Waseda University Doctoral Dissertation

Study on Grid-based Path Planning using Diagonal Node Expansion and Auto-splitting D* lite Algorithms

Shin-nyeong HEO

Graduate School of Information, Production and Systems
Waseda University
Oct 2022

Abstract

This study is a global path planning and local trajectory planning that execute while rescue robots and first responders search for survivors and perform complex missions in a disaster area. The research theme is to conduct rescue in the disaster area and transport using automatic vehicles to a safe area. In this situation, it is assumed that a server for the global path planning is connected in real-time with a camera sensor of a UAV. The path planning method uses grid-based global path planning based on a map taken from the UAV. Before planning the path based on the map from the UAV, the composition of the study varies depending on whether static or dynamic obstacles.

Firstly, a diagonal node expansion algorithm in static obstacle areas is needed to reduce expanded nodes better than Dijkstra and A* algorithms. The less total number of expanded nodes leads to faster calculation time in the grid-based global path planning methods. Thus, it is required to increase the searching speed of nodes searched by Lidar and Radar sensors. Also, path smoothing to reduce zig-zag issues in the path is important. The zig-zag issues occur during path planning, and influence the robot's movement. Additionally, simulations for dynamic obstacles are carried out to help understand the grid-based path planning method for the next idea.

Secondly, a grid-based global path planning method for dynamic obstacle areas named Auto-splitting D* lite algorithm is considered. Most of the research focus on this part to limit the environment of the grid-based global path planning method to improve the performance of existing algorithms by dividing a wide map. To divide the wide map, a clustering method is applied, and it is influenced to improve calculation speed. In a dynamic obstacle environment, a method reduces the expanded node when observation information is updated, rather than the existing expanded node being reduced by one search. Also, the calculation speed of the path is significantly slower depending on the distribution of obstacles on the wide map. Additionally, a new evaluation method is proposed to replace time complexity with other methods to evaluate the performance of algorithms. The evaluation method of each algorithm using the order and dominant value of Big-O notation attempts to solve the difficulty of evaluating in an environment where there is no reference algorithm.

Thirdly, a local trajectory planning method in dynamic obstacle environments is considered. In this part, we focus on the local trajectory planning using autonomous vehicles as well as mobile robots applied in a complex environment. Using the reference path by the Auto-splitting D* lite algorithm, an easy method to calculate various trajectory candidates is introduced. Along with the planned path of the global path planning method, the trajectory method for lane changes on roads considering other vehicles' trajectories in complex environments is considered. Also, an improvement in calculation time and accuracy is tried by selecting one of the different trajectories generated according to the behavioral form. An explanation of each chapter for the above three topics is written as follows.

Chapter 1 describes the background and purpose of the study, then the reasons and circumstances for the necessity of the path planning and trajectory planning of the rescue robot and the autonomous vehicle are described.

Chapter 2 introduces related work classified into six categories according to the path planning topic. The six path planning categories consisted of (1) Sampling-based algorithms, (2) Grid-based algorithms, (3) Bio-inspired algorithms, (4) Neural network algorithms, (5) Mathematic model-based algorithms, and (6) Multi-fusion based algorithms. This research mainly focuses on a grid-based global path planning method based on A* and D* lite algorithms and is combined with a local trajectory planning method. Therefore, the final trajectories of robots and vehicles are generated using a Frenet frame conversion. Thus, the final path and trajectories take the form of the multi-fusion based algorithm mainly focused on the Grid-based algorithm.

Chapter 3 describes a grid-based global path planning method in static obstacle areas. We propose a Diagonal node expansion algorithm, which expands only diagonal nodes of eight peripheral nodes of a selected current node. Also, we introduce a modified Shoemake scheme to quickly smooth zig-zag paths resulting from existing zig-zag issues and further extended diagonal nodes in this chapter. The Diagonal node expansion algorithm reduces the expanded nodes and the calculation time. If the size of a map and information are large, the calculation time will be reduced compared with an original grid-based path. This algorithm will help find the shortest and safest path for the rescue robots and the first responders. The Diagonal node expansion algorithm reduces the calculation time of global path planning in the static known area was reduced by 45% (51[grid]$\times$51[grid]) to 49% (1601[grid]$\times$1601[grid]) in a real map (125[m]$\times$125[m]).

Chapter 4 describes a grid-based global path planning method in the dynamic obstacle environments with a wide map. To reduce the expanded nodes of the global path on the wide map, a splitting D* lite algorithm is proposed to split the map in advance so that the expanded nodes are ignored if the obstacle node information is updated from other areas of the split map, which is not the current area of the selected map. Then, we introduce an Auto-splitting D* lite algorithm with an auto-clustering method so that a split map of the same size can generate expanded nodes with uniform clustering points. Also, we introduce an expected value of re-planned node (ERPN) method that quantifies the performance using the expected value in which the expanded node occurs. The proposed Auto-splitting D* lite algorithm solves the issue of the unnecessary areas by the traditional D* lite algorithm and alleviates the over-calculation issue of dynamic path planners in a large area. The Auto-splitting D* lite algorithm removes the nodes of unnecessary areas when a new path is updated in 58% of a city map (1.2[km]$\times$1.2[km]) and 46% of a rural map (400[m]$\times$400[m]) compared with the traditional D* lite algorithm.

Chapter 5 describes a local trajectory planning method using the Auto-splitting D* lite algorithm. The local trajectory planning methods in a complex environment are mainly discussed. This chapter introduces a method to convert the Auto-splitting D*lite into a reference path of the Frenet Frame, and to use the reference path as an axis of the Frenet Frame coordinate. By transforming the coordinate system, the Frenet Frame trajectory planning method is applied for selecting one of several sets of vehicle trajectories to avoid actual obstacles using a reference path containing information such as a velocity. Additionally, Kalman filter is applied to increase location accuracy. The final trajectories showed 8% improvement on the x-axis and 42% improvement on the y-axis when Kalman filter is used compared with the Frenet Frame trajectory planning (2.4[km]$\times$2.4[km], 400[grid]$\times$400[grid] map). In the experimental scenario, the y-axis shows that the vehicle's lateral movement improves.

Finally, Chapter 6 summarizes the research conclusion and addresses future work. To sum up, this dissertation proposed two global path planning algorithms for the static and dynamic environment. Also, one local trajectory planning application method is introduced to link with the dynamic partially known global path planning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Since 2020, various disaster situations such as fires, landslides, earthquakes, and tsunamis have occurred due to climate change and the activity of the Pacific Fire Ring. As a countermeasure, research is focused on rescue robots and autonomous vehicles for disaster area assistants, which help people evacuate. Especially, fires are caused by hydrological disasters, meteorological disasters, and climatological disasters, and the disaster cases increased 400% during 30 years in Fig. 1.1. This rapid increase also has a great impact on the growth of the market for disasters. Therefore, this research introduces global path planning methods for rescue robots, and local trajectory planning methods for patient transport after rescue is completed using rescue robots and autonomous vehicles.

Section 1.1 discusses the necessity for research on global path planning methods and local trajectory planning methods for robots and vehicles to perform rescue efficiently. Since fires, landslides, and earthquakes occur over a wide range, the shortest path plan for quick movement is necessary. To quickly perform a task on terrain that has changed due to a disaster situation, the global map to be required quickly updated the map, at least in a dynamic partially known obstacle environment. Also, the local trajectory is changed to control the robot's movement in the updated area on the map. Since the disaster situation severely damaged the city, it is considered to reduce the calculation time and improve the accuracy of the local trajectory planning for path planning tasks on a wide and complex map.

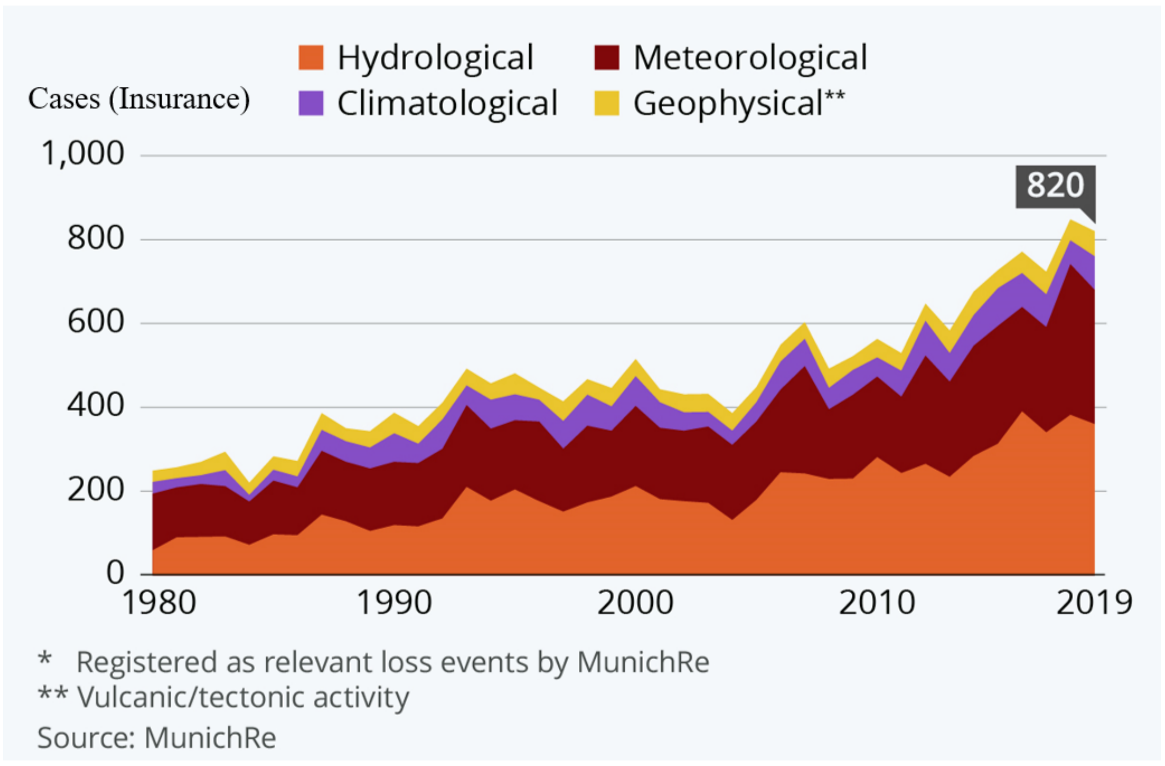


Fig. 1.1 Disaster Status

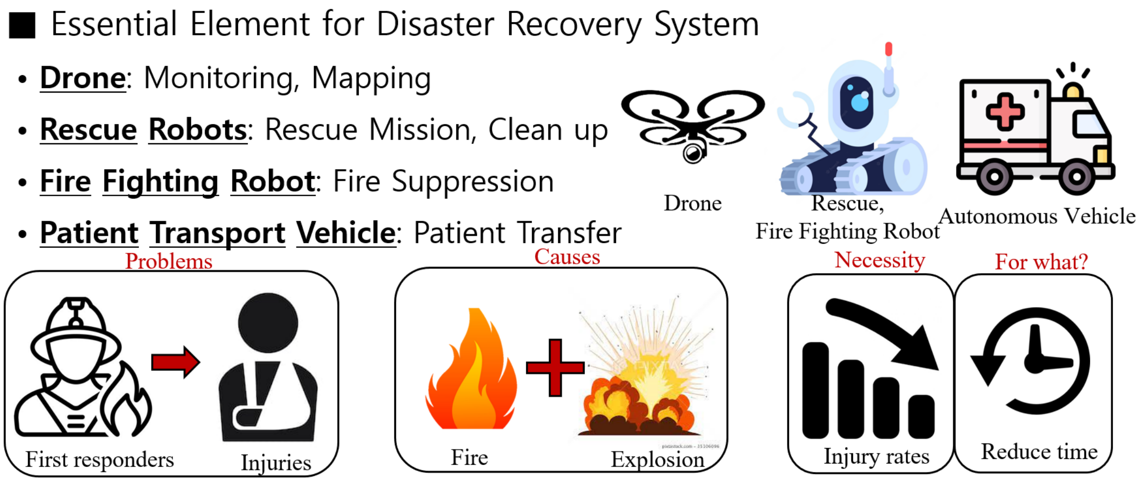

■ Essential Element for Disaster Recovery System
• Drone: Monitoring, Mapping
• Rescue Robots: Rescue Mission, Clean up
• Fire Fighting Robot: Fire Suppression
• Patient Transport Vehicle: Patient Transfer
Drone
Rescue,
Fire Fighting Robot
Autonomous Vehicle
Problems
First responders
Injuries
Causes
Fire
Explosion
Necessity
Injury rates
For what?
Reduce time

Fig. 1.2 Disaster Recovery System

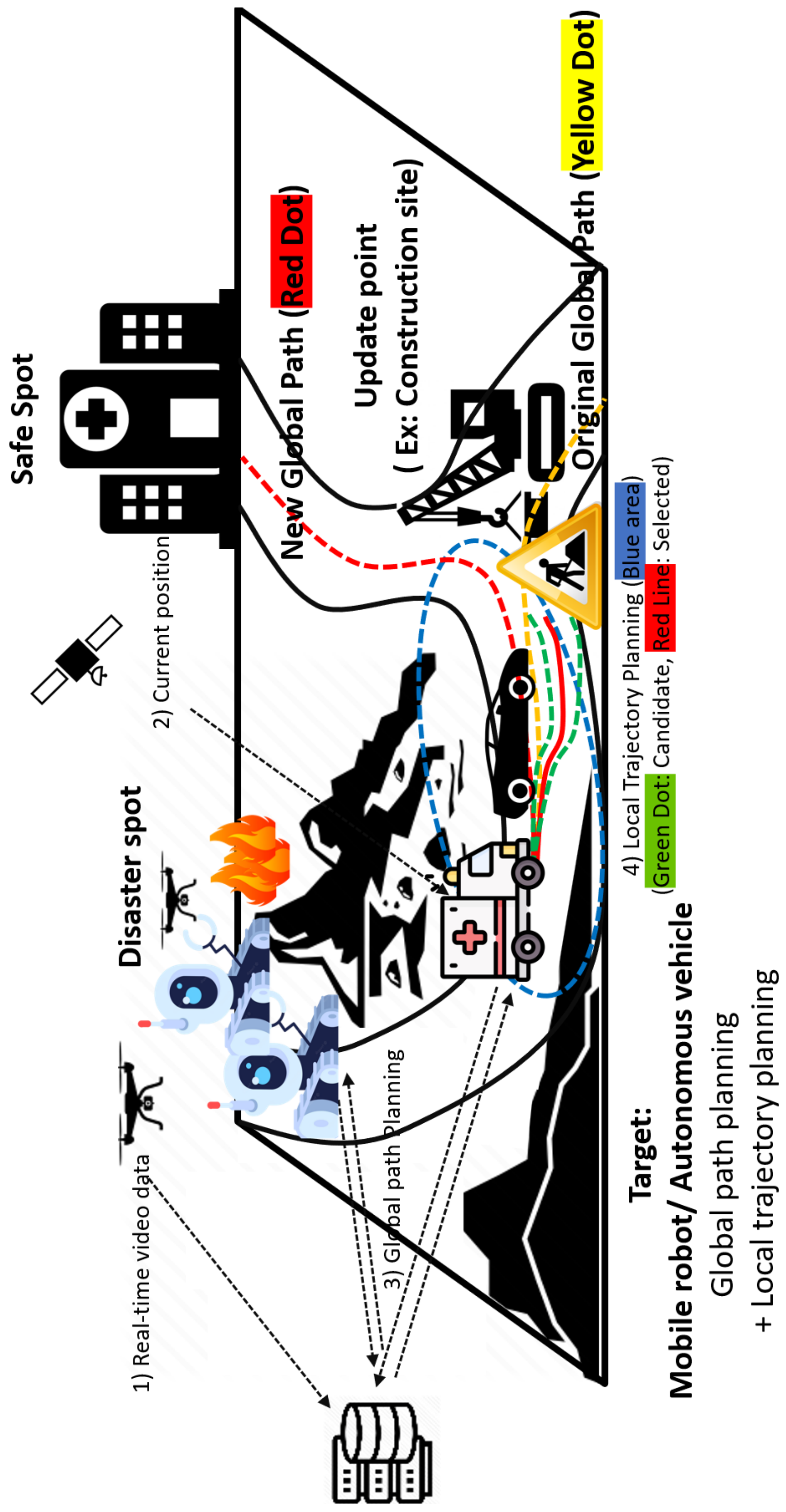
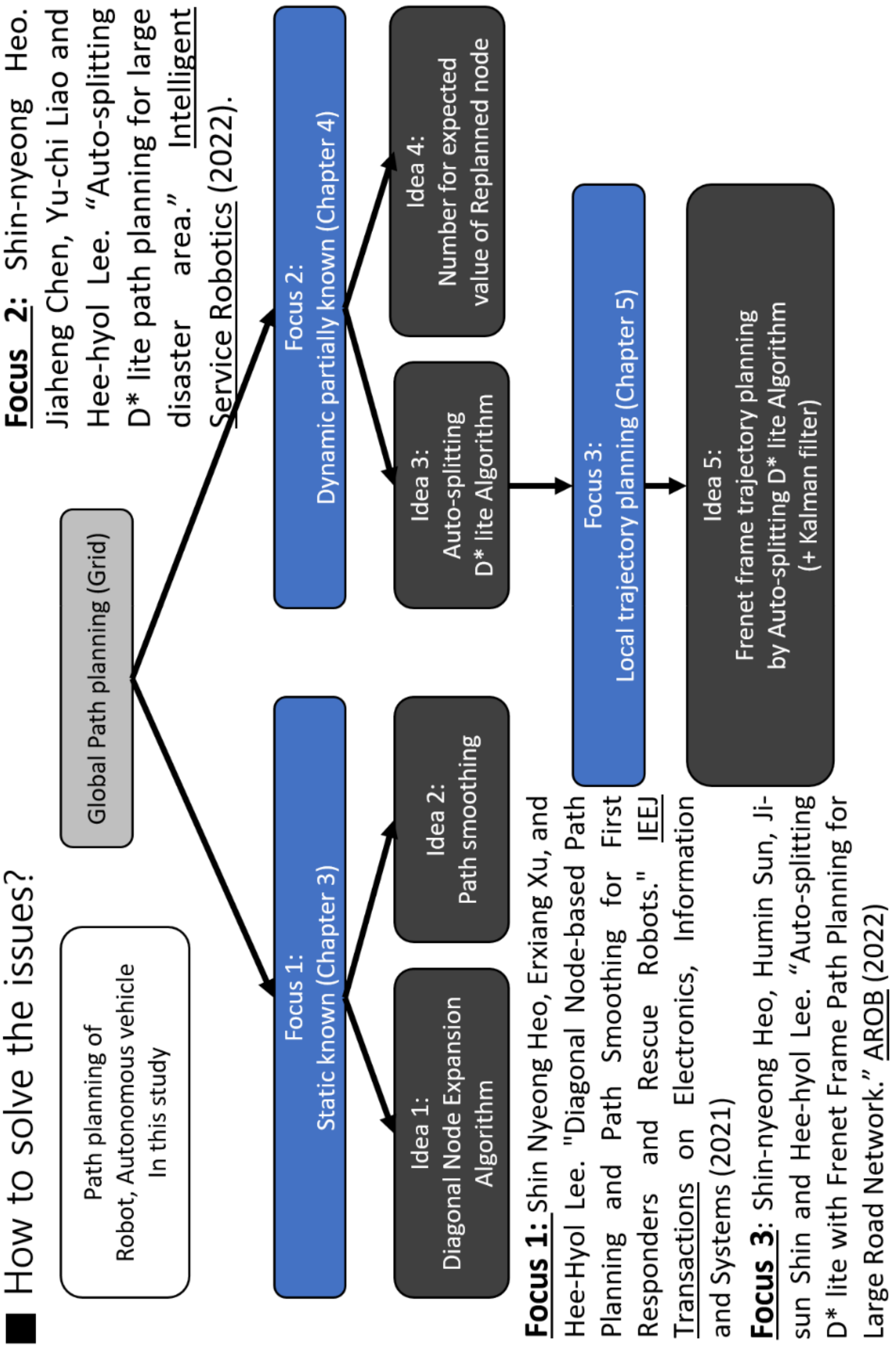

Safe Spot
Disaster spot
1) Real-time video data
2) Current position
3) Global path Planning
4) Local Trajectory Planning (Blue area)
(Green Dot: Candidate, Red Line: Selected)
New Global Path (Red Dot)
Update point
(Ex: Construction site)
Original Global Path (Yellow Dot)
Target:
Mobile robot/ Autonomous vehicle
Global path planning
+ Local trajectory planning

Fig. 1.3 Research Summary

Fig. 1.3 represents the research summary. As shown in the figure, the research targets are to develop global path planning methods and local trajectory planning methods for mobile robots and autonomous vehicles. Also, the research flows for path planning and trajectory planning are represented from 1) to 4). There is an updated point where the map is partially updated near the construction site, then a new global path is planned and local trajectories are generated. While generating the global path, the local trajectory planning is performed in a dynamic environment that oversees the movements and lane changes on roads of autonomous vehicles. Therefore, the research environment is possibly divided into (a) a wide map for search and safety (b) a partially known environment for recognizing sudden changes in the situation, and (c) a dynamic environment for realizing local trajectory planning in real-time. The research proposes new global path planning and local trajectory planning for each environment. Also, the research for path smoothing with evaluation methods is included.

1.2 Research Target

Section 1.2 describes research targets. As explained in Section 1.1 (background), the research focuses is on the global path planning and local trajectory planning. This research finds the shortest path and calculation time of the grid-based global path planning among global path planning algorithms. The local trajectory planning to be done later is also a research goal of fast calculation speed. However, the difference is that it additionally aims for high accuracy considering position estimation. Therefore, the flow of the research is a) Since optimal path planning is required on a wide map, the research is conducted in order of static known based on the grid-based path planning method. b) To solve the detailed issues of the proposing idea, we add a clustering method in the static environment part with path smoothing. Then, the dynamic environment part research is focused on. c) Finally, an accurate and fast trajectory planning method is applied with the planned path from the previous global path planning method.


How to solve the issues?
Path planning of Robot, Autonomous vehicle In this study
Global Path planning (Grid)
Focus 1: Static known (Chapter 3)
Focus 2: Dynamic partially known (Chapter 4)
Idea 1: Diagonal Node Expansion Algorithm
Idea 2: Path smoothing
Idea 3: Auto-splitting D* lite Algorithm
Idea 4: Number for expected value of Replanned node
Focus 3: Local trajectory planning (Chapter 5)
Idea 5: Frenet frame trajectory planning by Auto-splitting D* lite Algorithm (+ Kalman filter)

Focus 2: Shin-nyeong Heo. Jiaheng Chen, Yu-chi Liao and Hee-hyol Lee. "Auto-splitting D* lite path planning for large disaster area." Intelligent Service Robotics (2022).

Focus 1: Shin Nyeong Heo, Erxiang Xu, and Hee-Hyol Lee. "Diagonal Node-based Path Planning and Path Smoothing for First Responders and Rescue Robots." IEEJ Transactions on Electronics, Information and Systems (2021)

Focus 3: Shin-nyeong Heo, Humin Sun, Ji-sun Shin and Hee-hyol Lee. "Auto-splitting D* lite with Frenet Frame Path Planning for Large Road Network." AROB (2022)

Fig. 1.4 Research Target

Research necessities, issues, and ideas are shown in Figure 1.4 with divided into focus 1, focus 2, and focus 3.

In the focus 1, the first detailed focus is a research on the reduction of an expanded node, which is a common issue in grid-based global path planning methods in static known such as Dijkstra, A* [1]. It aims to reduce the total number of expanded nodes that are common issues in the grid-based global path planning methods. Therefore, a faster path planning method is achieved by increasing the searching speed of nodes detected by Lidar and Radar sensors. The first issue can be divided into two categories depending on the environment. The first category would be the grid-based global path planning method in the known area, which has an issue being the calculation time of A* based algorithms increases depending on offered information and map size. The other category would be the grid-based global path planning methods in partially known areas, which have a longer calculation time than A* based algorithms.

The second detailed focus is a research of the path smoothing to reduce zig-zag issues of the path. It aims to improve safety by reducing the zig-zag issues during the proposed global path planning method. In this part, many path smoothing methods are discussed. Mainly, spline curves, interpolations, and composite methods are traditional ways to smooth the zig-zag issue on grid-based path planning methods. B-spline [2], cubic spline [3], [4] and etc. are classes of spline curves [5]. The classes mentioned above are difficult to make the desired curve because each control point is hard to decide. LERP (Linear interpolation) [6], SLERP (Spherical linear interpolation), Bézier interpolation, polynomial interpolation, etc. [7] are classes of interpolation curves. These methods are suitable for fitting curves, but calculation time is long on a higher order. When the interpolations and the spline functions are used together, it is called spline interpolation. Convolution spherical spline interpolation, Cubic spline interpolation, and many other spline interpolations are typically used [8]. Composite methods are combined with other approaches. For example, Shoemake's scheme [9] using SLERP in Bézier curve on four-dimensional space is one of the well-used path smoothing methods. The four-dimensional space using quaternion is easy to be converted into 2D and 3D spaces. In addition, calculation time is shorter than the other path smoothing method.

Therefore, to solve each detailed issue in the focus 1, the following ideas (idea 1 and idea 2) are introduced in this study. (a) To propose a diagonal path planning, which applies to all grid-based algorithms using Euclidian distance and expands only four of the eight adjacent nodes of the selected current node. (b) To apply a modified Shoemake's scheme to quickly smooth the existing zig-zag issues and the zig-zag path resulting from further extending diagonal nodes. Moreover, the basic area is static, but there is an additional explanation for the dynamic area. This iss written to help explain the dynamic area in the next focus. In the dynamic area, the calculation time of D* lite based algorithms [10], [11] is increased depending on the information and map size same as static area.

In the focus 2, three detailed research focus is described. The first detailed focus is to improve the performance of the existing algorithms by dividing the wide map by limiting the environment of the grid-based global path planning algorithm to the dynamic partially known area. In a dynamic and partially known environment, a method to reduce the expanded nodes when updating observation information is more important than reducing the existing expanded nodes in a single search. To clarify, this means that unnecessary areas with expanded nodes exist in D* based path planning methods.

The second detailed focus is to improve computational speed. We conduct a study applying a clustering method that uniformly divides maps according to the distribution of obstacles. On a wide map, the calculation speed of the path is significantly slower depending on the distribution of obstacles. This is because the following issues exist in the existing clustering algorithm. First, the calculation time to determine the k-value is long in k-means clustering-based algorithms. Also, the calculation time is longer than the centroid searching (repeated calculation) in mean-shift clustering-based.

The third detailed focus is an alternative study of time complexity to evaluate the performance of algorithms. The path planning algorithms, which use a heuristic search, have a time complexity of tree search based on the node, but it is interpreted in a time complexity based on a grid map because the time complexity used for performance evaluation varies with the heuristic and branching factor (child node). It means the time complexity varies depending on the heuristics and branching factor. Also, most heuristic search algorithms apply the sorting methods (ex: Binary heap, quick sorting) when it used in $n \times n$ maps, but the time complexity also varies depending on the sorting method. Simply, the evaluation method of each algorithm using the order and dominant value of Big-O notation has difficulty of evaluating in an environment where there is no reference algorithm.

To solve each detailed issue in focus 2, the following three ideas are introduced in this study. (a) To reduce the expanded nodes of the global path on the wide map, a splitting D* lite algorithm is proposed to split the map in advance so that the expanded nodes are ignored if the obstacle node information is updated from other areas of the split map, which is not the current area of the selected map. (b) An auto-splitting D*lite with an auto-clustering method is proposed so that a split map of the same size can create expanded nodes with uniform clustering points. (c) To propose a Number for the Expected value of Replanned Node (ERPN) method that quantifies the performance of the algorithm using the expected value of the probability principle in which the expanded node occurs.

In the focus 3, one detailed research focus is described, and one idea is introduced. The detailed focus is the local trajectory planning for usage in mobile robots, autonomous vehicles on roads, and complex environments targeted after applying the global path planning methods. A method of selecting lane changes and avoiding obstacles in urban areas and complex environments is considered along with the planned global path planning. There is a critical issue with traditional Cartesian frame trajectory planning methods. The Cartesian frame trajectory planning method has the advantage of easy application because it has the same coordination as the global path planning method. However, the Cartesian frame trajectory planning method has disadvantages in selecting one of several sets of vehicle trajectories to avoid actual obstacles using a reference path. In particular, to make several trajectories suitable for the situation based on the reference path, it is necessary to create a trajectory in consideration of speed and acceleration. At this time, if the Cartesian frame is used, the equation is complex. Therefore, the idea to solve issues is to use the reference path as an axis of the Frenet Frame coordinate. By transforming the coordinate system, the Frenet Frame trajectory planning method is applied for selecting one of several sets of vehicle trajectories to avoid actual obstacles using a reference path containing information such as a velocity. Additionally, Kalman filter is applied to increase location accuracy.

Chapter 2

Related Work

2.1 Introduction of Path Plnanning Algorithms

In section 2.1, several path planning methods are introduced according to the type and method of the robot. Related work on disaster-rescuing robots and autonomous driving are being studied. Mobile robots equipped with robot modules, such as robot arms, fire suppression modules, cutter-feller modules for breakthroughs, etc., for various missions are introduced. Specifically, path planning methods are required by wheel-based robots, and control methods are included in various robot modules. In this case, path planning methods focusing on shortest path planning, path smoothing, and trajectory planning are often used in combination. Representative research is listed as follows.

- Howe & Howe's Thermite RS-T3, RS2-T2 (a mobile robot with 2,500 gallons of water and fire suppression foam and robot arms for missions) [12]
- Shar Robotics's Barrakuda and Bulkhead [13]
- Zebro (firefighting robots using cluster algorithms) of Robotics Without Borders (disaster robot advocacy group) [14].

Autonomous vehicles and humanoid rescue robots belong to other categories, many of them require sensors restrictions, but the same algorithms are applied. Since autonomous vehicles need to recognize their surroundings at high speed and require an enormous amount of data, the current research trend is to plan paths with cameras. In addition, there are many studies on camera-oriented path planning methods because humanoid rescue robots also perform missions based on human motifs, despite the fact that their moving speed is not fast. In other words, sampling-based algorithms and neural network-based behavioral trajectory planning algorithms are widely used on autonomous vehicles and humanoid-type rescue robots. Representative research is listed as follows.

- Autonomous vehicles of Tesla
- Atlas of Boston dynamics (a representative humanoid rescue robot) [15].

Many path planning methods are classified according to their features, and the pros and cons of the selected algorithms for this study is described in Section 2.1. It was reprocessed by referring to Sara Abdallaoui's algorithm classification [16]. The classifications of this study were conducted focusing on key points among the references, and further details are described in the research aims section of each chapter.

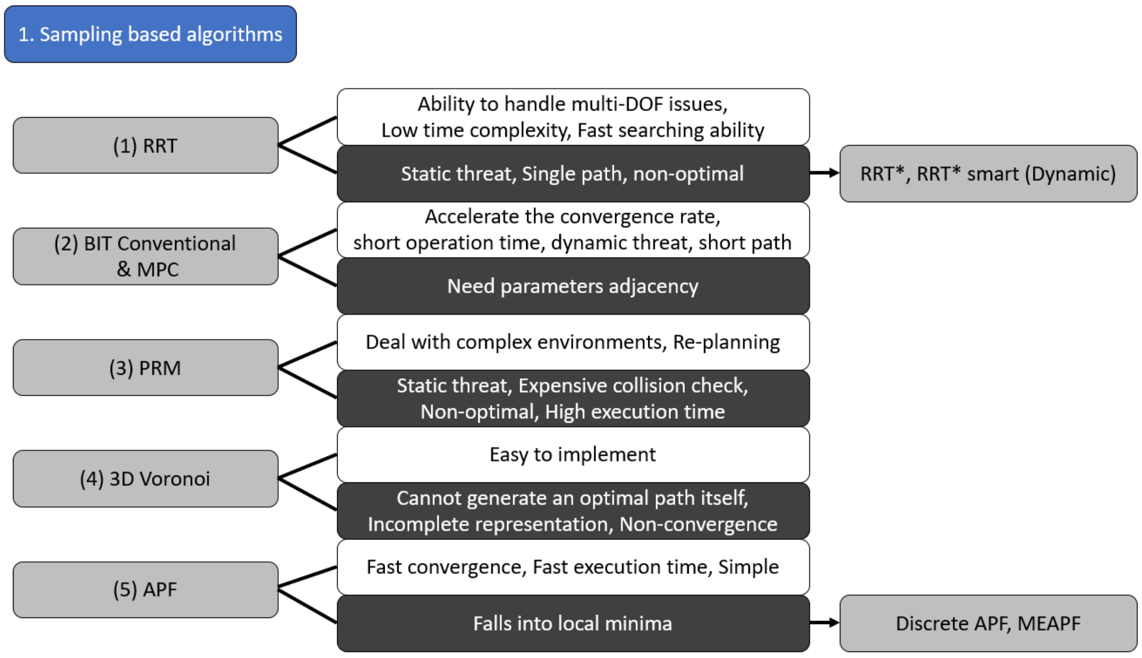

1. Sampling based algorithms
(1) RRT
Ability to handle multi-DOF issues, Low time complexity, Fast searching ability
Static threat, Single path, non-optimal
RRT*, RRT* smart (Dynamic)
(2) BIT Conventional & MPC
Accelerate the convergence rate, short operation time, dynamic threat, short path
Need parameters adjacency
(3) PRM
Deal with complex environments, Re-planning
Static threat, Expensive collision check, Non-optimal, High execution time
(4) 3D Voronoi
Easy to implement
Cannot generate an optimal path itself, Incomplete representation, Non-convergence
(5) APF
Fast convergence, Fast execution time, Simple
Falls into local minima
Discrete APF, MEAPF

Fig. 2.1 Categorization of Sampling based Algorithms

Fig. 2.1 is a simple introduction of sampling-based algorithms. Sampling-based algorithms can be divided into active and passive ones. It is decided by whether a planned path is generated from connecting the points that are randomly extracted from state space. The representative algorithms of an active sampling-based algorithm are RRT [17], BIT conventional [18] and MPC algorithms, which are shown in the figure. The representative processing procedure of an active algorithm is to select many paths to achieve the best possible path. The passive algorithm refers to an algorithm that generates only one path from the start point to the target point, such as PRM [19], 3D Voronoi [20], and APF [21] algorithms, which is usually used in combination with a searching algorithm that selects the best path among all possible paths. A well-known study for the sampling-based algorithm is the Google Waymo Project, which uses Lidar and image-based sensors to implement global and local trajectory planning by applying RRT*, RRT* smart-based algorithms, and deep learning technologies that solve static issues of RRT, which the path is planned based on sampling points. In this study, as explained in Chapter 2, sampling-based algorithm is excluded by the following environmental conditions of the shortest path.

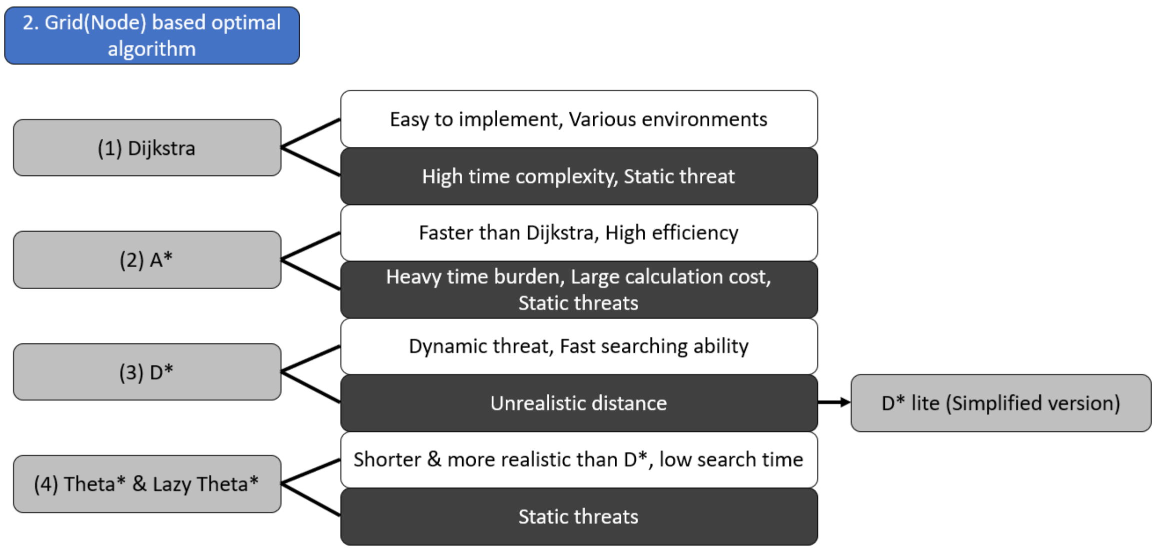

2. Grid(Node) based optimal algorithm
(1) Dijkstra
Easy to implement, Various environments
High time complexity, Static threat
(2) A*
Faster than Dijkstra, High efficiency
Heavy time burden, Large calculation cost, Static threats
(3) D*
Dynamic threat, Fast searching ability
Unrealistic distance
D* lite (Simplified version)
(4) Theta* & Lazy Theta*
Shorter & more realistic than D*, low search time
Static threats

Fig. 2.2 Categorization of Grid based Algorithms

Fig. 2.2 is a simple introduction of grid-based algorithms. The grid-based algorithms focus on searching for the shortest path, unlike the sampling-based algorithm. These methods find the shortest path by exploring among a set of cells in the map, where information sensing and processing procedures are already executed. The grid-based algorithms are categorized into 1. grid search and 2. graph search in the field of autonomous driving. Both the categorized parts are possible to be applied with the same algorithm, but there are differences in the application method. A grid search step is mapping the environment to a set of cells, and each cell indicates the presence or absence of an obstacle at the corresponding location. Therefore, this approach is mapping low-speed image data, which is not suitable for high-speed driving if the mapping time is taken into account. This method is called grid-based algorithm or grid search. On the other hand, graph search includes the process of discretizing a map into a graph. Unlike nodes in a certain grid, various shapes such as tree shapes and geometric shapes can be used during the discretization process. The same algorithm is used as a node in terms of algorithms, which is a little more suitable for high-speed driving but shows lower accuracy than grid search. This study uses the grid search method, which is easy to apply because the goal of this research is to improve the performance of existing algorithms with a multi-fusion algorithm searching for the shortest distance path with a global path planning method and a local trajectory planning method. Therefore, the associated algorithm is unified as the grid-based algorithm, not the node-based algorithm after this chapter. Some well-known representative research on grid-based algorithms is listed as follows.

