thus, R-maps can have elements in various sizes and shapes of cuboids instead of cubes, and much fewer number of elements than G-map and O-map. Table 4.1 compares the three map representations in different resolutions. R-map has a dramatically reduced number of elements and their connections, and the differences with the other two maps get bigger as the resolution gets higher. However, the R-map requires an investment of computational time to construct. Notice that the Matlab code for the R-map is not optimized, and the R-mapping time would be different depending on the implementation of the algorithm in the code.

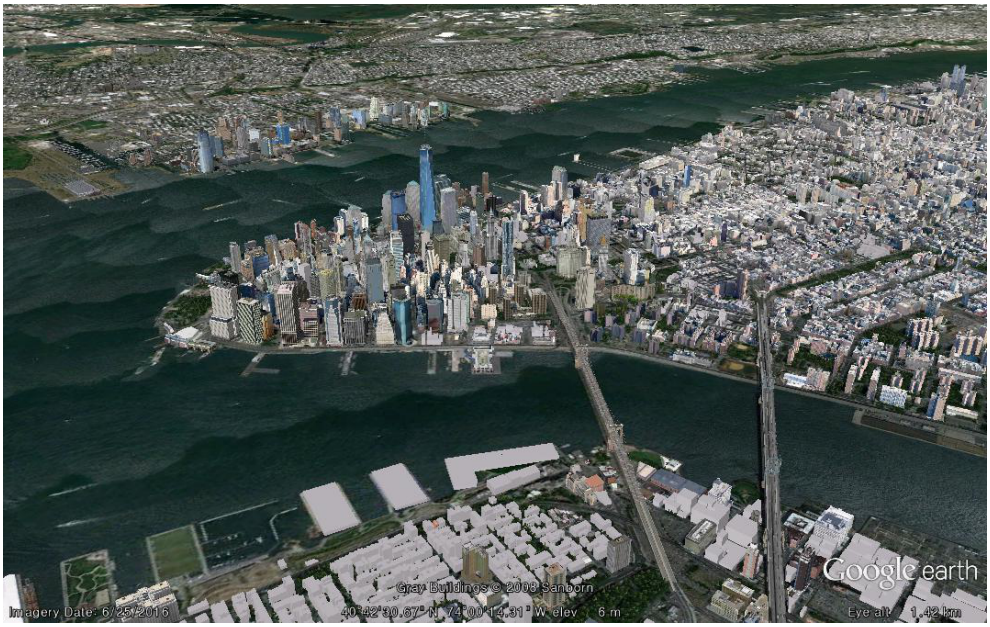
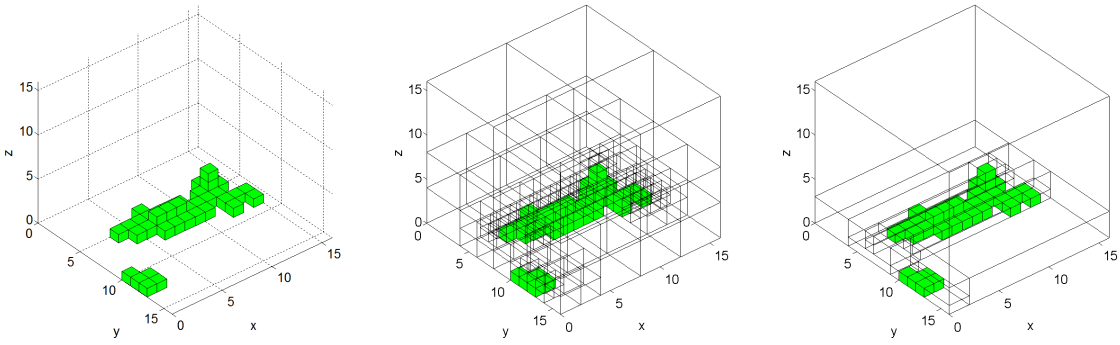


Figure 4.17. A 3D environment of New York City.

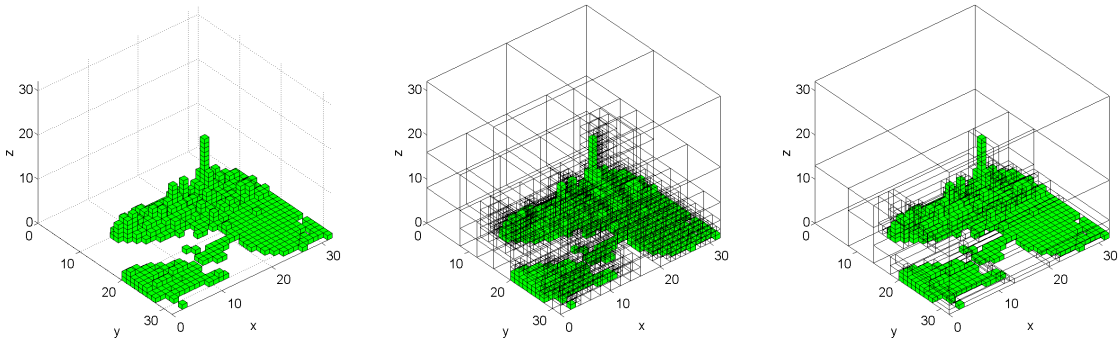
Table 4.1. Comparisons of mapping with different map representations.

Resolutions	**16×16×16**			**32×32×32**			**64×64×64**		
Map type	**G**	**O**	**R**	**G**	**O**	**R**	**G**	**O**	**R**
free elements	4,054	162	26	32,044	1,074	177	255,126	7,658	1,299
connections	22,724	938	140	185,458	6,326	1,072	1,498,296	43,970	8,176
mapping time (sec.)	-	0.98	1.32	-	8.58	38.99	-	229.19	1,807.26



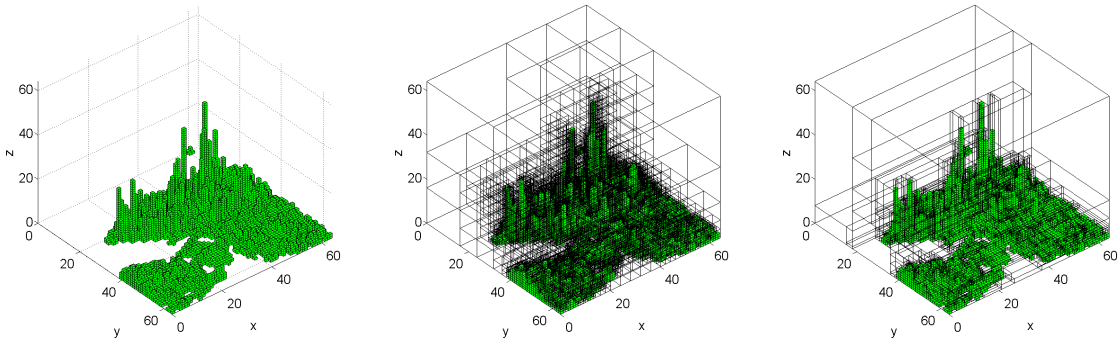
(a) G-map (grid removed) (b) O-map (c) R-map

Figure 4.18. Three different map representations in $(16 \times 16 \times 16)$ resolutions.



(a) G-map (grid removed) (b) O-map (c) R-map

Figure 4.19. Three different map representations in $(32 \times 32 \times 32)$ resolutions.



(a) G-map (grid removed) (b) O-map (c) R-map

Figure 4.20. Three different map representations in $(64 \times 64 \times 64)$ resolutions.

Chapter 5

3D Path Planning

Path planning considers free spaces in a map to find an optimal path avoiding obstacles between two points. Thus, computational performance of path planning depends on the number of free elements. Since R-maps have fewer free elements than conventional maps, a better performance is expected with R-maps. This Chapter explains how to determine a path from a start point to a finish point on an R-map. First, a series of connected cuboids between the two points will be selected by Dijkstra's algorithm [23]. The series of cuboids may differ by weight definitions. Second, the center points of sharing surfaces between the cuboids become waypoints, and an agent will pass through the waypoints to get to the finish point. Finally, two methods of interpolation are suggested to connect the waypoints to provide an exact path to follow. As results, path planning is accomplished on G-, O-, and R-maps of New York City to compare and discuss.

5.1 Selection of cuboids

The lists of dimensions and connections of cuboids (MECs) have acquired in the order of size from the R-mapping algorithm. To utilize the information in path planning problems, Dijkstra's algorithm is applied considering the cuboids as nodes and the connections as edges. Assume that there are start and finish points on a map and their locations are known, then it is easy to recognize which cuboids the points are within by simply looking at the corresponding elements of the points in the labeled G. Thus, start and finish nodes (cuboids) are defined, and Dijkstra's algorithm finds a series of nodes with the lowest cost from the start node to the finish node. Weights are assigned to the edges by weight definitions. Beginning at the start node, the costs to its neighbors are computed. Next, the lowest cost

node is defined as the current node, and the previous node is out of consideration. The total cost from the initial node to the current node is the summation of the costs along the path. There might be multiple paths to a certain node. In this case, the lowest-cost path will be the selected path and the others will be discarded.

Similar to 2D path planning, four weight definitions are considered again here. In each definition, $W(j,k)$ is a cost to travel from MEC_j to MEC_k.

Definition 5.1.1. $W_V(j,k) = k$. Weight by volume. Since the list of MECs is in the order of size, by simply giving a lower weight to a higher-ranked cuboid, the algorithm selects a path of larger volume.

Definition 5.1.2. $W_S(j,k) =$ (area of sharing surface between MEC_j and MEC_k). Weight by area of sharing surface. By giving a lower weight to an MEC, which has a larger area of sharing surface with its neighboring cuboid, the algorithm selects a wider path.

Definition 5.1.3. $W_D(j,k) =$ (distance between the center points of MEC_j and MEC_k). Weight by distance. By giving a lower weight to closer center points of two cuboids, the algorithm selects a shorter path to the destination, a classic application of Dijkstra's algorithm.

Definition 5.1.4. $W_N(j,k) = 1/$(the dimension of c_k). Weight by number of connections. To give a lower weight to a cuboid, which has more potential paths to the next cuboid, find the number of connections of each cuboid. Thus, the result is a path that has diverse alternative paths.

Note that $W_S(j,k) = W_S(k,j)$ and $W_D(j,k) = W_D(k,j)$, but $W_V(j,k) \neq W_V(k,j)$ and $W_N(j,k) \neq W_N(k,j)$ (cost going is not the same as cost returning).

As an example, recall the dimensions and connections of MECs from equations 4.26 and 4.32, and construct a relationship diagram as illustrated in figure 5.1. Here, the weights on the edges are the weight-by-distance (W_D). Also, the environment is a $(3 \times 3 \times 3)$ array, and define a start point at $(0.5, 0.5, 0.5)$ and a finish point at $(2.8, 2.8, 2.8)$, which are

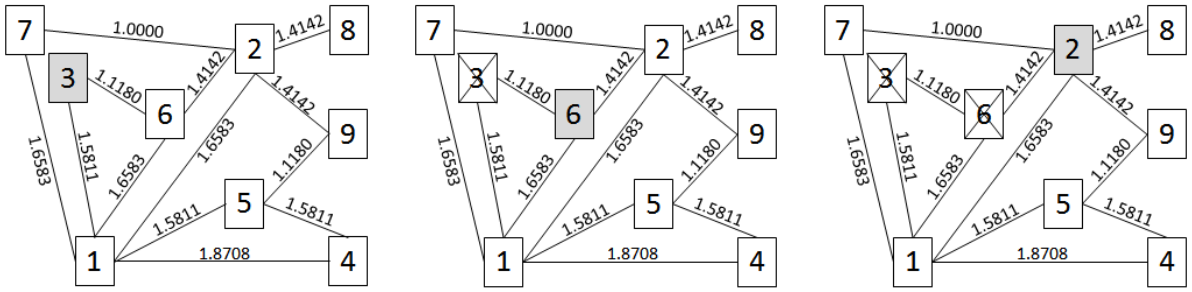


Figure 5.1. Dijkstra's algorithm selects the lowest-cost path to a finish point.

within cuboid-3 and cuboid-9, respectively. Thus, the path calculation begins at node-3 and continues until it considers node-9.

First, the current node is node-3 as shown in figure 5.1a, and the costs to its neighbors, nodes-1 and -6, are computed.

$$\text{At node-3,} \quad \text{for path 3-1,} \ \text{cost} = 1.5811 \tag{5.1a}$$

$$\text{for path 3-6,} \ \text{cost} = 1.1180 \tag{5.1b}$$

Next, previously considered node-3 is out of consideration, and the lowest-cost path in equation 5.1 becomes the current node, which is node-6 as shown in figure 5.1b. The total cost from the initial node via the current node to its neighbor is the summation of each weight on the path.

$$\text{At node-6,} \quad \text{for path 3-6-1,} \ \text{cost} = 1.1180 + 1.6583 = 2.7763 \ \text{(discard)} \tag{5.2a}$$

$$\text{for path 3-6-2,} \ \text{cost} = 1.1180 + 1.4142 = 2.5322 \tag{5.2b}$$

Here, since two different paths from node-3 to node-1 have been considered in equations 5.1a and 5.2a, select the lowest-cost path and discard the other as shown in equation 5.2a.