- Dijkstra example - [22]
- A* example / LPA* example - [23]
- D* example - [24]
- D* lite example - [20]
- Theta*/Lazy Theta* - [25].

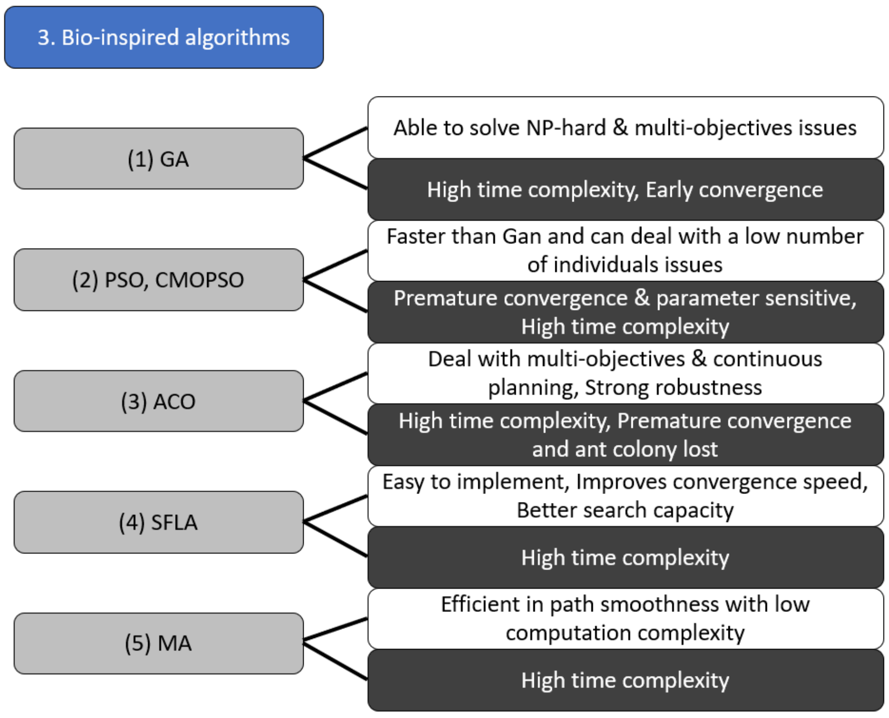

3. Bio-inspired algorithms
(1) GA
Able to solve NP-hard & multi-objectives issues
High time complexity, Early convergence
(2) PSO, CMOPSO
Faster than Gan and can deal with a low number of individuals issues
Premature convergence & parameter sensitive, High time complexity
(3) ACO
Deal with multi-objectives & continuous planning, Strong robustness
High time complexity, Premature convergence and ant colony lost
(4) SFLA
Easy to implement, Improves convergence speed, Better search capacity
High time complexity
(5) MA
Efficient in path smoothness with low computation complexity
High time complexity

Fig. 2.3 Categorization of Bio-inspired Algorithms

Fig. 2.3 is a simple introduction of the bio-inspired algorithms. Bio-inspired algorithms usually search for shortest paths with stochastic approaches. In particular, they are characterized by solving the nonlinear problem of 'NP', and the time complexity of these kinds of algorithms is usually high. This research excludes the bio-inspired algorithms due to their high time complexity, which is necessary to be considered while widening the map. Some well-known representative research on bio-inspired algorithms is listed as follows.

- GA based example - [26]
- PSO,CMOPSO example - [27], [28]
- ACO example - [29]
- SPLA example - [30], [31]
- , MA example - [32]

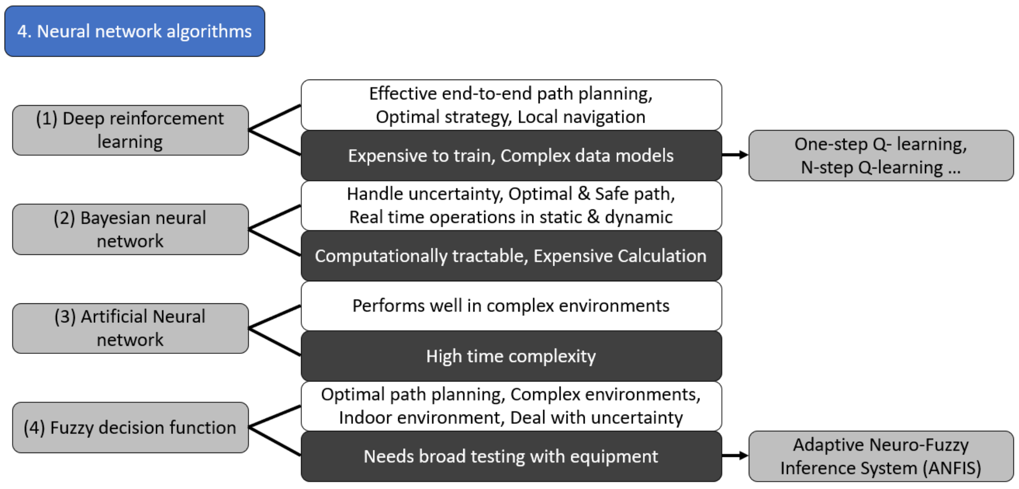

4. Neural network algorithms
(1) Deep reinforcement learning
Effective end-to-end path planning, Optimal strategy, Local navigation
Expensive to train, Complex data models
One-step Q- learning, N-step Q-learning ...
(2) Bayesian neural network
Handle uncertainty, Optimal & Safe path, Real time operations in static & dynamic
Computationally tractable, Expensive Calculation
(3) Artificial Neural network
Performs well in complex environments
High time complexity
(4) Fuzzy decision function
Optimal path planning, Complex environments, Indoor environment, Deal with uncertainty
Needs broad testing with equipment
Adaptive Neuro-Fuzzy Inference System (ANFIS)

Fig. 2.4 Categorization of Neural Network Algorithms

Fig. 2.4 is a simple introduction of neural network-based algorithms. As a field that is currently in the spotlight, neural network algorithms can be processed in all fields such as path planning, local trajectory planning, and motion planning. It has many strengths in the trajectory planning part, which learns sensor data to maintain lane change, lane-keeping, and speed. The execution time in the global path planning and trajectory planning is also fast, but physical costs and enormous time are required for learning. One of the best self-driving systems developed by Tesla currently uses a sampling-based algorithm with only eight image cameras but operates with more than 5,000 people in the training team to train numerous camera information. For this reason, this research excludes studies on the subject of neural network algorithms, but the method of applying its basic principles is considered for future research. Some well-known representative research on the neural network algorithms is listed below.

- Deep reinforcement Learning - [33], [34], [35], [36]
- Bayesian neural network - [37]
- Artificial neural network - [38]
- Fuzzy decision Function - [39]

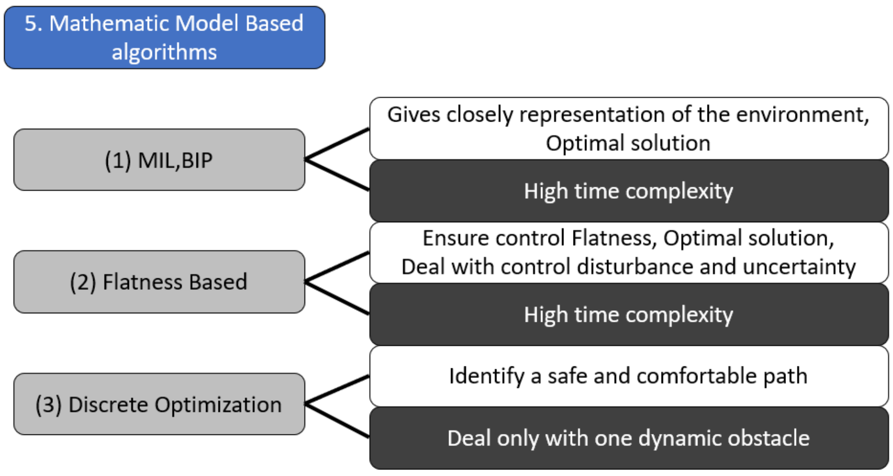

5. Mathematic Model Based algorithms
(1) MIL,BIP
Gives closely representation of the environment, Optimal solution
High time complexity
(2) Flatness Based
Ensure control Flatness, Optimal solution, Deal with control disturbance and uncertainty
High time complexity
(3) Discrete Optimization
Identify a safe and comfortable path
Deal only with one dynamic obstacle

Fig. 2.5 Categorization of Mathematic Model based Algorithms

Fig. 2.5 is a simple introduction of the mathematical model-based algorithms. The mathematical model-based algorithms can be divided into two subcategories: linear algorithms and optimal control. It is usually based on kinematic interpretations and geometric approaches. Obtaining optimal solutions using cost functions is a feature of mathematical model-based algorithms. The mathematic model-based algorithms are also included in a traditional vehicle control method, which determines the speed and acceleration of a vehicle by calculating all complex situations and sizing them by variable. The mathematic model-based algorithm is used only for control methods and is excluded from this research because most of the research uses global path planning and local trajectory planning. Some well-known representative research of the mathematic model-based algorithms is listed as follows.

- MIL, BIP - [40]
- Flatness based - [20]
- Discrete Optimization - [41]

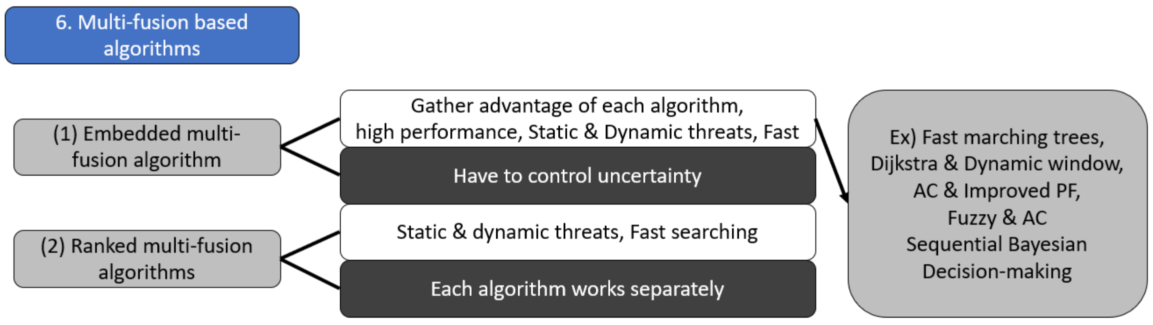

6. Multi-fusion based algorithms
(1) Embedded multi-fusion algorithm
Gather advantage of each algorithm, high performance, Static & Dynamic threats, Fast
Have to control uncertainty
(2) Ranked multi-fusion algorithms
Static & dynamic threats, Fast searching
Each algorithm works separately
Ex) Fast marching trees, Dijkstra & Dynamic window, AC & Improved PF, Fuzzy & AC Sequential Bayesian Decision-making

Fig. 2.6 Categorization of Multi-fusion based Algorithms

Fig. 2.6 is a simple introduction of the multi-fusion-based algorithms. Currently, most of the studies are multi-fusion-based algorithms. In most cases, the multi-fusion-based algorithms can be applied quickly to static and dynamic environments at the same time with high performance. Considering the research objectives and environment, the subject of this research would be using the grid-based global path planning method and the Frenet-base local trajectory planning method, which can also be viewed as a Multi-fusion based algorithm. To sum up, the proposed algorithm in this research is a multi-fusion-based algorithm based on the grid-based algorithms with the Frenet frame local trajectory planning methods.

2.2 Research Flow

As explained in the related research, this research is about the path planning methods based on A* and D* lite algorithms for the grid-based global path planning methods. The trajectory of the final autonomous vehicles for patient transport are generated using the Frenet frame conversion method. Therefore, the final trajectory is in the form of multi-fusion-based algorithm combined with the grid-based algorithm with trajectory planning using the frame conversion. Most of the related work in this paper are focused on the new development of the grid-based algorithm. The algorithms proposed in Chapters 3 through 5 are introduced according to the obstacle environment. Especially, the first section of each chapter represents deatil reasearch aims. These sections are introduce advantages and disadvantages for the grid-based algorithms with the detailed issues depending on the environment of maps and obstacles, and explain the detailed research aims to be addressed from the issue. The flow of overall research is shown in the figure 2.7.

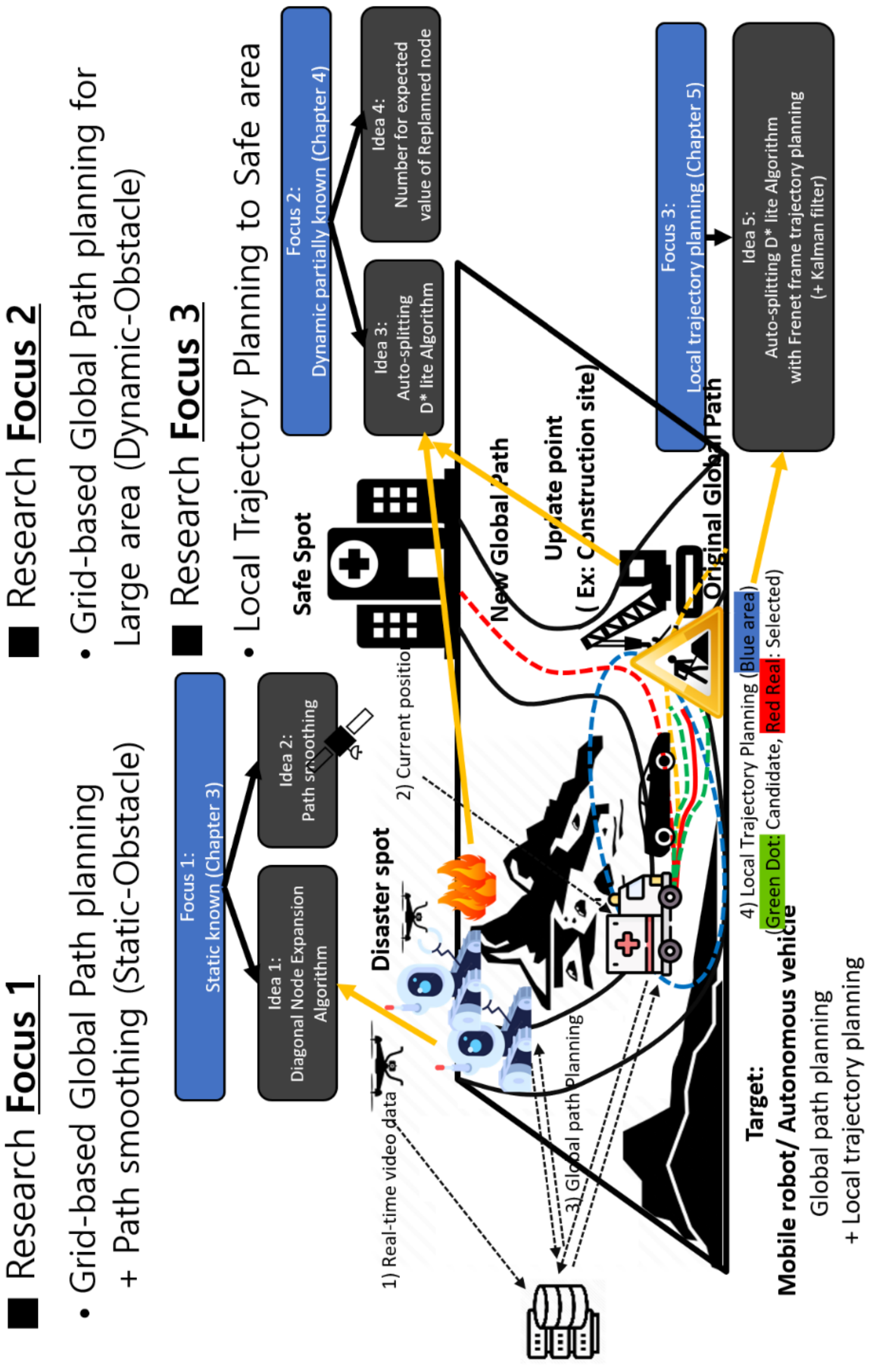

Research Focus 1
• Grid-based Global Path planning
+ Path smoothing (Static-Obstacle)
Research Focus 2
• Grid-based Global Path planning for Large area (Dynamic-Obstacle)
Research Focus 3
• Local Trajectory Planning to Safe area
Focus 1: Static known (Chapter 3)
Idea 1: Diagonal Node Expansion Algorithm
Idea 2: Path smoothing
Focus 2: Dynamic partially known (Chapter 4)
Idea 3: Auto-splitting D* lite Algorithm
Idea 4: Number for expected value of Replanned node
Focus 3: Local trajectory planning (Chapter 5)
Idea 5: Auto-splitting D* lite Algorithm with Frenet frame trajectory planning (+ Kalman filter)
Safe Spot
Disaster spot
1) Real-time video data
2) Current position
3) Global path Planning
4) Local Trajectory Planning (Blue area)
Green Dot: Candidate, Red Real: Selected)
New Global Path
Update point
(Ex: Construction site)
Original Global Path
Target:
Mobile robot/ Autonomous vehicle
Global path planning
+ Local trajectory planning

Fig. 2.7 Research Flow

A brief summary of each chapter is as follows. The grid-based global path planning methods in the static known obstacle areas are described in Chapter 3. We propose a diagonal node expansion algorithm, which is applicable to all grid-based algorithms using the euclidian distance and expands only four of the eight peripheral nodes of the selected current node. Also, we introduce a modified Shoemake's scheme to quickly smooth zig-zag paths resulting from existing zig-zag issues and further extend diagonal nodes in this chapter. The diagonal node expansion algorithm reduces the expanded nodes and the calculation time. If the size of a map and information is large, the calculation time will be reduced compared with an original grid-based path. This algorithm will help find the shortest and safest path for the rescue robots and the first responders The main environment is explained in the static area, but before entering Chapter 4, the diagonal node expansion in the dynamic area is also described for the rescue robots. However, a more suitable algorithm for rescue robot in dynmaic area is introduced in Chapter 4. In Chapter 4, we describe the grid-based global path planning method in the dynamic and partially known environment.To reduce the expanded nodes of the global path on the wide map, a splitting D* lite algorithm is proposed to split the map in advance so that the expanded node is ignored if the obstacle node information is updated from other areas of the split map, which is not the current area of the selected map. Then, an auto-splitting D*lite with an auto-clustering method is introduced so that a split map of the same size can generate an expanded node with uniform clustering points. Also, an expected value of replanned node (ERPN) method that quantifies the performance of an algorithm is considered. The expected value using the probability principle is applied, in which the expanded node occurs. In Chapter 5, local trajectory planning methods in dynamic partially known environment are described. The local trajectory planning methods that are used in a complex environment are mainly discussed. The way to produce faster and more accurate results than the existing local trajectory planning methods in the dynamic partially known obstacle area is focused in this chapter. Along with the planned global path, a method of selecting lane changes and avoiding obstacles in urban areas and complex environments is simulated. This chapter introduces a method to convert the Auto-splitting D*lite into a reference path of the Frenet Frame, and to use the reference path as an axis of the Frenet Frame coordinate. By transforming the coordinate system, the Frenet Frame trajectory planning method is applied for selecting one of several sets of vehicle trajectories to avoid actual obstacles using a reference path containing information such as a velocity.

Chapter 3

Diagonal Grid-based Global Path Planning for Static Obstacle Environment

3.1 Research Aims

The main targets in this chapter are the path planning and the path smoothing methods for the first responders and rescue robots to assist rescue in static known area. The first responders need the shortest and fast path planning. Therefore, a static obstacles environment based path planning is considered. On the other hand, the rescue robots cannot react rapidly to changing areas such as radical fire, explosion, and collapse. Therefore, a dynamic obstacles environment based path planning is one of the solutions to handle this dynamic situation. These path planning are possible to divide into the static known environment and the dynamic partially known environment [20]. In this study, we will introduce how to shorten the calculation time of the path for two grid-based path planning algorithms represented by these two environments. The grid-based path is finding the smallest cost algorithm, so the proposed path planning has the shortest distance of the path, and faster calculation for a large area. Also, the proposed path planning is a solution for safety using a weighted procedure. It mainly describes static obstacle path planning for first responders, and additionly D*lite algorithm is introduced for the rescue robot.

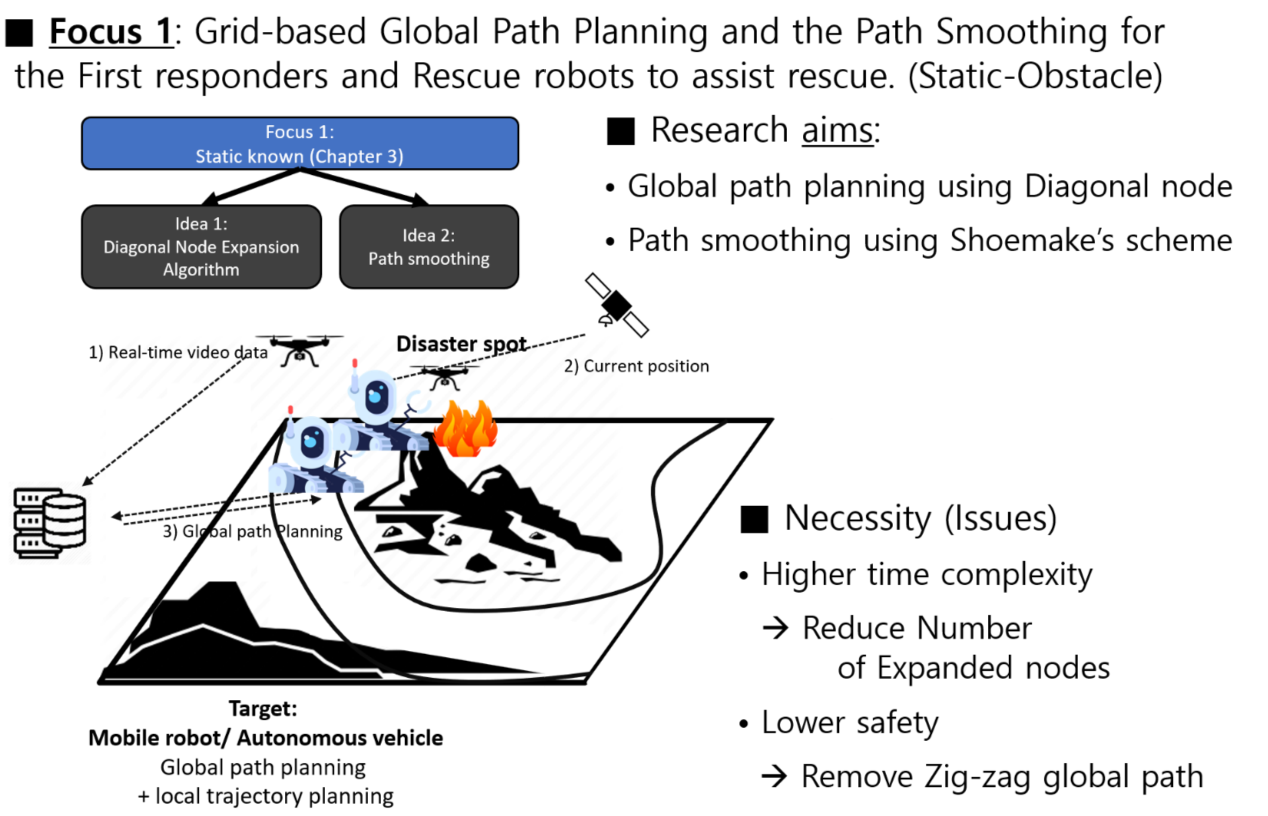

■ Focus 1: Grid-based Global Path Planning and the Path Smoothing for the First responders and Rescue robots to assist rescue. (Static-Obstacle)
Focus 1: Static known (Chapter 3)
Idea 1: Diagonal Node Expansion Algorithm
Idea 2: Path smoothing
■ Research aims:
• Global path planning using Diagonal node
• Path smoothing using Shoemake's scheme
1) Real-time video data
Disaster spot
2) Current position
3) Global path Planning
■ Necessity (Issues)
• Higher time complexity
→ Reduce Number of Expanded nodes
• Lower safety
→ Remove Zig-zag global path
Target:
Mobile robot/ Autonomous vehicle
Global path planning
+ local trajectory planning

Fig. 3.1 Research Aims (Chapter 3)

One of the common issues of path planning methods in a known environment is the long calculation time. The following algorithms are typical: A* algorithm [42] is a representative in the static known obstacles environment search with a stationary start, stationary goal, and stationary obstacles in two-dimension (2D). GAA* algorithm [43] is an adaptive searching algorithm in the static known obstacles environment and it uses stationary start, stationary goal, and movable obstacles in 2D. It implements forward and backward steps similar to D* lite. Wang et al. [44], [45] discuss research on path planning for the first responder with a stationary start, movable goal, and movable obstacles using A* algorithm. Delmerico and Jefferey [46] apply the A* and rescue robot in a static known obstacles environment with a stationary start, stationary goal, and stationary obstacles. Especially, they use A* with a three-dimension (3D) map environment, but path planning is in 2D environment. The drawback that including from A* to Delmerico et al. is that calculation time increases depending on given information and map size. One of the common issues of path planning algorithms in a dynamic partially known obstacles environment is also a long calculation time. Some typical algorithms and their conditions are provided below briefly. Most of grid-based path planning algorithms under the dynamic partially known obstacles condition are related to D* algorithm [10] and D* lite algorithm [11]. The calculation time of these algorithms is longer than A* because the recalculation procedure on every new environment updating is needed. D* lite is a faster and optimized version of LPA* (Lifelong plan A*) [47]. It has a stationary start, stationary goal, and movable obstacles. Especially, movable obstacles are recognized as a new updating target in every recalculation procedure. MTD* lite (Moving Target D*) [48] focuses on stationary start, but it has a movable goal in a dynamic partially known obstacles environment. It also has movable but uncertain obstacles in 2D situation. Field D* [49] focuses on nodes connection and faster calculation on an interline node in a dynamic partially known obstacles environment. It assumes stationary start, stationary goal, and movable obstacles in 2D. 3D Field D* [50] is developed in 3D dynamic partially known obstacles environment. It assumes stationary start, stationary goal, and movable as well as uncertain obstacles in 3D. Disadvantages of those algorithms including from D* lite to 3D field D* have the same issue as the A*, that is the calculation time is longer. To solve the common issues of previous research for path planning, this study proposes a diagonal node exapnsion algrorithm in the static known obstacles environment for first responders and the dynamic partially known obstacles environment for rescue robots. Most of the algorithm introductions are described in the static known obstacle area. The diagonal node expansion algorithm in the static known obstacles environment is based on A* algorithm. The diagonal node expansion algorithm in the dynamic partially known obstacles environment is based on D* lite algorithm. Both proposed algorithms decrease the path calculation time for solving common issues of previous research. In summary, this research has two improvement points in the path planning method and the path smoothing method part to reduce the calculation time. On the path planning method part, the diagonal node expansion algorithm is applied for reducing calculation time in the static known obstacles area for the first responders and the dynamic partially known obstacle area for the rescue robots. On the path smoothing part, a modified Shoemake's scheme is applied to solve the zig-zag issue on 3D space. Later in this chapter, the static known obstacles environment is simply named known evironment, and the dynamic partially known obstacles environment is simply named partially known evironment.

3.2 Diagonal Node Expansion Algorithm and Path Smoothing Method

3.2.1 Diagonal Node Expansion

Diagonal node expansion algorithm is an algorithm that aims at reducing the calculation time for the elimination of a node in a conventional grid-based algorithms. The related work stated that the grid-based algorithm is based on A* and D* lite depending on the environment. The two algorithms have the same structure and shape of an enclosed node, so the node elimination process is equally applicable except node expansion and operation. The approximate idea for the node elimination is as follows: The left side of 3.2 shows open-nodes of the original A* and D* lite. The right side of 3.2 shows open-nodes and disable nodes of the diagonal node expansion algorithm.

The operating principles of A* are as follows before applying the node elimination process and the diagonal node: for every node n encountered during the search, A* maintains three values. First, g-value is defined as $g(n)$, which is infinity initially and has the length of the shortest discovered path from the start node to node n. Second, h-value is defined as $h(n) := H\left(n, n_{\text{goal}}\right)$, which estimates the distance from a goal node to node n. Finally, f-value is defined as $f(n) := g(n) + h(n)$, which estimates the distance from the start node via node n to the goal node. For the algorithm to find the actual shortest path, the heuristic function $h(n)$ must be admissible, because $h(n)$ never overestimates an actual cost to get the nearest goal node. This algorithm maintains two sets OPEN and CLOSED: the OPEN set saves all the nodes that can be detected but not accessible yet; the CLOSED set saves the nodes already visited. The heuristic function $h(n)$ is sorted in the OPEN set according to the result of $f(n)$, which selects the minimum cost for each node to access. The result of $f(n)$ finds the relatively shortest way to the destination.

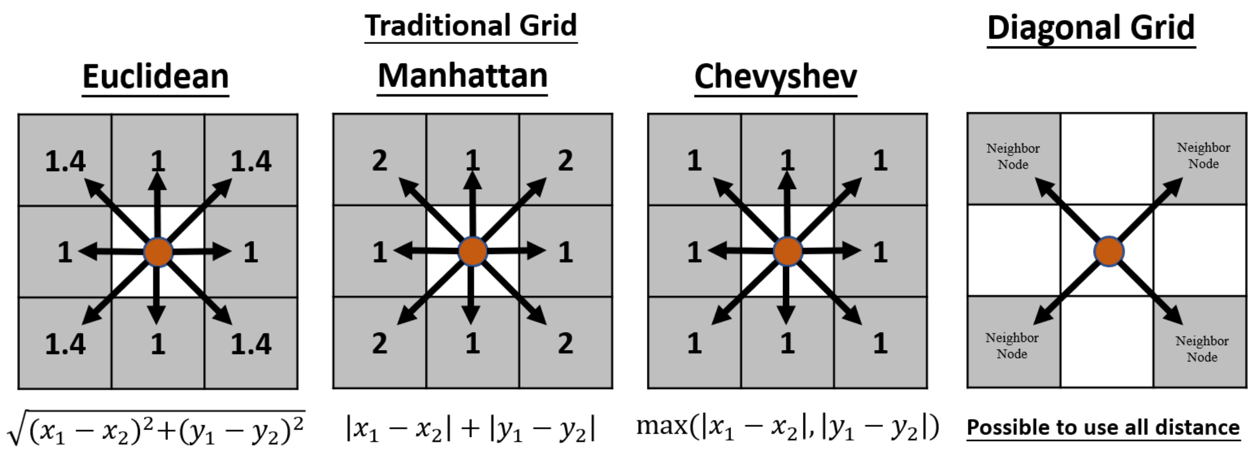


Fig. 3.2 Comparision of Neighbour Node

3.2.1.1 Diagonal A* Path Planning

The diagonal A* consists of the elimination of the node and the brief modification of the procedure in the traditional A*. Following steps describe the procedure of diagonal A* illustrated in 3.3: Step 1: All nodes require the states that are not visited, and initialize OPEN set and CLOSED set. Step 2: Start node need to put on the list of OPEN set. The current node is the node with the lowest f-cost in OPEN set with calculating the f-cost of all nodes. Step 3: Remove the current node from OPEN set and add a current node to CLOSED set. Step 4: Check whether the current node is a target node. Step 5: Move to a next state that is not visited diagonal neighbor nodes with the smallest f-cost and repeat the above steps, which checks diagonal neighbors and mark visited state. Step 6: If the diagonal neighbor nodes are not traversable or are in CLOSED set, skip to the next smallest f-cost neighbor. Step 7: If the new path to a neighbor is shorter or the neighbor node is not in OPEN set, update the f-cost of neighbor and add the neighbor to OPEN set.