Next, node-6 is out of consideration, and the lowest-cost path, equation 5.2b, becomes the current node, node-2. The total costs to its neighbors are calculated in the same manner.

$$\text{At node-2,} \quad \text{for path 3-6-2-7,} \ \text{cost} = 1.1180 + 1.4142 + 1.000 = 3.5322 \tag{5.3a}$$

$$\text{for path 3-6-2-8,} \ \text{cost} = 1.1180 + 1.4142 + 1.4142 = 3.9464 \tag{5.3b}$$

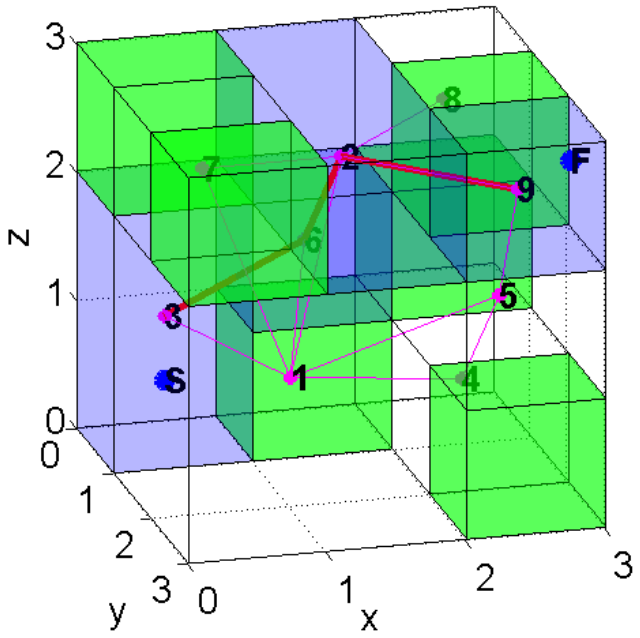
$$\text{for path 3-6-2-9,} \ \text{cost} = 1.1180 + 1.4142 + 1.4142 = 3.9464 \tag{5.3c}$$

Finally, the path between node-3 and node-9 is determined as computed in equation 5.3c.

Figure 5.2a represents the relationship with the nodes (pink dots) and the edges (pink line segments) in the 3D space. The start and the finish points are the blue dots with the letters 'S' and 'F', respectively, the selected cuboids are the blue cuboids. Also, the center points of the selected cuboids are connected with the red line segments. Figures 5.2b to 5.2d show the selected cuboids in 2D matrices.

5.2 Waypoint navigation

An agent can travel to the finish point by moving freely through the selected cuboids. However, for efficient travelling, it would be better if the agent has an exact path to follow. To make sure the path is inside of the selected cuboids and the agent travels avoiding obstacles, control points or waypoints are defined. Here, the waypoints are the center points of the sharing surfaces between two cuboids. For example, in figure 5.2c, cuboid-3 and cuboid-6 have a sharing surface between the first and the second columns in the first row, thus the waypoint between the two cuboids is the center point of the sharing surface. Table 5.1 shows the waypoints of the selected cuboids in figure 5.2a. Such waypoints will be connected by interpolation methods: linear interpolation for straight line segments, and spline interpolation for smooth curve segments.



(a) 3D view

x

y	1	2	3
1	0	2	8
2	7	2	0
3	0	2	9

(b) 3rd layer

x

y	1	2	3
1	3	6	0
2	1	1	5
3	1	1	5

(c) 2nd layer

x

y	1	2	3
1	3	0	4
2	1	1	4
3	1	1	0

(d) 1st layer

Figure 5.2. The start point is in cuboid-3, and the finish point is in cuboid-9. A series of cuboids that connects cuboids-3 and cuboid-9 in the shortest distance is 3-6-2-9.

Table 5.1. Waypoints in figure 5.2a.

Waypoint	**Coordinates**			**Cuboid**
i	x_i	y_i	z_i	
1	0.5	0.5	0.5	3
2	1.0	0.5	1.5	3, 6
3	1.5	0.5	2.0	6, 2
4	2.0	2.5	2.5	2, 9
5	2.8	2.8	2.8	9

5.3 Linear interpolation

Linear interpolation connects $(n+1)$ waypoints with n line segments. Each line segment is expressed as $L_i(s)$.

$$L_i(s) = x(s)\hat{i} + y(s)\hat{j} + z(s)\hat{k} \tag{5.4}$$

Here, $s \in [0, 1]$ and $i = 1, \cdots, n$. As illustrated in figure 5.3, each line segment starts at $s = 0$ and ends at $s = 1$. To specify the coefficient equations, $x(s)$, $y(s)$, and $z(s)$, first define $\vec{v}$

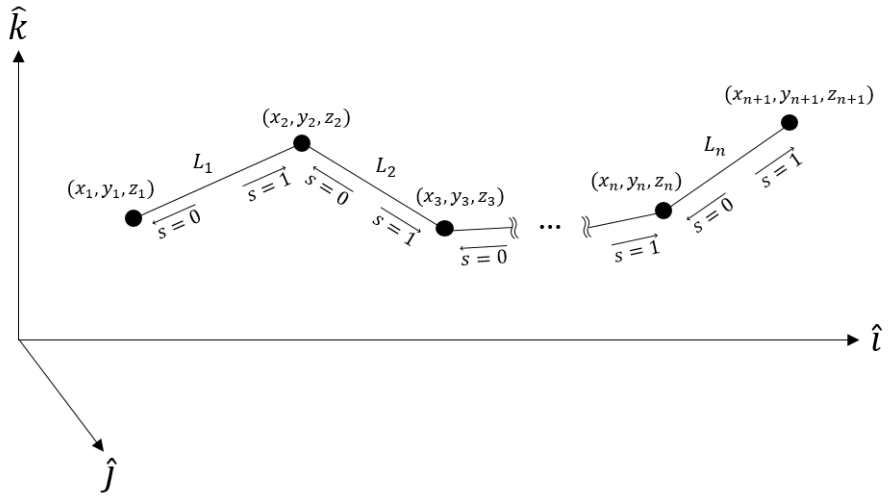


Figure 5.3. The $(n+1)$ waypoints are connected by n line segments. Each line segment can be found by interpolation methods.

as a direction vector, which has the same direction with a line between two points.

$$\vec{v} = r_{i+1} - r_i = (x_{i+1} - x_i)\hat{i} + (y_{i+1} - y_i)\hat{j} + (z_{i+1} - z_i)\hat{k} \tag{5.5}$$

Then, s times of $\vec{v}$ will be the same magnitude with the line. Finally, by moving to the point i, the vector will be identical with the line segment, L_i.

$$L_i(s) = \vec{v}s + r_i \tag{5.6}$$

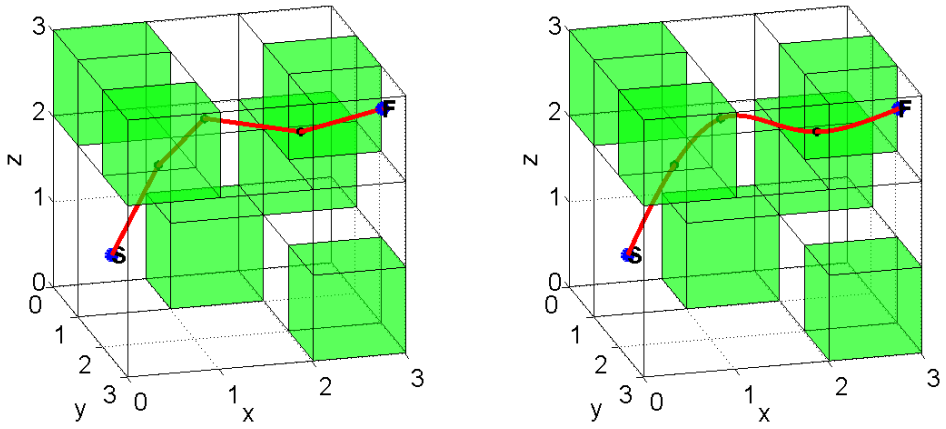
Therefore, each coefficient term in equation 5.4 can be obtained.

$$x(s) = (x_{i+1} - x_i)s + x_i \tag{5.7a}$$

$$y(s) = (y_{i+1} - y_i)s + y_i \tag{5.7b}$$

$$z(s) = (z_{i+1} - z_i)s + z_i \tag{5.7c}$$

For example, using the waypoints from table 5.1, equations of line segments are given. There are five waypoints and four line segments.



(a) Linear interpolation (b) Spline interpolation

Figure 5.4. Two methods of waypoint navigation.

$$L_1(s) = \begin{bmatrix} x(s) \\ y(s) \\ z(s) \end{bmatrix} = \begin{bmatrix} 0.5s + 0.5 \\ 0.5 \\ 1.0s + 0.5 \end{bmatrix} \begin{matrix} \text{if } 0.5 \leq x \leq 1.0 \\ \text{if } 0.5 \leq y \leq 0.5 \\ \text{if } 0.5 \leq z \leq 1.5 \end{matrix} \tag{5.8}$$

$$L_2(s) = \begin{bmatrix} x(s) \\ y(s) \\ z(s) \end{bmatrix} = \begin{bmatrix} 0.5s + 1.0 \\ 0.5 \\ 0.5s + 1.5 \end{bmatrix} \begin{matrix} \text{if } 1.0 \leq x \leq 1.5 \\ \text{if } 0.5 \leq y \leq 0.5 \\ \text{if } 1.5 \leq z \leq 2.0 \end{matrix} \tag{5.9}$$

$$L_3(s) = \begin{bmatrix} x(s) \\ y(s) \\ z(s) \end{bmatrix} = \begin{bmatrix} 0.5s + 1.5 \\ 2.0s + 0.5 \\ 0.5s + 2.0 \end{bmatrix} \begin{matrix} \text{if } 1.5 \leq x \leq 2.0 \\ \text{if } 0.5 \leq y \leq 2.5 \\ \text{if } 2.0 \leq z \leq 2.5 \end{matrix} \tag{5.10}$$

$$L_4(s) = \begin{bmatrix} x(s) \\ y(s) \\ z(s) \end{bmatrix} = \begin{bmatrix} 0.8s + 2.0 \\ 0.3s + 2.5 \\ 0.3s + 2.5 \end{bmatrix} \begin{matrix} \text{if } 2.0 \leq x \leq 2.8 \\ \text{if } 2.5 \leq y \leq 2.8 \\ \text{if } 2.5 \leq z \leq 2.8 \end{matrix} \tag{5.11}$$

Finally, equations 5.8 to 5.11 provide a path to follow as shown in figure 5.4a.

5.4 Spline interpolation

Spline interpolation connects $(n+1)$ waypoints with n curve segments. Each curve segment is expressed as $S_i(s)$.

$$S_i(s) = x(s)\hat{i} + y(s)\hat{j} + z(s)\hat{k} \tag{5.12}$$

Here, $s \in [0,1]$ and $i = 1, \cdots, n$. Figure 5.3 is applied again, however the notation of the line segments will be S_i instead of L_i. The coefficient terms in equation 5.12 are cubic functions as a path will be curves.

$$x(s) = a_x s^3 + b_x s^2 + c_x s + d_x \tag{5.13a}$$

$$y(s) = a_y s^3 + b_y s^2 + c_y s + d_y \tag{5.13b}$$

$$z(s) = a_z s^3 + b_z s^2 + c_z s + d_z \tag{5.13c}$$

Thus, total n line segments have $12n$ unknown coefficients. Those coefficients can be determined by several constraints.

First, curves should pass through interior waypoints, which means that each interior waypoint has two line segments from both directions.

$$S_i(1) = S_{i+1}(0) = x_{i+1}\hat{i} + y_{i+1}\hat{j} + z_{i+1}\hat{k} \tag{5.14}$$

Here, $i = 1$ to $(n-1)$. This constraint makes $(2n-2)$ equations.

Second, the first and the last line segments should pass the start and the finish points, respectively

$$S_1(0) = x_1\hat{i} + y_1\hat{j} + z_1\hat{k} \tag{5.15a}$$

$$S_n(1) = x_{n+1}\hat{i} + y_{n+1}\hat{j} + z_{n+1}\hat{k} \tag{5.15b}$$

This constraint makes 2 equations.

Third, the first derivatives of two curves at each interior waypoint should be the same.

$$S_i'(1) = S_{i+1}'(0) \tag{5.16}$$

Here, $i = 1$ to $(n-1)$. This constraint makes $(n-1)$ equations.

Fourth, the second derivatives of two curves at each interior waypoint should be the same.

$$S_i''(1) = S_{i+1}''(0) \tag{5.17}$$

Here $i = 1$ to $(n-1)$. This constraint makes $(n-1)$ equations.

Fifth, the second derivatives at the start and the finish points should be zeros.

$$S_1''(0) = 0 \tag{5.18a}$$

$$S_n''(1) = 0 \tag{5.18b}$$

This constraint makes 2 equations.

The five constraints make $4n$ of S_i equations and each of them consists of three of $x(s)$, $y(s)$, and $z(s)$ equations, thus $(4n \times 3)$ equations are enough to find $12n$ unknown coefficients.