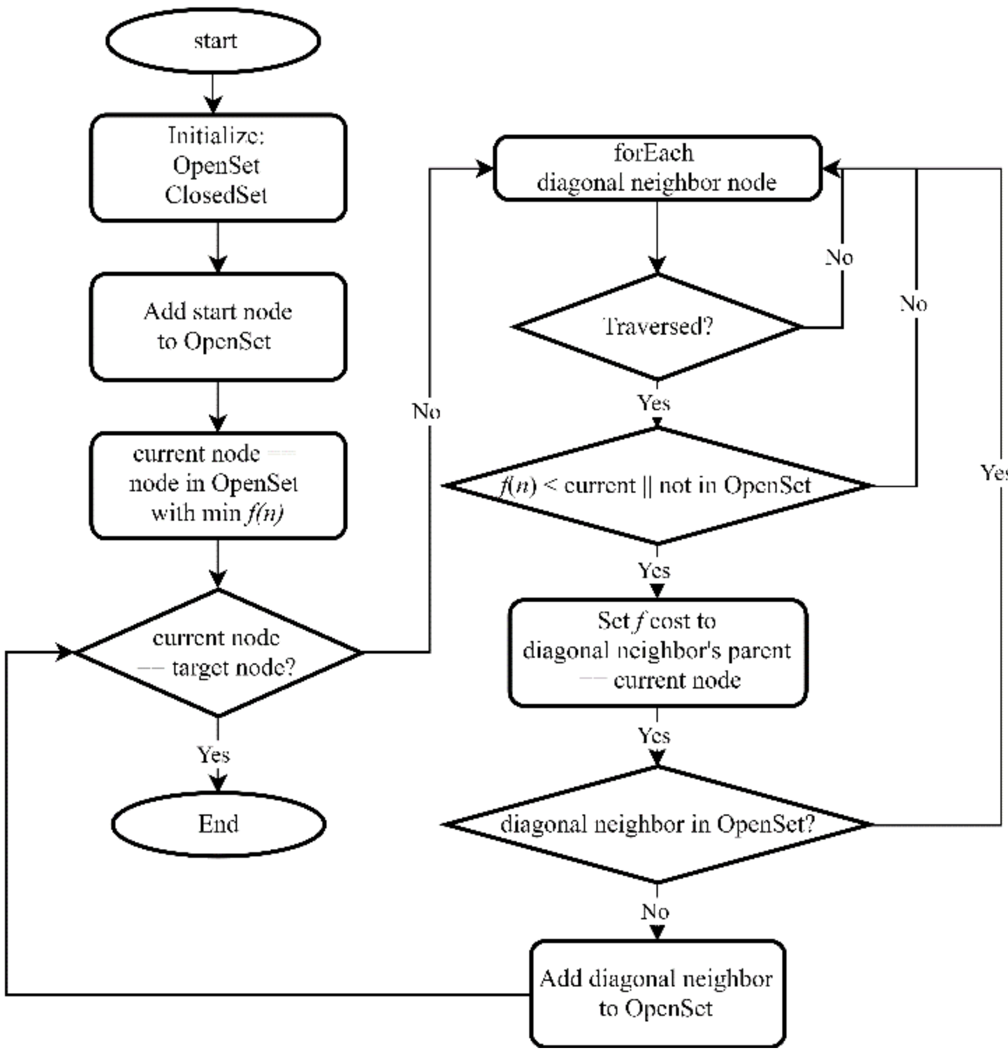

start
Initialize: OpenSet ClosedSet
Add start node to OpenSet
current node == node in OpenSet with min $f(n)$
current node == target node?
Yes
End
No
forEach diagonal neighbor node
Traversed?
No
Yes
$f(n)$ < current || not in OpenSet
No
Yes
Set f cost to diagonal neighbor's parent == current node
Yes
diagonal neighbor in OpenSet?
Yes
No
Add diagonal neighbor to OpenSet

Fig. 3.3 Flowchart of A* Algorithm

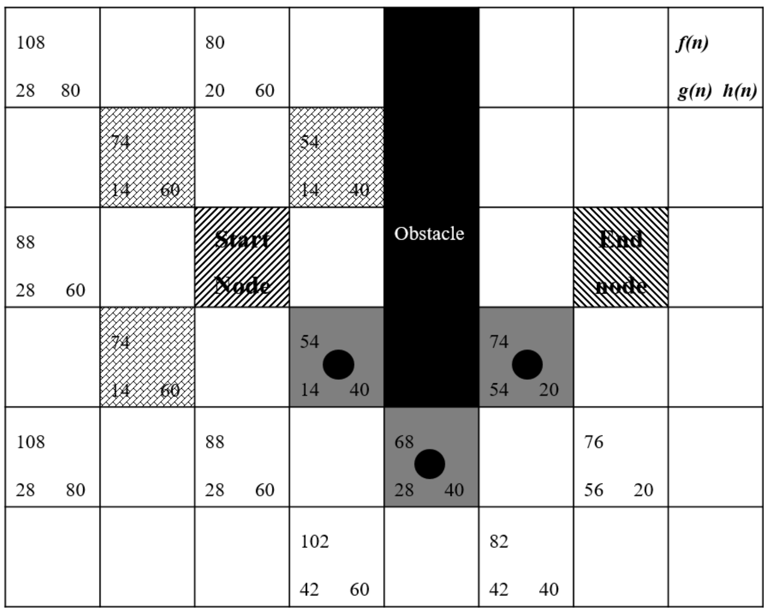


Fig. 3.4 Cost Map of A* Algorithm

From Step 1 to Step 7, all procedures of the diagonal A* with the cost information of $f(n)$, $g(n)$, and $h(n)$ are illustrated in 3.3 as an example. Especially, the heuristic function $h(n)$ in the diagonal A* uses a Manhattan distance, because the expanded nodes in OPEN set using a Euclidian distance are bigger than the Manhattan distance. As shown in 3.5, the black nodes represent both ends, and the gray nodes are the expanded nodes in the OPEN set. Simply, allowable grid movements are possible to summarize: (a) any directions (b) eight directions (c) four cross-line directions (d) four diagonal-line directions (Node elimination is not allowed to cross-line directions)

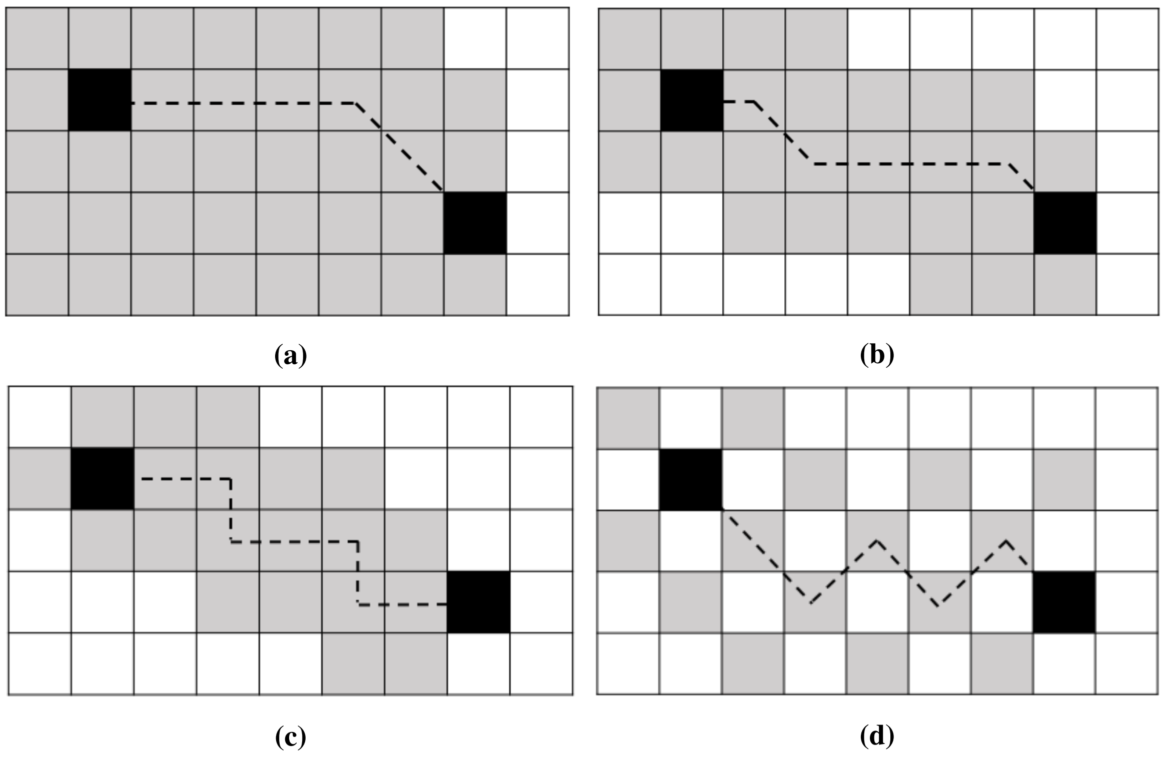


Fig. 3.5 Various Distance for Heuristic Fuction

One more combination might make the assumption, which is diagonal distance and diagonal node. This combination perform the same as (d), but calculation time is longer because a procedure of the heuristic function is slightly increased. Therefore, the Manhattan distance is used in this diagonal A* algorithm. All this procedure for path planning in a known environment has the purpose of reducing calculation time. And then, the first responder is quickly reached to the target position using the diagonal A*. This expanded node has a profound effect on calculation time. Discussion for time complexity depended on the expanded node is on the last part of this study.

3.2.1.2 Diagonal D* lite Path Planning

This section is an additional dynamic part of this chapter. It shows that the first responder as well as the Diagonal node expansion algorithm is available in dynamic environments when the rescue robot moves. However, a more suitable algorithm for rescue robot in dynmaic area is introduced in Chapter 4. A diagonal D* lite path planning is based on an original D* lite path planning, and the original D* lite is an upgraded version of LPA*, which can adapt to change in the path planning without recalculating the entire search. D* lite is one of most popular goal-directed navigation algorithms and widely used in partially known environment. It is adaption of LPA*, which is an incremental derivation of A*. It determines the same paths as D* algorithm and moves the agent the same way but it is algorithmically different. Therefore, the diagonal D* lite also determines how to efficiently update the shortest path under changing edge costs using diagonal nodes with the Manhattan distance. The efficiency is achieved by only updating the values that need to be updated to find the shortest path as same as the D* lite.

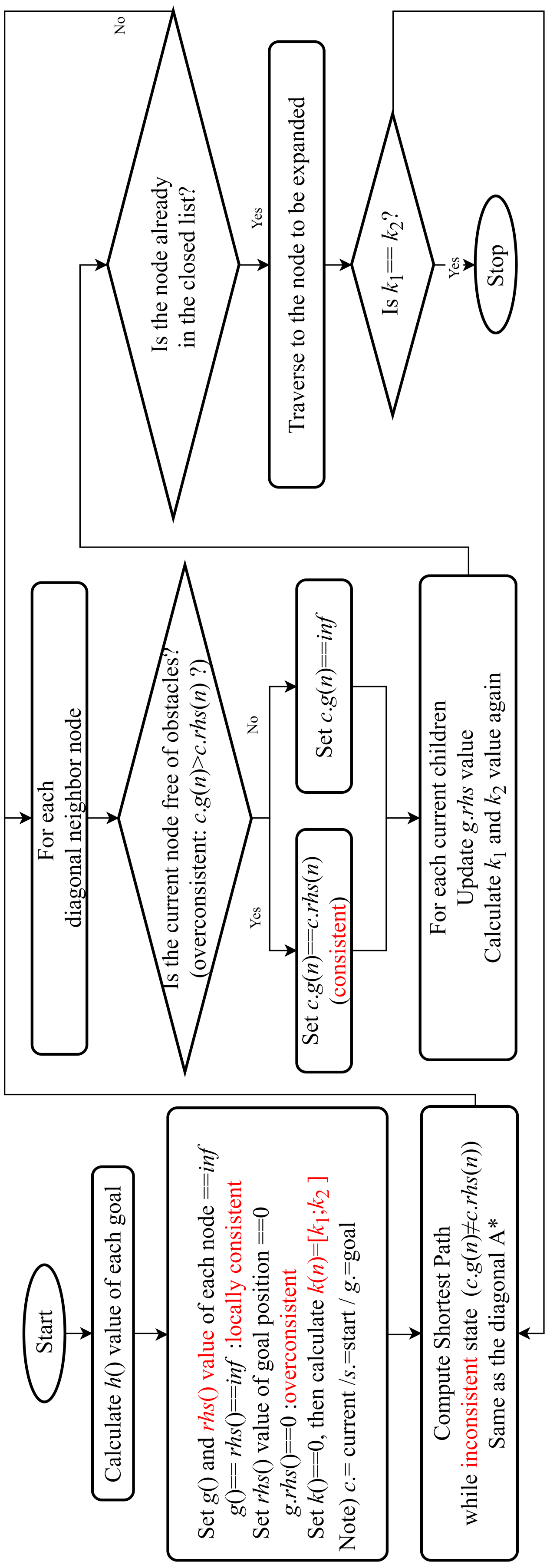

Start
Calculate $h()$ value of each goal
Set $g()$ and $rhs()$ value of each node ==inf
$g()$== $rhs()$==inf :locally consistent
Set $rhs()$ value of goal position ==0
$g.rhs()$==0 :overconsistent
Set $k()$==0, then calculate $k(n)$=[k_1;k_2]
Note) $c.$= current / $s.$=start / $g.$=goal
Compute Shortest Path
while inconsistent state ($c.g(n)$≠$c.rhs(n)$)
Same as the diagonal A*
For each
diagonal neighbor node
Is the current node free of obstacles?
(overconsistent: $c.g(n)$>$c.rhs(n)$?)
Yes
No
Set $c.g(n)$==$c.rhs(n)$
(consistent)
Set $c.g(n)$==inf
For each current children
Update $g.rhs$ value
Calculate k_1 and k_2 value again
Is the node already
in the closed list?
No
Yes
Traverse to the node to be expanded
Is k_1== k_2?
Yes
Stop

Fig. 3.6 Flowchart of Traditional D* lite

Unlike the diagonal A*, there are largely three differences as shown by the red text in 3.6. Most of these differences are in line with the D* lite. First, there are two estimates of the start distance $g'(n)$ for each node. The $g(n)$ in the diagonal D* lite is the same as the diagonal A*, but one more $rhs(n)$ value is included called as a right-hand side or look-ahead value based on the g-values. Therefore, $g'(n)$ is selected $g(n)$ or $rhs(n)$ depends on the consistency. The $rhs(n)$ is defined in Eq. 3.1, and a different feature compared to the LPA* is finding the shortest path from a goal node to a start node by minimizing rhs value.

D* lite maintains an estimate $g(n)$ of the start distance $g'(n)$ of each vertex n, analogous to the g-values of an A* search. D* lite carries them forward from search to search. D* lite also maintains a second kind of estimate of the start distances; the rhs-values one-step-lookahead values based on the g-values and thus potential better informed than the g-values. g-values and rhs-values should always satisfy the following equation.

$$rhs(n) := \begin{cases} 0 & \text{if } n = n_{\text{goal}} \\ \min_{n' \in \text{ succ } (n)} \left(c\left(n, n'\right) + g\left(n'\right)\right) & : \text{ otherwise} \end{cases} \tag{3.1}$$

Second, another difference between the diagonal A* and the diagonal D* lite is a key modifier, which is $k()$ value. The key modifier is a value used to sort the OPEN set, and the sorting method of the OPEN set uses a heap reordering [51]. The key modifier $k()$ are defined as $k(n) := [min(g(n), rhs(n) + h(n)); min(g(n), rhs(n)]$. The key modifiers are updated when an environment changes with the new amount of a rescue robot has traveled. The third difference between the diagonal A* and diagonal D* lite is the existence of consistency. This consistency is based on M.Likhachev and S.Koenig [10], [11], and a brief description is re-written as follows: Two-states are depending on the relationship between $g(n)$ and $rhs(n)$: Locally consistent is defined as $g(n) = rhs(n)$; locally inconsistent is defined as $g(n) \neq rhs(n)$. Especially, the inconsistent fall into two categories, which called over-consistent and under-consistent: Locally over-consistent is defined as $g(n) > rhs(n)$; locally under-consistent is defined as $g(n) < rhs(n)$. The following flow-chart of 3.6 can be configured using the three differences described above and the diagonal A* algorithm described the previous chapter. In a note of caution, the computed shortest path procedure in 3.6

is the same as the diagonal A*, but always execute the inconsistent states with the diagonal node. The inconsistent states contain all the previous locally inconsistent nodes as well as the new nodes, which are recently made inconsistent states by the changes. This algorithm is constantly checking for environment changes in the grid node with the OPEN set, which is never reset. Briefly, the detail operations have differences to the diagonal A*, but node selection has the same behavior. Until this section, the diagonal node expansion algorithms are applied in the known environment and partially known environment. However, the diagonal node expansion algorithm occurs "zig-zag" issue. Therefore, a mitigation method to solve the zig-zag issue is described in the next section, then an appropriate application method is explained.

3.2.2 Path Smoothing in 3D Environment

Path planner of this research focuses on 3D space. The 3D space is more convenient to predict intuitive distance when the first responder is going to the goal point. It is useful for the first responder and rescue robot when the 3D environment is used to find evacuated people. In addition, nowadays real-time 3D map building is possible because of applications for drones such as Drone Deploy [52], Pix4D [53] mapper. As developing the technique related to the drone, path smoothing in the 3D environment after path planning becomes important. Especially, path smoothing has a characteristic to change the node-based path to a smooth path. After path smoothing is applied, the disadvantage of the node-based algorithm is solved to reduce the slow calculation time on a large map. In other words, path planning and path smoothing execute on a small grid map, then the result of path smoothing has the possibility to convert a bigger size of the map in order to make a smooth path.

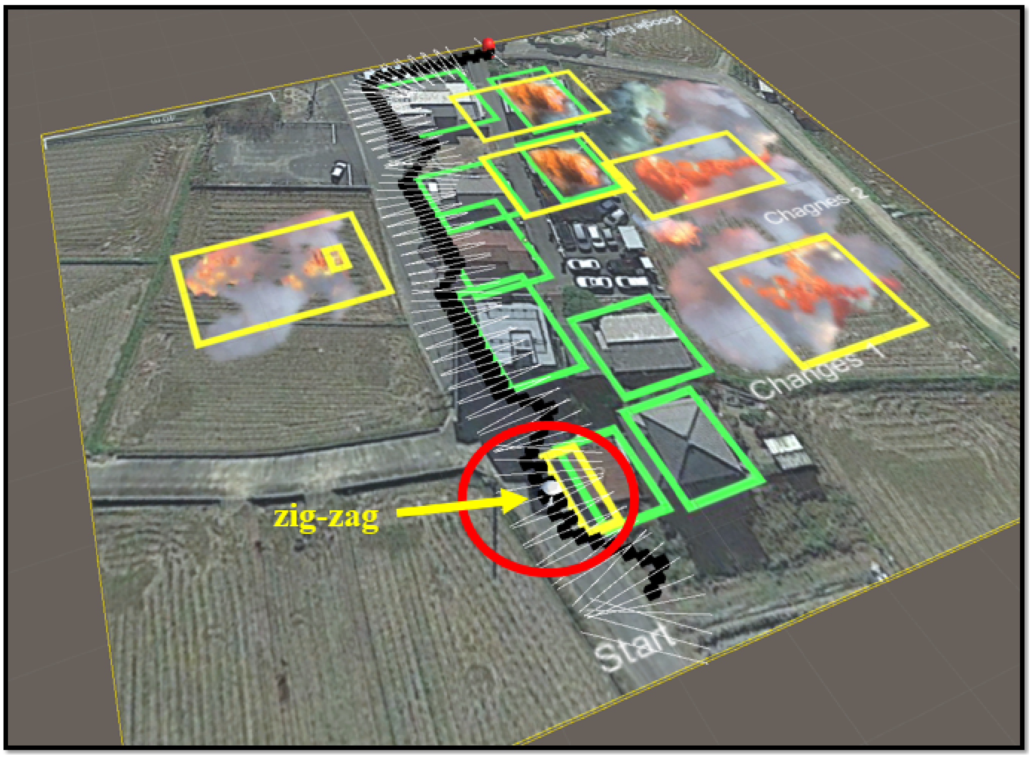


Fig. 3.7 Zig-zag Issues after Diagonal Node Expansion Algorithm

3.7 shows the zig-zag issue of 3D map in this research. To solve the zig-zag issue after diagonal node expansion algorithm, one of the composite methods is described below.

3.2.2.1 Path Smoothing using Modified Shoemake's Scheme

The combination of a composite method proposed in this study is Shoemake's scheme using LERP called as the modified Shoemake's scheme. As mentioned in the related work, many methods are considered to apply path smoothing to solve a zig-zag issue from the diagonal node expansion as path smoother using variety splines, Interpolation path smoother (IPS), and combinations of curvatures [54], [55], [56], [57]. The Shoemake's scheme is based on a quaternion space and rotation. Original Shoemake's scheme generates the smooth path to link several points such as zig-zag issue using SLERP (Spherical Linear Interpolation) and Bézier curve in the quaternion space. Instead of SLERP, a LERP (Linear Interpolation) function is applied because fast calculation time is a top priority in this research. An advantage to apply the modified Shoemake's scheme is easy converting to rotation matrices in the quaternion space. Especially, Euler angles are needed to show an animation of the path using 2D on a 3D map, and the quaternion space is easily converted from the Euler angles to the quaternion conversion. The following description shows the relation between quaternion and rotation [58], [59]. A Hamilton algebra (quaternion algebra) is defined as Eq. 3.2.

$$\mathbb{H} = \{t + xi + yj + zk \mid t, x, y, z \in \mathbb{R}\} \tag{3.2}$$

Here, i, j, and k are the three imaginary units, and it was defined by Hamilton [60] as Eq. 3.3.

$$i^2 = j^2 = k^2 = ijk = -1 \tag{3.3}$$

The product of two imaginary units satisfies these fundamental rules as Eq. 3.4.

$$\begin{array}{lll} ij = k & jk = i & ki = j \\ ik = -j & jk = -k & kj = -i \end{array} \tag{3.4}$$

A quaternion qn is denoted in Eq. 3.5.

$$\begin{aligned} q_n = t_n + (i, j, k)\vec{q}_n, \quad q_n \in \mathbb{H}, \quad n \in \mathbb{N} \quad (n: \text{ an index number}) \\ t_n \in \mathbb{R}, \quad \vec{q}_n \in \mathbb{R}^3 \end{aligned} \tag{3.5}$$

where, t_n is a real number of the real part and $\vec{q}_n$ is a real number of the imaginary part for the quaternion, respectively.

$$\vec{q}_n = \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} \in \mathbb{R}^3 \tag{3.6}$$

These sets of multiplication of quaternion product are satisfied as Eq. 3.7 when n =1 and 2.

$$\begin{aligned} \vec{q}_1\vec{q}_2 &\triangleq (-1)\,(\vec{q}_1 \times \vec{q}_2) + \vec{q}_1 \times \vec{q}_2 \\ q_1 q_2 &\triangleq t_1 t_2 (-1)\,(\vec{q}_1 \times \vec{q}_2) + (i,j,k)\,(t_1\vec{q}_2 + t_2\vec{q}_1 + (\vec{q}_1 \times \vec{q}_2)) \end{aligned} \tag{3.7}$$

An inverse element can be constructed for every quaternion number as Eq. 3.8, and a norm of a quaternion is given by Eq. 3.9.

$$\forall q_n \in \mathbb{H} : q_n^{-1} = t_n - (i,j,k)\vec{q}_n \tag{3.8}$$

$$\|q_n\| = \sqrt{q_n q_n^{-1}} \tag{3.9}$$

To illustrate a quaternion, the next procedures need how the space of rotation is constructed. Every rotation in a three-dimensional space is realized from the rotation by angle and axis. The set of possible rotations by any angle S(a) is defined as Eq. 3.10, and a unit hypersphere is defined as Eq. 3.11, which is extended four-dimensional rotations.

$$S(a); a \neq 0\,[^\circ] \tag{3.10}$$

$$S^3 = \{q_n \in \mathbb{H} : |q_n| = 1\} \tag{3.11}$$

As an example, consider a vector $\vec{v}$,

$$\vec{v} \in \mathbb{R}^3 \tag{3.12}$$

The vector $\vec{v}$ is represented as

$$v = (i,j,k)\vec{v} \tag{3.13}$$

Then, the rotation of $\vec{v}'$ is realized by using a quaternion q as

$$\vec{v}' = \mathrm{Rot}(\vec{v}) = qvq^{-1} \tag{3.14}$$

An example of 3D expressions on quaternion(4D) shows in 3.8 .

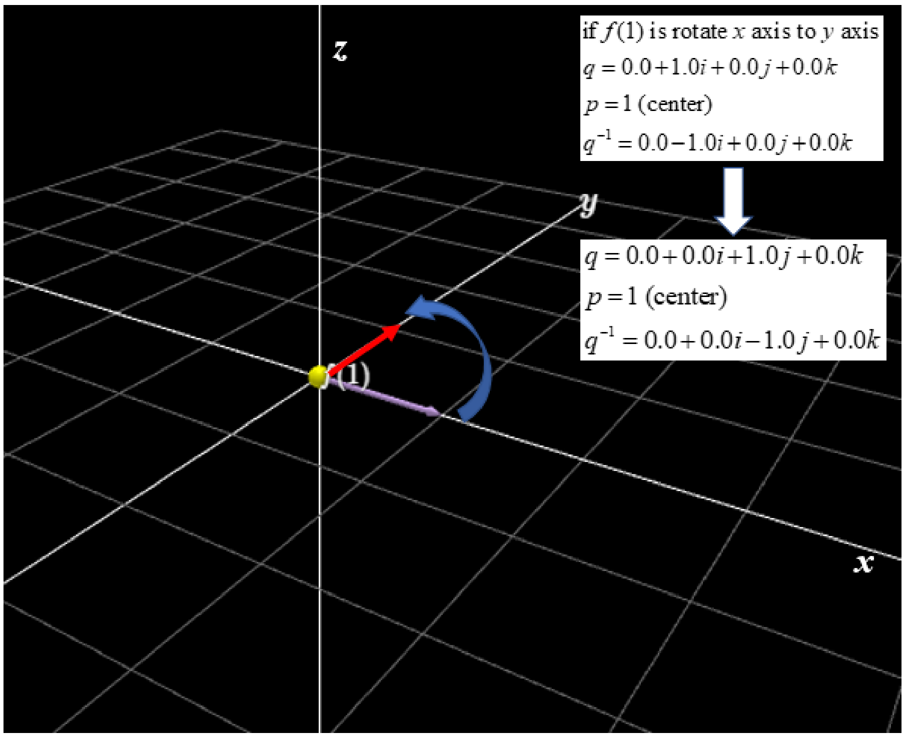

z
y
x
(1)
if $f(1)$ is rotate x axis to y axis
$q = 0.0+1.0i+0.0j+0.0k$
$p = 1$ (center)
$q^{-1} = 0.0-1.0i+0.0j+0.0k$
$q = 0.0+0.0i+1.0j+0.0k$
$p = 1$ (center)
$q^{-1} = 0.0+0.0i-1.0j+0.0k$

Fig. 3.8 Principle of Quaternion Space Conversion

After the rotation, the line connection is needed to show the computed path between two points to illustrate LERP. Eq. 3.15 represents the definition of LERP.

$$\begin{aligned}
&\mathrm{LERP}\,(p_0, p_1; t) \triangleq (1-t)p_1 + tp_1 \\
&p_0 : \text{start point} , \quad p_1 : \text{end point} \\
&t \in [0,1]
\end{aligned} \tag{3.15}$$

Finally, an animated smooth curve represents when several quaternion qn is used and linked to all the procedures as Eq. 3.16.

$$\mathrm{LERP}\left(q_n, q_{n+1}; t\right) = (1-t)q_n + tq_{n+1} \tag{3.16}$$

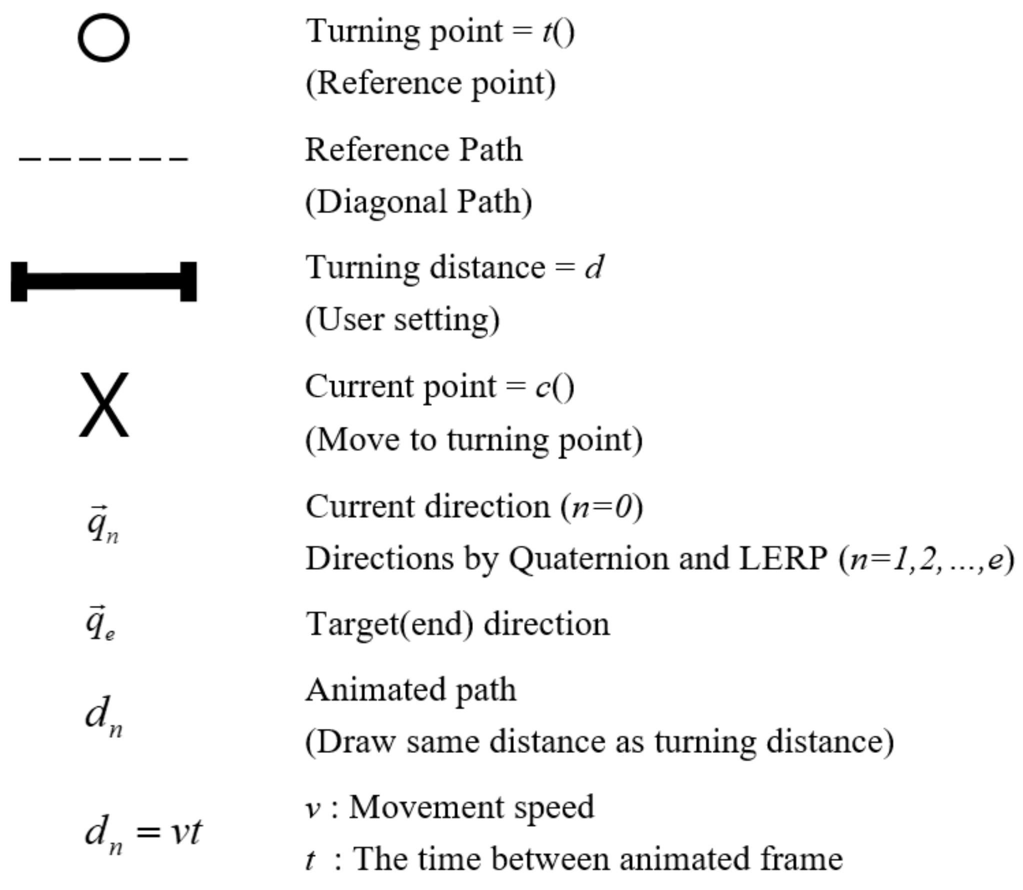
Follwing scheme is the modified Shoemake's scheme. After applying the modified Shoemake's scheme, the referenced path, which is the diagonal path having zig-zag issues, are animated a new smoothing path in real-time for the first responders and rescue robots.

Modified Shoemake's Scheme:

- 1. To work exclusively in $\mathbb{H}$, it needs to convert the original data into a quaternion q_n.
- 2. To design a unit quaternion r_n, convert a rotation matrix into r_n.
- 3. To perform the rotation(conversion), calculate $q_{n+1} = r_n q_n r_n^{-1}$.
- 4. Repeat for any other rotation r^n_{n+1}, r^n_{n+2} , etc.
- 5. If the procedure is stopped after the first rotation, calculate the in-between via $\mathrm{LERP}\left(q_n, q_{n+1}; t\right) = (1-t)q_n + tq_{n+1}$ with $t \in [0,1]$, yielding the numbers $q_n, q_{n+t_1}, q_{n+t_2}, \cdots, q_{n+1}$ along a smooth curve.
- 6. Afterward, link a set of quaternion numbers along the path while rotating.

3.2.2.2 Bounded Curvature Procedure

To show the animated path, a 2D map is converted to 3D and reconstructed on a 3D map using a simulation tool called Unity-3D. Therefore, the path is shown as 2D on the surface of the 3D map. The modified Shoemake's scheme is converted to a program function, which has procedures as the Modified Shoemake's scheme. The scheme is applied to 4 steps called as "Bounded curvature procedure" in this study. Notations of the bounded curvature are shown in 3.9 and procedures are shown in 3.10.



○	Turning point = $t()$ (Reference point)
– – – – – –	Reference Path (Diagonal Path)
├──┤	Turning distance = d (User setting)
X	Current point = $c()$ (Move to turning point)
$\vec{q}_n$	Current direction ($n=0$) Directions by Quaternion and LERP ($n=1,2,\ldots,e$)
$\vec{q}_e$	Target(end) direction
d_n	Animated path (Draw same distance as turning distance)
$d_n = vt$	v : Movement speed t : The time between animated frame

Fig. 3.9 Notations of Bounded Curvature Procedure

Step 1 of 3.10 shows how to load the reference path. The start point and the endpoint are the same as the calculated diagonal path. Step 2 shows how to set a turning distance and movement speed. Step 3 shows how to calculate an animated path using the modified Shoemake's scheme. Finally, the animation curve will be illustrated after all frame of the quaternion points using LERP is linked. For guidance, depending on turning distance and turning boundary, the animated path will be smoother. It needs a user-setting parameter about turning distance and movement speed in the simulation. Detail settings are described in Chapter 4.

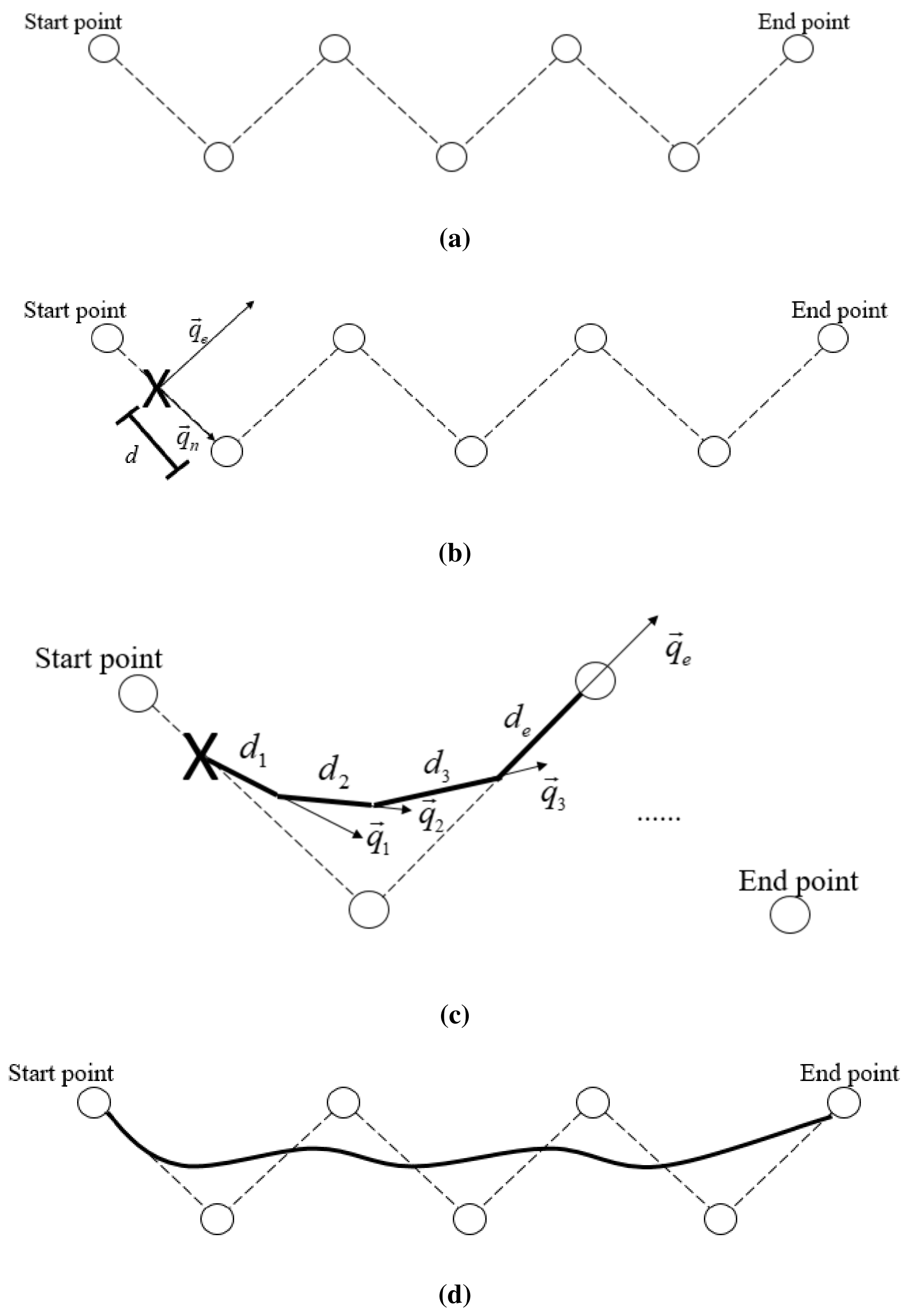


Fig. 3.10 4-steps of Bounded Curvature Procedure

3.11 is a top view of a test map and the path planning result in a known area. In all subsequent scenarios, the test map also used a 720×720[pixel], and the drone was recording 150[m] above. The distance of the x-y axis is 125[m]. However, the y-axis over 40 (40 between 50) was cut off in the figure. The calculation time of each path smoothing is represented in Table 3.1. The diagonal A* with Shoemake's scheme using LERP, SLERP, and Bézier curve are tested followed by the bounded curvature procedure as 3.10. There are very small differences in calculation time as 0.1 [ms] orders after the path smoothing is applied. It may seem like a small difference, but the size of the map is 101×101[grid], which is relatively small. However, the wider difference will be observed if the map is larger. Therefore, path smoothing should have as small influence as on the path planning algorithm.

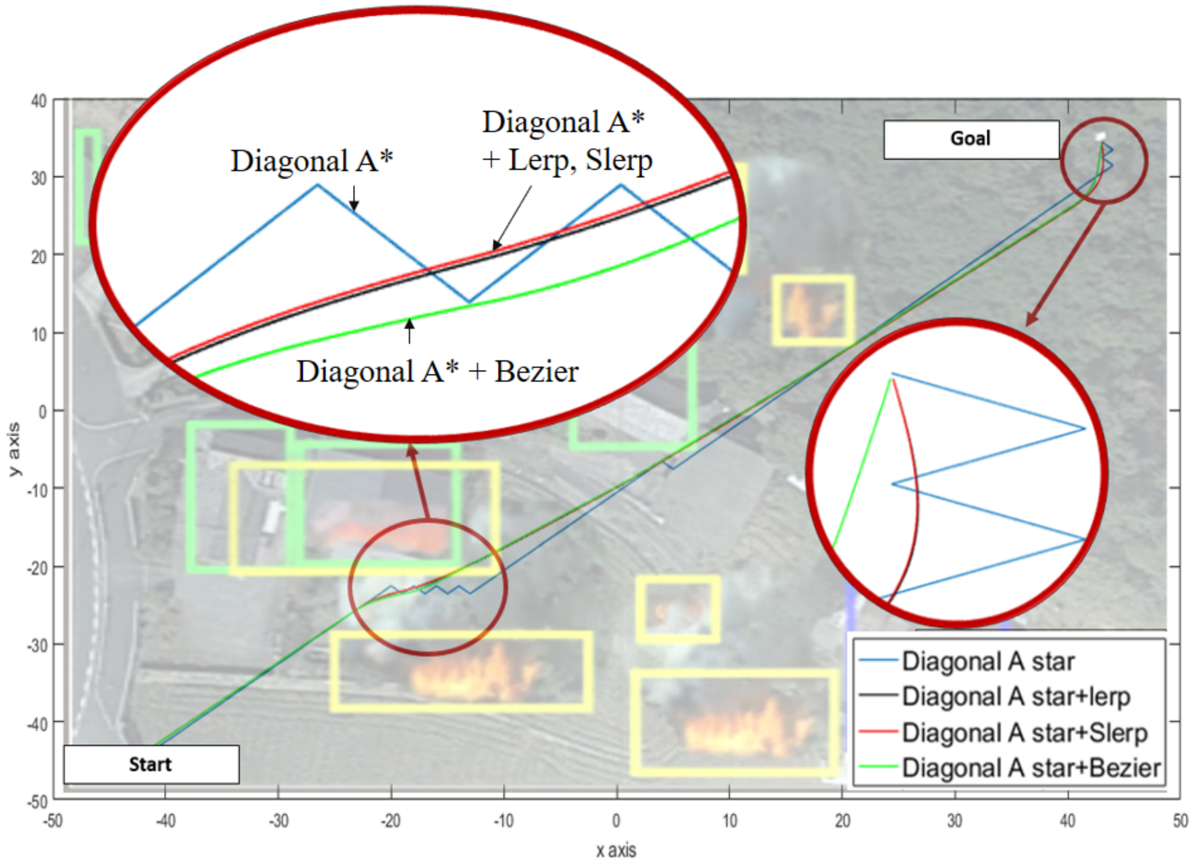


Fig. 3.11 Path Smoothing after Diagonal A* Path Planning

Table 3.1 Path Calculation Time of Different Curvature

Algorithms	Calculation time
A*	35[ms]
Diagonal A*	18 [ms]
Diagonal A*+LERP	18.1 [ms]
Diagonal A*+SLERP	18.1 [ms]
Diagonal A*+ Bézier curve	18.3 [ms]

3.2.2.3 Time Complexity Analysis

The calculation time of the node-based path planning algorithms mentioned above is based on the expansion of the nodes collected on the open and closed list by the non-descending order. Therefore, the data structure is more important than the algorithm structure [61]. Most of the searching algorithm's typical approach is to use a binary heap as the data structure. This research also uses the binary heap in the known and partially known path planning, both cases. The two algorithms have the same data structure using the binary heap, so a Big-O notation [62] for time complexity is the same. Based on the Big-O notation, the time complexity of A* is $\mathrm{O}(n\log(n))$, where n is a total number of nodes in the map in the best case.