As an example, using the waypoints from table 5.1, equations of curve segments are provided.

$$S_1(s) = \begin{bmatrix} x(s) \\ y(s) \\ z(s) \end{bmatrix} = \begin{bmatrix} 0.0530s^3 + 0.0000s^2 + 0.4137s + 0.5000 & \text{if } 0.5 \leq x \leq 1.0 \\ -0.0894s^3 + 0.0000s^2 + 0.0999s + 0.5000 & \text{if } 0.5 \leq y \leq 0.5 \\ -0.0788s^3 + 0.0000s^2 + 1.0338s + 0.5000 & \text{if } 0.5 \leq z \leq 1.5 \end{bmatrix} \quad (5.19)$$

$$S_2(s) = \begin{bmatrix} x(s) \\ y(s) \\ z(s) \end{bmatrix} = \begin{bmatrix} -0.1951s^3 + 0.1680s^2 + 0.5913s + 1.0000 & \text{if } 1.0 \leq x \leq 1.5 \\ 0.6198s^3 - 0.2835s^2 - 0.1999s + 0.5000 & \text{if } 0.5 \leq y \leq 0.5 \\ 0.0496s^3 - 0.2498s^2 + 0.7696s + 1.5000 & \text{if } 1.5 \leq z \leq 2.0 \end{bmatrix} \quad (5.20)$$

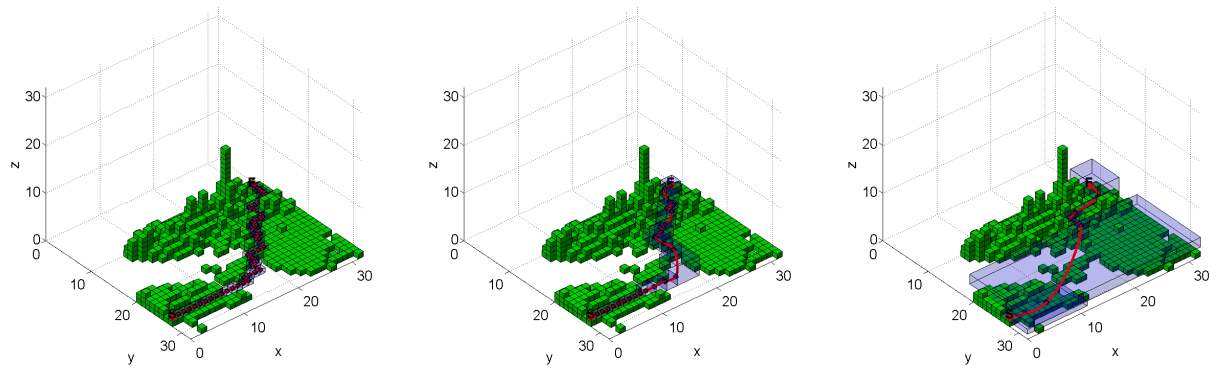
$$S_3(s) = \begin{bmatrix} x(s) \\ y(s) \\ z(s) \end{bmatrix} = \begin{bmatrix} 0.1675s^3 - 0.3241s^2 + 0.4600s + 1.5000 & \text{if } 1.5 \leq x \leq 2.0 \\ -0.5324s^3 + 1.2801s^2 + 0.6381s + 0.5000 & \text{if } 0.5 \leq y \leq 2.5 \\ 0.0332s^3 - 0.1248s^2 + 0.4546s + 2.0000 & \text{if } 2.0 \leq z \leq 2.5 \end{bmatrix} \quad (5.21)$$

$$S_4(s) = \begin{bmatrix} x(s) \\ y(s) \\ z(s) \end{bmatrix} = \begin{bmatrix} -0.1429s^3 + 0.4078s^2 + 0.5820s + 2.0000 & \text{if } 2.0 \leq x \leq 2.8 \\ 0.3664s^3 - 1.0461s^2 + 0.9789s + 2.5000 & \text{if } 2.5 \leq y \leq 2.8 \\ -0.0071s^3 + 0.0203s^2 + 0.3024s + 2.5000 & \text{if } 2.5 \leq z \leq 2.8 \end{bmatrix} \quad (5.22)$$

The path calculated from equation 5.19 to 5.22 is illustrated in figure 5.4b.

5.5 Result

A series of connected cuboids between two points can be selected by Dijkstra's algorithm, also two interpolation methods to calculate an exact path passing through the cuboids avoiding obstacles are suggested. As results, this process is applied to New York City environment in figure 5.5. This figure shows paths on the three different map representations with the weight-by-distance (W_D), and their resolutions are ($32 \times 32 \times 32$). The R-map



(a) G-map (b) O-map (c) R-map

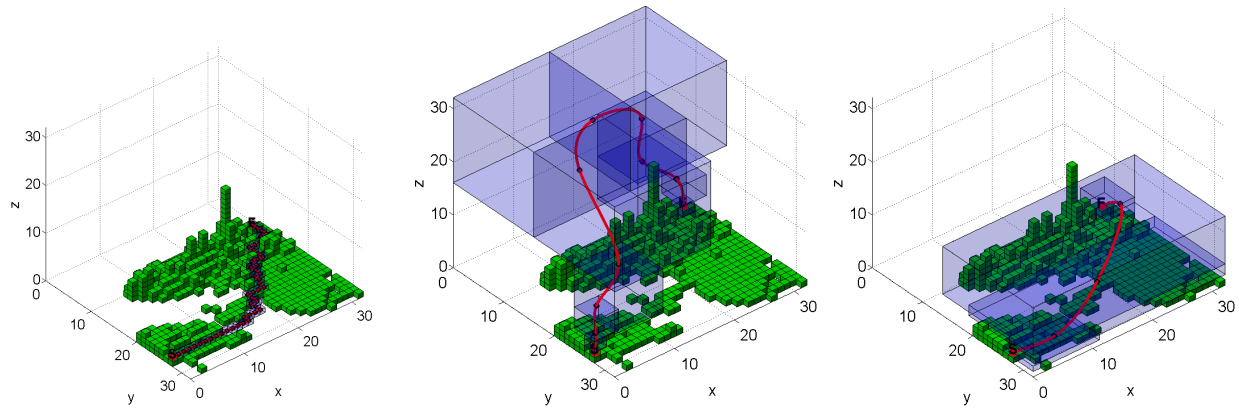
Figure 5.5. Path planning with weight by distance, W_D, in $(32 \times 32 \times 32)$ resolutions.

Table 5.2. Comparisons of mapping with different map representations.

Resolutions	**16×16×16**			**32×32×32**			**64×64×64**		
Map type	**G**	**O**	**R**	**G**	**O**	**R**	**G**	**O**	**R**
free elements	4,054	162	26	32,044	1,074	177	255,126	7,658	1,299
connections	22,724	938	140	185,458	6,326	1,072	1,498,296	43,970	8,176
mapping time (sec.)	-	0.98	1.32	-	8.58	38.99	-	229.19	1,807.26
path planning time (sec.)	1.17	0.13	0.07	8.11	0.38	0.13	71.06	2.00	0.51

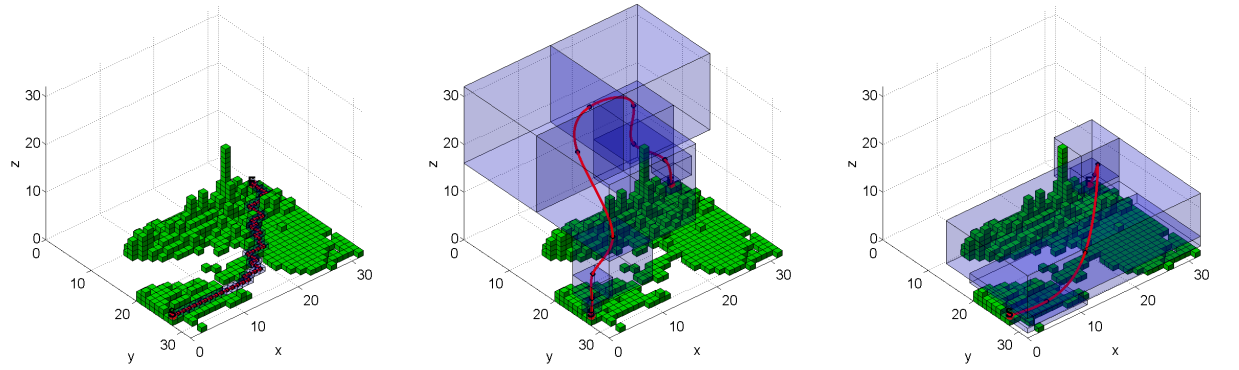
has advantages in path planning comparing with the other maps. As mentioned in 3D R-mapping section, R-maps require more computational time in mapping than the other maps because of additional algorithms. However, as shown in table 5.2, path planning on R-maps is the fastest among the three map types due to a reduced number of elements, and the differences in time become larger as the resolution rises. Also, the larger volume of elements in R-maps allows smoother curves and fewer turns of a path. Therefore, for certain vehicles or applications, the paths generated from the R-map may be inherently desirable. Considering the combined computational expenses of mapping and path planning, the R-map may be more efficient for applications that require multiple path planning operations in a given environment. For the $(16 \times 16 \times 16)$ resolution, the R-map approach would be more efficient than the G-map or O-map for any application requiring 6 or more paths to be planned. For the $(64 \times 64 \times 64)$ resolution, the R-map approach is more efficient for any application requiring 1,060 or more paths.

The results of path planning using the other three weights ($W_V, W_S,$ and W_N) are also shown in figures 5.6 to 5.8. Some paths produced by those weights can be similar, because a cuboid with a larger volume has a bigger surface area and more connections to its neighboring cuboids.



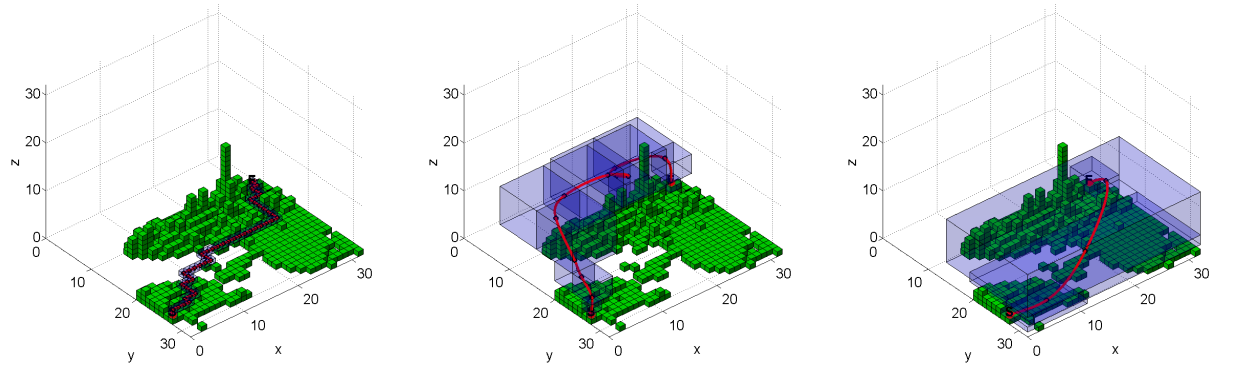
(a) G-map (b) O-map (c) R-map

Figure 5.6. Path planning with weight by volume, W_V, in $(32 \times 32 \times 32)$ resolutions.



(a) G-map (b) O-map (c) R-map

Figure 5.7. Path planning with weight by area of sharing surface, W_S, in $(32 \times 32 \times 32)$ resolutions.



(a) G-map (b) O-map (c) R-map

Figure 5.8. Path planning with weight by number of connections, W_N, in $(32 \times 32 \times 32)$ resolutions.

Chapter 6

Target Interception

This Chapter considers the guidance problem for target interception. The approach here is to treat the problem as path planning to a moving target. Conventional strategies such as pursuit and proportional guidance laws are still widely used in ground and aerial navigation systems [25–35]. However, this problem becomes challenging in a cluttered environment to perform the interception while avoiding obstacles. For target interception in a cluttered environment, the guidance laws can be implemented on the R-map in a receding horizon approach, which takes advantage of the computationally efficient path planning. Target interception approaches used here have two assumptions: a map environment and target's dynamic information are known. Under the same assumptions, other approaches focus on obstacles by means of keeping a safety distance between a pursuer and obstacles to avoid collision [40, 41], however R-map approaches focus on empty spaces and a pursuer moves along the selected cuboids. This Chapter has three sections. First, the traditional navigation methods are reviewed and simulated in New York City environment. Second, two approaches to determine the desired final location to use the navigation method on the R-map are provided. Finally, both traditional and R-map approaches are compared and discussed.

6.1 Traditional Approaches

Conventional navigation methods such as pursuit and proportional navigation are easy to develop and implement in navigation systems. However, those methods do not consider the presence of obstacles in the environment. In this section, pure pursuit and pure proportional navigation methods are briefly introduced and simulated to compare with the navigation on R-maps in the last section.

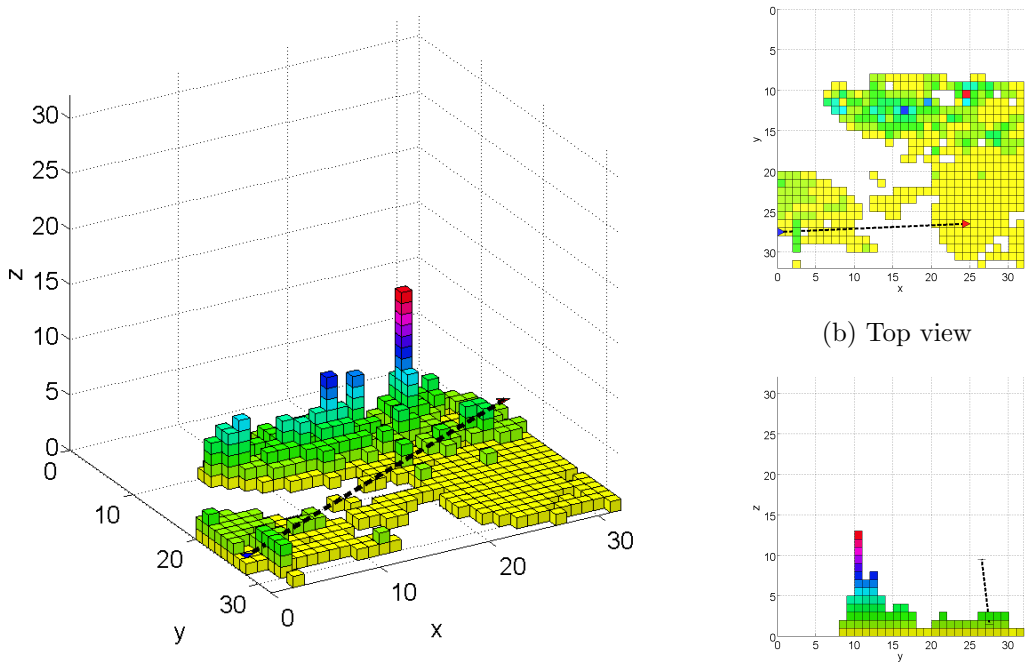
6.1.1 Pursuit Navigation

Pursuit navigation is a method to guide a pursuer toward a target. Specifically, the pursuer's location is a start point and the target's location is a finish point, and path planning is simple as finding a line segment between the two points. Accordingly, the pursuer's velocity always points toward the target's current location, and its trajectory will be a target's tail-tracking path.