However, the diagonal node expansion algorithm's best cases are represented by 0.5 node-expansion, and it is influenced by the total number of data amounts. The number of basic operations of the whole algorithm can be expressed as:

$$\begin{aligned} c(n) &= k_{\text{pre}} + k_{\text{main}} + k_{\text{post}} \\ k_{\text{main}} &= 0.5 \cdot \mathrm{O}(n\log(n)) \end{aligned} \tag{3.17}$$

where, n is the number of nodes in the map, k_{pre} is a constant number of basic operations before entering the main loop, k_{post} is a constant number of basic operations after the main while-loop. kmain is $0.5 \cdot \mathrm{O}(n\log(n))$. It can be written that the computational complexity of the diagonal node expansion algorithm is expressed as: $\mathrm{O}(n\log(n))$ type, and the dominant is 0.5. If the whole $c(n)$ maintains the characteristic of $\mathrm{O}(n\log(n))$, the k_{post} should be a constant value. This point is why this research is used the Shomeake's scheme for path smoothing in the quaternion space. The k_{post} is an additional procedure that can quickly solve zig-zag issues in the path that occurs after diagonal node expansion algorithm is applied. The focusing of the k_{post} is on reprocessing and stability of the path shape.

If the time complexity of each procedure is shown, k_{pre} has a time complexity of $\mathrm{O}(1)$, because it is a setting value. The k_{post} has $\mathrm{O}(n)$ time complexity because it includes the re-processing of the data of path smoothing. The weight also has a time complexity of $\mathrm{O}(n)$ in the same meaning.

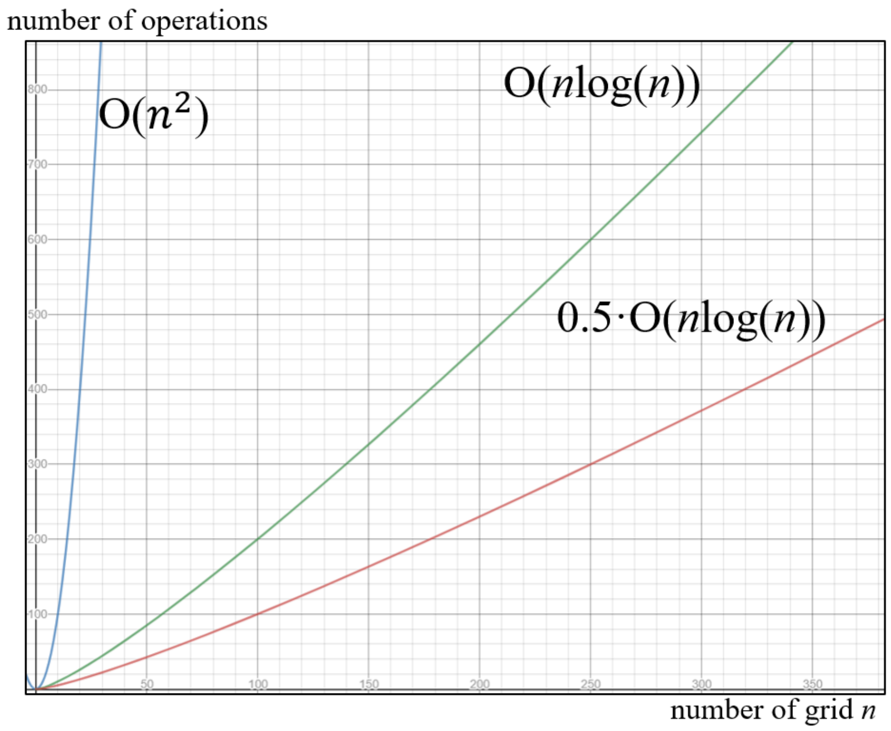


Fig. 3.12 Theoratical Time Complexity of A* and D* based Algorithms

3.2 shows the diagonal node expansion algorithm and path smoothing methods reduce the number of expanded nodes and calculation time. The expanded node of the proposed diagonal node expansion algorithm is reduced by 40% compared with the original path. However, the calculation time of the diagonal A* reduced almost 45% and the diagonal D* lite reduced 40%. There is a difference between the decreased rate of the number of expanded nodes and the calculation time. The reason for the difference will be described in the next simulation. The last factor is the path length, and the proposed algorithm with path smoothing which has longer paths are more safe and smooth.

Table 3.2 Number of Expanded Node and Calculation Time for Scenario

	A*	Diagonal A*	Diagonal A* +path smoothing +weight
Known envrionment	Number of Open nodes: 109 Number of Closed nodes: 1066 Number of Expanded nodes: 1175 Calculation time: 11 [ms] Path length: 172 m	Number of Open nodes: 39 Number of Closed nodes: 676 Number of Expanded nodes: 715 (40% less) Calculation time: 6 [ms] (45% less) Path length:179 m	Number of Open nodes: 39 Number of Closed nodes: 676 Number of Expanded nodes: 715 (40%less) Calculation time: 6.1 [ms] (45%less) Path length: 192 m
	D* lite	Diagonal D* lite	Diagonal D* lite +path smoothing +weight
Partially known environment	Total number of Open nodes: 312 Total number of Closed nodes: 7120 Total number of Expanded nodes: 7432 First calculation:10 [ms] Recalculation after fire 1: 27 [ms] Recalculation after fire 2: 36 [ms] Total: 73 [ms] Path length:294 m	Total number of Open nodes: 101 Total number of Closed nodes: 4424 Total number of Expanded nodes: 4525 (40%less) First calculation:7 [ms] Recalculation after fire 1: 14 [ms] Recalculation after fire 2: 22 [ms] Total: 43 [ms] (41%less) Path length:301 m	Total number of Open nodes: 101 Total number of Closed nodes: 4424 Total number of Expanded nodes: 4525 (40%less) First calculation:7.1 [ms] Recalculation after fire 1: 14.4 [ms] Recalculation after fire 2: 22.5 [ms] Total: 44 [ms] (40%less) Path length:321 m

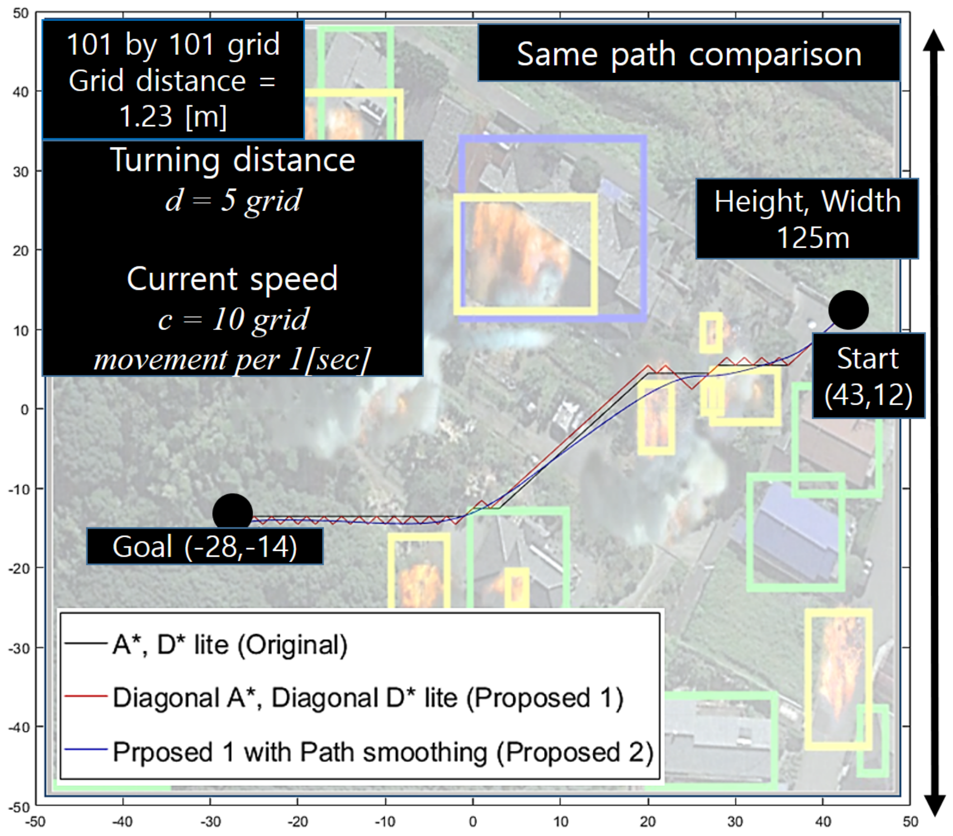


Fig. 3.13 Path Planning Simulation of Known Enviornment

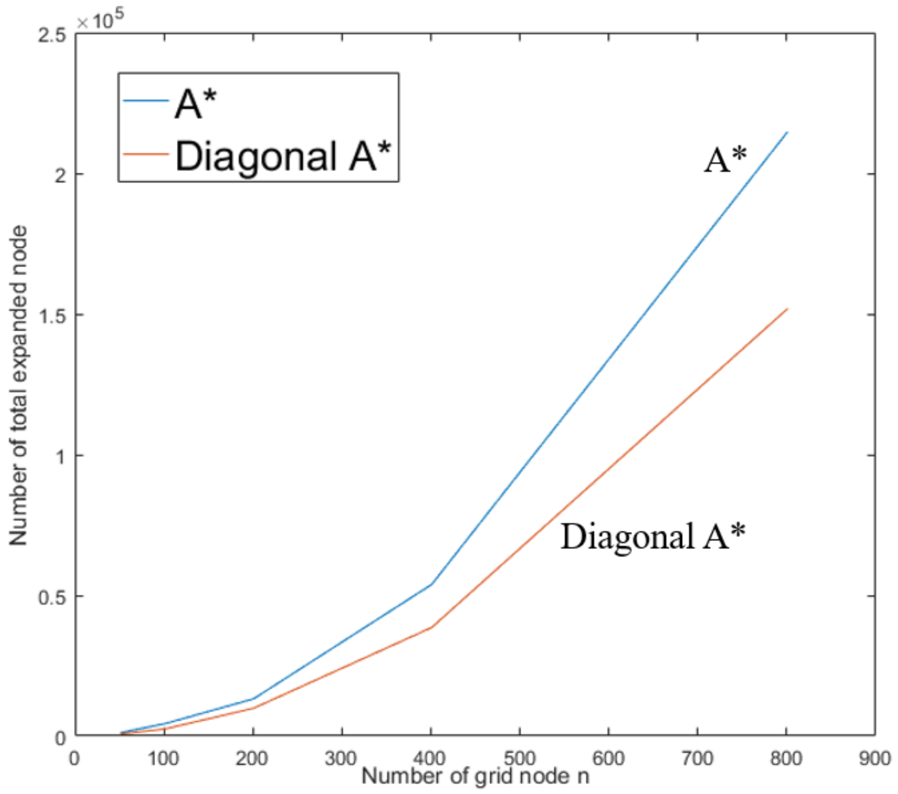


Fig. 3.14 Real Time Complexity of A* and Diagonal A* algorithms

To compare the precise performance for the expanded node and calculation time of the algorithms, the condition of the two different environments adjusts the same as possible. For the precise comparison, speed is set to a constant speed, and the turning boundary is set to a larger boundary for perfectly removing the zig-zag. The turning distance is 5[grid] and the current speed is 10[grid]. 3.13 shows the same path in the known and partially known environment because an updated path in the partially known area is not influenced by the planned path. The weighted procedure is excluded for an accurate comparison of algorithms. The number of expanded nodes and calculation times are shown in 3.3. The diagonal A* with path smoothing is reducing the number of expanded nodes and the calculation time 51% and 45% less compared with the original A*. Also, the total expanded node and total calculation time of the D* lite are reduced by 32% and 31%. The decreased ratio between the number of expanded nodes and the calculation times are not much because of less influencing factor. Thus, under the same conditions when the influencing factor is not presented, the diagonal node expansion algorithm shows a performance improvement of 50% in the known environment, but a performance improvement of 30% or more in the partially known environment.

Table 3.3 Expanded Node and Cacluation Time Comparision (Traditional vs Diagonal)

	Algorithms	Expanded node and calculation time
Known Eenvironment	A*	Number of Open nodes: 169 Number of Closed nodes: 1958 Number of Expanded nodes: 2127 Calculation time: 18 [ms]
	Diagonal A* + Smoothing	Number of Open nodes: 101 Number of Closed nodes: 986 Number of Expanded nodes: 1087 (51% less) Calculation time: 10 [ms] (45% less)
Partially known environment	D* lite	Number of Open nodes: 592 Number of Closed nodes: 9332 Number of Expanded nodes: 9924 First calculation: 28 [ms] Recalculation after fire 1: 36 [ms] Recalculation after fire 2: 39 [ms] Total: 103 [ms]
	Diagonal D* lite + Smoothing	Number of Open nodes: 408 Number of Closed nodes: 6439 Number of Expanded nodes: 6847 (32% less) First calculation: 22 [ms] Recalculation after fire 1: 25 [ms] Recalculation after fire 2: 25 [ms] Total: 72 [ms] (31% less)

Also, the proposed path smoothing using the modified Shoemake's scheme, which has bounded curvature procedure, is successfully applied to the diagonal node expansion algorithm and it does not influence the total calculation time of the diagonal node expansion algorithm. The smoothed path is better than the original path, and it solves the zig-zag issues.

3.4 shows the experiment results of time complexity depends on the map size. The scenario is the same as 3.2 and compared following the time complexity. The number of expanded nodes always has the same number regardless of the many trials of simulations. However, the performance of the calculation time is reduced by 45% to 50% in all circumstances. If the diagonal node expansion algorithm is applied, the expanded nodes have 40% reduction rate if the grid distance is small. But as the map is larger, it shows 30%. There is a reason that the two rates are not the same. The rate of calculation time reduction is possible to increase even if the rate of expanded nodes is decreased. The reason is path changes because of the small size of the node. The path becomes closer to obstacles due to the property of the algorithm that finds the smallest cost because of decreasing the size of a node. The decreased rated of the expanded node is begin to starting from 201×201[grid]. Therefore, even if the total number of nodes is reduced by 50%, the number of expanded nodes is reduced by 30%. It means the whole procedure including map searching is around 45% between 50% including error.

However, the time complexity performance comparison of algorithms uses node expansion rather than computational time because of the wide error boundary. In addition, the path smoothing and weighted procedure (k_{pre} and k_{post}) have not much calculation increment even if the map is larger. The rate of increment is theoretically $\mathrm{O}(n)$, so the k_{pre} and k_{post} are experimentally satisfied to have a fast time as possible with a small number of increment rates. 3.14 shows the real simulation of the time complexity. The A* shows $\mathrm{O}(nlog(n))$ and the diagonal A* shows $0.6 \cdot \mathrm{O}(n\log(n))$ until the number of grid n is 201, and after it shows $0.7 \cdot \mathrm{O}(n\log(n))$. An average dominant is 0.7 compared with the A*. As the map is larger, it is difficult to construct a completely same proportion of paths, obstacles, etc., so it can be confirmed 10% between 20% of the error compared with the best case of theoretical time complexity comparison.

Table 3.4 Expanded Node and Calculation Time Depends on Grid Size

Number of grid	A*	Diagonal A*	Diagonal A* + path smoothing + weight
51 by 51 (Perfectly same as Scenario Grid distance : 2.45 [m]	Number of Open nodes: 109 Number of Closed nodes: 1066 Number of Expanded nodes: 1175 Calculation time : 11 [ms]	Number of Open nodes: 39 Number of Closed nodes: 676 Number of Expanded nodes: 715 (40% less) Calculation time : 6 [ms] (45% less)	Number of Expanded nodes is the same as Diagonal A* Calculation time : 6 [ms]
101 by 101 (Same Scenario - different scale) Grid distance: 1.23 [m]	Number of Open nodes: 280 Number of Closed nodes: 4075 Number of Expanded nodes: 4355 Calculation time: 24 [ms]	Number of Open nodes: 99 Number of Closed nodes: 2396 Number of Expanded nodes: 2495 (43% less) Calculation time : 19 [ms] (44% less)	Number of Expanded nodes is the same as Diagonal A* Calculation time : 17 [ms]
201 by 201 (Same Scenario - different scale) Grid distance : 0.66 [m]	Number of Open nodes: 458 Number of Closed nodes: 12788 Number of Expanded nodes: 13246 Calculation time: 128 [ms]	Number of Open nodes: 215 Number of Closed nodes: 9679 Number of Expanded nodes: 9894 (25% less) Calculation time : 72 [ms] (44% less)	Number of Expanded nodes is the same as Diagonal A* Calculation time : 74 [ms]
401 by 401 (Same Scenario - different scale) Grid distance : 0.33 [m]	Number of Open nodes: 942 Number of Closed nodes: 52932 Number of Expanded nodes: 53874 Calculation time : 505 [ms]	Number of Open nodes: 426 Number of Closed nodes: 38149 Number of Expanded nodes: 38575 (29% less) Calculation time : 263 [ms] (48% less)	Number of Expanded nodes is the same as Diagonal A* Calculation time : 268 [ms]
801 by 801 (Same Scenario - different scale) Grid distance : 0.16 [m]	Number of Open nodes: 1921 Number of Closed nodes: 212852 Number of Expanded nodes: 214773 Calculation time : 2232 [ms]	Number of Open nodes: 859 Number of Closed nodes: 151053 Number of Expanded nodes: 151912 (29% less) Calculation time : 1216 [ms] (46% less)	Number of Expanded nodes is the same as Diagonal A* Calculation time : 1219 [ms]
1601 by 1601 (Same Scenario - different scale) Grid distance : 0.08 [m]	Number of Open nodes: 3980 Number of Closed nodes: 857042 Number of Expanded nodes: 861022 Calculation time : 9572 [ms]	Number of Open nodes: 1755 Number of Closed nodes: 600788 Number of Expanded nodes: 602543 (30% less) Calculation time : 4953 [ms] (49% less)	Number of Expanded nodes is the same as Diagonal A* Calculation time : 4968 [ms]

3.3 Simulations

3.3.1 Target Descriptiuon

A target is introduced to show the effectiveness of the proposed method. The target consists of three parts as UAV, server, and object as shown in 3.15. There are similar systems [63], [64], network structure, and applications [65], [66].

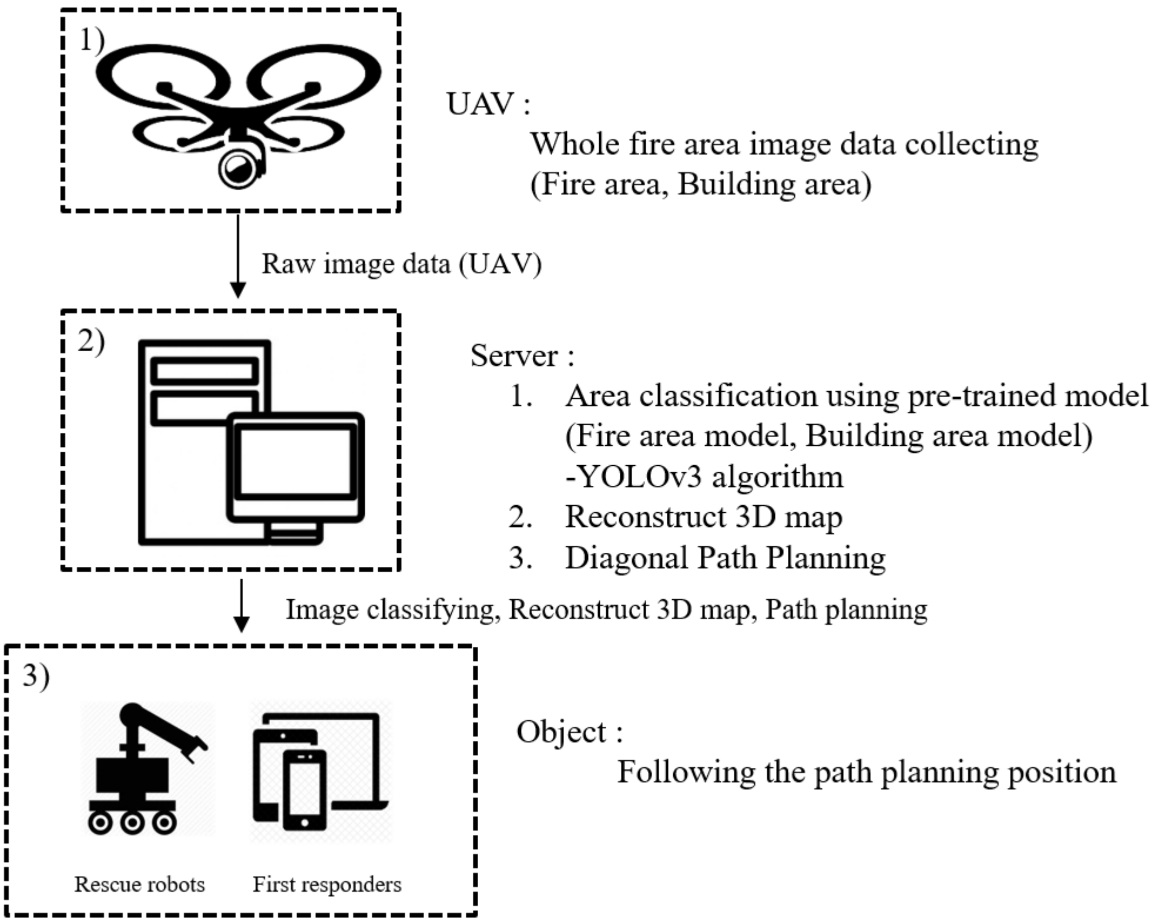

1)
UAV :
Whole fire area image data collecting
(Fire area, Building area)
Raw image data (UAV)
2)
Server :
1. Area classification using pre-trained model
(Fire area model, Building area model)
-YOLOv3 algorithm
2. Reconstruct 3D map
3. Diagonal Path Planning
Image classifying, Reconstruct 3D map, Path planning
3)
Object :
Following the path planning position
Rescue robots
First responders

Fig. 3.15 Simulation Target

The three-step of 3.15 is described below:

- Step 1: UAV records the images and sends them to a server.
- Step 2: Server executes the image processing using YOLOv3 algorithm, map reconstruction, and path planning.

- Step 3: Objects (first responders and rescue robots) are executed using the animated path from the server.

Step 1 and Step 3 are the main contents of data processing and communication between devices. The description of step 2 is as follows: all image data are given from a UAV after data collection, then the YOLOv3 algorithm finds the fire area and building area. The advantages of the YOLOv3 algorithm [67] are fast speed on real-time images, which has high frames per second (Fps), compared with any other training methods [68]. In a real situation, the images from the UAV send 30[Fps] images to a server, but in this study, simulation is carried out using referencing images for testing. However, making a pre-trained model should be preceded and implemented on the server before it finds the fire area and building area. After making the pre-trained model, the server is finding the fire area and building area using the YOLOv3 algorithm from current recording images. Then, a 3D map is reconstructed from a 2D map. Additionally, Unity-3D can be replaced with 3D Geographic Information System (GIS) data. The diagonal node expansion algorithm and path smoothing procedure are used after reconstructing the 3D map. Finally, the first responders and the rescue robots are following the planned path using a receiver. From this target, two scenarios are selected to show advantages and differences compared with the traditional algorithm and diagonal node expansion algorithm.

3.3.2 YOLOv3 Implementation

An overview of implemented the YOLOv3 including a pre-trained model is shown in 3.16.

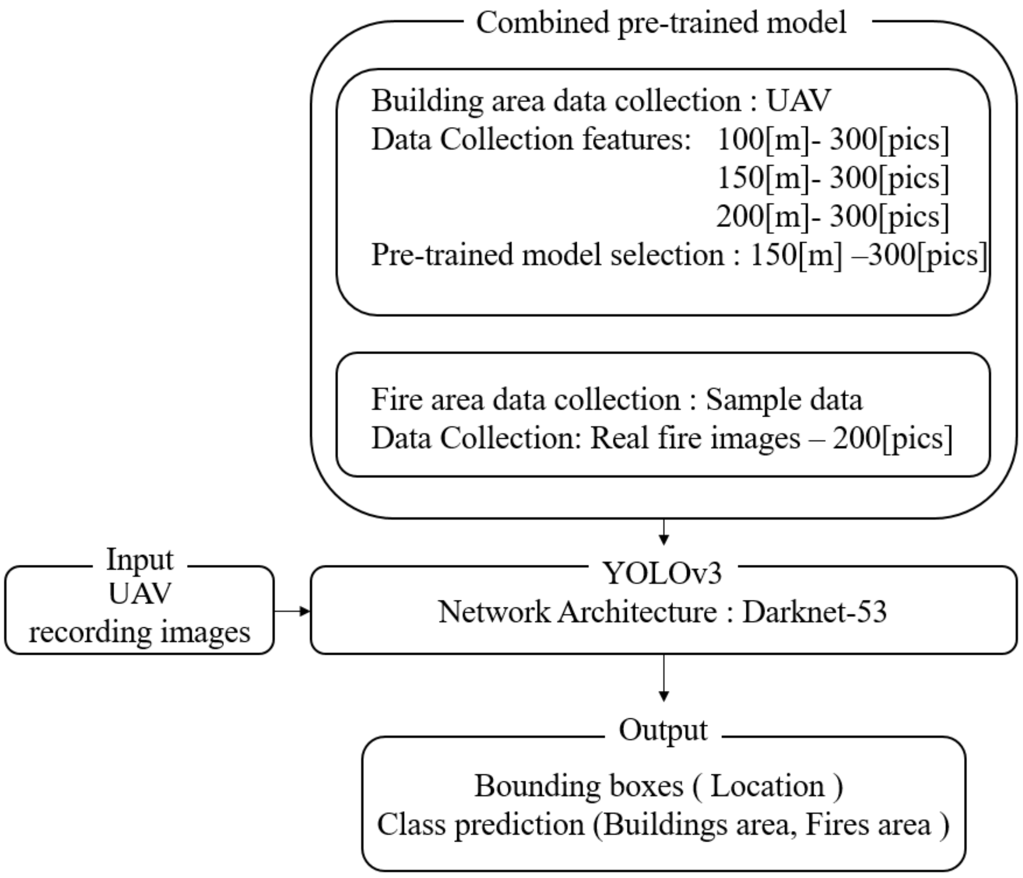

Combined pre-trained model
Building area data collection : UAV
Data Collection features: 100[m]- 300[pics]
150[m]- 300[pics]
200[m]- 300[pics]
Pre-trained model selection : 150[m] –300[pics]
Fire area data collection : Sample data
Data Collection: Real fire images – 200[pics]
Input
UAV
recording images
YOLOv3
Network Architecture : Darknet-53
Output
Bounding boxes (Location)
Class prediction (Buildings area, Fires area)

Fig. 3.16 Flow of YOLOv3 Application

To make a pre-trained model using the YOLOv3, UAV collects every 300 pictures depends on height in 100[m], 150[m], and 200[m]. before making each pre-trained model and these models are already implemented in the server. The 900 pictures have features as similar as possible since it does not have differences between the pre-trained model and input data. Similar to the building area, a pre-trained model of the fire area uses 200 real fire images. Fully 300 times training epochs take about 5[hours] for fire area detection and 12[hours] for building area detection by each height. Recording images from UAVs are given to input, then the YOLO network is running and finds bounding boxes and classes. After generating the pre-trained model, the specification of the fire area model result is shown in 3.5, and the specification of the building area model result is shown in 3.6. Especially, the building area detection at 150[m] is selected because the object recognition error rate is smaller than the others.

Table 3.5 Training Model Result of Fire Area

List	YOLOv3 Detector
Fire point detected	416
Error point detected	12
Object Recognition error rate	2.9%
Bounding boxes location accuracy	75.6%

Table 3.6 Training Model Result of Building Area

	YOLOv3 detection under UAVs height		
	100m	150m	200m
Building point detected	2014	2019	2621
Error point detected	301	136	313
Object recognition error rate	14.9%	6.8%	11.9%
Bounding boxes location accuracy	72.5%	74.2%	76.7%

Note)

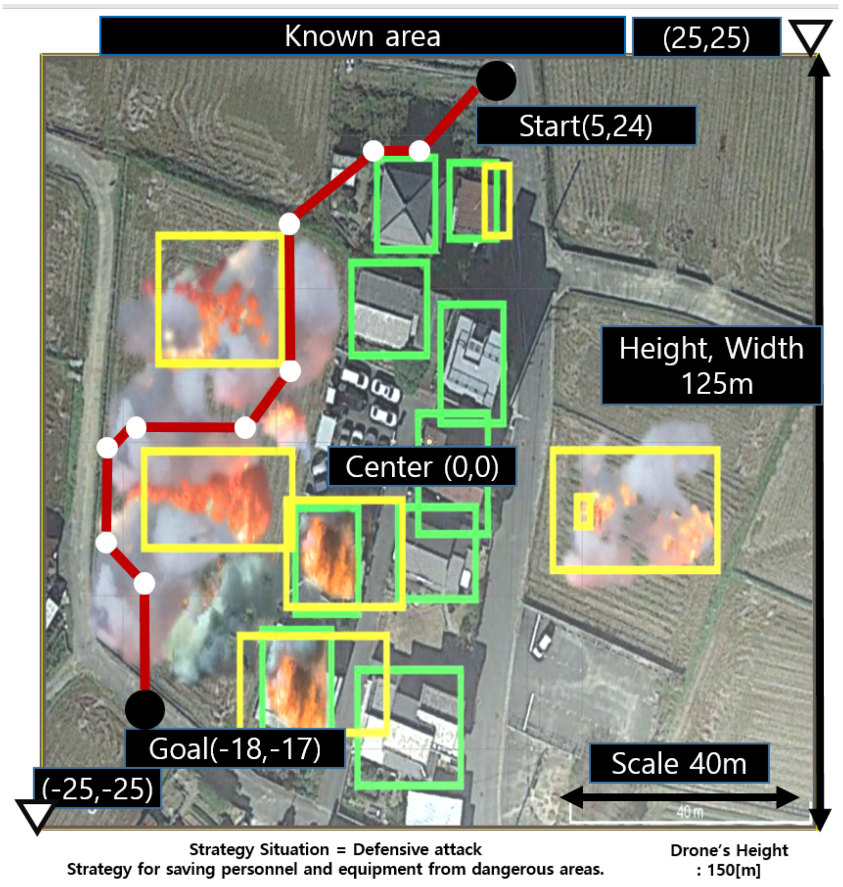
Object recognition error rate $= \frac{\text{Detected error point}}{\text{Detected object recognition point}} [\%]$

Bounding boxes location accuracy $= 100 - \frac{\text{Inaccurate bounding boxes generated}}{\text{Detected bounding boxes}} [\%]$

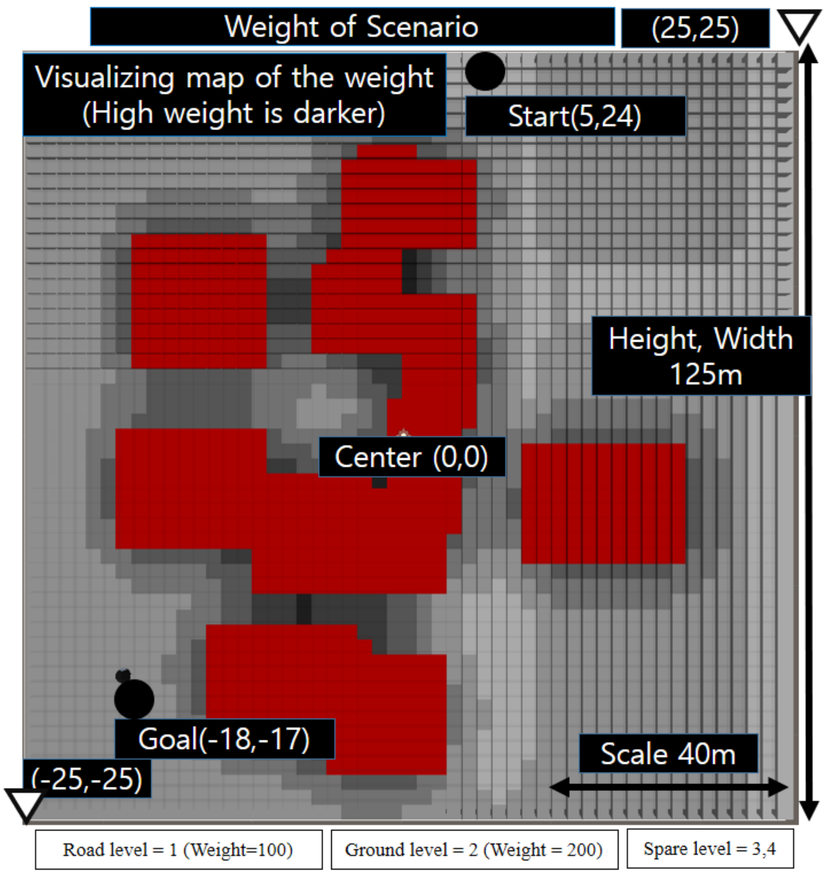
For object recognition, a specific hardware list is used as follows. CPU: i7-7700HQ, GPU: GTX 1060, GPU Memory: 6GB. YOLOv3 uses Tensor Flow in Python. Additional GUI uses PyCharm. GUI information is composed of building areas and fire areas. Especially, the building areas are separated depending on size by large, medium, and small.

3.3.3 Scenario for Different Path Depends on Known and Partially Known Environment

A scenario shows path differences between the known environment and a partially known environment. This scenario assumes that first responders and rescue robots are going to a goal position where people to be relieved, but fire is spreading to a wide area. The first responders and the rescue robots are starting from the wide-road, and they save the trapped person in the goal position using the road. The strategy to select the wider road for saving the person is a top priority. Therefore, the first responders make the rescue strategy using the information of map and known path to find the evacuated people. The rescue robots are assumed to assist the first responders. 3.17 shows the known environment and 3.18 shows the partially known environment. The map information in the known environment consists of a stationary start, stationary goal, stationary fire area, and stationary building area. The partially known environment consists of a stationary start, stationary goal, sudden fire area, and movable building area (on the fire situation). The yellow rectangles show the detected fire area, and the green colors show the detected building area from YOLOv3 object detection. Small (under 400[m^2]), medium (around 400-600[m^2]) and large size (around over 600[m^2]) of the building are represented by yellow, blue, and red rectangles, respectively.



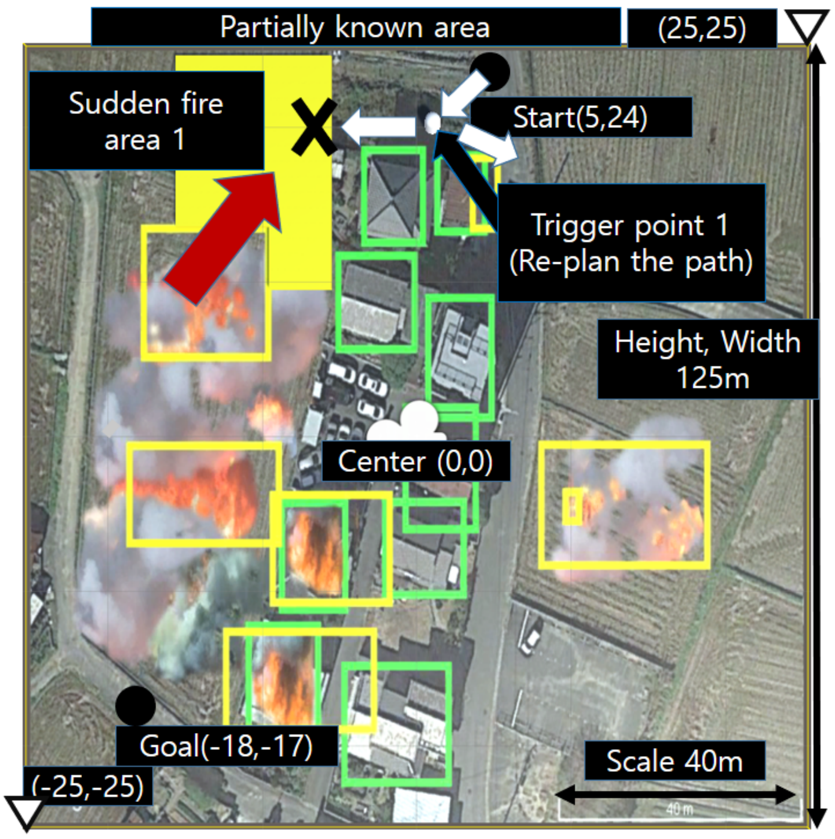
(a)



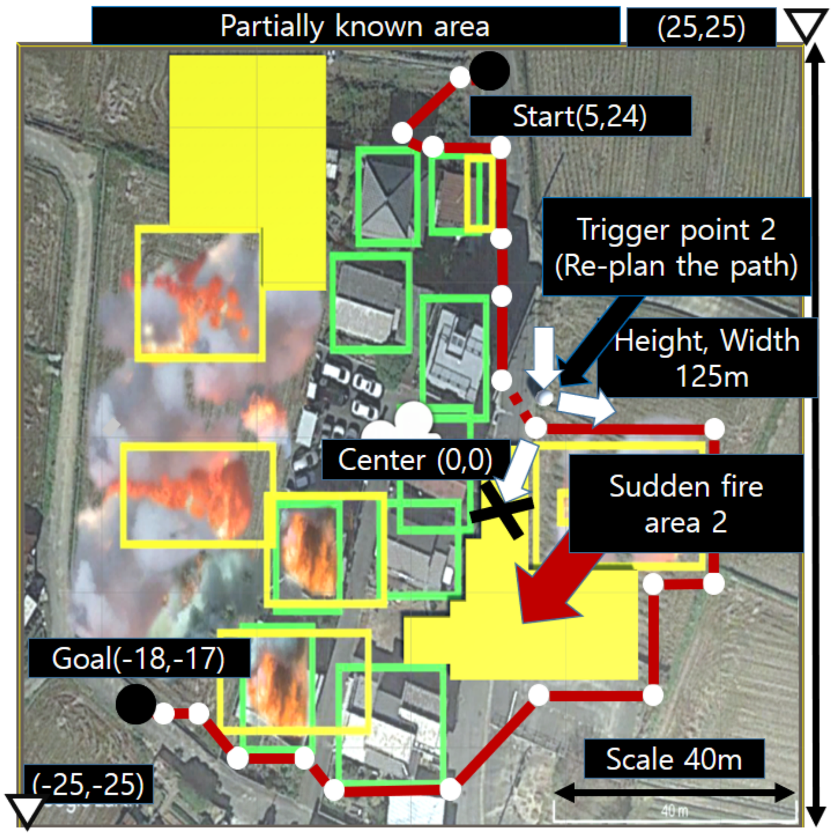
(b)

Fig. 3.17 Scenario to Known Area Path Planning

A sudden fire area appears after meeting every trigger in the partially known environment, but the planned path is blocked when the sudden fire areas are appearing. It means the moving of rescue robot is considered as possibly react radical area changing such as the sudden fire area, flame, and explosion cases. Important parameters of the simulation and parameters of the bounded curvature procedure are as follows: 720$\times$720[pixel] images are divided into 51$\times$51[grid]. 1[grid] distance is 2.45[m] in a real situation where the drone is around 150[m]'s height. Turning distance d is 2[grid] boundary. Therefore, 2[grid] is 4.9[m] for the turning distance. The 2[grid] boundary is enough for turning that close to human's turning boundary when they are running. It is the same boundary and speed for the normal moving of the rescue robot. The current speed c is the 1-2[grid] random speed movement per 1[sec]. The 1-2[grid] random speed movement per 1[sec] is around 9-18[km] per hour. And the speed is close to the average speed of human's running speed. This setting value makes the path more realistic the same as the movement of the first responders and rescue robots. Also, the turning distance influenced to path smoothing then looks more smoothly visualize if turning distance is large.



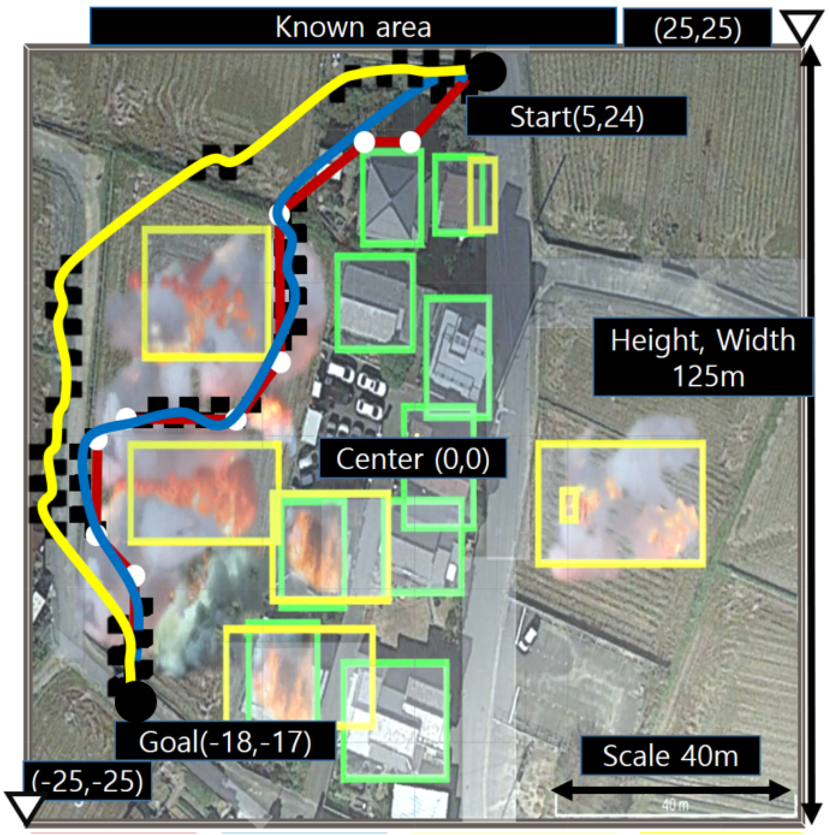
(a)



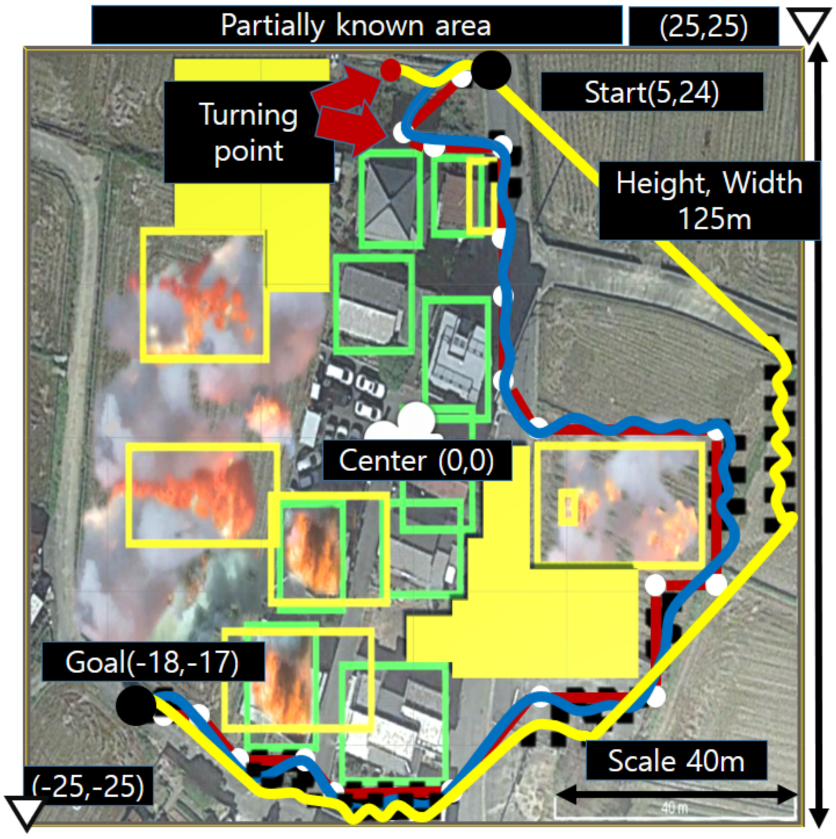
(b)

Fig. 3.18 Scenario to Partially Known Area Path Planning

As shown in 3.17(environment of known area), the first responder is following the planned path such as 3.19-(a). Similar in 3.18(environment of partially known area), the rescue robot has also followed the path such as 3.19-(b), but the path in the partially known area is updated when the rescue robot meets every trigger. The start position of both cases has the same start position and goal position. The center of map is (0,0), start position is (5,24) and goal position is (-18, -17).



(a)



(b)

Fig. 3.19 Simulation Result. (a) Known area , (b) Partially known area

3.19 represents a simulation result of two different environments. The (a) is the known area and following the path without any change of environment from the start position to the goal position. The A* shows the shortest distance [(a) - 1) red], so the total path length is shorter and arrival time is fast. But it is likely to pass through dangerous areas. The diagonal A* with path smoothing [(a) – 2) blue] has a significantly smooth path and the smaller calculation time. Also, the zig-zag issue can be seen as being little solved by path smoothing. The path of the diagonal A* with path smoothing and weighted procedure [(a) -3) yellow] have a longer path than the diagonal A*, but it ensures fast calculation speed and high safety. Same as (a), (b) is the partially known area and following the path with two-time environment changes. The partially known paths are confirmed the planned path changes.

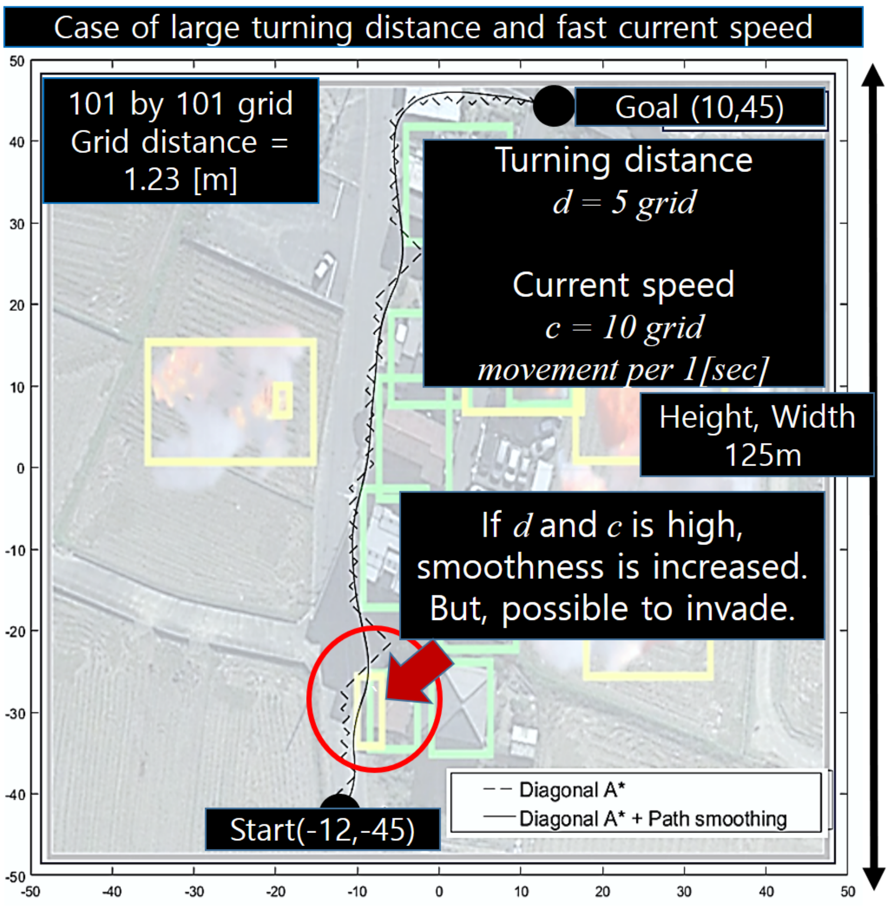


Fig. 3.20 Simulation for Higher Speed Parameters

Especially, the turning point(red) of [(b) -3) yellow] is a different position because of the high weight. Finally, 3.20 shows the first responder and rescue robot maintain a higher speed and large boundary of turning ability (turning distance). The above situation is assumed at a higher speed to compare with section 4.3. In this case, the simulation solved the zig-zag issues more perfectly compared with the path, which has a random speed and small turning distance. In real situations, the sensor value will be the input parameter of turning distance and current speed.

3.4 Conclusion

In this study, the diagonal node expansion algorithm was introduced in a known environment searching and partially known environment searching. The A* based path planning was applied in a known environment and the D* lite based path planning was applied in a partially known environment to path plan safely for the first responders and gave the path information to the rescue robot. The diagonal node expansion algorithm reduced the expanded node and calculation time at least 30% depending on information quantity in the grid node. Especially, 40% is reduced the number of expanded node in static environment by using the diagonal node expansion algorithm for target scenario(Real map size: 125[m]$\times$125[m], Grid: 51[grid]$\times$51[grid]). If the size of the map or information is large, the calculation time will be reduced even more compared with an original node-based path. This method will help to find the shortest and safest path for the first responder and rescue robot. After the path planning, the modified Shoemake's scheme and bounded curvature procedure were applied for path smoothing on the 3D environment. This proposed method was removed zig-zag issue after the diagonal node expansion algorithm applied on the 3D environment with less expanded node and fast calculation, especially in a partially known area. If new information is updated on the map, path planning and path smoothing should be re-planed. The proposed path smoothing method did not influence too much of calculation time. Also, the weighted procedure was possible to apply for safety. These additional procedures which are path smoothing and the weighted procedure had small calculation time. If the map is larger, the smaller calculation time of the additional procedure had advantages of total calculation time. Finally, the number of expanded nodes and calculation time of the proposed diagonal node expansiona algorithm and path smoothing were reduced compared with the time of the original node-based path planning, and then the path was changed to the safe path for the first responder.

Chapter 4

Auto-Splitting D* lite Global Path Planning for Dynamic Partially Known Area

4.1 Research Aims

In this study, the research target is to design a global path planning method for rescue robots in a large disaster area. The environments are generally dynamic and partially known areas, which is why global path planning is adopted. The global path planning method is capable of giving more information in the dynamic and partially known area than static and known area. Furthermore, this information increases the survival rate of those people in danger. It means that the path planning method requires a shorter path and a faster execution time. The shorter path can decrease the arrival time to the destination. Also, the faster execution time for updating the path improves the performance of the algorithm when the environment is dynamic.

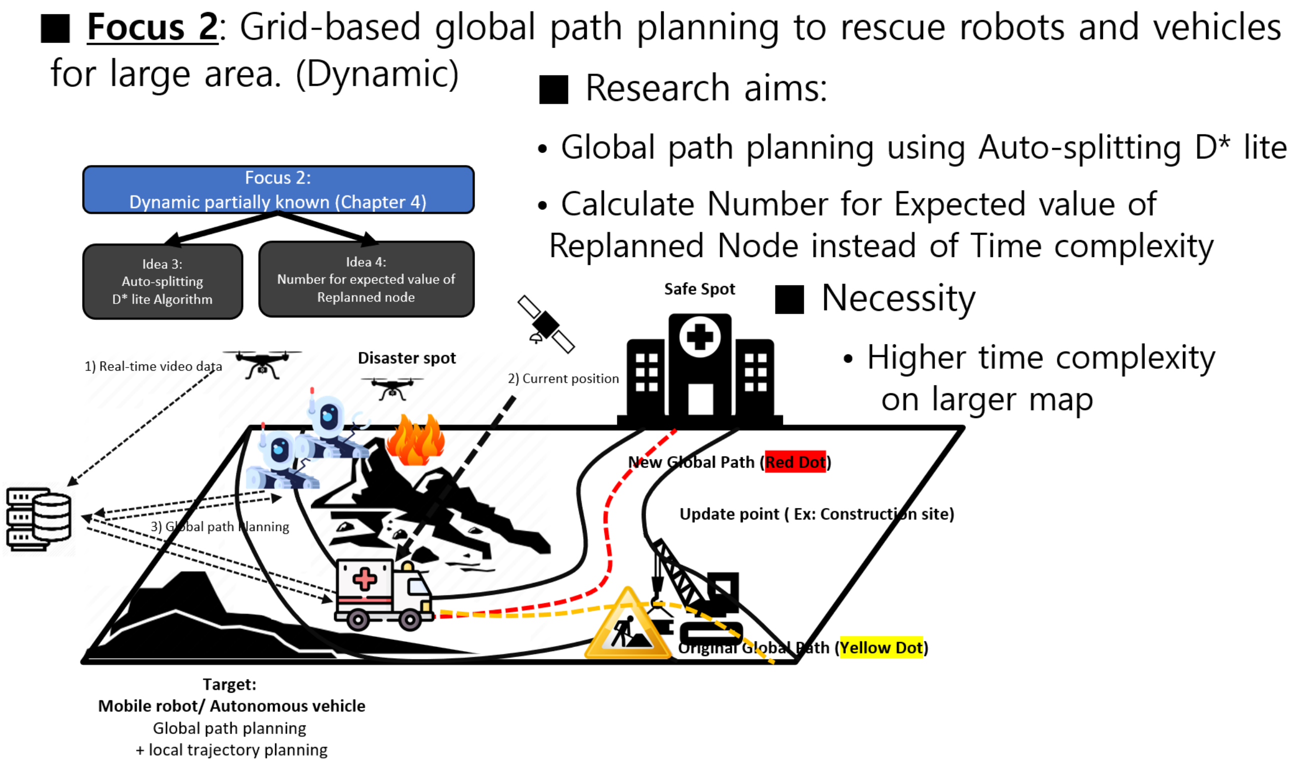

■ Focus 2: Grid-based global path planning to rescue robots and vehicles for large area. (Dynamic)
■ Research aims:
• Global path planning using Auto-splitting D* lite
• Calculate Number for Expected value of Replanned Node instead of Time complexity
■ Necessity
• Higher time complexity on larger map
Focus 2: Dynamic partially known (Chapter 4)
Idea 3: Auto-splitting D* lite Algorithm
Idea 4: Number for expected value of Replanned node
1) Real-time video data
Disaster spot
2) Current position
Safe Spot
3) Global path planning
New Global Path (Red Dot)
Update point (Ex: Construction site)
Original Global Path (Yellow Dot)
Target: Mobile robot/ Autonomous vehicle
Global path planning
+ local trajectory planning

Fig. 4.1 Research Aims (Chapter 4)

Fig. 4.2 shows the flow of the research. Most of the algorithms are processed on the server. The server sends the information of the planned path based on the robot's current location along with the obstacle information provided by the UAVs to the rescue robot. Then, the robot follows the global planned path with an implemented automatic control system. this study consists of five parts to show the effectiveness of a proposed global path planning method.

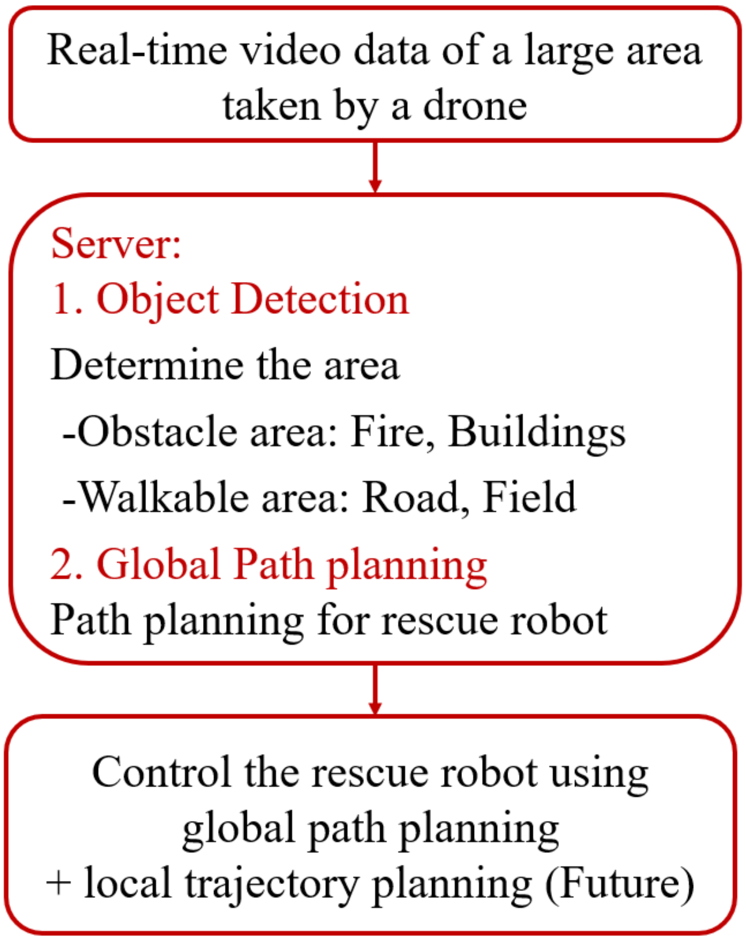

Real-time video data of a large area taken by a drone
Server:
1. Object Detection
Determine the area
-Obstacle area: Fire, Buildings
-Walkable area: Road, Field
2. Global Path planning
Path planning for rescue robot
Control the rescue robot using global path planning + local trajectory planning (Future)

Fig. 4.2 Flow of Research Overview (Robots are composed in the order of data flow in Fig. 4.1, and this research focuses on 2. Global path planning method. A typical example of 1. Object detection is the use of a You Only Look Once (YOLO) algorithm.

Previous research on the global path planning methods in the dynamic and partially known area for large areas is described below. The common target of the algorithms is to achieve a shorter path and a faster execution time. The dynamic global path planning methods are divided into several representative categories based on the characteristics of the algorithm. Grid-based algorithms, typically D* and D*lite algorithms, have an advantage in dynamic environments due to their fast performance. Nevertheless, their performance decreases significantly as a map grows in size [10,11,49,69]. In order to maintain fast computational performance, methods to reduce expanded nodes have been proposed in several ways. A typical method is a weighted method [70, 71]. However, the weighted methods can sometimes perform worse than the traditional algorithms [72]. In many situations, there is a general trend where a higher weight in the weighted cost method leads to a faster search. Nevertheless, there are also circumstances where a higher weight leads to a slower search. The weighted search is fast if there is a strong correlation between the estimated cost of getting to a goal node from current node and the number of nodes between current node and the nearest goal measured in edge count. In general, the weighted cost reduces the number of expanded nodes, but in situations where the costs are too highly weighted, the number of expanded nodes cannot be reduced due to the reason that the map becomes similar to a free-cost map. Hence, the performance will be the same or even worse than the traditional algorithm. Therefore, it is necessary that the grid-based algorithms require to reduce expanded nodes in other ways. Sampling-based algorithms, typically RRT and RRT* algorithms, are less capable of performing tasks in dynamic environments, but have better performance as the map grows in size [73–76]. The traditional approaches such as the potential fields and their related algorithms are not suitable for the dynamic and partially known environment, and the execution time is not fast enough [77–81]. Discrete optimization algorithms such as particle swarm optimization are used in both global and local path planning methods. Nonetheless, most algorithms can only deal with static obstacles. Their performance for moving obstacles is not sufficient enough to perform dynamic path planning in the large area [82–84]. Now, the current trend is to use sampling-based algorithms along with discrete optimization algorithms, and the machine learning method with the grid-based algorithm. [85–87] However, as the calculation performance of various chipsets has increased rapidly due to the development of the semiconductor industry, the research aimed at fast global path planning in large areas is actively underway. Consequently, the grid-based global path planning methods are being considered again to obtain high speed in the dynamic and partially known area. The grid-based global path planning methods work well for the dynamic and partially known areas, yet large dimension area processing is always the weak point. Especially, unnecessary dynamic areas exist in D* based algorithms [88–91].

To reduce the unnecessary areas, which means unnecessarily expanded nodes, of the D* based algorithm, this research proposes a way to pre-split a map using an Auto-Splitting D* lite algorithm denoted as AS-D* lite. The AS-D* lite algorithm uses an automatic clustering algorithm that calculates the optimal map segmentation automatically using the information of the obstacle's location. On the other hand, how to split an area automatically is just as important as removing unnecessary nodes for path planning. For splitting the map automatically, an automatic-clustering method is commonly used. The automatic-clustering methods are divided into different types such as centroid-based, connectivity-based, and density-based. The centroid-based automatic clustering is appropriate for determining the number of clusters for unlabeled data [92, 93]. The connectivity-based automatic clustering is suitable for hierarchical clustering [94, 95]. The density-based automatic clustering applies autonomous machine learning techniques and is widely used for finding clusters of any arbitrary shape, not only spheres [96, 97]. However, the centroid-clustering method is considered for splitting the maps through the path planning conditions, which are large disaster areas including dynamic and partially known, to achieve a fully automatic splitting method with a shorter calculation time. Nevertheless, there are some issues in the previous centroid-based automatic clustering methods with the grid-based global path planning methods for the dynamic environments. The centroid-based automatic clustering methods usually require a long processing time. This is a common issue that occurs in these algorithms. A *k*-means clustering and a mean-shift clustering-based algorithms are typical centroid-based algorithms. The *k*-means clustering-based algorithm takes a long time to determine the *k*-value. This means that the grid-based path planning method takes a long time to achieve an optimal effect by classifying a map based on the obstacles in the map [98]. The mean-shift clustering-based algorithms also have similar issues such as the repeated calculations. The repeated calculation carried out to find a center point rather than the *k*-value [99, 100]. To shorten the processing time for determining the *k*-value comparing with the other methods, a gap statistics is applied to find an optimal *k*-value, which is a splitting number, for the *k*-means algorithms. Moreover, a Voronoi algorithm is combined with the *k*-means algorithm to draw a segmented map automatically with the optimal *k*-value.

To sum up, Section 4.1 describes the current issues of previous global path planning methods in the dynamic and partially known area. Also, auto-clustering methods are summarized to select a better method to apply with the proposed global path planning method, which are solutions to the current issues. Section 4.2 describes details of the proposed algorithm, and the performance is analyzed through a time complexity method in special cases. Section 4.3 shows the simulation of special cases, algorithm performances, large city map simulations and large rural map simulations. Section 4.4 describes the conclusions of this study.

4.2 Auto-Splitting D* lite

4.2.1 Auto-Splitting Method for Auto-Clustering and Drawing

As explained in the Section 4.1, it is necessary to pre-divide a map to eliminate unnecessary areas when planning the path with the D* based algorithm on the large map. Accordingly, the automatic clustering method is used as a way to separate the map in this study. However, the previous centroid-based automatic clustering methods require a long processing time [98, 101]. Especially, the *k*-means clustering-based algorithm takes a long time to determine the *k*-value optimally. Therefore, the gap statistics is used to obtain the optimal *k*-value, which is the splitting number, for the *k*-means algorithms to achieve a shorter processing time comparing with all the other methods. The Voronoi diagram is combined with the *k*-means algorithm to draw a splitting map automatically with the optimal *k*-value.

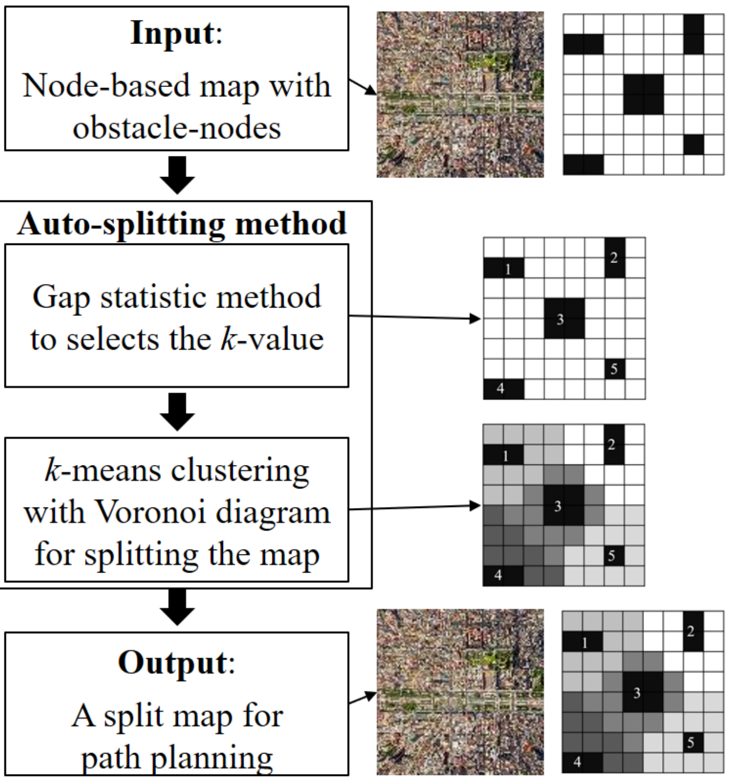

Input:
Node-based map with obstacle-nodes
Auto-splitting method
Gap statistic method to selects the k-value
k-means clustering with Voronoi diagram for splitting the map
Output:
A split map for path planning
1
2
3
4
5

Fig. 4.3 Overview of Auto-splting Method

Fig. 4.3 shows the overall flow for the auto-splitting method. The auto-splitting method proceeds in the following steps:

Input: node-based map with obstacle-nodes.

Step 1. Select the k-value by the gap statistic.

Step 2. Divide the map by the k-means clustering algorithm with the Voronoi diagram.

Output: a split map for path planning.

The information of the map retrieved by a drone is used as the input, and the grid map is generated according to the location of the obstacles. With the information collected, the optimal k-value can be found by the gap statistics. The details are described in Subsection 4.2.1.1. Then, the map is split by the line

4.2.1.1 Gap Statistics

The gap statistics method is applied for selecting the k-value before the k-means algorithm splits the map. There are other automatic k-means clustering methods such as combining a Silhouette coefficient [102] with the k-means algorithm. However, most of the previous methods require a long calculation time to determine the k-value [98, 101]. Furthermore, determining the k-value with learning algorithms for complete automation tends to make the proceeding time longer [103]. Centroid clustering methods such as mean-shift clustering-based methods also require longer calculation time because of the centroid finding issue, which means repeated calculations.

The selected method based on the gap statistics is discussed by Tibshirani et al. [104] for estimating the number of clusters in a set of data. In this study, the gap statistics is used to find an optimal k-value for the k-means algorithm with rapid calculation in the pre-processing step. It should be noted that the first step in the Auto-Splitting D* lite (AS-D* lite) algorithm is operated under a static environment. This method shows a faster processing time than the k-means with the Silhouette method, which has been frequently used.

List of Symbols in Gap statistics

n	Number of node (x-axis)
m	Number of node (y-axis)
N	Total number of obstacle node
(x,y)	Position of each node
G	Set of ground nodes
O	Set of obstacle nodes
R	Set of randomized obstacle nodes
$o_i(x,y)$	Element of obstacle nodes
(x',y')	Randomized position of each nodes
$r_i^d(x',y')$	Element of randomized obstacle nodes
M	Sample size determination
Z	Z score
EBM	Error bound for the mean
W_k	Within-cluster sum of squares using obstacle nodes
W_{kb}^*	Within-cluster sum of squares using randomized nodes
sd_k	Standard deviation
s_k	Simulation error

Let us consider an n[node]$\times m$[node] map having a total of N obstacle nodes. The ground node is represented as $G = \{(x,y)|x = 1,2,\ldots,n, y = 1,2,\ldots,m\}$. A set of obstacle nodes O is a subset, which belongs to the set of ground nodes G. The obstacle nodes O is denoted as $O = \{o_i(x,y)|i = 1,2,\ldots,N\}$. Also, let us consider a set of randomized obstacle nodes R having a total of B patterns, which is represented as $R = \{r_i^d(x',y')|i = 1,2,\ldots,N, d = 1,2,\ldots,B\}$. The gap statistics is used to determine an optimal k-value compared to each pattern. The position of random obstacle nodes are represented as (x',y') by randomizing function $rand()$. The random obstacle nodes are limited based on the obstacle nodes as given by

$$\begin{cases} \min_x(o_i(x,y)) < r_i^d(x',y') < \max_x(o_i(x,y)) \\ \min_y(o_i(x,y)) < r_i^d(x',y') < \max_y(o_i(x,y)) \end{cases} \tag{4.1}$$

The number of total patterns B is recommended to simplify the calculation process using a function $floor()$ with the sample size determination M as given by

$$M \triangleq \left(\frac{Z\sigma}{EBM/2}\right)^2 \tag{4.2}$$

$$floor(M) = max\{B \in Integer|B \leq M\} \tag{4.3}$$

Where Z is a standard z-score, and EBM is the error bound for a population mean. For the calculation example, if a randomized data set is assumed to follow $N(0,1)$ when the desired confidence interval is 90%, then the z-score is 1.645. When the desired margin of error is ±0.5%, then the EBM is 1. In this situation, the confidence interval is 90%±0.5%, then the value of $floor(M)$ is $floor(10.82)$=10 from equation (4.2). Other distributions, such as a uniform distribution, are possible to apply for the confidence interval. Depends on the distribution, the standard deviation changes the B value.

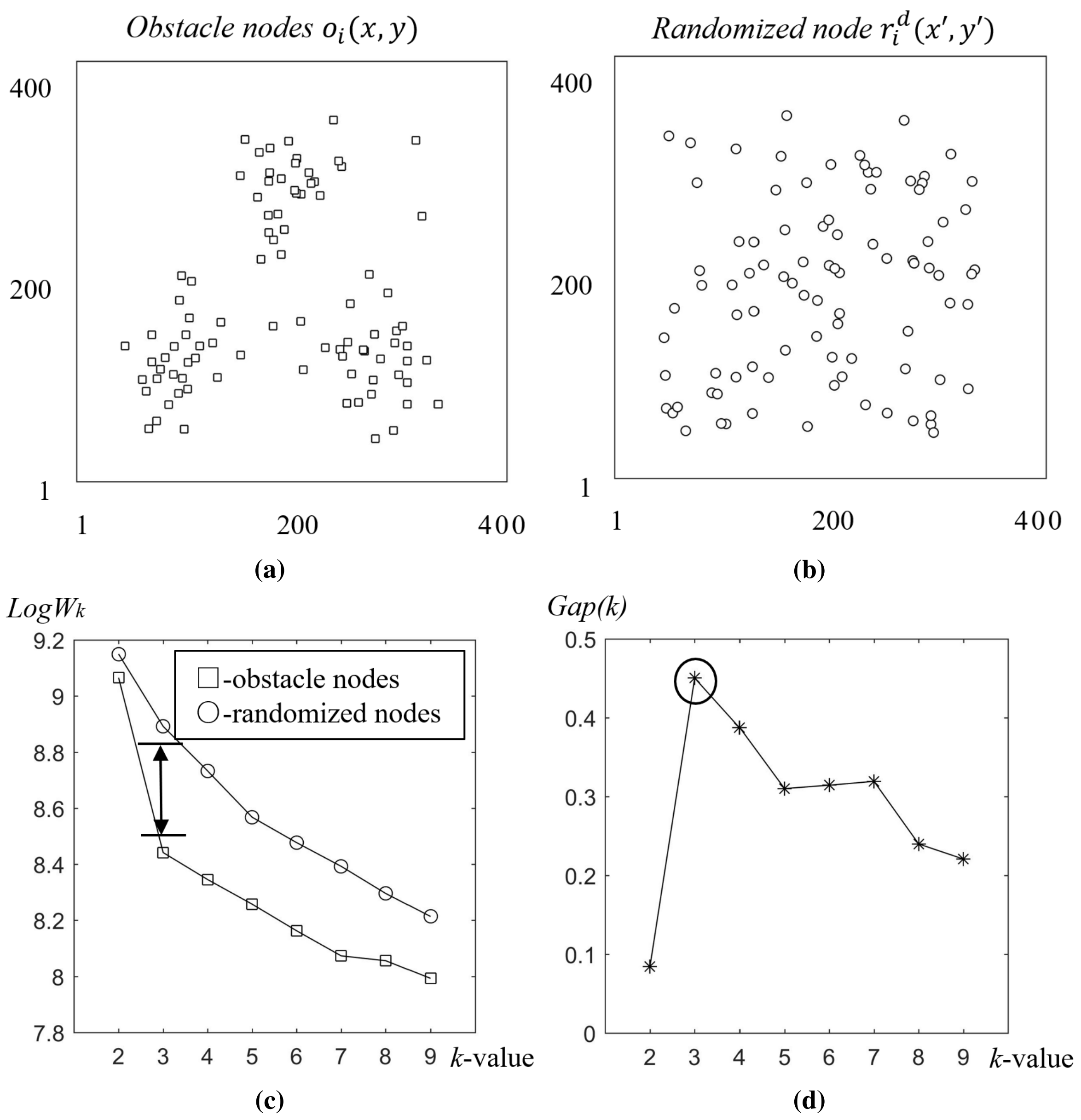


Fig. 4.4 Procedure of Gap Statistics. (a) Detecting the postion of obstacle nodes, (b) Generating the location of the random node based on the position of the obstacle nodes, (c) Calculating within-cluster sum of squres, (d) Calculating the $Gap(k)$ and using method to select k-value

The gap statistics $Gap(k)$ for k-value is defined as

$$Gap(k) \triangleq (\frac{1}{B}) \sum_{b=1}^{B} \log(W_{kb}^{*}) - \log(W_k) \tag{4.4}$$

Where W_k is calculated by the obstacle nodes, and W_{kb}^{*} is calculated by the randomized nodes in the map, respectively. The standard deviation sd_k is calculated by

$$sd_k = \left[\left(\frac{1}{B} \right) \sum_{b=1}^{B} \left\{ \log\left(W_{kb}^{*}\right) - \left(\frac{1}{B} \right) \sum_{b=1}^{B} \log\left(W_{kb}^{*}\right) \right\}^2 \right]^{\frac{1}{2}} \tag{4.5}$$

The simulation error s_k is calculated by the standard deviation sd_k and the value B such as

$$s_k = sd_k \sqrt{(1+1/B)} \tag{4.6}$$

The optimal k-value is the smallest k-value which satisfies the equation:

$$Gap(k) \geq Gap(k+1) - s_{k+1} \tag{4.7}$$

Fig. 4.4 shows the working principle of the gap statistics. From Fig. 4.4 (d), the best k-value is indicated as $k = 3$. The gap statistics is applied for selecting the best k-value before using the k-means clustering algorithm combined with the Voronoi diagram to split a large map.