Figure 6.1 shows a simulation at t_0. Here, the pursuer (blue triangle) and the target (red triangle) are presented in the environment of New York City. The computed interception-path (black dashed-line) is a straight line segment between the pursuer and the target. As the target moves, the interception path is updated, and the pursuer follows the path until the next update. Figure 6.2 shows the simulation at the interception moment, t_I. Here, the trajectories of the pursuer (blue line) and the target (red line) are illustrated. The speeds of the target and the pursuer are constant as 1.0 and 1.5 3D grid cells per time unit, respectively.

6.1.2 Proportional Navigation

Proportional navigation is often implemented as a guidance law giving a commanded acceleration to drive the line-of-sight rate to zero, steering the pursuer on to an interception triangle. The implementation used here assumes the current position and the velocity of the target are known. Additionally, the speeds of the pursuer and the target are assumed to be constant, and the direction of the pursuer's velocity can be arbitrarily selected. A simulation will be conducted against a maneuvering target. The approach to find a commanded acceleration, $\boldsymbol{a}_c$, in a 3D environment is referred from Moran [37].

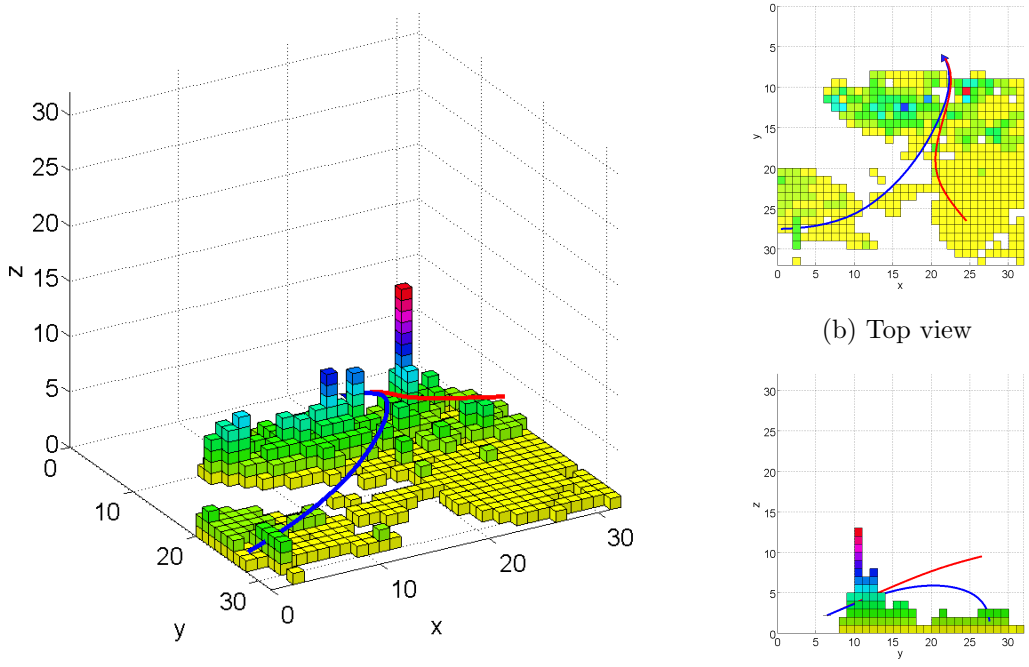


(a) 3D view

(b) Top view

(c) Side view

Figure 6.1. Pure pursuit navigation: the computed interception path at t_0.



(a) 3D view

(b) Top view

(c) Side view

Figure 6.2. Pure pursuit navigation: the final trajectories at t_I.

Initially, the positions, $\boldsymbol{r}_P$ and $\boldsymbol{r}_T$, and the velocities, $\boldsymbol{v}_P$ and $\boldsymbol{v}_T$, of a pursuer and a target are defined.

$$\boldsymbol{r}_P = \begin{bmatrix} r_{Px} \\ r_{Py} \\ r_{Pz} \end{bmatrix}, \qquad \boldsymbol{r}_T = \begin{bmatrix} r_{Tx} \\ r_{Ty} \\ r_{Tz} \end{bmatrix} \tag{6.1a}$$

$$\boldsymbol{v}_P = \begin{bmatrix} v_{Px} \\ v_{Py} \\ v_{Pz} \end{bmatrix}, \qquad \boldsymbol{v}_T = \begin{bmatrix} v_{Tx} \\ v_{Ty} \\ v_{Tz} \end{bmatrix} \tag{6.1b}$$

Here, the subscripts x, y, and z indicate elements associated with corresponding axis.

The relative position and the relative velocity of the target respect to the pursuer, $\boldsymbol{r}_{TP}$ and $\boldsymbol{v}_{TP}$, are the subtractions of the vectors from equation 6.1.

$$\boldsymbol{r}_{TP} = \begin{bmatrix} r_{TPx} \\ r_{TPy} \\ r_{TPz} \end{bmatrix} = \boldsymbol{r}_T - \boldsymbol{r}_P \tag{6.2a}$$

$$\boldsymbol{v}_{TP} = \begin{bmatrix} v_{TPx} \\ v_{TPy} \\ v_{TPz} \end{bmatrix} = \boldsymbol{v}_T - \boldsymbol{v}_P \tag{6.2b}$$

Also, the distance between the pursuer and the target projected on the three planes is defined as $\boldsymbol{D}$.

$$\boldsymbol{D} = \begin{bmatrix} D_{xy} \\ D_{yz} \\ D_{zx} \end{bmatrix} = \begin{bmatrix} \sqrt{r_{TPx}^2 + r_{TPy}^2} \\ \sqrt{r_{TPy}^2 + r_{TPz}^2} \\ \sqrt{r_{TPz}^2 + r_{TPx}^2} \end{bmatrix} \tag{6.3}$$

Then, the closing velocity, $\boldsymbol{v}_c$, projected on the planes are negative changing rate of the distance.

$$\boldsymbol{v}_c = \begin{bmatrix} v_{cxy} \\ v_{cyz} \\ v_{czx} \end{bmatrix} = -\dot{\boldsymbol{D}} = \begin{bmatrix} -\frac{r_{TPx}v_{TPx}+r_{TPy}v_{TPy}}{D_{xy}} \\ -\frac{r_{TPy}v_{TPy}+r_{TPz}v_{TPz}}{D_{yz}} \\ -\frac{r_{TPz}v_{TPz}+r_{TPx}v_{TPx}}{D_{zx}} \end{bmatrix} \tag{6.4}$$

The line-of-sight (LOS) angle, $\boldsymbol{\lambda}$, from the pursuer to the target projected on the planes is

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_{xy} \\ \lambda_{yz} \\ \lambda_{zx} \end{bmatrix} = atan \begin{bmatrix} \frac{r_{TPy}}{r_{TPx}} \\ \frac{r_{TPz}}{r_{TPy}} \\ \frac{r_{TPx}}{r_{TPz}} \end{bmatrix} \tag{6.5}$$

and its changing rate is computed as

$$\dot{\boldsymbol{\lambda}} = \begin{bmatrix} \dot{\lambda}_{xy} \\ \dot{\lambda}_{yz} \\ \dot{\lambda}_{zx} \end{bmatrix} = \begin{bmatrix} \frac{r_{TPx}v_{TPy}-r_{TPy}v_{TPx}}{r_{TPx}^2+r_{TPy}^2} \\ \frac{r_{TPy}v_{TPz}-r_{TPz}v_{TPy}}{r_{TPy}^2+r_{TPz}^2} \\ \frac{r_{TPz}v_{TPx}-r_{TPx}v_{TPz}}{r_{TPz}^2+r_{TPx}^2} \end{bmatrix} \tag{6.6}$$

Finally, the acceleration command, $\boldsymbol{a}_c$, projected on the planes is obtained by the multiplication of the closing velocity and the LOS rate.

$$\boldsymbol{a}_c = \begin{bmatrix} a_{cxy} \\ a_{cyz} \\ a_{czx} \end{bmatrix} = N\boldsymbol{v}_c \circ \dot{\boldsymbol{\lambda}} \tag{6.7}$$

Here, N is the proportionality constant that is generally between 3 and 5. Also, the circle notation between $\boldsymbol{v}_c$ and $\dot{\boldsymbol{\lambda}}$ is the element-wise multiplication (or Hadamard product), which multiplies element by element of two vectors. Notice that equation 6.7 is for non-maneuvering target. For maneuvering target, target's acceleration term is added to $\boldsymbol{a}_c$. This guidance law

for maneuvering target is called augmented proportional navigation (APN) [36].

$$\boldsymbol{a}_c = \begin{bmatrix} a_{cxy} \\ a_{cyz} \\ a_{czx} \end{bmatrix} = N\boldsymbol{v}_c \circ \dot{\boldsymbol{\lambda}} + \frac{N\boldsymbol{a}_T}{2} \tag{6.8}$$

Here, $\boldsymbol{a}_T$ is the target's acceleration. The elements of the acceleration command in equation 6.8 can be separated to the elements associated with the x-, y-, and z-axis by trigonometry, thus the acceleration along each axis will be the summation of associated elements, which becomes the pursuer's acceleration, $\boldsymbol{a}_P$.

$$\boldsymbol{a}_P = \begin{bmatrix} a_{Px} \\ a_{Py} \\ a_{Pz} \end{bmatrix} = \begin{bmatrix} a_{cxy}sin(\lambda_{xy}) - a_{czx}cos(\lambda_{zx}) \\ a_{cyz}sin(\lambda_{yz}) - a_{cxy}cos(\lambda_{xy}) \\ -a_{czx}sin(\lambda_{xz}) + a_{cyz}cos(\lambda_{yz}) \end{bmatrix} \tag{6.9}$$

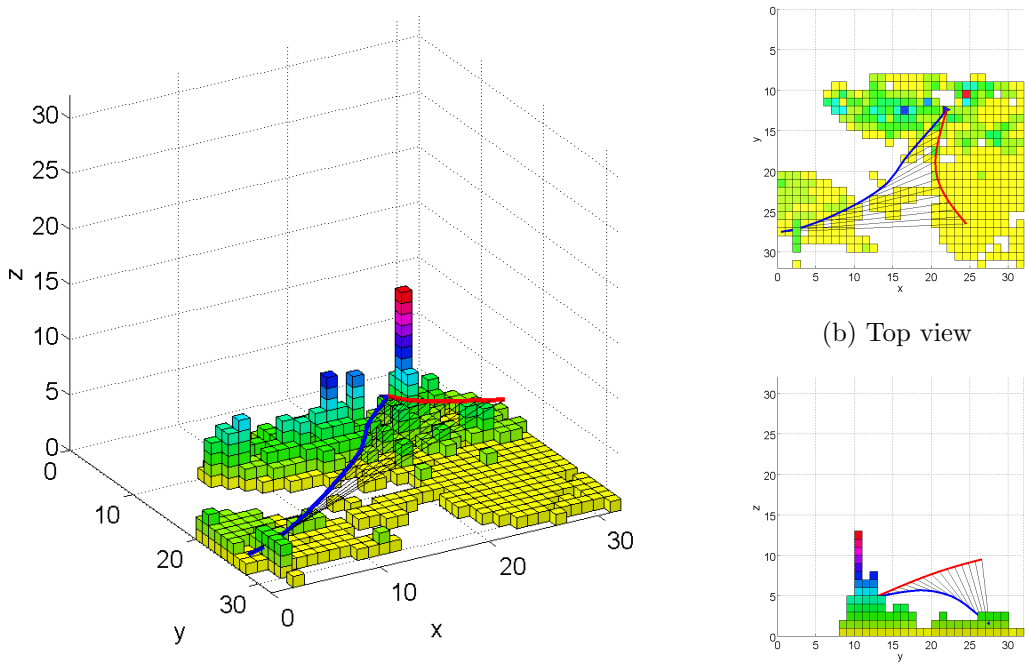
Therefore, the pursuer's next required velocity and position to keep the pursuer on the interception path are calculated by the equations of motion using the current dynamic information, $\boldsymbol{r}_P$, $\boldsymbol{v}_P$, and $\boldsymbol{a}_P$ from equations 6.1 and 6.9.

$$\boldsymbol{v}_{P(i+1)} = \boldsymbol{a}_{P(i)}dt + \boldsymbol{v}_{P(i)} \tag{6.10a}$$

$$\boldsymbol{r}_{P(i+1)} = \boldsymbol{r}_{P(i)} + \frac{1}{2}(\boldsymbol{v}_{P(i+1)} + \boldsymbol{v}_{P(i)})dt \tag{6.10b}$$

Here, dt is a time interval, and the subscripts (i) and $(i+1)$ mean the current and the next values after dt, respectively. This process is repeated as the target's dynamic information is updated.

A simulation is shown in figure 6.3. The figure shows the trajectories of the pursuer (blue line) and the target (red line) at the interception moment, t_I. The LOS (black line



(a) 3D view

(b) Top view

(c) Side view

Figure 6.3. Pure proportional navigation: the final trajectories and the LOS at t_I.

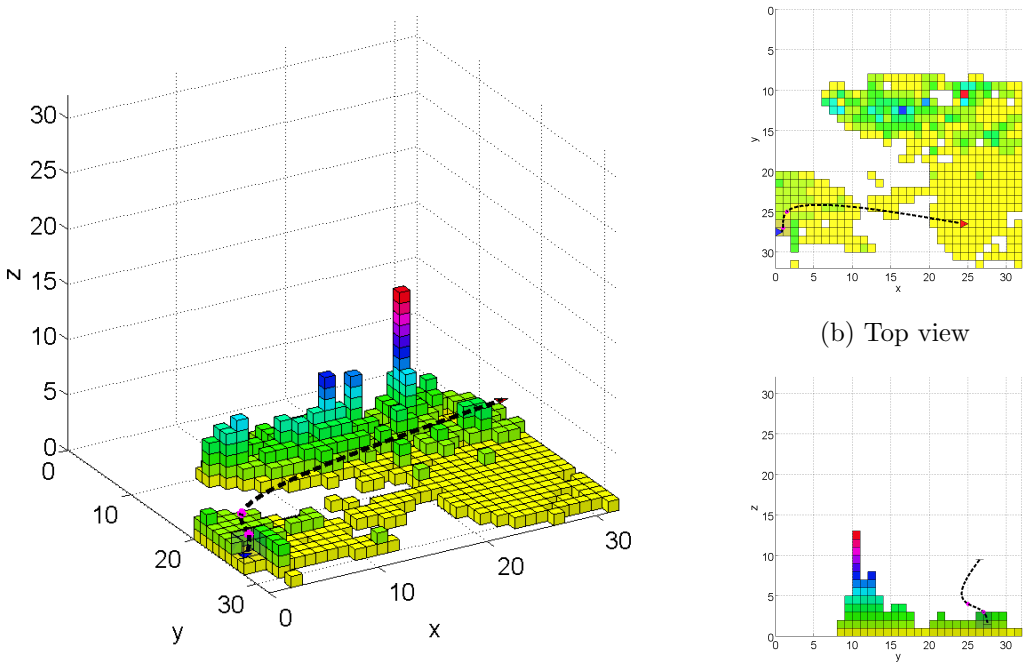
segments) at each sensing moment is also illustrated. The speeds of the target and the pursuer are constant as 1.0 and 1.5 3D grid cells per time unit, respectively.