4.2.1.2 Combined k-means Clustering with Voronoi Diagram

The k-means clustering algorithm is combined with a Voronoi diagram for splitting the large map, and determines the splitting line automatically. Also, the auto-splitting method needs to initialize the positions of the cluster centers and the initial k-value. The k-means algorithm is not faster than heuristic algorithms. However, the AS-D* lite determines an optimal k-value more conveniently than the heuristic algorithms [101, 105]. The AS-D* lite is briefly represented in three steps as follows:

Step 1. Input every cluster center from the k-means clustering.

Step 2. Draw the Voronoi diagram based on the cluster centers.

Step 3. Draw the splitting line after the Voronoi diagram is finished.

List of Symbols in *k*-means with Voronoi

CR	Set of cluster regions
G	Set of ground nodes
O^s	Sub set elements of obstacle nodes in each cluster center
N	Total number of obstacle node
N_s	Total number of elements belong to O^s
d_{js}	Distance between obstacle nodes and cluster centers
$o_j^s(x,y)$	Position of obstacle nodes
$c_s(x,y)$	Position of each cluster center
c_s	Set of cluster center
d_s	Distance between ground node and cluster centers
W_k	Within-cluster sum of squares using obstacle nodes

The k-means clustering is a method for separating data into k sets, and is applied for partitioning obstacles into k regions on a two-dimensional map. Then, the map is partitioned into k regions, which is the same as the k-value used in the k-means clustering algorithm. The map is composed of obstacle nodes and ground nodes. The position data of obstacle nodes needs to be collected from the map, and an initial k-value needs to set before applying the k-means clustering method. Then, the two-dimensional coordinate of each obstacle node in the map is stored. The initial k-value is set as $k_{initial} \geq 2$.

Also, the initial k-value represents the lower limit of an expected number of clusters. The Voronoi diagram is applied to classify each ground node in the map to one of the cluster regions. The large map is divided into k cluster regions, and the nodes which have the same distance to two different regions are defined as the splitting lines.

The cluster regions are represented as CR, and each cluster region is represented as $CR = \{CR_s | s = 1, 2, \ldots, k\}$, where $G = \bigcup_{s=1}^{k} CR_s$. Each cluster center is denoted as $c_s(x,y)$ in the clusrter region CR_s. The obstacle nodes in each cluster region are represented as $O^s = \{o_j^s | j = 1, 2, \ldots, N_s\}$, for example $\{o_1^s, o_2^s, \ldots, o_{N_s}^s\}$ in CR_s. Therefore, the O^s has the total of N_s obstacle nodes. The total number of obstacle nodes is equal to the sum of the number of nodes in each cluster region $N = \sum_{s=1}^{k} N_s$. To draw the Voronoi diagram, a distance d_{js} between obstacle nodes $o_j^s(x,y)$ and cluster centers $c_s(x,y)$ in the cluster region CR_s are defined as

$$d_{js} = |o_j^s(x,y) - c_s(x,y)|^2 \tag{4.8}$$

A value of the within-cluster sum of squares W_k for each cluster region CR_s is defined as

$$W_k = \sum_{s=1}^{k} \frac{1}{2N_s} \sum_{o_j^s, c_s \in CR_s} d_{js} \tag{4.9}$$

The initial position of each cluster center is set randomly. A final position of each cluster center is set by

$$\underset{c_s}{\operatorname{argmin}} W_k \tag{4.10}$$

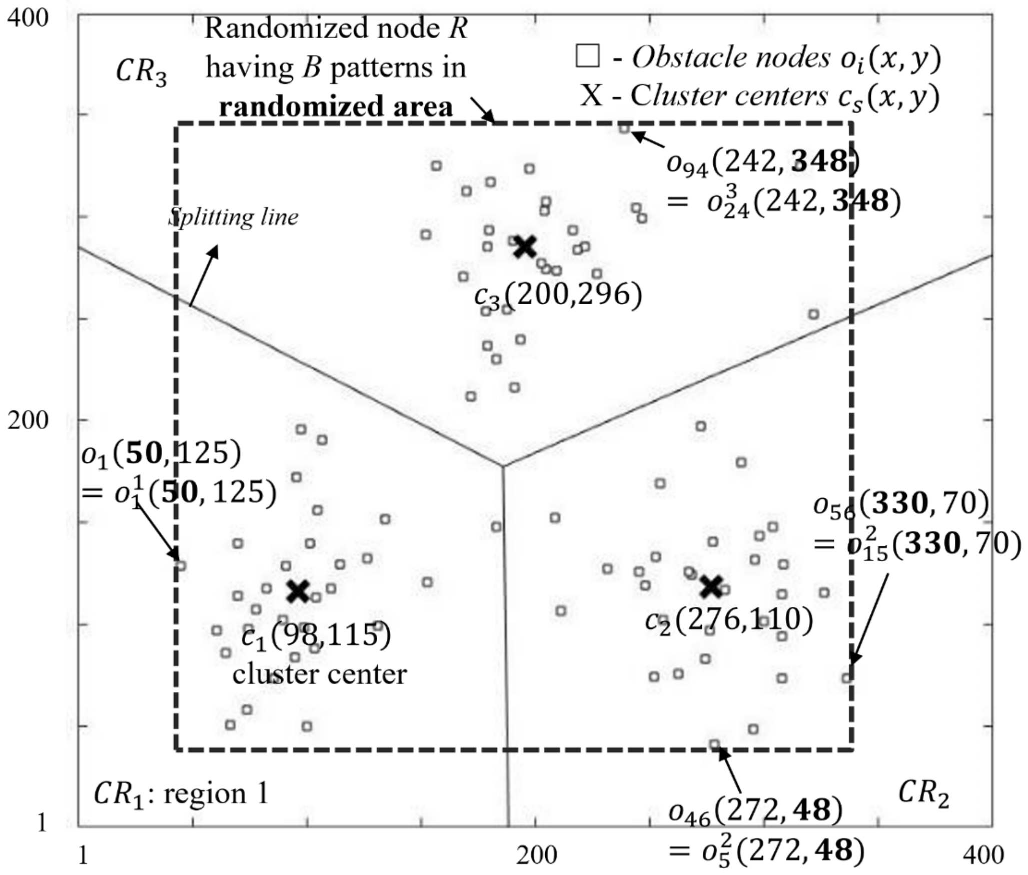


Fig. 4.5 Splitting Map by k-means with Voronoi Diagram. Step 1: Make the randomized area to find the k-means clustering ; Step 2: Find the best k-value ; Step 3: Input c_1, c_2, c_3 from the k-means clustering ; Step 4: Divide the cluster region CR using the distance between $o_j^s(x,y)$ and $c_s(x,y)$; Step 5: Calculate and draw the Voronoi diagram using equation (4.12)

According to the position of cluster centers, the map can be divided into a few regions using the gap statistics method. To divide the splitting line of each separate region, the Voronoi diagram is applied. The Voronoi diagram is a data structure investigated extensively in the domain of computational geometry. Originally, the Voronoi diagram characterizes the regions of proximity for a set of sites in a plane where the distance is defined by the Euclidean distance [101]. In this study, the Voronoi diagram is applied for grouping every point into the nearest cluster region.

For the Voronoi diagram, the ground nodes and the cluster center of the map selected by the k-means algorithm are required. The distance between a ground node (x,y) and the cluster center $c_s(x,y)$ in the cluster regions CR_s are defined as

$$d_s = |(x,y) - c_s(x,y)|^2 \tag{4.11}$$

Then, the regions are divided by the splitting lines based on the condition given as

$$d_s = d_{s'},\ c_s \neq c_{s'} \tag{4.12}$$

After applying the k-means clustering algorithm with the Voronoi diagram, the large map can be split into k regions with splitting lines shown on the map. An example of the k-means clustering combined with the Voronoi diagram splitting a map is shown in Fig. 4.5.

4.2.2 Auto-Splitting D* lite

The issues of the D* based algorithms containing unnecessary areas are mentioned in the previous sections. To reduce the unnecessary areas of the D* based algorithms, this study proposes a way to pre-split the map using the AS-D* lite. The AS-D*lite adopts an automatic clustering algorithm that automatically calculates the optimal map segmentation based on the location information of obstacles.

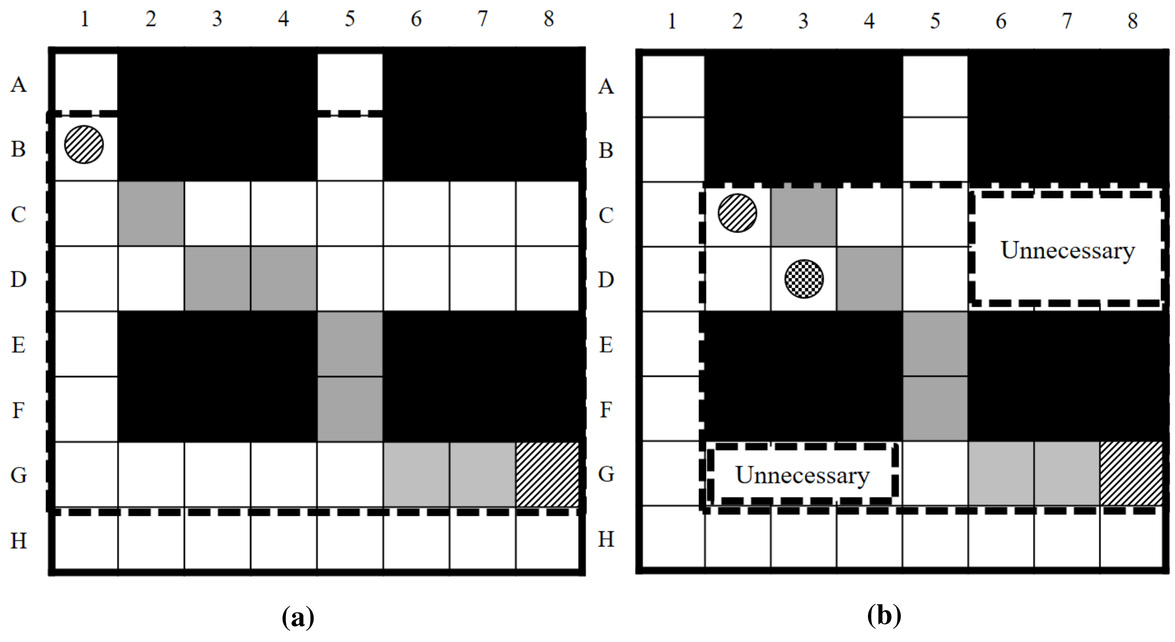


Fig. 4.6 Unnecessary Dynamic Areas. (a) Expanded area in traditional D* lite, (b) Unnecessary area while dynamic path planning runs

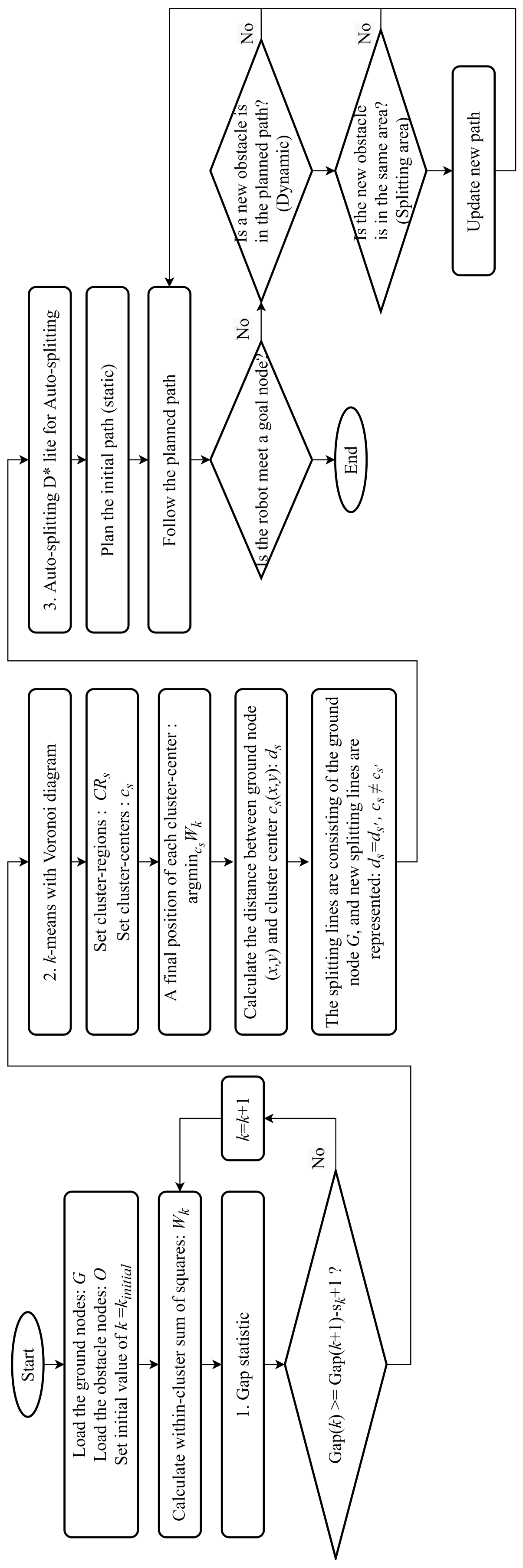

Start
Load the ground nodes: G
Load the obstacle nodes: O
Set initial value of $k = k_{initial}$
Calculate within-cluster sum of squares: W_k
1. Gap statistic
Gap(k) >= Gap(k+1)-s$_k$+1 ?
No
k=k+1
2. k-means with Voronoi diagram
Set cluster-regions : CR_s
Set cluster-centers : c_s
A final position of each cluster-center : argmin$_{c_s}W_k$
Calculate the distance between ground node (x,y) and cluster center $c_s(x,y)$: d_s
The splitting lines are consisting of the ground node G, and new splitting lines are represented: $d_s = d_{s'}$, $c_s \neq c_{s'}$
3. Auto-splitting D* lite for Auto-splitting
Plan the initial path (static)
Follow the planned path
Is the robot meet a goal node?
No
End
Is a new obstacle is in the planned path? (Dynamic)
No
Is the new obstacle is in the same area? (Splitting area)
No
Update new path

Fig. 4.7 Flow Chart of Auto-Splitting D* lite

The unnecessary areas are defined as the areas that are not required for the path planning method when calculating from a current node to a target node. Let us have a close look at Fig. 4.6, it can be seen that C6, C7, C8, D6, D7, D8, G2, G3, and G4 are the unnecessary areas. To reduce these unnecessary areas, the method of generating the splitting map in advance when calculating the path is described in Fig. 4.7. It is represented in three steps as follows:

Step 1. Plan the shortest path in an initial map state. (static path planning)

Step 2. Split the large map using the auto-splitting method. (auto-splitting method)

Step 3. Update the path if the map changes in the splitting area where the robot exist. (dynamic path planning to avoid moving obstacle) Else, keep the path. (dynamic path planning to reduce unnecessary areas)

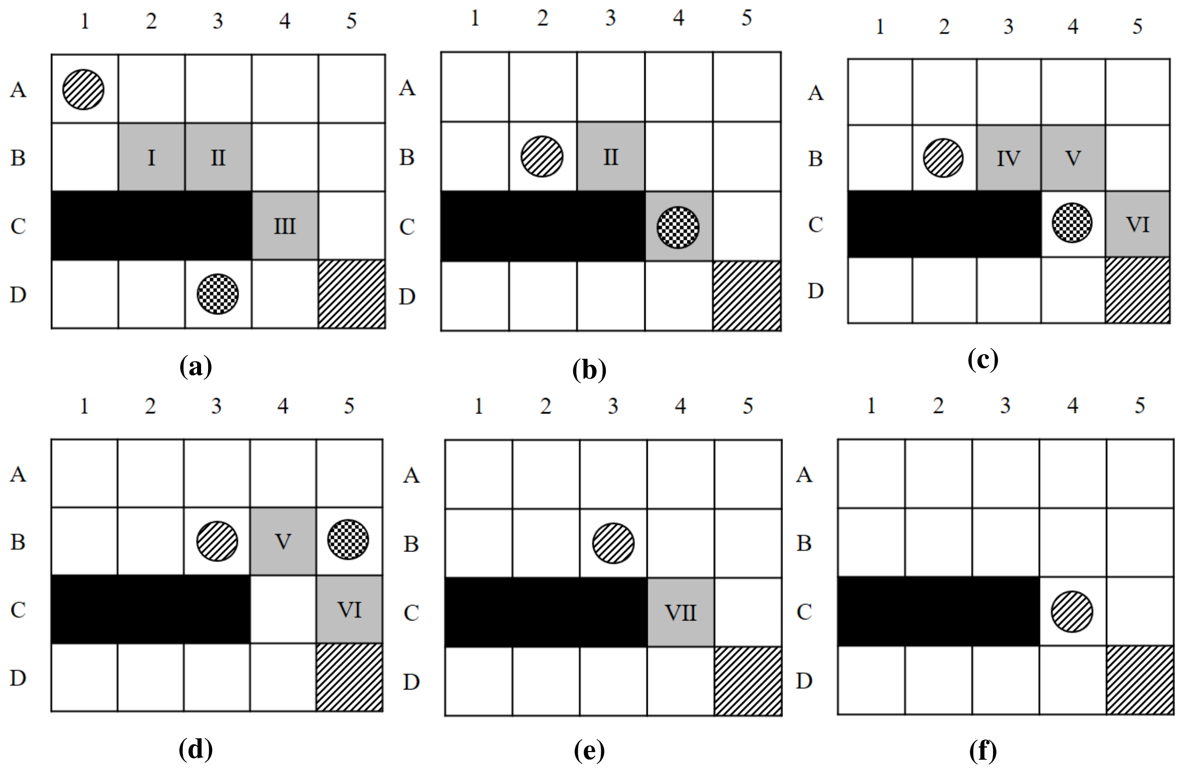


Fig. 4.8 Re-planned Nodes by Traditional D* lite. (a) Planned path (S-I-II-III-G), (b) Dyamic osbtacle block the path (II-III-G), (c) New path generated (IV-V-VI-G), (d) Dyamic osbtacle escape the path, (e) New path generated (VII-G), (f) Find the goal (G)

Fig. 4.8 shows an example of re-planned nodes by traditional D* lite, while Fig. 4.9 shows the re-planned nodes by the AS-D*lite accordingly. Circle with slashed-area in Fig. 4.8 (a) represents a current position, Node with slashed-area represents a goal node (D5), and Circle with checked-flag (D3) represents the moving obstacle (D3→ C4 → B5), respectively. Also, Black nodes represent the obstacle nodes (C1, C2, and C3), White nodes represent the movable nodes, and Gray nodes represent the planned nodes, respectively.

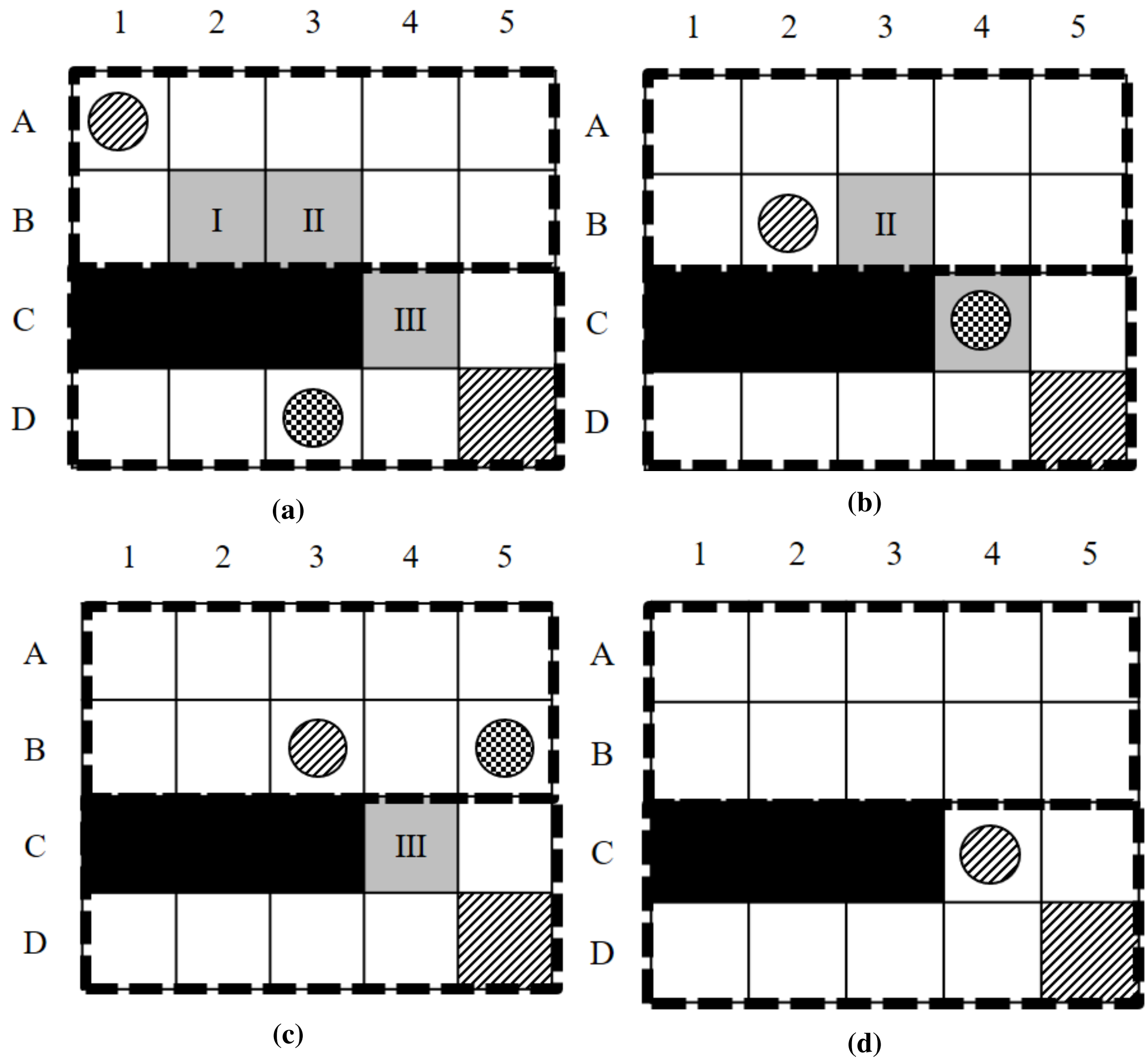


Fig. 4.9 Re-planned Nodes by Auto-Splitting D* lite. (a) Planned path (S-I-II-III-G), (b) Dyamic osbtacle block the path (II-III-G), (c) Unnecessary node reducement (III-G), (d) Moved to next area

As shown in Fig. 4.9 (a), the rescue robot(A1) is in area 1 and moving to the goal(D5). At this point, the planned path is A1, B2, B3, C4, and D5, and the moving obstacle(D5) appears in the area 2. Fig. 4.9 (b) shows the second step of the rescue robot(B2), and it remains in the area 1 and is moving to the goal(D5). The remaining planned path at this step is B2, B3, C4, and D5. The moving obstacle(C4) is currently moving in the area 2 and moves to the planned path B3, C4, and D5, but the

AS-D* lite algorithm does not re-plan the path because the rescue robot and the moving obstacle are not existing in the same area. Fig. 4.9 (c) shows the third step of the rescue robot(B3), and it still remains in the area 1 and is moving to the goal(D5). The remaining planned path is C4 and D5. The moving obstacle(B3) is now in the area 1, but it is not blocking the path. Therefore, the algorithm does not re-plan the path. Fig. 4.9 (d) shows the fourth step of the rescue robot(C4). The rescue robot enters the area 2 and is moving to the goal(D5). The only remaining node of the planned path is D5. Now, the moving obstacle has disappeared. The goal and planned path remained in the same area. The AS-D* lite shows a significant difference in the total number of re-planned nodes(RPN) comparing to the traditional D* lite. Here, RPN is defined as the number of re-planned nodes. Then, the 7 re-planned nodes(RPN) are generated in the traditional D* lite in total, but only 3 re-planned nodes(RPN) are generated in the AS-D* lite in the case of Fig. 4.8 and Fig. 4.9.

4.2.3 Analysis of Auto-Splitting D* lite

The proposed AS-D* lite is an algorithm that removes unnecessary areas with the auto-clustered methods based on the obstacles' locations. Therefore, theoretical analysis such as the time complexity of the AS-D*lite is discussed. In general, it is difficult to calculate the time complexity of the AS-D* lite algorithm. First, the auto clustering algorithm re-calculates when the static map updates globally. However, the re-plans whenever the dynamic obstacle changes. Secondly, the number of obstacle nodes is a unit for the auto-clustering, but the number of grids n in a map is another unit for the time complexity in path planning. Therefore, an expected value for the number of re-planned nodes $E(RPN)$ is newly defined to represent as the calculation time that varies according to the map size and k-value. $E(RPN)$ is introduced to analyze the traditional D* lite and the AS-D* lite using the total number of re-planned nodes (RPN) as shown in Fig.4.10.

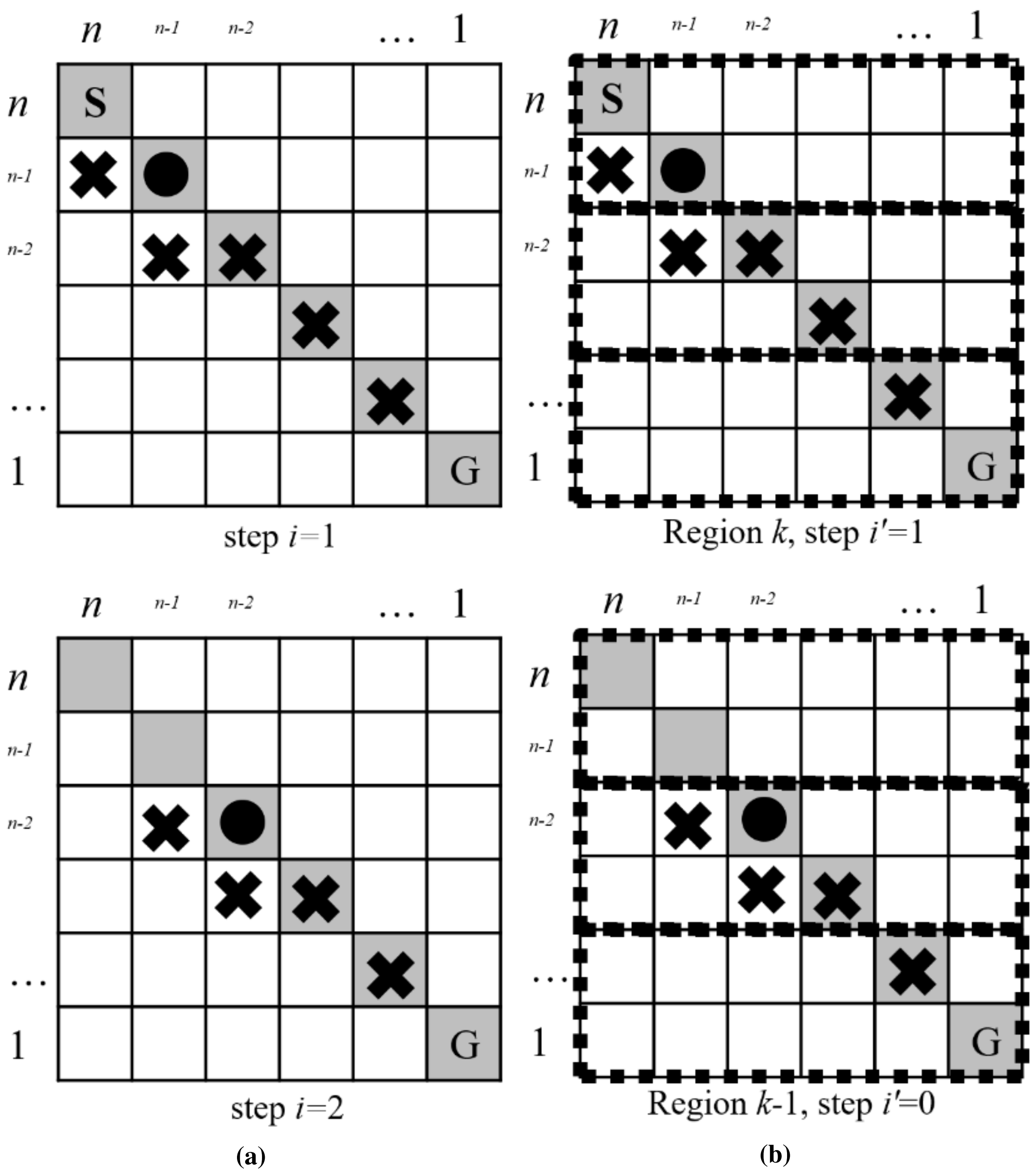


Fig. 4.10 Number of Re-planned Node(RPN) in $n \times n$ Nodes. (a) Re-planned nodes of Traditional D* lite until step $i = n - 1$, (b) Re-planned nodes of AS-D* lite until step $i' = m - 1$ (m represents the number of the planned nodes in region k)

In Fig.4.10, S node represents a start node (x_n, y_n), G node represents a goal node (x_1, y_1), Gray nodes represents planned path nodes and Black circle represents an obstacle node. Especially, the obstacle node that positioned in planned path node means the obstacle blocks the planned path. Continuously, X marks represent re-planned node r, and Dashed lines represent k-region $(1, 2, \dots, j, \dots, k)$ which is the same number of k-value. Finally, $i(i = 0, 1, 2, \dots, n-1)$ is the number of step for tradtional D* lite, and i' is for AS-D* lite.

For the traditional D* lite:

Let us consider a square map of n[node]$\times n$[node], a start node (x_n, y_n) and a goal node (x_1, y_1) in the map. Also, a robot moves from the start node to the goal-node along the planned path nodes as $(x_n, y_n), (x_{n-1}, y_{n-1}), (x_{n-2}, y_{n-2}), \ldots, (x_1, y_1)$. The robot moves one node in every step. Assume a random obstacle node with $U(1, n^2)$ appears anywhere on the map.

List of Symbols for $E(RPN)$ - Traditional D* lite

r	Number of the re-planned node
i	Number of steps. $(i = 0, 1, 2, \ldots, n-1)$
T	Event of the planned path being re-planned if the random obstacle appears on any planned path node that blocks the path.
T_n	Event T happens from (x_n, y_n) to (x_1, y_1)
$R(T_n)$	Number of the re-planned node when T_n occurs
$P(T_n)$	Probability of a re-planned path generated

If the robot is moving on the planned path and the random obstacle appears on any planned path node that blocks the path, then the traditional D* lite generates $r(r = n, n-1, \ldots, 1)$ re-planned nodes. The location of the robot (x_n, y_n) changes to the (x_{n-1}, y_{n-1}). After one step, the previous node (x_n, y_n) is deleted from the planned path nodes. When the robot doesn't have any movement, then the number of step is $i = 0$; when the robot moves from (x_n, y_n) to (x_{n-1}, y_{n-1}), the number of step is $i = 1$; and the maximum number of steps represents as $i = n-1$. For the traditional D* lite, the expected value for the number of the re-planned nodes $E(RPN)$ is represented as

$$
\begin{aligned}
E(RPN) &= \sum_{i=0}^{n-1} P(T_{n-i})\, R(T_{n-i}) \\
&= \sum_{i=0}^{n-1} \frac{1}{n^2}(n-i)^2
\end{aligned}
\tag{4.13}
$$

Table 4.1 Procedure of $E(RPN)$ in Traditional D* lite

T_n	(x_n, y_n) to (x_1, y_1)
$R(T_n)$	n
$P(T_n)$	$\frac{n}{n^2}$
$E(T_n)$	$\frac{n}{n^2} \cdot n$
T_{n-i}	(x_{n-i}, y_{n-i}) to (x_1, y_1)
$R(T_{n-i})$	$n-i$
$P(T_{n-i})$	$\frac{n-i}{n^2}$
$E(T_{n-i})$	$\frac{n-i}{n^2} \cdot n-i$
T_1	(x_1, y_1) to (x_1, y_1)
$R(T_1)$	1
$P(T_1)$	$\frac{1}{n^2}$
$E(T_1)$	$\frac{1}{n^2} \cdot 1$

Table 4.1 shows the outlines of the entire process for computing $E(RPN)$ in the Traditional D*lite.

For the Auto-Splitting D* lite:

Let us consider the square map which is split into k regions by the AS-D* lite algorithm. Also, assume each k-region $(1, 2, \ldots, j, \ldots, k)$ contains the same number of planned path nodes n/k (natural number). Here, k is a number of regions split by the k-value, and $j(j = k, k-1, \ldots, 1)$ are any area among the regions automatically split by k. Then, the planned path nodes starting from k region through j region to the final region 1 are represented as

$$\begin{aligned}
&\left(x_{\frac{nk}{k}},\, y_{\frac{nk}{k}}\right), \left(x_{\frac{nk}{k}-1},\, y_{\frac{nk}{k}-1}\right), \left(x_{\frac{nk}{k}-2},\, y_{\frac{nk}{k}-2}\right), \\
&\cdots, \left(x_{\frac{nk}{k}-\left(\frac{n}{k}-1\right)},\, y_{\frac{nk}{k}-\left(\frac{n}{k}-1\right)}\right), \cdots, \left(x_{\frac{nj}{k}},\, y_{\frac{nj}{k}}\right), \left(x_{\frac{nj}{k}-1},\, y_{\frac{nj}{k}-1}\right), \left(x_{\frac{nj}{k}-2},\, y_{\frac{nj}{k}-2}\right), \\
&\cdots, \left(x_{\frac{nj}{k}-\left(\frac{n}{k}-1\right)},\, y_{\frac{nj}{k}-\left(\frac{n}{k}-1\right)}\right), \\
&\cdots, (x_1,\, y_1)
\end{aligned}$$

List of Symbols for $E(RPN)$ - AS-D* lite

r	Number of the re-planned node in each region
m	Number of the planned nodes in region k
i'	Number of steps in each region. ($i' = 0, 1, 2, \ldots, m-1$)
A	Event of the planned path being re-planned if the random obstacle appears on any planned path node that blocks the path
A_{km}	Event A happens from (x_{km}, y_{km}) to (x_1, y_1) in region k
$R(A_{km})$	Number of the re-planned node when A_{km} occurs
$P(A_{km})$	Probability of a re-planned path generated

If the robot is moving on the planned path in the region j, and the random obstacle appears on any planned-path node that blocks the path in the region j, then the AS-D* lite generates the number of re-planned nodes $r(r = n, n-1, \ldots, 1)$. The node of the robot (x_{km}, y_{km}) changes to the (x_{km-1}, y_{km-1}) in the region k. After one step, the previous node (x_{km}, y_{km}) is deleted from planned path nodes. For the AS-D* lite, the expected value for the number of the re-planned nodes $E(RPN)$ is represented as

$$\begin{aligned} E\left(RPN\right) &= \sum_{i'=0}^{m-1}\sum_{j=1}^{k} P\left(A_{km-i'}\right) R\left(A_{km-i'}\right) \\ &= \sum_{i'=0}^{m-1}\sum_{j=1}^{k} \frac{1}{n^2}\left(m-i'\right)\left(jm-i'\right) \end{aligned} \tag{4.14}$$

Table 4.2 shows the outlines of the entire process for computing $E(RPN)$ in the AS-D*lite. Not only area j but also area k should be calculated and added. The $E(RPN)$ in different map size n for equations (4.13) and (4.14) are shown in Fig. 4.11. The x-axis shows the size of the map (n[node]$\times n$[node]) and the y-axis shows the expected value for the number of the re-planned nodes $E(RPN)$.