6.2 R-map Approaches

Obstacle avoidance can be achieved by applying the R-map based path planning. The process of path planning at each planning iteration has two steps: first, select a desired final location, and second, apply the path planning method from the previous Chapter. In this section, two methods to determine the desired final location are provided.

6.2.1 Pursuit Navigation

As described in the previous section, the desired final location in pursuit navigation is the target's current location. With the two points of the start and the finish, the path planning method from the previous Chapter can be applied to plan an obstacle collision-free

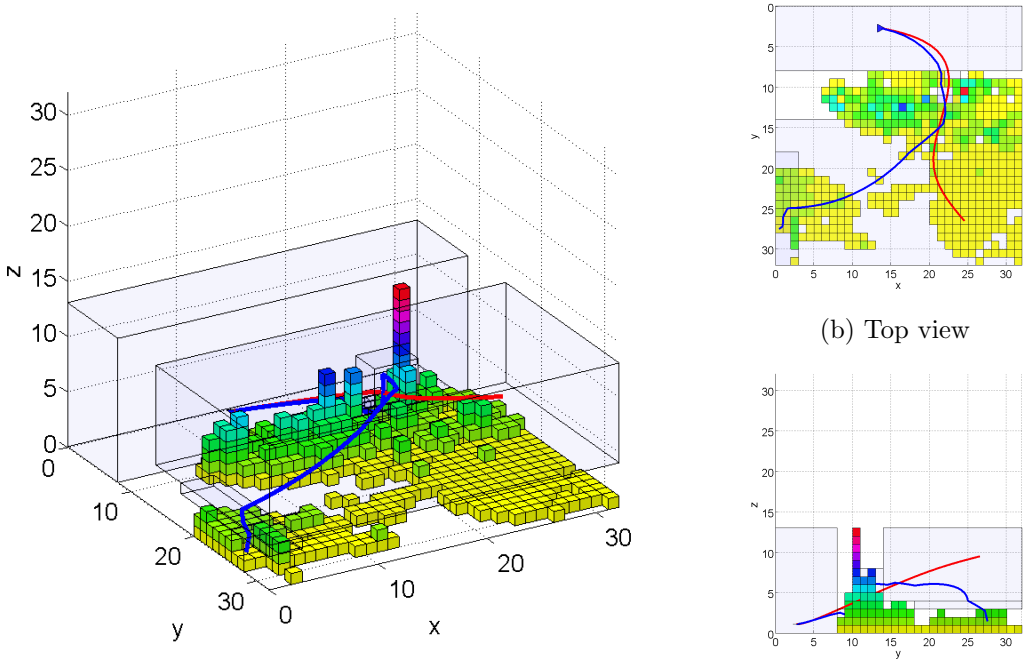


(a) 3D view

(b) Top view

(c) Side view

Figure 6.4. Pursuit navigation on R-map: the computed interception path at t_0.



(a) 3D view

(b) Top view

(c) Side view

Figure 6.5. Pursuit navigation on R-map: the final trajectories at t_I.

path between the two points. This planning process is repeated as the target's location is updated.

Figure 6.4 shows a simulation at t_0. The computed interception-path (black dashed-line) passes the center points (pink dots) of the selected cuboids, thus it is an obstacle collision-free path. Also, the cuboids that the pursuer has been inside are illustrated as blue transparent-cuboids. As the target moves, the interception path is updated, and the pursuer follows it until the next planning iteration. Figure 6.5 shows the simulation at the interception moment, t_I. Here, the pursuer's trajectory (blue line) is obstacle collision-free. The speeds of the target and the pursuer are constant as 1.0 and 1.5 3D grid cells, respectively.

6.2.2 Interception-Triangle Navigation

For pursuit navigation, it is actually desired for the pursuer to be at this location at the current time. However, the interception triangle determines the desired final location and an actual pursuer trajectory that can be followed over time if the target is moving with constant velocity. The presence of obstacles, though, forces deviation from this path, and requires a receding horizon implementation. The detailed process is given here. It is assumed that the target's velocity and the pursuer's speed are known.

The target's location at time t is defined with the initial position, $\boldsymbol{r}_{T,0}$, and the assumed constant velocity, $\boldsymbol{v}_T$.

$$\boldsymbol{r}_T(t) = \boldsymbol{r}_{T,0} + \boldsymbol{v}_T t \tag{6.11}$$

Also, the pursuer's location at time t is defined with the initial position, $\boldsymbol{r}_{P,0}$, and the constant velocity, $\boldsymbol{v}_P = v_P \hat{\boldsymbol{l}}$.

$$\boldsymbol{r}_P(t) = \boldsymbol{r}_{P,0} + \boldsymbol{v}_P t \tag{6.12}$$

Here, the magnitude v_P is known, but the direction $\hat{\boldsymbol{l}}$ is unknown. Solve for $\hat{\boldsymbol{l}}$ to make the pursuer intercept the target at an unknown interception time, t_I.

$$\boldsymbol{r}_P(t_I) = \boldsymbol{r}_T(t_I) \tag{6.13a}$$

$$\boldsymbol{r}_{P,0} + v_P\hat{\boldsymbol{l}}t_I = \boldsymbol{r}_{T,0} + \boldsymbol{v}_T t_I \tag{6.13b}$$

This vector equations can be expressed in components of an arbitrary coordinate frame to produce three quadratic equations in terms of the unknowns l_1, l_2, l_3 and t_I. The constraint that $l_1^2 + l_2^2 + l_3^2 = 1$ gives a fourth quadratic equation. Solving a system of four quadratic equations for four unknowns is generally difficult. However, the process can be simplified by selecting an advantageous coordinate system. First, equation 6.13b is rearranged.

$$\left(v_P\hat{\boldsymbol{l}} - \boldsymbol{v}_T\right) t_I = \boldsymbol{r}_{T,0} - \boldsymbol{r}_{P,0} \tag{6.14}$$

Next, the coordinate system is defined as

$$\hat{\boldsymbol{e}}_1 = \frac{\boldsymbol{r}_{T,0} - \boldsymbol{r}_{P,0}}{\|\boldsymbol{r}_{T,0} - \boldsymbol{r}_{P,0}\|} \tag{6.15a}$$

$$\hat{\boldsymbol{e}}_2 = \hat{\boldsymbol{e}}_3 \times \hat{\boldsymbol{e}}_1 \tag{6.15b}$$

$$\hat{\boldsymbol{e}}_3 = \frac{\hat{\boldsymbol{e}}_1 \times \boldsymbol{v}_T}{\|\hat{\boldsymbol{e}}_1 \times \boldsymbol{v}_T\|} \tag{6.15c}$$

Then, the factors in equation 6.14 can be written in terms of these coordinate vectors.

$$\boldsymbol{r}_{T,0} - \boldsymbol{r}_{P,0} = \|\boldsymbol{r}_{T,0} - \boldsymbol{r}_{P,0}\|\,\hat{\boldsymbol{e}}_2 \tag{6.16a}$$

$$\boldsymbol{v}_T = v_{T,1}\hat{\boldsymbol{e}}_1 + v_{T,2}\hat{\boldsymbol{e}}_2 \tag{6.16b}$$

$$\hat{\boldsymbol{l}} = l_1\hat{\boldsymbol{e}}_1 + l_2\hat{\boldsymbol{e}}_2 + l_3\hat{\boldsymbol{e}}_3 \tag{6.16c}$$

Note that $\boldsymbol{v}_T$ has no $\hat{\boldsymbol{e}}_3$ component, as $\hat{\boldsymbol{e}}_3$ is defined to be perpendicular to $\boldsymbol{v}_T$. And, l_1, l_2 and l_3 are the unknowns to be found.

Substitute equation 6.16 to equation 6.14.

$$(v_P l_1 - v_{T,1})\, t_I \hat{\boldsymbol{e}}_1 + (v_P l_2 - v_{T,2})\, t_I \hat{\boldsymbol{e}}_2 + v_P l_3 t_I \hat{\boldsymbol{e}}_3 = \|\boldsymbol{r}_{T,0} - \boldsymbol{r}_{P,0}\|\, \hat{\boldsymbol{e}}_1 \tag{6.17}$$

From equation 6.17, it is recognized that the $\hat{\boldsymbol{e}}_3$ component of $\boldsymbol{v}_P$ is zero, because the pursuer's velocity will be coplanar with the relative position and the target's velocity. Further, the pursuer and the target have equal velocity components in the $\hat{\boldsymbol{e}}_2$ direction. Equation 6.17 yields the following.

$$l_3 = 0 \tag{6.18a}$$

$$l_2 = \frac{v_{T,2}}{v_P} \tag{6.18b}$$

$$l_1 = \sqrt{1 - l_2^2} = \sqrt{1 - \left(\frac{v_{T,2}}{v_P}\right)^2} \tag{6.18c}$$

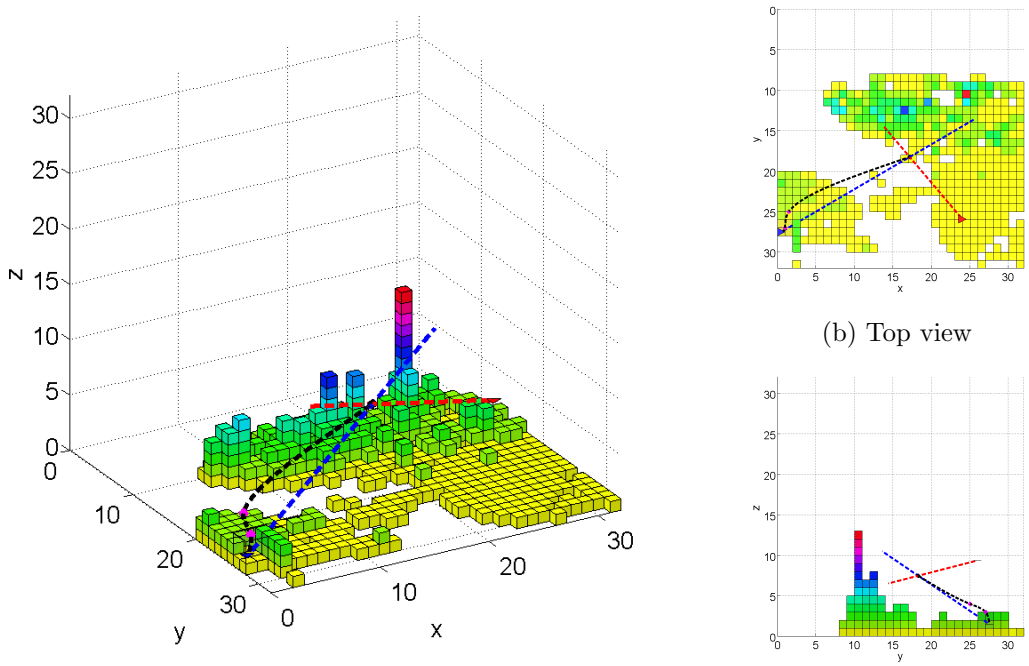
$$t_I = \frac{\|\boldsymbol{r}_{T,0} - \boldsymbol{r}_{P,0}\|}{v_P l_1 - v_{T,1}} = \frac{\|\boldsymbol{r}_{T,0} - \boldsymbol{r}_{P,0}\|}{\sqrt{v_P^2 - v_{T,2}^2} - v_{T,1}} \tag{6.18d}$$

Note that the positive square root is selected in equations 6.18c and 6.18d to give the smallest possible real solution for t_I.

The solution from equation 6.18 is for components of $\hat{\boldsymbol{l}}$ in the e coordinate frame, and in terms of $\boldsymbol{v}_T$ components in the e coordinate frame. The components of $\boldsymbol{v}_T$ are likely given in terms of some other coordinate frame i, and the solution for $\hat{\boldsymbol{l}}$ should be in the same coordinate frame. Therefore, it is required to transform between these coordinate frames. A rotation matrix is defined as

$$[\boldsymbol{C}] = \begin{bmatrix} [\hat{\boldsymbol{e}}_1]_i & [\hat{\boldsymbol{e}}_2]_i & [\hat{\boldsymbol{e}}_3]_i \end{bmatrix} \tag{6.19}$$

where, $[\hat{\boldsymbol{e}}_1]_i$, $[\hat{\boldsymbol{e}}_2]_i$, and $[\hat{\boldsymbol{e}}_3]_i$ are computed by evaluating equation 6.15 using $\boldsymbol{r}_{T,0}$, $\boldsymbol{r}_{P,0}$, and $\boldsymbol{v}_T$ in i components. Then at the start of the method, $\boldsymbol{v}_T$ is transformed from i to e components.

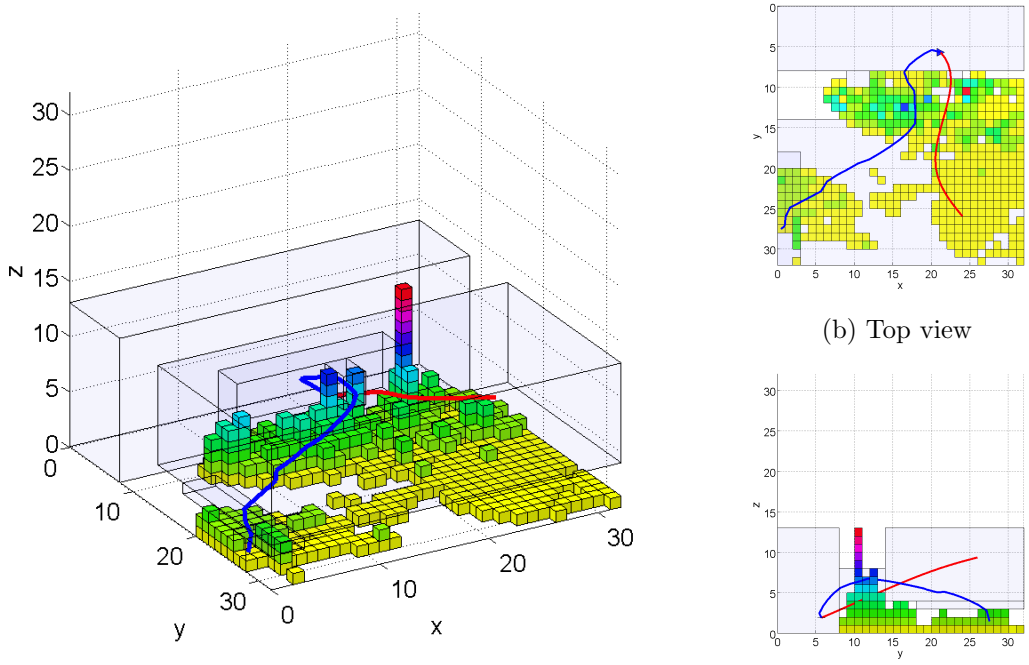


(a) 3D view

(b) Top view

(c) Side view

Figure 6.6. Interception-triangle navigation on R-map: the computed interception path at t_0.



(a) 3D view

(b) Top view

(c) Side view

Figure 6.7. Interception-triangle navigation on R-map: the final trajectories at t_I.

$$[\boldsymbol{v}_T]_e = [\boldsymbol{C}]^\intercal [\boldsymbol{v}_T]_i \tag{6.20}$$

Once the solution for $\hat{\boldsymbol{l}}$ is computed from equations 6.18a, 6.18b, and 6.18c, it can be transformed from e to i components.

$$\left[\hat{\boldsymbol{l}}\right]_i = [\boldsymbol{C}] \left[\hat{\boldsymbol{l}}\right]_e \tag{6.21}$$

Finally, with this $\hat{\boldsymbol{l}}$, the interception point can be calculated by evaluating either equation 6.11 or 6.12 at t_I. This process iterates as the target's dynamic information is updated.

Figure 6.6 shows a simulation with this method. Here, the speeds of the target and the pursuer are constant as 1.0 and 1.5 3D grid cells per time unit, respectively. The velocities of the target (red dashed-line) and the pursuer (blue dashed-line) are illustrated. The intersection point of those two lines is the desired final location. If the intersection point is inside an obstacle, the algorithm selects the closest out-of-obstacle point from it on the line of target's velocity, as an alternative final location. When both target and pursuer have constant velocities and are not affected by obstacles, the blue dashed-line is the actual path that the pursuer can follow in time to intercept the target. However, due to the maneuvering target in this simulation and the pursuer's need to avoid obstacles, the actual trajectory generally deviates (see appendix A for more simulations).

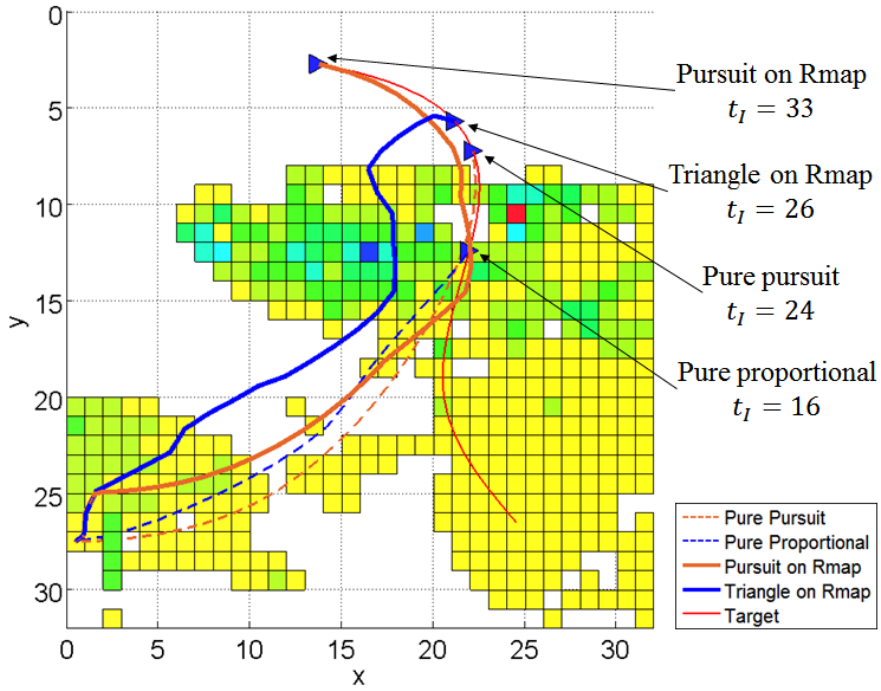
6.3 Discussion

The four approaches require different sensing information to compute an interception path: only the LOS for pure pursuit, the closing velocity and the LOS (and the target's acceleration if it maneuvers) for pure proportional, the target's position vector for pursuit on R-map, and the position and velocity vectors for triangle on R-map.

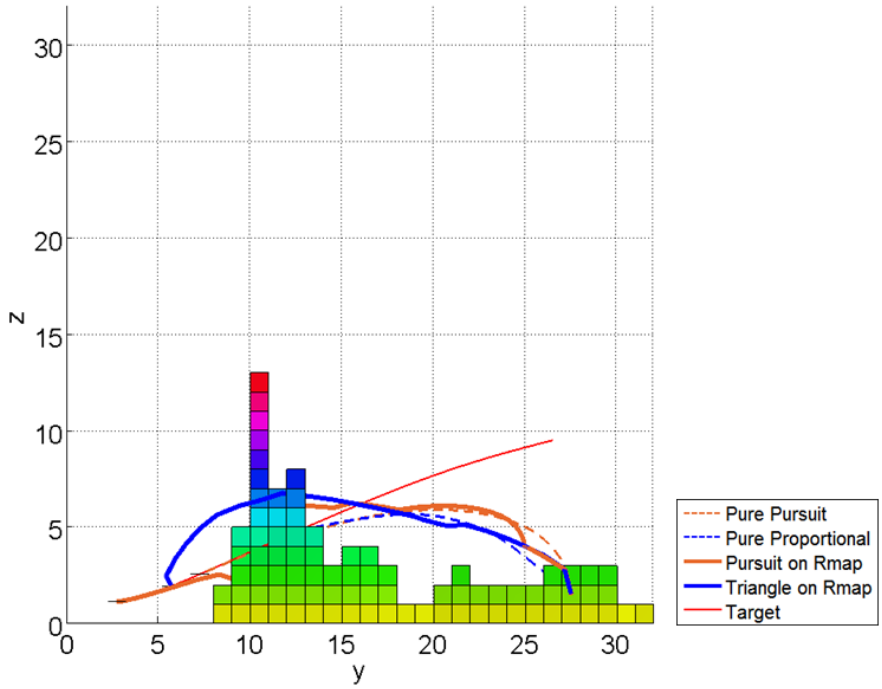
Figure 6.8 compares the paths from the four approaches. Here, the interception time of each approach is as following: pure pursuit $t_I = 24$, pure proportional $t_I = 16$, pursuit

on Rmap $t_I = 33$, and triangle on Rmap $t_I = 26$. The paths from the two R-map approaches require more time to intercept the target. Notice that the interception time will vary depending on the environment.

Figure 6.9 shows the pursuer's heading angle. It is assumed that the pursuer is initially heading to the North (zero angle at t_0), and the clockwise direction is the positive azimuth angle. Figure 6.10 shows the change of the pursuer's heading angle. As shown in this figure, the paths from the traditional approaches, figures 6.10a and 6.10b, do not have radical changes (paths are smooth curves). However, the paths from the R-map approaches, figures 6.10c and 6.10d, require several maneuvers to avoid obstacles.

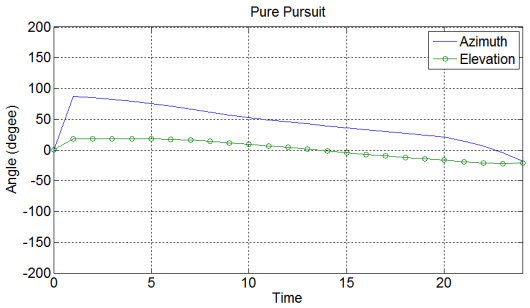


(a) Top view

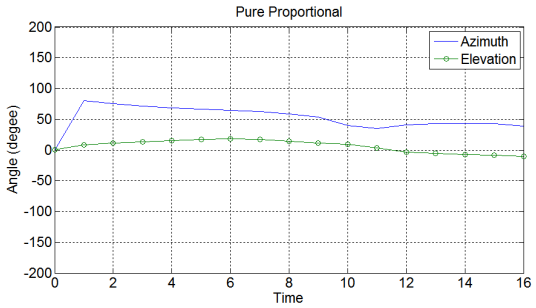


(b) Side view

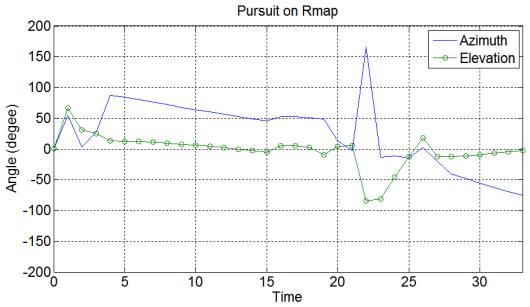
Figure 6.8. Simulation 1: four target interception strategies.



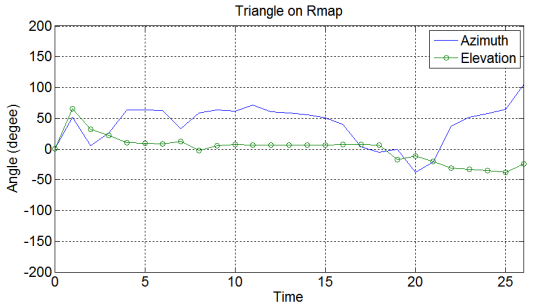
(a) Pure pursuit



(b) Pure proportional

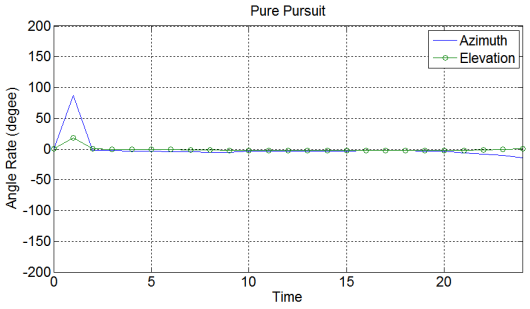


(c) Pursuit on Rmap

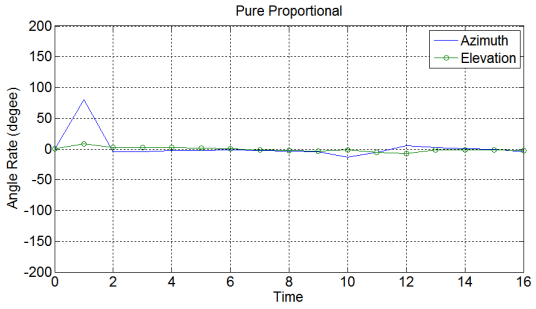


(d) Triangle on Rmap

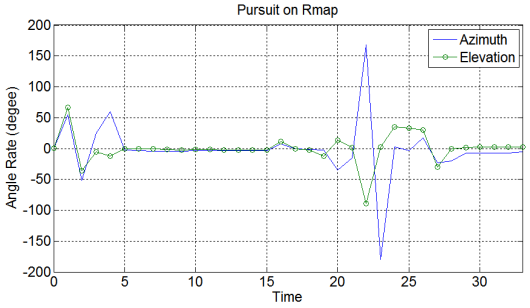
Figure 6.9. Simulation 1: angles of the pursuer's heading.



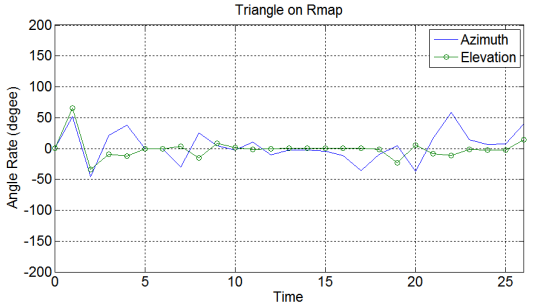
(a) Pure pursuit



(b) Pure proportional



(c) Pursuit on Rmap



(d) Triangle on Rmap

Figure 6.10. Simulation 1: change of angles of the pursuer's heading.

Chapter 7

Conclusions

This dissertation explored development of R-maps for 2D and 3D environments. R-maps help to build higher-resolution maps with fewer elements by applying the maximal empty rectangle problem. The R-map representation has several advantages. First, by focusing on the maximal empty rectangles (cuboids), it is naturally suited to obstacle avoidance. Second, by using a reduced number of elements, R-maps are more efficient for path-planning algorithms such as Dijkstra's algorithm. Last, by integration of empty cells into a larger rectangle, R-maps reduce redundant data, thus they are efficient for data saving. Therefore, R-maps could also be beneficial for sharing map information between cooperative agents with limited communication bandwidth.