Table 4.2 Procedure of $E(RPN)$ in AS-D* lite

A_{km}	First node to last node in region k
$R(A_{km})$	n
$P(A_{km})$	$\frac{n/k}{n^2}$
$E(A_{km})$	$\frac{n/k}{n^2}\cdot n$
$A_{km-i'}$	i' step of node to last node in region k
$R(A_{km-i'})$	$\frac{(nk)}{k}-i'$
$P(A_{km-i'})$	$\frac{(n/k)-i\prime}{n^2}$
$E(A_{km-i'})$	$\frac{(n/k)-i\prime}{n^2}\cdot(\frac{nk}{k}-i')$
A_{11}	Last node to last node in region k
$R(A_{11})$	1
$P(A_{11})$	$\frac{1}{n^2}$
$E(A_{11})$	$\frac{1}{n^2}\cdot 1$

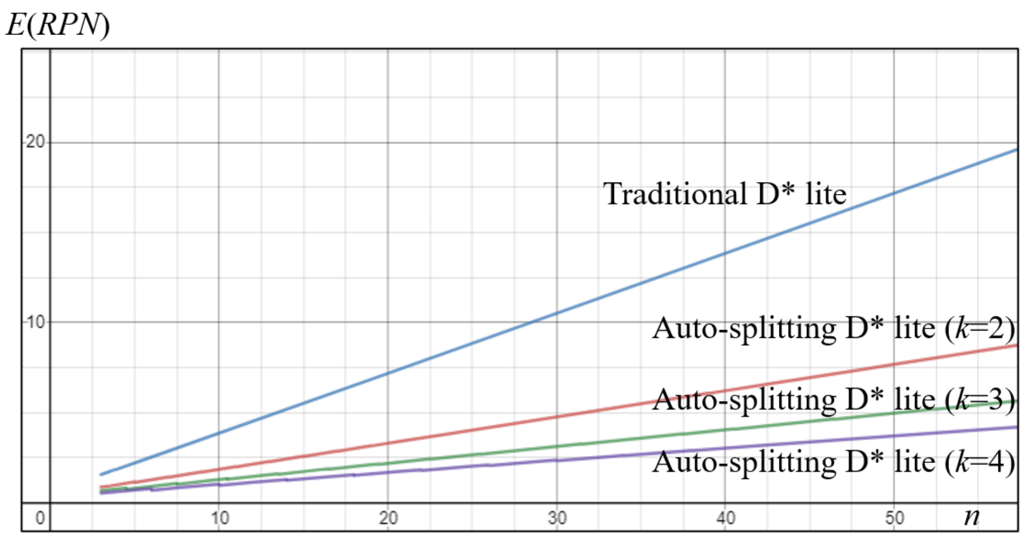


Fig. 4.11 $E(RPN)$ in Different Map Sizes n for Equations (4.13) and (4.14)

The calculation time of the grid-based path planning methods is based on the expansion of the nodes collected on the open and closed list by the non-descending order. Therefore, the data structure and input arrays are more important than the structure of the algorithm. The time complexity is as follows if we focus on one-dimensional node movement. D* lite algorithms run like an A* algorithms, so the time complexity of the first iteration is $O(n)$ (where n is the total number of nodes in graphs) in the worst case. For the path updates, the number of obstacles that have been updated can be much more than A* in the worst case. However, this research takes place on a two-dimensional map rather than a one-dimensional graph. Accordingly, the time complexity increases to $O(n^2)$, and this is also the reason why the binary heap sorting $O(nlog(n))$ is basically used to reduce the time complexity. Most of the searching algorithm's typical approach is to use a binary heap if the structure is the same as this research [106]. This research also uses the binary heap, so a Big-O notation for time complexity is the same. Based on the Big-O notation, the time complexity of D* lite is $O(nlog(n))$, where n is the total number of the nodes in the map in the best case. However, the AS-D* lite algorithm's best cases are represented by $1/k$ node-expansion, and it is influenced by the total amount of the data. The number of basic operations of the whole algorithm can be expressed as:

$$\begin{aligned} T_{total} &= T_{pre} + T_{main} + T_{post} \\ T_{main} &= 1/k\text{O}(nlog(n)) \end{aligned} \tag{4.15}$$

where, n is the number of nodes in the map; T_{pre} is a constant number of basic operations before entering the main loop; T_{pre} is a constant number of basic operations after the main while-loop. T_{main} is $1/k\text{O}(nlog(n))$. It can be written that the time complexity of the AS-D* lite algorithm is expressed as $\text{O}(nlog(n))$ in worst cases, and the dominant is $1/k$ compared to traditional D* lite theoretically. The time complexity of heuristic search algorithms such as BFS (Breadth-first search), Dijkstra's, and A* changes depending on the sorting methods [107,108]. So it is better to compare the expected value for the number of re-planned nodes and the average update nodes in different map sizes as shown in Fig. 4.12 and Fig.4.13. The details are described in the simulation section.

To analyze the time complexity of the preprocessing part T_{pre} of the map segmentation, it is necessary to understand the theoretical time complexity of the k-means clustering and the time complexity of other methods comparable to the k-means. The k-means based clustering methods such as the k-means, a k-modes [109], and a k-medoids [110] are compared, and the density-based mean-shift algorithm [111] is compared as follows.

To analyze the time complexity of the k-means based algorithm according to this study, a few variables c, k, and i need to be introduced beforehand. The c variable represents the number of obstacle nodes in the k-means based algorithm. Note that we used the commonly expressed n variable in the T_{main} part, while the number of bobstacle c is used instead of n variable in the T_{pre} part. As the size of the map increases, the number of obstacles also increases, so variable c is adjusted. k represents the number of regions and is fixed to analyze the time complexity according to the size of the map. i is the iterations until convergence, and it is set with the maximum iterations in this study. As an important condition, $k << c$ and $i << c$ are required.

Firstly, the k-means-based clustering methods have $\text{O}(cki)$ time complexity. In this study, the k-means has aims to partition the number of obstacles c into k regions in which each obstacle belongs to the regions with the nearest mean (cluster centers or cluster centroid). The increment of obstacles can be interpreted in the same way as the map size increase. For accurate analysis with other algorithms, an increase in the c value is focused, k=10 is fixed, and the maximum iterations are set to 100. Therefore, it is predictable that the time complexity increases with c.

Secondly, the k-modes-based clustering also have $\text{O}(cki)$ time complexity. The k-modes is an extended method of the k-means but using the nearest modes (most frequent value) instead of the nearest mean. Using the nearest modes in the k-modes has advantages to categorical data types. Compared with k-mean, the information of obstacle's position is assumed as categorical data. All the values of c,k and i are the same as the k-means.

Thirdly, a representative algorithm of the k-medoids-based clustering called PAM (Partitioning around medoids) has O$(ik(c-k)^2)$ time complexity [112]. In contrast to the k-means clustering algorithms, the k-medoids clustering algorithms choose actual data points as centers (medoids or exemplars). Thereby allows for greater interpretability of the cluster centers than in the k-means algorithms, where the center of a cluster is not necessarily one of the input data points. All the remaining medoids $(c-k)$ aimed to find the set of medoids that has the minimum cost function. The loop would be k for looping through all the medoids $(c-k)$. Then it will be $(c-k)$ to loop through all the non-medoid data points. Then $(c-k)$ again for choosing the random medoid. Therefore, the k-medoids have O$(ik(c-k)^2)$ time complexity if the iteration is i. The analysis focuses on the increase in c value, k=10 is fixed same as the other methods. However, the maximum iterations are set to 10 because of the slow convergence.

Finally, the mean-shift-based clustering methods have O(c^2i) time complexity. The mean-shift clustering algorithm is a mode-seeking algorithm. The mean-shift is a hill-climbing algorithm that involves shifting kernels iteratively to a higher density region until convergence. Therefore, a bandwidth of the kernel is used to simulate random samples. It means, instead of c, the bandwidth would increase when the map size changes. At every iteration, the kernel is shifted to the centroid or the mean of the points within it. In the case of mean-shift algorithms, the time complexity can be expressed in the same way as the k-means algorithms, but due to the characteristic of the mean-shift algorithms, c increases only when the bandwidth of the kernel increases. Therefore, the bandwidth of the kernel is adjusted in this study. The bandwidth according to the map size is as follows: 20×20-1.8, 40×40-3.6, 60×60-5.4, 80×80-7.2, 100×100-9, 120×120-10.8, 140×140-12.6, 160×160-14.4, 180×180-16.2, 200×200-18.

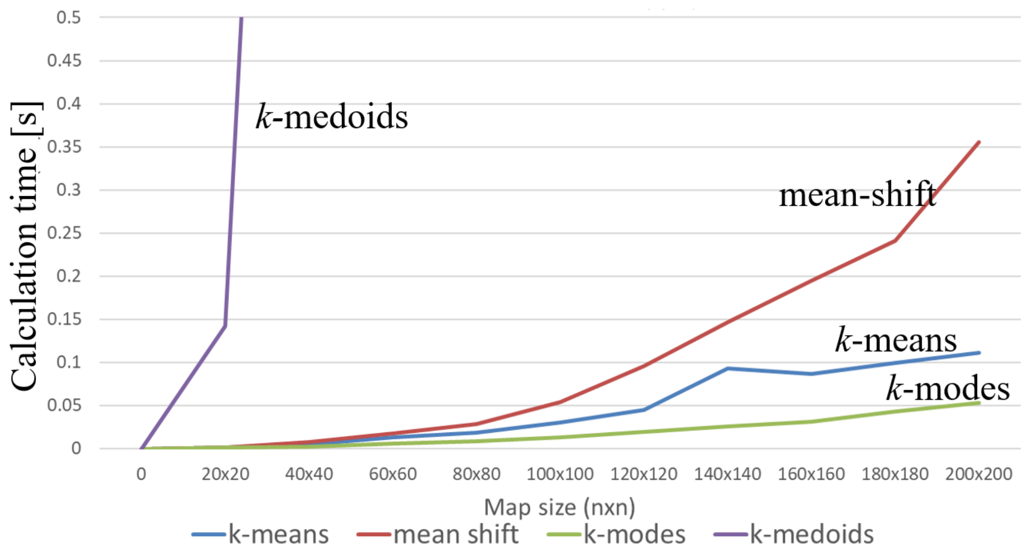


Fig. 4.12 Real-Time Performance (T_{pre})

Fig.4.12 shows the simulation of real-time performance for the T_{pre}. Random obstacles are set to 25% of the map. Therefore, c is $0.25n^2$, but the mean-shift uses the bandwidth. The important point is that even if c is $0.25n^2$, the actual c and n should not be substituted. Since this represents the order in the time complexity, T_{main} and T_{pre} are separated. The result shows that the changes in the calculation time are almost the same as the theoretical time complexity. In terms of performance, the k-modes algorithms are the fastest, and the k-medoids algorithms are judged to be difficult to use. In terms of the scalability of the algorithm, the k-means algorithm has the best balance. Therefore, the k-means algorithm is selected. The k-modes algorithm is also possible to be selected due to its advantages for data types with categorical variables.

In this section, the reason for the algorithm selection is explained through time complexity analysis in various situations of T_{main} and T_{pre}. In addition to the clustering algorithm, T_{pre} includes a method for determining the k value and separating the map. Since calculations are added according to the time complexity of these clustering algorithms, the order follows the higher order of the clustering algorithm. The real-time performance for the entire T_{pre} is shown in Section 4.1. Auto-splitting method for Auto-clustering and drawing.

4.3 Simulations

4.3.1 Auto-Splitting Method for Auto-Clustering and Drawing

Simulation for the AS-D*lite algorithm is carried out in the Unity 3D environment. Simulation PC has a performance of CPU: i7-7700HQ, GPU: GTX 1060, RAM: 16GB. In the AS-D*lite, the auto-clustering methods are able to segment the maps automatically. The ways to determine the *k*-value are compared with the auto-clustering methods such as the Silhouette coefficient and the gap statistics. All algorithms are applied with the Voronoi diagram. The simulation condition of the gap statics and the Silhouette coefficient is given at 40,000 nodes, which will be the same as the city map simulation shown later.

Table 4.3 Calculation Time of Auto Clustering Method

Map size	*k*-means	mean-shift	*k*-modes	*k*-medoids
20×20	0.0014	0.0015	0.0011	0.1419
40×40	0.0049	0.0075	0.0027	1.9937
60×60	0.0128	0.0176	0.0060	9.7364
80×80	0.0184	0.0290	0.0085	31.2622
100×100	0.0303	0.0537	0.0131	77.6559
120×120	0.0453	0.0958	0.0192	162.7345
140×140	0.0927	0.1467	0.0259	296.2298
160×160	0.0870	0.1949	0.0311	509.5957
180×180	0.0994	0.2408	0.0427	798.9193
200×200	0.1115	0.3558	0.0534	1227.0020

Table 4.4 Calculation Time of Auto-splitting Method(combinations)

Method	Calculation time
Silhouette coefficient approach with *k*-means	11.47[s]
Gap statistics approach with *k*-means	2.50[s]

Theoretically, the calculation time itself is not large if excluding the clustering part, but the total calculation process takes a long time in the simulation due to the reason that graphical expressions are used in the map. As shown in Table 4.3, the gap statistics are running with the *k*-means algorithm, so the time complexity is the same. However, the Silhouette coefficient has higher order than the time complexity of the *k*-means algorithm. Therefore, it takes more time to calculate. From Table 4.3 to Table 4.4, the *k*-means and *k*-modes algorithms have the lowest time complexity. The *k*-means algorithms use the nearest mean and have advantages for numerical variables. The *k*-modes algotirhms use the nearest modes (most frequent) and have advantages for categorical variables. Both *k*-means based clustering algorithms are fast for larger maps. We decide to apply the *k*-means algorithm because of the scalability it has. Moreover, there is not much significant difference between the calculation pseed of *k*-modes algorithms and *k*-means algorithms when the the gap statistics are applied. When the gap statistics is applied, the *k*-value can be determined about 80% faster than the other traditional methods. The algorithm that determines the *k*-value only works once until the background map is completely changed. Moreover, this simulation focuses on the preprocessing part, so only the calculation time is analyzed as shown in Table 4.4. The performance of the AS-D*lite is evaluated using the expected value for the number of the re-planned nodes $E(RPN)$.

4.3.2 Auto-Splitting D* lite

4.3.2.1 Simulation for Special Case of Auto-Splitting D* lite

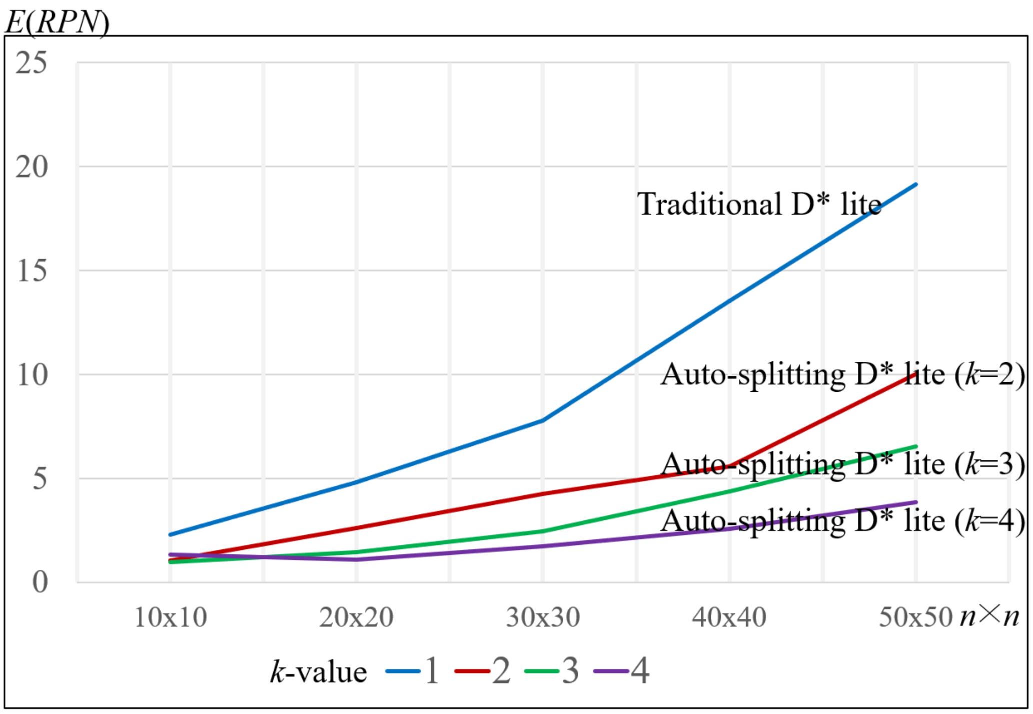


Fig. 4.13 $E(RPN)$ in Different Map Sizes n.

The following simulations are a validation test using $E(RPN)$. Performance of the traditional D* lite and the AS-D* lite under the condition of *k*=2,3, and 4 is compared with each other. The total re-planned nodes for each case n are calculated. The special case of the AS-D* lite assuming each *k*-region $(1,2,\dots,j,\dots,k)$ has the same number of planned-path nodes as $m = n/k$ (natural number) described in Section 4.3.1. The simulation conditions are set to satisfy the natural numbers as follows: 50×50 is replaced by 51×51 or 52×52 when *k*=3 or 4, respectively; 40×40 is replaced by 42×42 when *k*=3; 30×30 is replaced by 32×32 when *k*=4; 20×20 is replaced by 21×21 when *k*=3; 10×10 is replaced by 12×12 when *k*=3 and 4. All simulations are conducted 5,000 times under the same conditions as shown in Fig. 4.11 to validate the expected value for the number of the re-planned nodes $E(RPN)$.

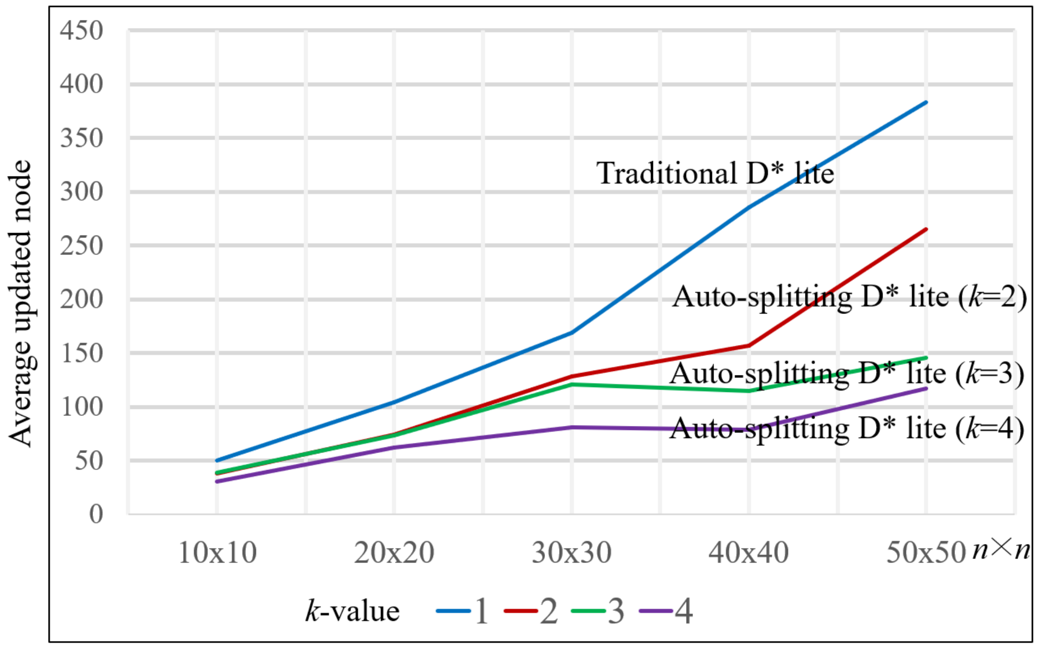
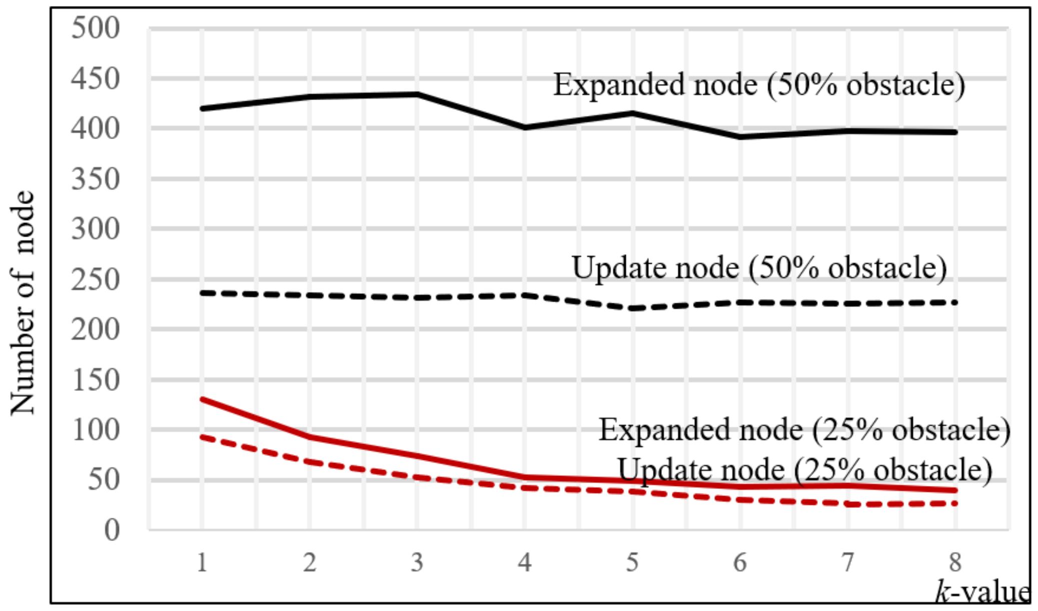


Fig. 4.14 Average Updated Node in Different Map Sizes

Fig. 4.13 shows $E(RPN)$ in different map sizes using simulations. The overall error rate is around 10% comparing to Fig. 4.11. Fig. 4.14 shows the average updated nodes in different map sizes. The updated node contains 8 surrounded nodes of the current node. Also, the updated nodes are possible to overlap with each other if the re-planned paths are created multiple times. Unlike $E(RPN)$, the average number of updated nodes is re-calculated when any value in the open-list of the D* lite algorithms changes [11].

4.3.2.2 Simulation for Different Ratio of Random Obstacles

Fig. 4.15 shows simulation results with different amounts of the obstacles in the same map size. The simulation shows the difference in the reduction rate of the updated nodes resulting from the change in the value for the number of regions k in a map of 2,500[node] for 3 cases: random static obstacles, two random dynamic obstacles, and a random start position. In particular, the random static obstacles refer to the density of obstacles on the map. The density for 25% of obstacles on the map is set to represent as a rural area, and 50% of obstacles on the map are set to represent as a city area. Each simulation is repeated 500 times, where region k=1 shows the traditional D* lite and region $k \geq 2$ shows the AS-D* lite.



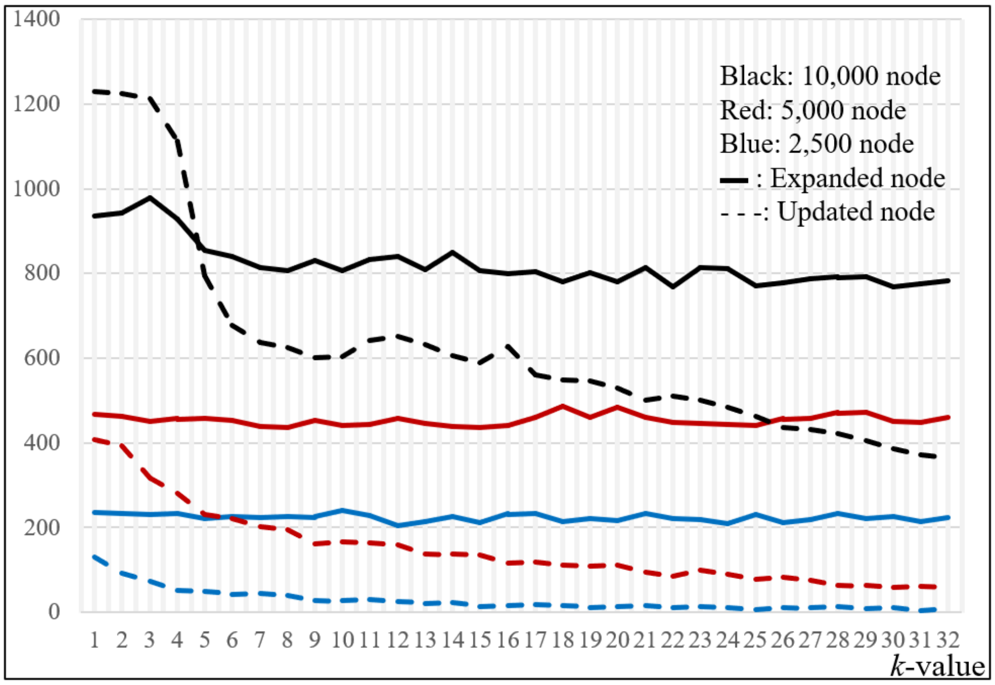
(k=1 : Traditional D* lite, $k \geq 2$: AS-D* lite)

Fig. 4.15 Number of Expanded Node and Updated Node for Different Ratio of Random Obstacles

Black lines represent the simulations in an environment with 50% of static obstacles, and Red lines represent the simulations in an environment with 25% of static obstacles. Solid lines show the expanded node, and Dotted lines show the updated node. As shown in Fig. 4.15, the updated node and the expanded node are reduced by 50%, when the density of the random obstacle is reduced by 50%. Also, large sizes of the map are simulated to show the efficiency of the AS-D* lite.

4.3.2.3 Simulation for Different Map Sizes

Fig. 4.16 shows a simulation in the map of different sizes with the same proportion of random obstacles. The maps of 2,500[node], 5,000[node], and 10,000[node] are compared to each other as shown in Fig. 4.17. The random static obstacles are set at 25% for all three maps, but the number of random dynamic obstacles is set at 2, 4, and 8 for the three maps, respectively. Each simulation is repeated 500 times. The k-value is set from 1 to 32 for showing the difference in the number of expanded nodes and the updated node.



(k=1 : Traditional D* lite, $k \geq 2$: AS-D* lite)

Fig. 4.16 Number of Expanded Node and Updated Node for Different Map Sizes

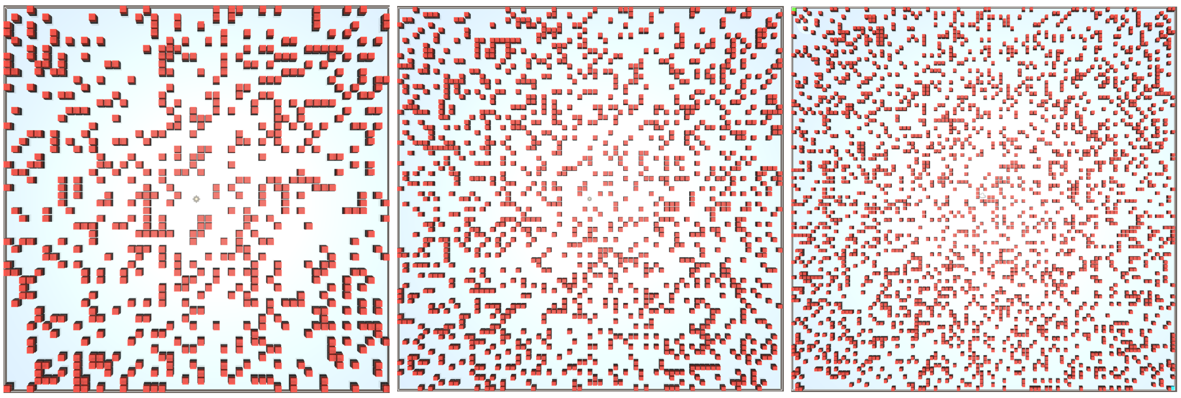
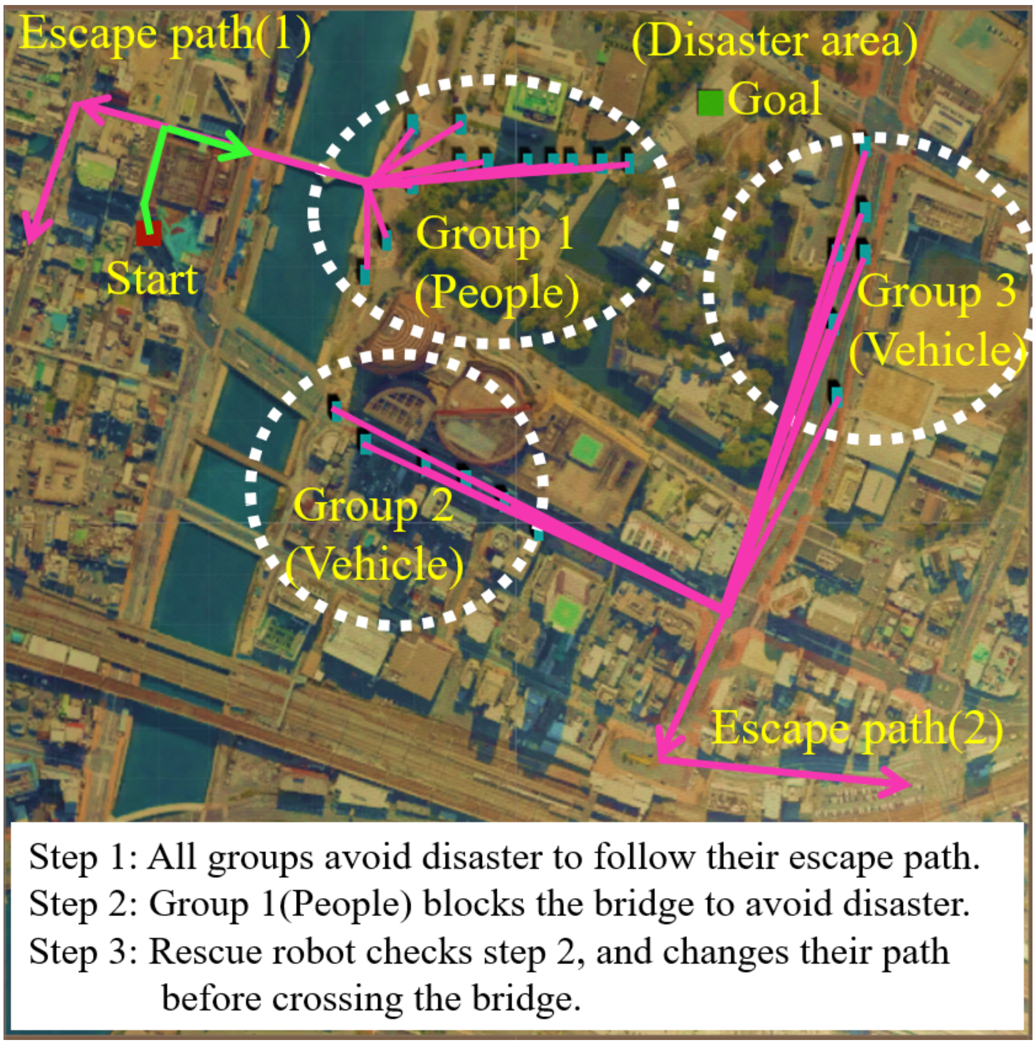


Fig. 4.17 Node Number of Test Map Size. (a) 2,500 [node], (b) Approximately 5,000 [node] (exactly 5,041 [node]), (c) 10,000 [node]

The overall result can be seen that the performance of the AS-D*lite algorithm is better as the map size increases. The results of the AS-D*lite algorithm shows that number of region *k*=7 or 8 is automatically selected on 2,500[node], *k*=4 on 5,000[node], and *k*=5 or 6 on 10,000[node]. The *k*-value varies greatly depending on the random obstacle of the reference data set.

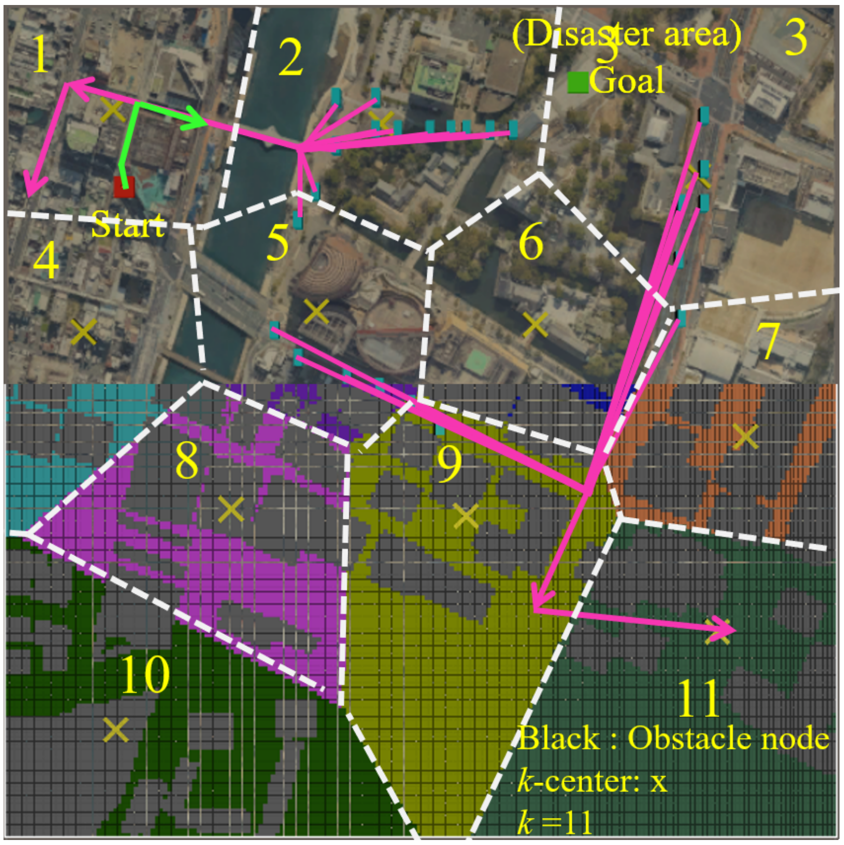
4.3.2.4 City Map Simulation with Fire (Scenario 1)



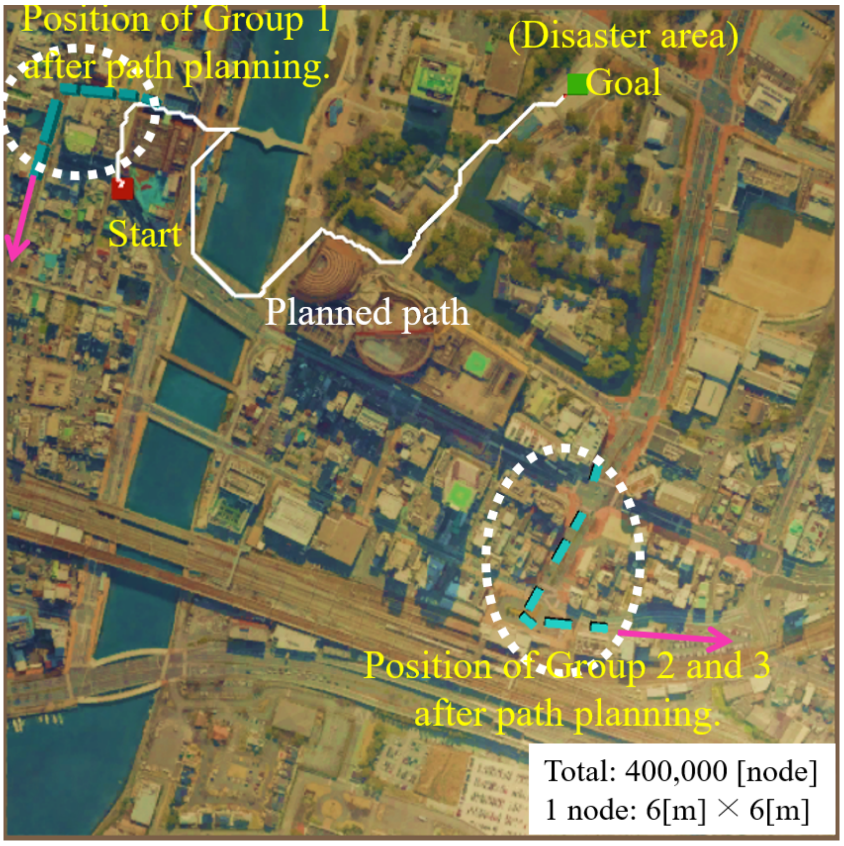
Step 1: All groups avoid disaster to follow their escape path.
Step 2: Group 1(People) blocks the bridge to avoid disaster.
Step 3: Rescue robot checks step 2, and changes their path before crossing the bridge.

Fig. 4.18 Scenario 1 of Rescue Robot Path Planning in Large City Map

This simulation is carried out on a city map for a larger environment near Kokura Castle in Kitakyushu, Japan. Image data of 2048×2048[pixel] taken at a height of 1[km] is selected. The actual map has an axial distance of 1.2[km] and a map size of 40,000[node] is constructed. The size of each node is 6[m]×6[m]. The size of one node was calculated based on the area of the two-lane road in Japan. The width of the lane on the Japanese road is a general two-lane road which is commonly seen, and it is approximately 3.0[m] wide per lane, 3.25[m] on the main road and 3.5–3.75[m] on the highway. The minimum possible range of global path planning was set. Also, it is possible with a smaller node size on a larger map.