Further, this work introduced a new approach to map-merging problems using the idea of R-map. The key technique in this approach is extracting the properties and connections of rectangles from maps that allows to match orientations and scales, and find overlapping points. Because the three factors are unknown, the final merged map is an estimated map by minimizing differences of the factors. To expect a good fit of two maps in a merged map, initial maps require two conditions. First, the environment in each map should have at least one pair of parallel edges and longer makes a better estimation. Thus, empty spaces such as rectangles or trapezoids are essential to place the maps in the orthogonal orientations by aligning the parallel edges with the map grids. Second, the initial maps should have at least three of separate rectangular spaces or a non-rectangular space to find three rectangles as shared-triangles. Hence, the R-map algorithm produces rectangles, and the shared-triangle algorithm is able to find three of common rectangles. Fortunately, many practical environments meet the two conditions.

This work also discussed 3D R-map based path planning on the environment of New York City. The paths can be diverse by the different weight definitions, thus an agent can select a weight depending on the goal of its mission. In the case of a moving target, navigation methods were applied to the R-map to intercept the target while avoiding obstacles. Two approaches were introduced to determine the desired final location, and compared with the traditional approaches. Because the R-map yields rapid path planning, R-maps could be a solution to the real-time path-planning problems such as target interception in cluttered environments.

The contribution of developing the 3D R-map regarding path planning and target interception is that the R-map can make any path planning algorithm efficient. This dissertation illustrated path planning with Dijkstra's algorithm, however other map-based algorithms such as A* and RRT (rapidly exploring random tree) also can be adopted and accomplish rapid path planning.

Future work could include further investigation on cooperation of multiple agents. For example, tracking/intercepting multiple targets avoiding obstacles, making a swarm in the largest cuboid, or hiding beside obstacles by moving to small cuboids. In addition, the idea of R-map could be applied to the field of image processing by means of multi-resolution images to save data amount.

Bibliography

[1] Thrun, S., "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, 2002, pp.1-35.

[2] Latombe, J.C., "Robot motion planning," Springer Science and Business Media, Vol. 124, 2012.

[3] Cowlagi, R. V., and Tsiotras, P., "Multiresolution Path Planning with Wavelets: A Local Replanning Approach," *American Control Conference*, IEEE, Seattle, WA, 2008, pp. 1220-1225

[4] Behnke, S., "Local multiresolution path planning," *Robot Soccer World Cup*, Springer Berlin Heidelberg, pp. 332-343, July 2003.

[5] Tsiotras, P., and Bakolas, E., A hierarchical on-line path planning scheme using wavelets, *European Control Conference*, Kos, Greece, 2007.

[6] Hwang, J. Y., Kim, J. S., Lim, S. S., and Park, K. H., A fast path planning by path graph optimization, *IEEE Trans. Systems, Man, and Cybernetics*, IEEE Vol. 33, No. 1, pp. 121127, January 2003.

[7] Katevas, N.I., Tzafestas, S.G., and Pnevmatikatos, C.G., "The approximate cell decomposition with local node refinement global path planning method: path nodes refinement and curve parametric interpolation," *Journal of intelligent and robotic systems*, Vol. 22, No.3, pp.289-314, 1998.

[8] Prazenica, R., Kurdila, A., and Sharpely, R., "Receding Horizon Control for MAVs with Vision-Based State and Obstacle Estimation," *AIAA Guidance, Navigation, and Control Conference*, AIAA, SC, 2007

[9] Prazenica, R., Kurdila, A., Sharpely, R., and Evers, J., "Multiresolution and Adaptive Path Planning for Maneuver of Micro-Air-Vehicles in Urban Environments," *AIAA Guidance, Navigation, and Control Conference*, AIAA, San Francisco, CA, 2005, pp. 1-12

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

[11] Kuipers, B., and Byun, Y.T., "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *Robotics and autonomous systems*, Vol. 8, No. 1, pp.47-63, 1991.

[12] Simhon, S., and Dudek, G., "A global topological map formed by local metric maps," *Intelligent Robots and Systems Proceedings*, IEEE/RSJ International Conference Vol. 3, pp. 1708-1714 October 1998.

[13] J. Park, and A. Sinclair, "Mapping for path planning using maximal empty rectangles," *AIAA Guidance, Navigation, and Control (GNC) Conference*, pp. 5096, 19-22 August 2013.

[14] Konolige, K., Fox, D., Limketkai, B., Ko, J., and Stewart, B., "Map merging for distributed robot navigation," *Intelligent Robots and Systems (IROS 2003) Proceedings*, IEEE/RSJ International Conference, Vol. 1, pp. 212-217, October 2003.

[15] Arras, K.O., Tomatis, N., Jensen, B.T., and Siegwart, R., "Multisensor on-the-fly localization: Precision and reliability for applications," *Robotics and Autonomous Systems*, Vol. 34, No. 2, pp.131-143, 2001.

[16] Fox, D., Burgard, W., Kruppa, H., and Thrun, S., "A probabilistic approach to collaborative multi-robot localization," *Autonomous robots*, Vol. 8, No. 3, pp.325-344, 2000.

[17] Carpin, S., Birk, A., and Jucikas, V., "On map merging," *Robotics and autonomous systems*, Vol. 53, No. 1, pp.1-14, 2005.

[18] Tardós, J., Neira, J., Newman, P., and Leonard, J., "Robust mapping and localization in indoor environments using sonar data," *The International Journal of Robotics Research*, vol. 21, no. 4, pp. 311-330, April 2002.

[19] Vandevoorde, D., "The Maximal Rectangle Problem," *Dr. Dobb's Journal*, Vol. 23, No. 4, pp. 28-32, 1998

[20] Hwang, Y.K., and Ahuja, N., "Gross motion planninga survey," *ACM Computing Surveys (CSUR)*, Vol. 24, No. 3, pp.219-291, 1992.

[21] DeSouza, G.N., and Kak, A.C., "Vision for mobile robot navigation: A survey," *IEEE transactions on pattern analysis and machine intelligence*, Vol. 24, No. 2, pp.237-267, 2002.

[22] LaValle, S.M., "Planning algorithms," *Cambridge university press*, 2006.

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

[24] Fischler, M., and Bolles, R., "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381-395, June 1981.

[25] Guelman, M., "A qualitative study of proportional navigation," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 4, pp.637-643, 1971.

[26] Bakolas, E., and Tsiotras, P., "Feedback navigation in an uncertain flowfield and connections with pursuit strategies," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 4, pp.1268-1279, 2012.

[27] Gutman, S., "On optimal guidance for homing missiles," *Journal of Guidance, Control, and Dynamics*, AIAA, Vol. 2, No. 4, 1979.

[28] Turetsky, V., and Shinar, J., "Missile guidance laws based on pursuitevasion game formulations," Automatica, Vol. 39, No. 4, pp.607-618, 2003.

[29] Kendoul, F., "Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems," *Journal of Field Robotics*, Vol. 29, No. 2, pp.315-378, 2012.

[30] Golan, O., and Shima, T., "Head pursuit guidance for hypervelocity interception," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pp. 4885, Providence, Rhode Island, 16-19 August 2004.

[31] Moon, J., Kim, K., and Kim, Y., "Design of missile guidance law via variable structure control," *Journal of Guidance, Control, and Dynamics*, Vol. 24, No.4, pp.659-664, 2001.

[32] Murtaugh, S.A., and Criel, H.E., "Fundamentals of proportional navigation," *IEEE spectrum*, Vol. 3, No. 12, pp.75-85, 1966.

[33] Kim, T.H., Park, B.G., and Tahk, M.J., "Bias-shaping method for biased proportional navigation with terminal-angle constraint," *Journal of Guidance, Control, and Dynamics*, AIAA, Vol. 36, No. 6, pp.1810-1816, 2013.

[34] Lee, C.H., Kim, T.H., and Tahk, M.J., "Interception angle control guidance using proportional navigation with error feedback," *Journal of Guidance, Control, and Dynamics*, AIAA, Vol. 36, No. 5, pp.1556-1561, 2013.

[35] Dhananjay, N., and Ghose, D., "Accurate time-to-go estimation for proportional navigation guidance," *Journal of Guidance, Control, and Dynamics*, AIAA, Vol. 37, No. 4, pp.1378-1383, 2014.

[36] Palumbo, N., Blauwkamp, R., and Lloyd, J., Modern homing missile guidance theory and techniques, *Johns Hopkins APL technical digest*, Vol. 29, No. 1, pp. 42, 2010.

[37] Moran, I., and Altilar, T., "Three plane approach for 3D true proportional navigation," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pp. 6457, August 2005.

[38] Shusterman, E., and Feder, M., "Image compression via improved quadtree decomposition algorithms," *IEEE Transactions on image processing*, Vol. 3, No. 2, pp. 207-215, March 1994.

[39] Medellin, H., Corney, J.R., Davies, J.B.C., Lim, T. and Ritchie, J.M., "Octree-based production of near net shape components," *IEEE Transactions on Automation Science and Engineering*, Vol. 5, No. 3, pp.457-466, July 2008.

[40] Triharminto, H.H., Adji, T.B. and Setiawan, N.A., "Dynamic uav path planning for moving target intercept in 3D," *IEEE In Instrumentation Control and Automation (ICA), 2nd International Conference on*, pp. 157-161, November 2011.

[41] Chen, Y., Cheng, L., Wu, H. and Yang, Y., "Receding horizon control for mobile robot path planning in unknown dynamic environments," *IEEE In Control and Decision Conference (2014 CCDC), The 26th Chinese*, pp. 1505-1509, June 2014.

Appendices

Appendix A

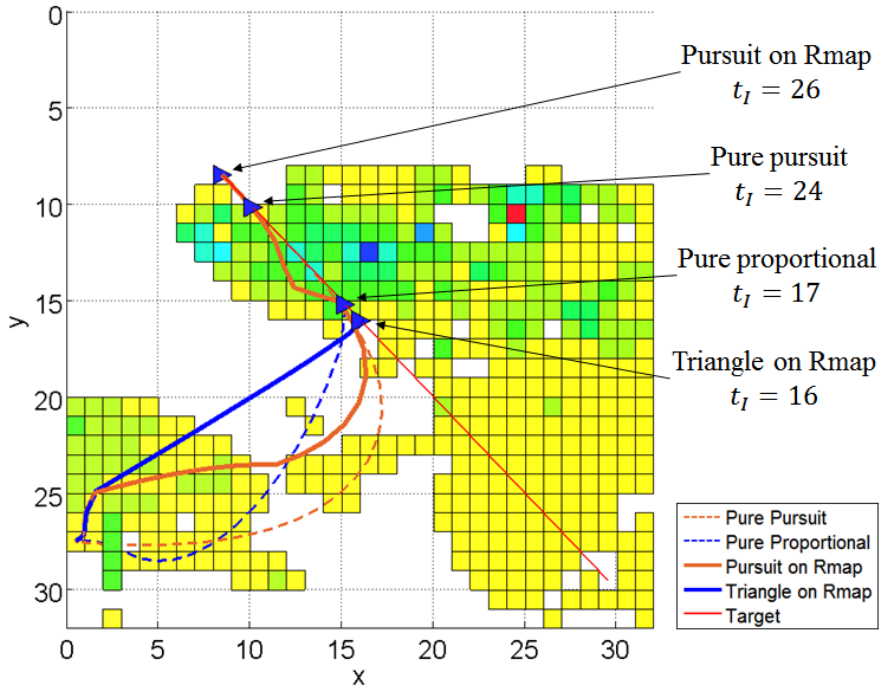
More simulations on target interception

A.1 Simulation 2

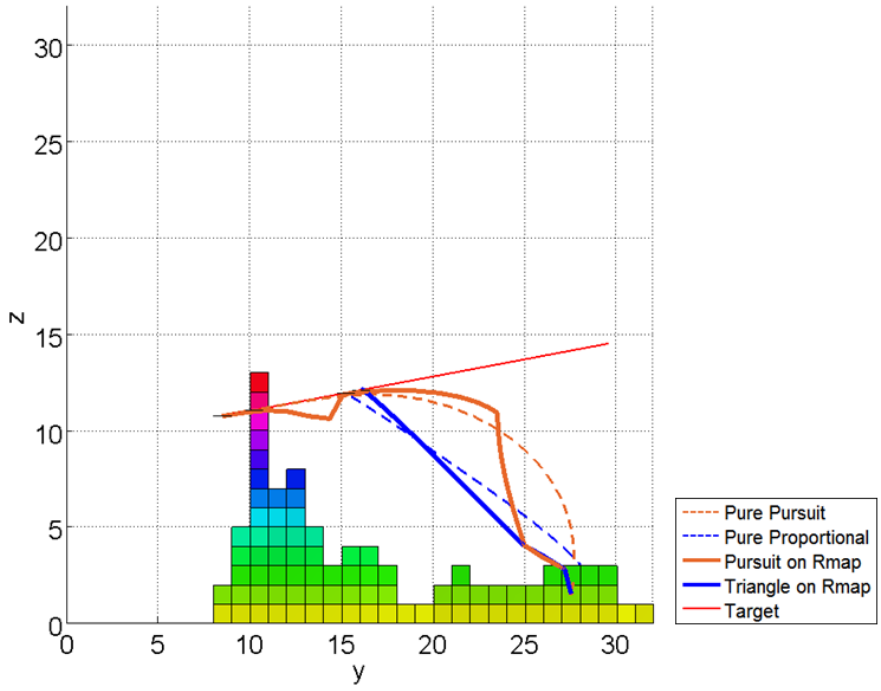
In this simulation, the target moves with constant velocity as illustrated in figure A.4. Here, the interception time of each approach is computed as following: $t_I = 24$ for pure pursuit, $t_I = 17$ for pure proportional, $t_I = 26$ for pursuit on R-map, and $t_I = 16$ for triangle on R-map. For the path from triangle on R-map, the algorithm computed a straight interception-path after avoiding obstacles (blue line), which is an actual path to follow in time to intercept the target, because the target has constant velocity and is in the same cuboid with the pursuer.

A.2 Simulation 3

In this simulation, the target maneuvers, but the initial locations of the pursuer and the target are different from simulation 1 in figure 6.8. The interception time of each approach is computed as following: $t_I = 16$ for pure pursuit, $t_I = 13$ for pure proportional, $t_I = 30$ for pursuit on R-map, and $t_I = 17$ for triangle on R-map. The order of interception times is the same with the simulation 1.

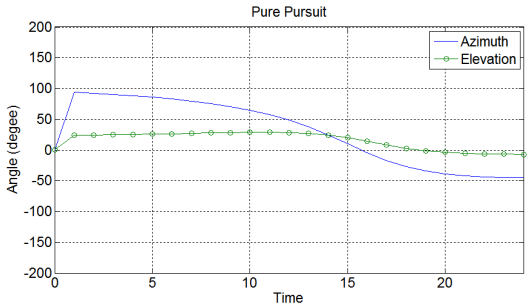


(a) Top view

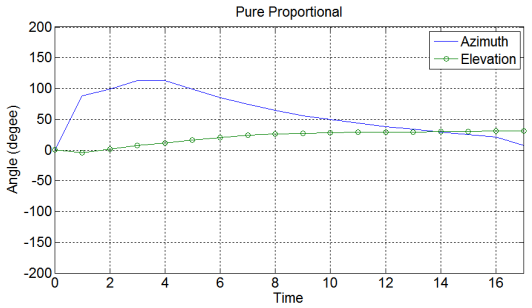


(b) Side view

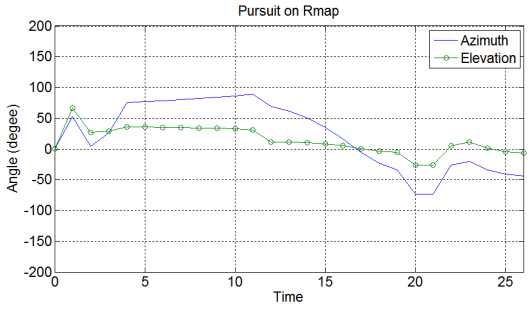
Figure A.1. Simulation 2: four target interception strategies.



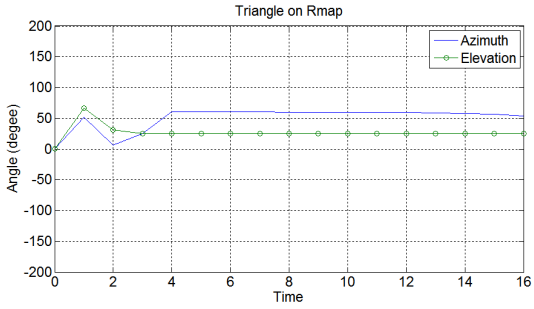
(a) Pure pursuit



(b) Pure proportional

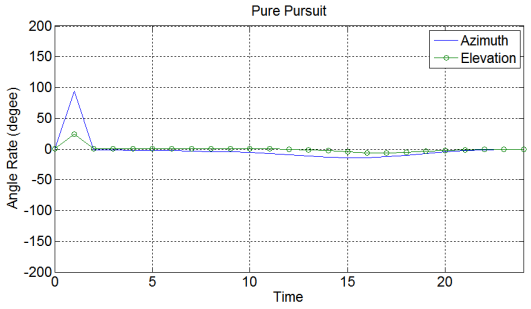


(c) Pursuit on Rmap

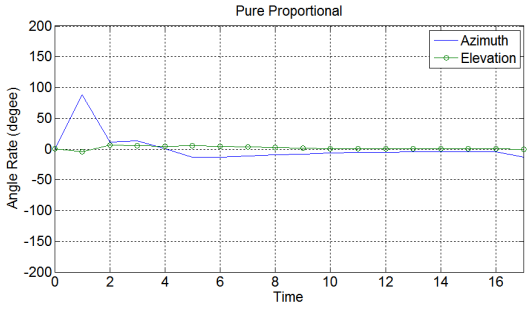


(d) Triangle on Rmap

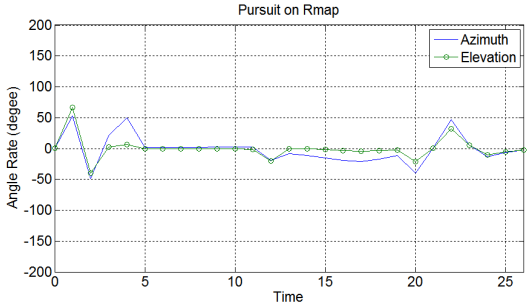
Figure A.2. Simulation 2: angles of the pursuer's heading.



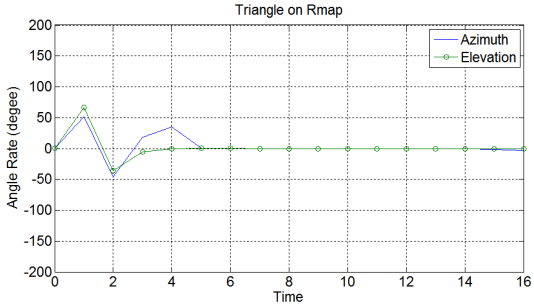
(a) Pure pursuit



(b) Pure proportional

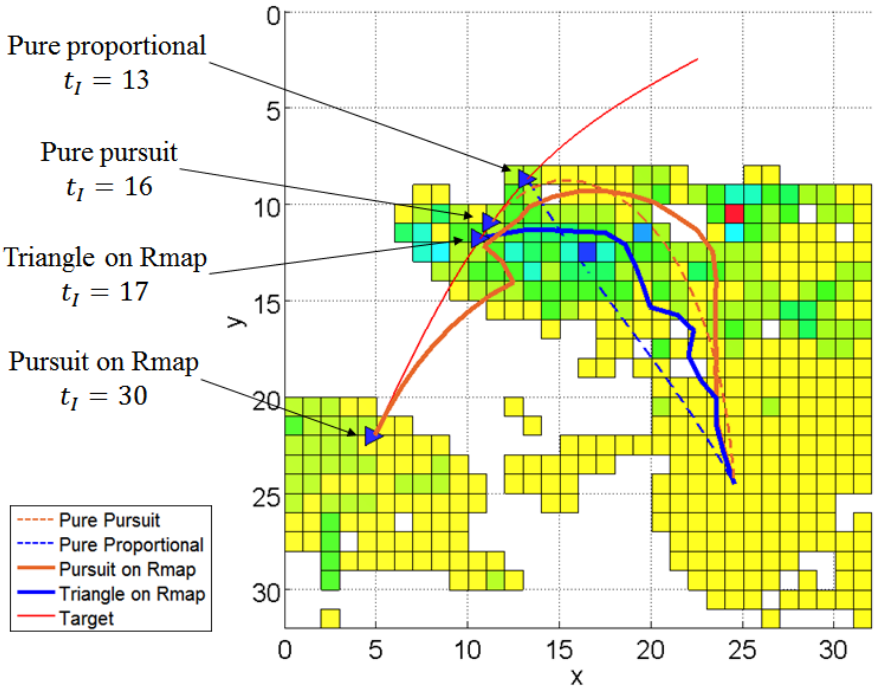


(c) Pursuit on Rmap

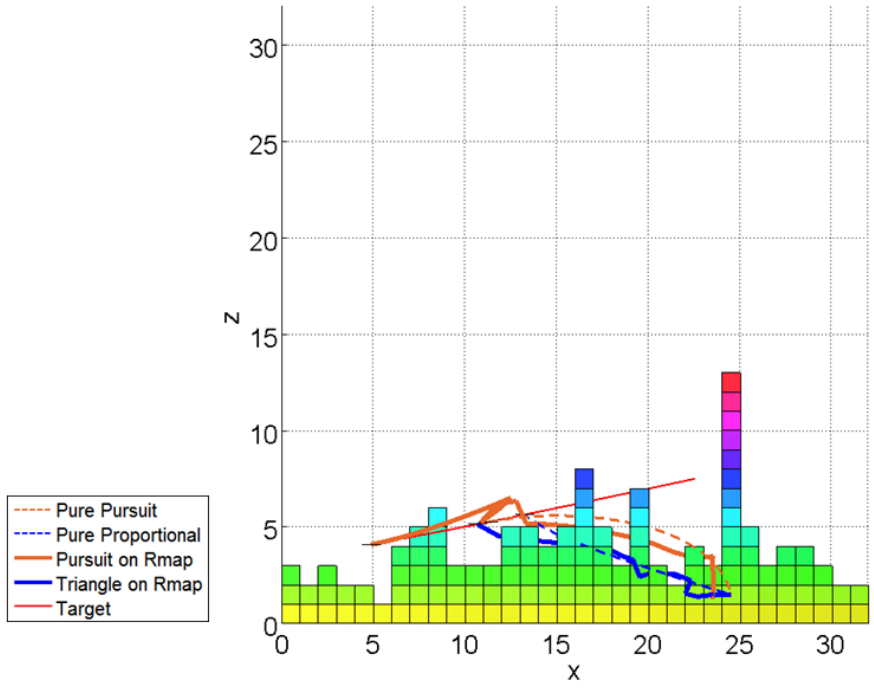


(d) Triangle on Rmap

Figure A.3. Simulation 2: change of angles of the pursuer's heading.

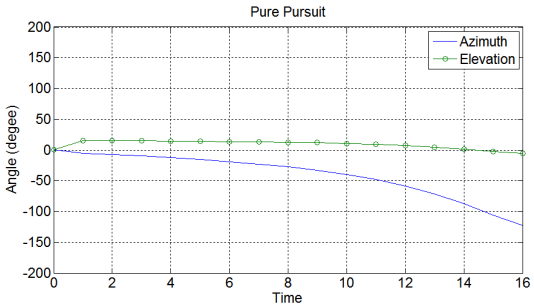


(a) Top view

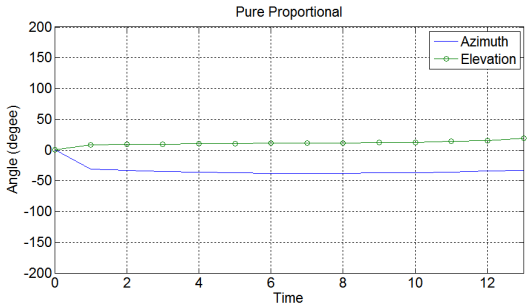


(b) Front view

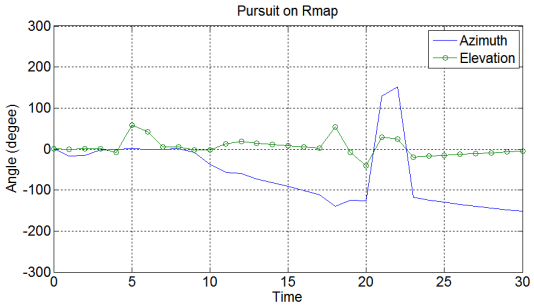
Figure A.4. Simulation 3: four target interception strategies.



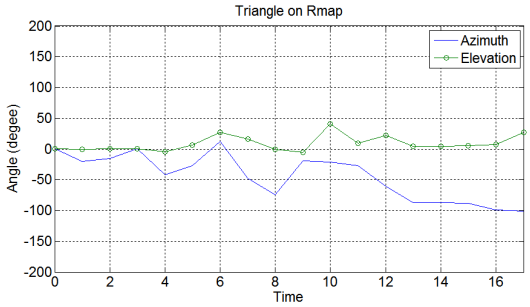
(a) Pure pursuit



(b) Pure proportional

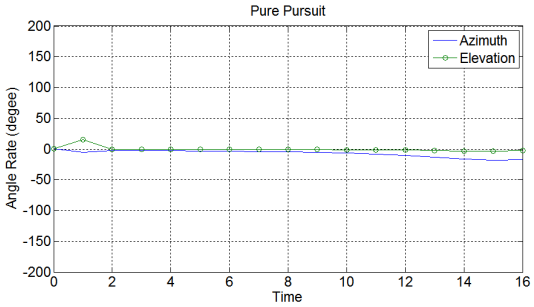


(c) Pursuit on Rmap

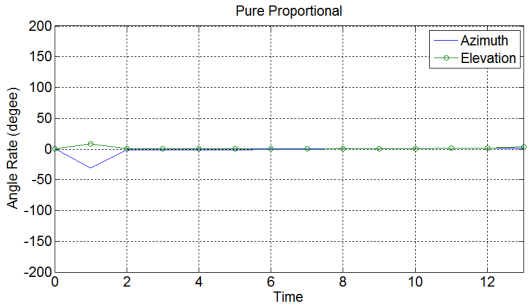


(d) Triangle on Rmap

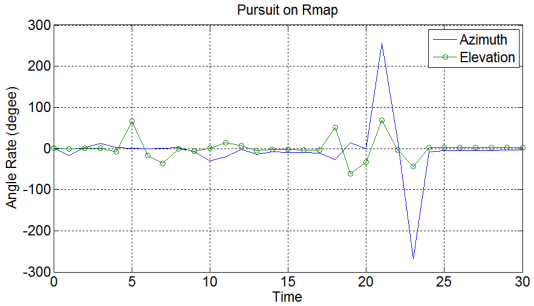
Figure A.5. Simulation 3: angles of the pursuer's heading.



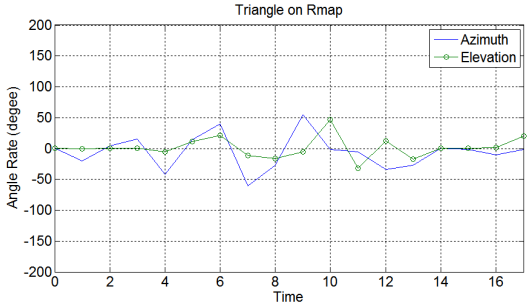
(a) Pure pursuit



(b) Pure proportional



(c) Pursuit on Rmap



(d) Triangle on Rmap

Figure A.6. Simulation 3: change of angles of the pursuer's heading.