(a)



(b)

Fig. 4.19 Path Planning in City Map. (a) Auto-splitting map (*k*=11 is automatically generated), (b) Planned path

Simulations of scenario 1 are shown in Fig. 4.18. Red spot is a starting point and Green is a goal point. Assuming that a fire or disaster occurred around the goal point, three dense groups move to two main escape points. At this point, group 1 is following the escape path (1); group 2 and group 3 are following the escape path (2). The rescue robot is finding the shortest path to the goal. The group 1 is escaping through a bridge, so the rescue robot re-calculates another path for dynamic obstacles. For the robot, the components of groups 1, 2, and 3 become dynamic obstacles. Fig. 4.19 (a) shows how the AS-D*lite splits the map. As a result, the k-value is selected as k=11 or 12. Compared to the traditional D*lite algorithm, the proposed AS-D*lite algorithm shows 18% reduction of the expanded nodes and 58% reduction of the updated nodes simulated in the city map case. The reduction of the expanded nodes means the reduction of unnecessary nodes, and the reduction of updated nodes means the reduction of re-planned nodes in the AS-D*lite algorithm. Fig. 4.19 (b) indicates a path of the robot according to the scenario. Unlike the ideal environment, the number of updated nodes and ratio of expanded nodes does not decrease as much as the ideal environment because of the high ratio of obstacles and non-uniform obstacle density in the actual environment. The number of updated nodes is possible to be reduced by $1/k$ in the theoretical best case.

4.3.2.5 Rural Map Simulation with Landslide (Scenario 2)

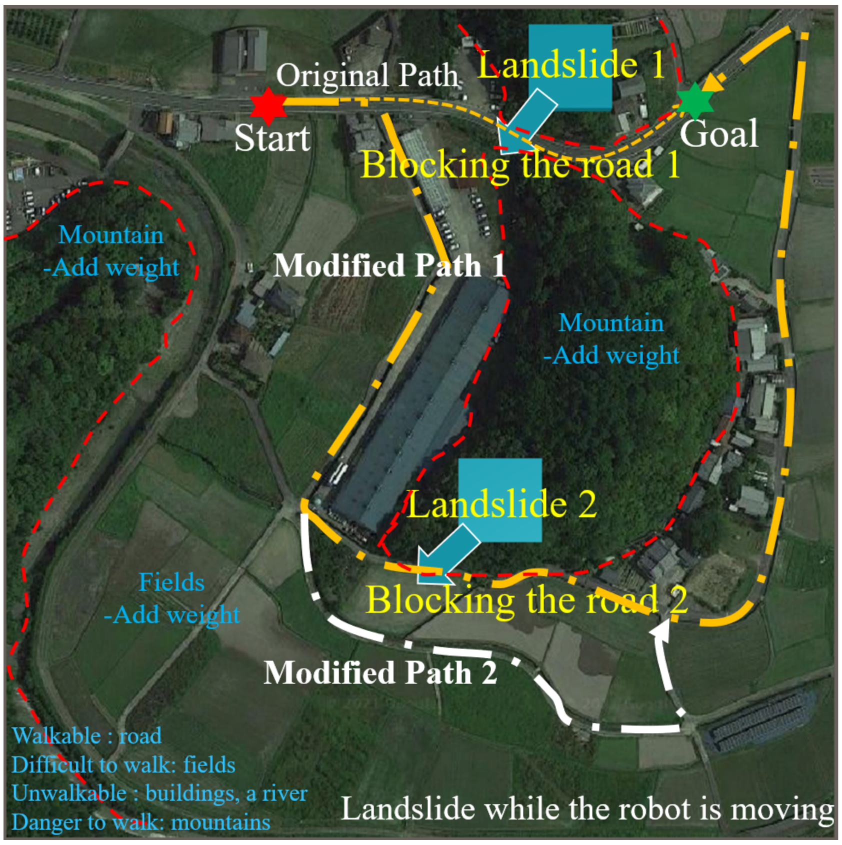
Second simulation is carried out on a rural map for a larger environment near Nishimuro District in Wakayama-city, Japan. Image data of 2048×2048[pixel] taken at the height of 0.5[km] is selected. The actual map has an axial distance of 0.4[km], and a map size of 40,000[node] is constructed. The size of each node is 1[m]×1[m]. Unlike the dynamic global path planning of the city map, the surrounding terrains are also dangerous in the case of landslides. Therefore, most of the environments (mountains, buildings, and rivers) except for roads are set as obstacles in this case. Each obstacle is given a base weight of 1, and each non-obstacle is given a base weight of 0. Moreover, the weight is separated into five different levels according to the potential danger each environment may cause. Fig. 4.20 (b) demonstrates how the different levels are given to every area on the map. Level 1 to level 5 represent from the most dangerous to the least dangerous. Level 1 stands for adding 50% of the base weight, whereas level 5 stands for adding 10% of the base weight. This weight doesn't exceed

1.5 times compare to the existing cost of the obstacle. Also, there are fields adjacent to landslides or lower ground which is also possible to be given a weight. Theoretically, the cost of obstacle nodes in traditional D*lite is infinite and is set by the user. However, since infinity cannot be used in actual applications, it was necessary to properly adjust the cost.

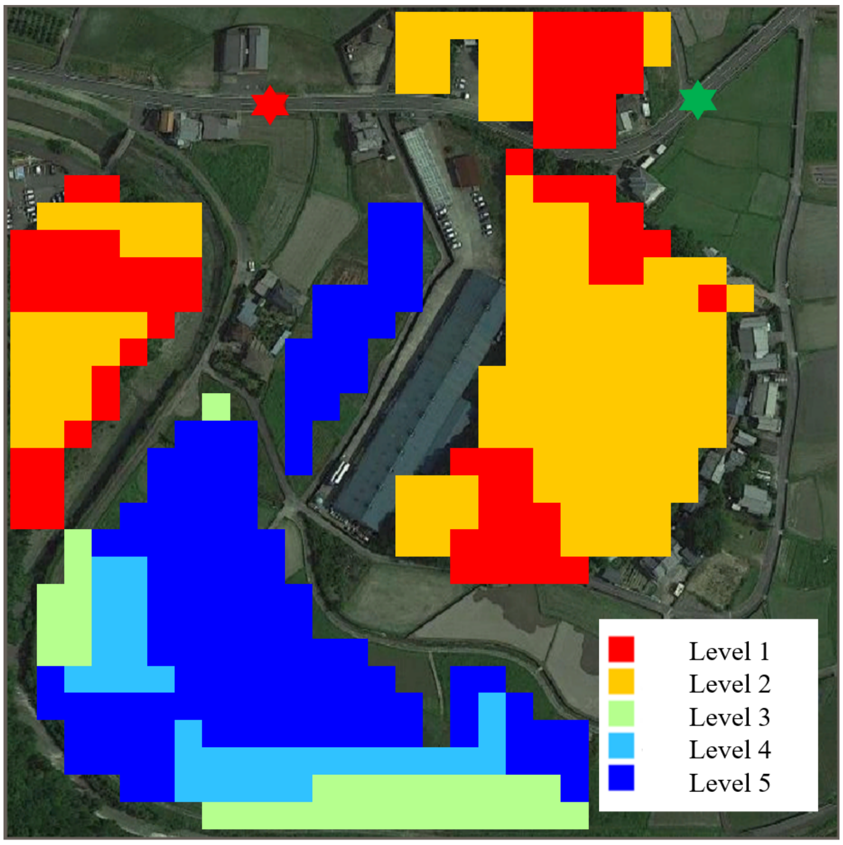
The scenario for rural maps is as follows:

Step 1. A rescue robot executes dynamic global path planning from the start position to the goal.

Step 2. A landslide occurs while the rescue robot is moving, blocking road 1 and road 2 at the same time.

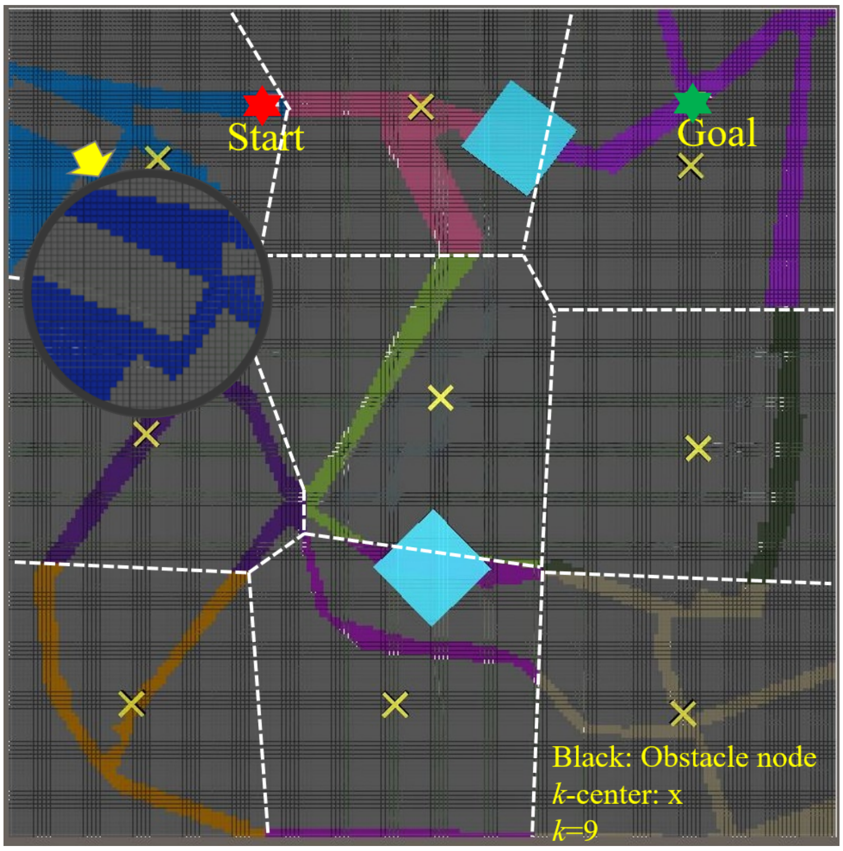


(a)

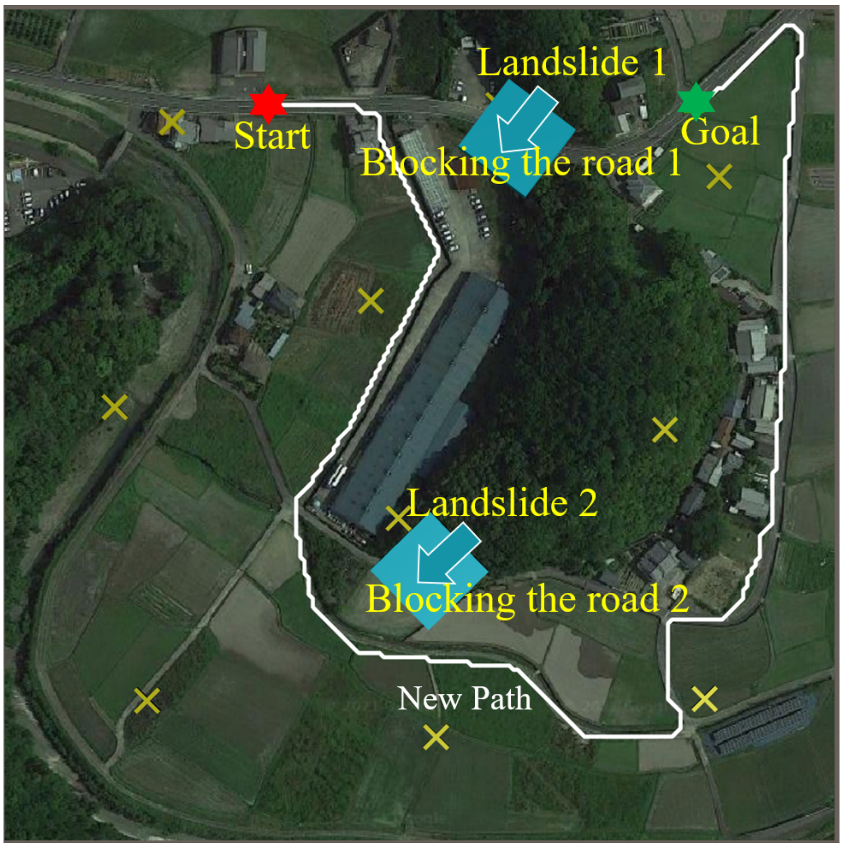


(b)

Fig. 4.20 Rescue Robot Path Planning in Rural Map. (a) Scenario 2 of rescue robot path planning in rural map, (b) Example of landslide alarm mapping



(a)



(b)

Fig. 4.21 Path Planning in Rural Map. (a) Auto-splitting map (k=9 or 10 is automatically generated), (b) Planned path

Simulations of scenario 2 are shown in Fig. 4.20 (a). The red spot is the starting point, and the green spot is the goal point. Assuming that landslides occurred on the mountain, the blue areas moved in the direction of the arrow and blocked the road. The rescue robot initially plans to travel in the shortest distance to get to the goal point, but primarily changes the path due to the influence of the landslide. The robot has encountered a landslide once again while moving on the first modified path, so it changed to the second modified path and arrived at the goal point. Fig. 4.21 (a) shows how the AS-D*lite splits the map. As a result, the *k*-value is selected as *k*=9 or 10. Compared to the traditional D*lite algorithm, the proposed AS-D*lite algorithm shows 11% reduction of the expanded nodes and 46% reduction of the updated nodes simulated in the rural map case. Fig. 4.21 (b) indicates the path of the robot according to the scenario. This simulation result shows that adding weight increases the calculation time compared to the city map due to the reason that every node is given a different level of weight. In this situation, field nodes with small costs are treated as obstacles, the number of the expanded nodes and updated nodes are reduced. Therefore, the calculation time is increased compare to the city map cases as described in section 2, which the weighted method can sometimes lead to a slower search. In this case, although the calculation time is increased compared to the city map, it still performs better than traditional methods.

4.4 Conclusion

In this study, the AS-D* lite algorithm was proposed to perform a dynamic and collision-free path planning task. Two important issues were considered and solved by path planning methods and auto-clustering methods. The first issue was the unnecessary area of the D* based algorithm. The proposed AS-D* lite algorithm has solved the issue of the unnecessary area by ceasing the unnecessary calculation of the traditional D* lite, and alleviated the over-calculation issue of dynamic path planners in the partially known and large area. Furthermore, the path planning method was faster with the AS-D* lite algorithms when the environment was large and dynamic. The second issue was the slow processing time for determining the *k*-value and the way to determine the *k*-value automatically. To shorten the processing time for determining the *k*-value, the gap statistics method was applied to find the optimal *k*-value. Moreover, the Voronoi diagram was combined with the *k*-means algorithm to draw a segmented map automatically. In particular, the $E(RPN)$ was introduced for representing the efficiency of the AS-D* lite algorithm according to the number of split maps and the size of different maps. The $E(RPN)$ is possible to be applied in various grid-based algorithms which operate in dynamic and partially known areas. To sum up, the unneecessary areas with expanded nodes exist in D* lite based path is reduced by the AS-D* lite about 58% on the city map and 46% on the rural map. This results are updated node, which is all node counted number after re-planning.

Chapter 5

Frenet Frame Trajectory Planning by Auto-splitting D* lite

5.1 Research Aims

Until Chapter 4, global path planning was described by classifying it into known, partially known, static, and dynamic according to the environment. In Chapter 5, we aim to plan the trajectory of real robots and vehicles using the path of the proposed global path planning, Auto-splitting D*lite (Dynamic Partially Known). In this research, two targets for path planning and trajectory planning are mainly focused to assist fully autonomous robots and vehicles. The first target is to make dynamic and global path planning based on a grid map for the autonomous vehicles in the large road networks. The second target is to combine the dynamic and global path planning with the local trajectory planning for changing road lanes. Combining a partially known global path planning with dynamic local trajectory planning for fully autonomous vehicles in a large road network will lead to more accurate and faster than other trajectory planning. Since Auto-splitting D*lite is described in detail in chapter 4, most of the descriptions are the Frenet frame trajectory planning with the proposed global path planning and the Kalman filtering to estimate final Frenet frame trajectory.

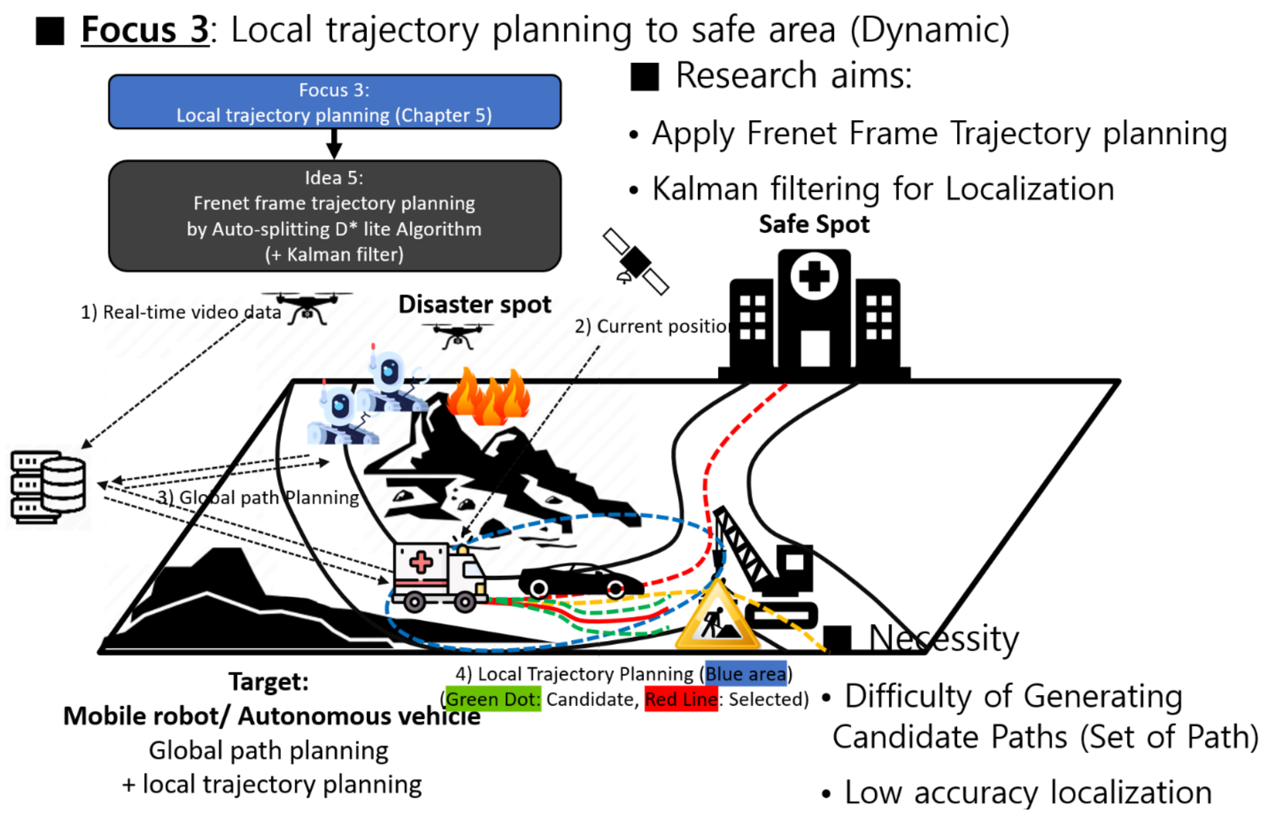

■ Focus 3: Local trajectory planning to safe area (Dynamic)
Focus 3: Local trajectory planning (Chapter 5)
Idea 5: Frenet frame trajectory planning by Auto-splitting D* lite Algorithm (+ Kalman filter)
■ Research aims:
• Apply Frenet Frame Trajectory planning
• Kalman filtering for Localization
Safe Spot
1) Real-time video data
Disaster spot
2) Current position
3) Global path Planning
4) Local Trajectory Planning (Blue area)
(Green Dot: Candidate, Red Line: Selected)
Target: Mobile robot/ Autonomous vehicle
Global path planning + local trajectory planning
■ Necessity
• Difficulty of Generating Candidate Paths (Set of Path)
• Low accuracy localization

Fig. 5.1 Research Aims (Chapter 5)

5.2 Auto-Splitting D* lite Algorithm with Frenet Frame

5.2.1 Frenet Frame Trajectory Planning

The advantages and disadvantages of the difference between the Cartesian frame and the Frenet frame described in related work is briefly as follows. The Cartesian frame is easy to interact with other global path planning algorithms, but it has a slow calculation time because of the many curvature parameters. 5.3 shows the differences between the Cartesian frame and the Frenet frame.

The Cartesian frame is usually called a World Frame W which is the inertial frame in 2D space. The most common coordinate is the rectangle coordinate $(x-y)$. The Frenet Frame F is a coordinate built on the curve. In 2D space, it is composed of four components: a point $(\vec{r})$ on the curve, tangential vector $(\vec{t_r})$, distance to the reference point (d), normal vector $(\vec{n_r})$. In the rectangle coordinate, the curve can be expressed with $f(x,y)=0$. If we introduce an intermediate parameter t, the curve can

■ Cartesian Frame

• General steps in Cartesian Frame: Modeling → Calculate Longitudinal / Lateral forces → Calculate Linear and angular velocity → Calculate Position (Different axis) → Make Candidate Trajectories → Choose Optimal Trajectory

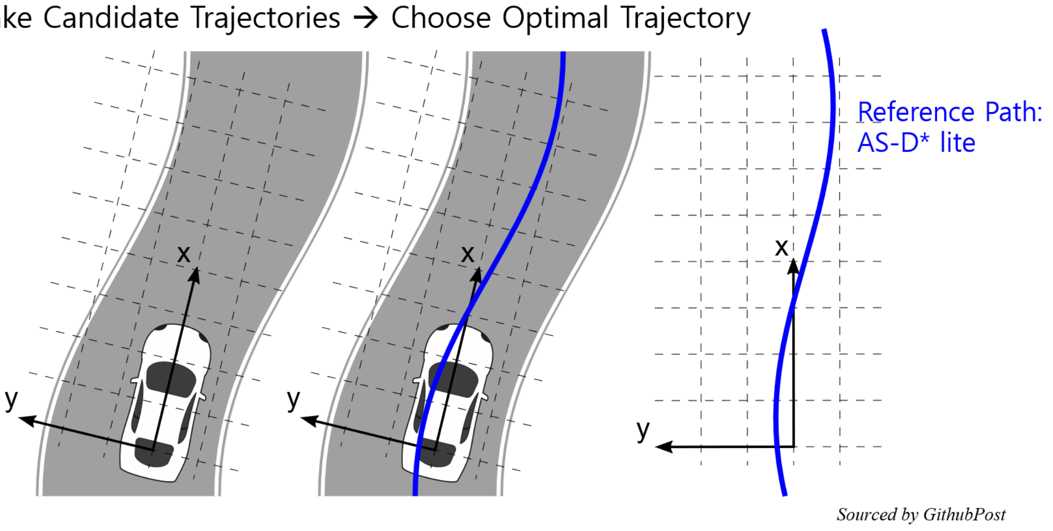
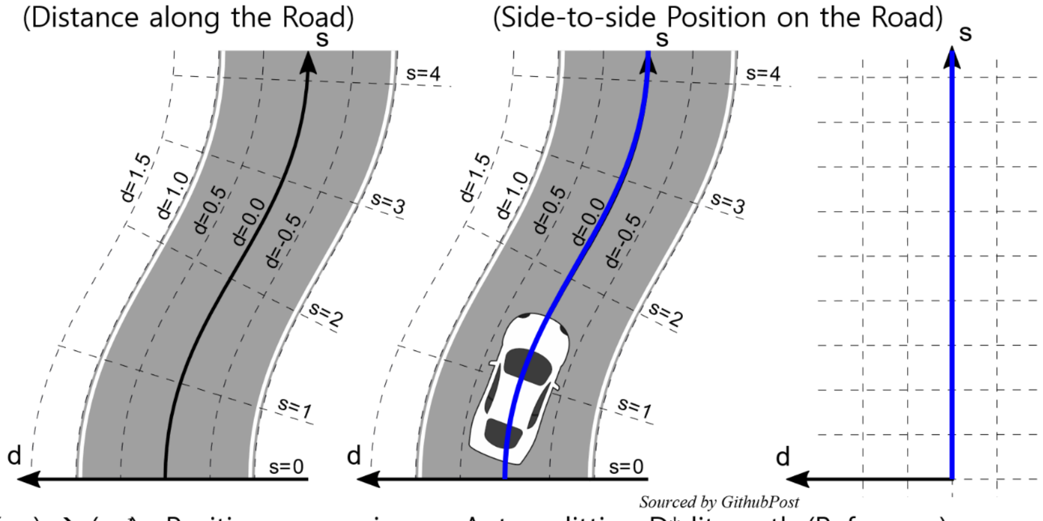


Fig. 5.2 Cartesian Frame-[Sourced by GithubPost]

■ Frenet Frame

• *s* - Longitudinal Displacement (Distance along the Road)

• *d*- Lateral Displacement (Side-to-side Position on the Road)



• (x,y) → (s,d) : Position conversion

• Frenet Frame conversion: Global path to Local path

• Auto-splitting D* lite path (Reference)
→ Make Quintic polynomial (5th Order) trajectory
→ Velocity, Acceleration, Curve is obtained

Fig. 5.3 Frenet Frame-[Sourced by GithubPost]

be expressed with $x(t)$ and $y(t)$. Given the Frenet frame, we can choose arc length s of the curve as the intermediate parameter. Givne that, any point in the 2D space can also be paramterized in the Frenet Frame after deciding the parameter, then the vehicle trajectory is formulated in the following equation.

$$\vec{x}(s(t), d(t)) = \vec{r}(s(t)) + d(\vec{n}_r s(t))) \tag{5.1}$$

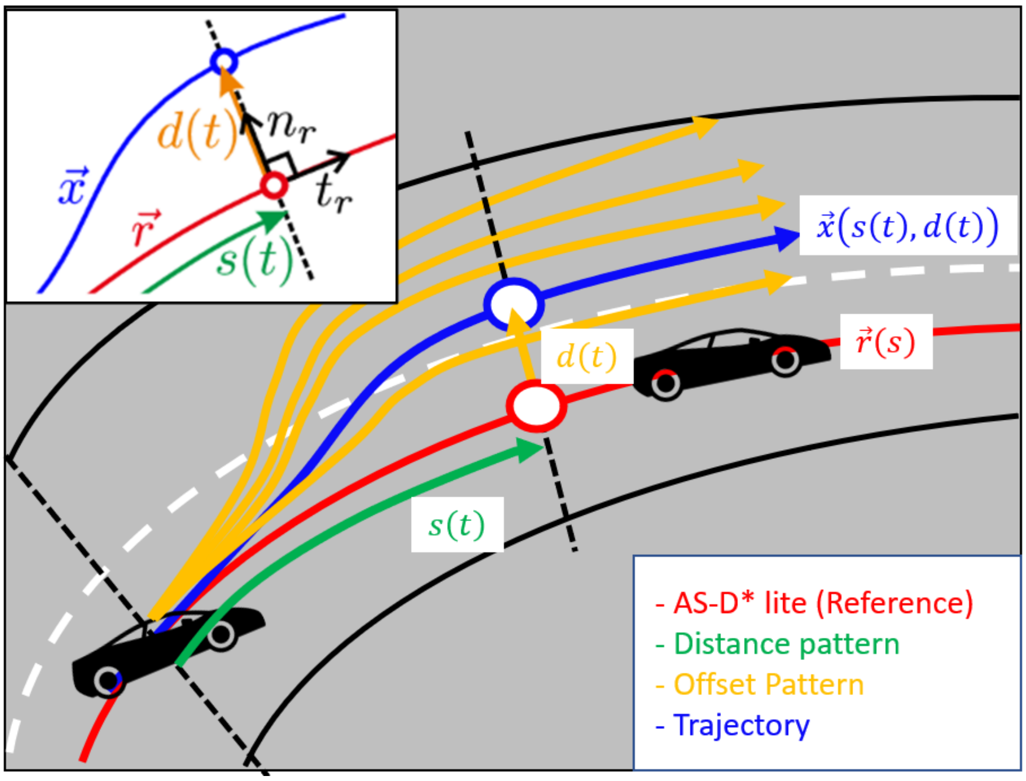


Fig. 5.4 Parameterizations of Frenet Frame

In this study, the reference path is changed from AS-D*lite to Quintic polynomial, and the parameters used for Frenet Frame transformation in Cartesian coordinate are

$$[x, y, \theta, \kappa, v, a] \leftrightarrow [s, \dot{s}, \ddot{s}, d, d', d''] \tag{5.2}$$

AS-D*lite is converted into a Frenet Frame using the kinetic polynomial, with the differential operations of s being $\dot{s}$ and the differential operations of d being d'. The coordinate transformation is performed by the following 12 formulas. Here, a small x represents the time point of the current reference path.

- $x_x = x_r - dsin\left(\theta_r\right)$
- $y_x = y_r + dcos\left(\theta_r\right)$
- $v_x = \sqrt{\left[\dot{s}\left(1-\kappa_r d\right)\right]^2 + \left(\dot{s}d'\right)^2}$
- $a_x = \ddot{s}\frac{1-\kappa_r d}{cos(\theta_x-\theta_r)} + \frac{d'^2}{cos(\theta_x-\theta_r)}\left[d'\left(k_x\frac{1-\kappa_r d}{cos(\theta_x-\theta_r)} - \kappa_r\right) - \left(\kappa'_r d + \kappa_r d'\right)\right]$
- $\theta_x = arctan\left(\frac{d'}{1-\kappa_r d}\right) + \theta_r \in [-\pi, \pi]$
- $\kappa_x = \left(\left(d'' + \left(\kappa'_r d + \kappa'_r d\right) tan\left(\theta_x - \theta_r\right)\right)\frac{cos^2(\theta_x-\theta_r)}{1-\kappa_r d} + k_r\right)\frac{cos(\theta_x-\theta_r)}{1-\kappa_r d}$
- $s = s_r$
- $\dot{s} = \frac{v_x cos(\theta_x-\theta_r)}{1-\kappa_r d}$
- $\ddot{s} = \frac{a_x cos(\theta_x-\theta_r) - \dot{s}\left[d'\left(\kappa\frac{1-\kappa_r d}{cos(\theta_x-\theta_r)} - \kappa_r\right) - \left(\kappa'_r d + \kappa_r d'\right)\right]}{1-\kappa_r d}$
- $d = \sqrt{\left(x_x - x_r\right)^2 + \left(y_x - y_r\right)^2} sign\left(\left(x_x - x_r\right)cos(\theta_r) - \left(y_x - y_r\right)sin(\theta_r)\right)$
- $d' = \left(1 - \kappa_r d\right)tan(\theta_x - \theta_r)$
- $d'' = -\left(\kappa'_r d + \kappa_r d'\right)tan(\theta_x - \theta_r) + \frac{(1-\kappa_r d)}{cos^2(\theta_x-\theta_r)}\left(\frac{1-\kappa_r d}{cos(\theta_x-\theta_r)}\kappa_x - \kappa_r\right)$

5.2.2 Apply Auto-Splitting D* lite Algorithm with Frenet Frame

After the understanding of frenet frame, a brief explanation of the procedure of the Frenet frame trajectory planning is as follows:

Step 1. Qunitic Polynomial Trajectory Generation.

Step 2. Trajectory check.

Step 3. Optimal trajectory selection

In step 1, the local trajectory planning is represented with a quintic polynomial. A unique quintic polynomial is represented in the Eq. 5.1. Notations of each parameter are described: t_s is the initial time of a target vehicle; t_e is the end time of the target vehicle; the initial state for the target vehicle position, velocity, and acceleration at time t represents $[f(t_s) \quad \dot{f}(t_s) \quad \ddot{f}(t_s)]$; the end state for the target vehicle position, velocity, and acceleration at time t represents$[f(t_e) \quad \dot{f}(t_e) \quad \ddot{f}(t_e)]$. Then the trajectory between t_s and t_e can be determined by solving this equation.

$$\begin{bmatrix} 1 & t_s & {t_s}^2 & {t_s}^3 & {t_s}^4 & {t_s}^5 \\ 0 & 1 & 2t_s & 3{t_s}^2 & 4{t_s}^3 & 5{t_s}^4 \\ 0 & 0 & 2 & 6t_s & 12{t_s}^2 & 20{t_s}^3 \\ 1 & t_e & {t_e}^2 & {t_e}^3 & {t_e}^4 & {t_e}^5 \\ 0 & 1 & 2t_e & 3{t_e}^2 & 4{t_e}^3 & 5{t_e}^4 \\ 0 & 0 & 2 & 6t_e & 12{t_e}^2 & 20{t_e}^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} f(t_s) \\ \dot{f}(t_s) \\ \ddot{f}(t_s) \\ f(t_e) \\ \dot{f}(t_e) \\ \ddot{f}(t_e) \end{bmatrix} \tag{5.3}$$

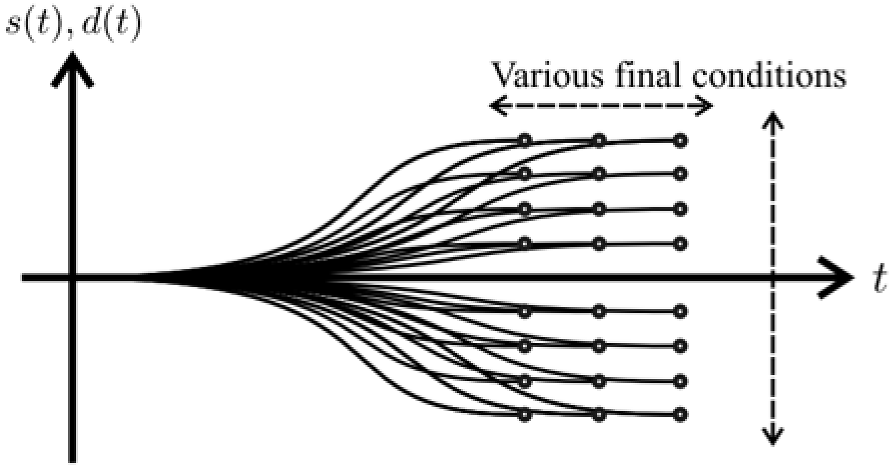


Fig. 5.5 Offset Patterns of Frenet Frame Trajectory- [Sourced by Keisuke Yoneda]

Various patterns using Quintic Polynomial are generated in the following way.

$$[d_1, d'_1, d''_1, \Delta\, T_d] \tag{5.4}$$

$$[\dot{s}_1 + \Delta\dot{s}, \ddot{s}_1, \Delta\, T_s] \tag{5.5}$$

$$s(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5 \tag{5.6}$$

$$d(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{5.7}$$

In step 2, the Frenet frame trajectory planning checks the velocity limitation, acceleration limitation, and curvature limitations. According to the limitation, several quintic polynomial trajectories will be produced. In step 3, the optimal trajectory selection is using a cost estimation function. The total cost of trajectories is represented as $J_{tot} = J_s + J_d$. The cost of lateral trajectories J_d and the cost of longitudinal trajectories J_s are calculated based on the Eq. 5.1 and many outer conditions. Also, weight parameters are recommended in traditional Frenet frame trajectory planning. A trajectory with a minimum cost is the final local trajectory.

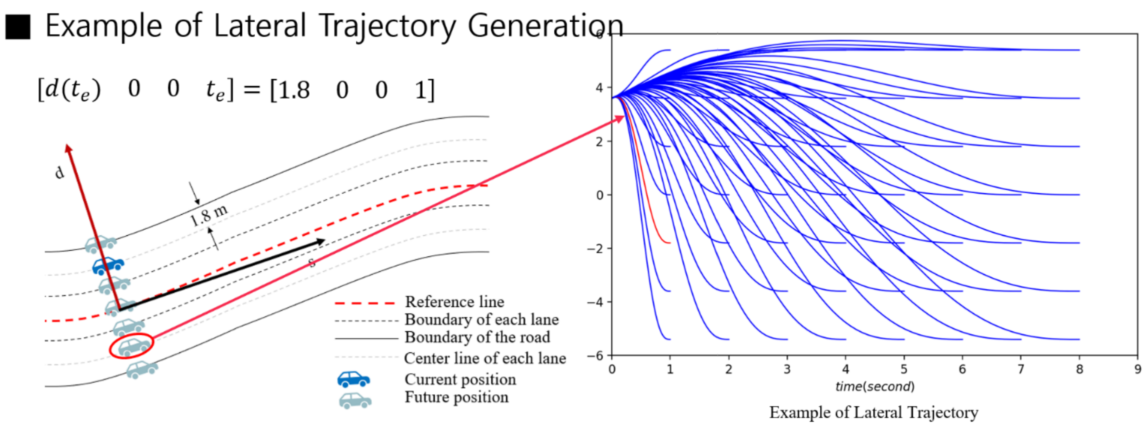


Fig. 5.6 Example of Lateral Trajectory Generation

The lateral trajectory can be calculated. Multiple end states can be obtained by sampling different lateral positions and times, then a cluster of trajectories is received. In this scenario, the lane width is 3.6m, light blue vehicle is the sampled position, dark blue vehicle is the current position. Sample these positions in the future 8 seconds, we can get a cluster of trajectories at this time, which is shown in Fig. 5.6. Due to the different end-state positions and times of sampling, different trajectory curves are obtained, and their comfort and duration are different. The generated trajectory will cover all situations when the sampling is dense enough.

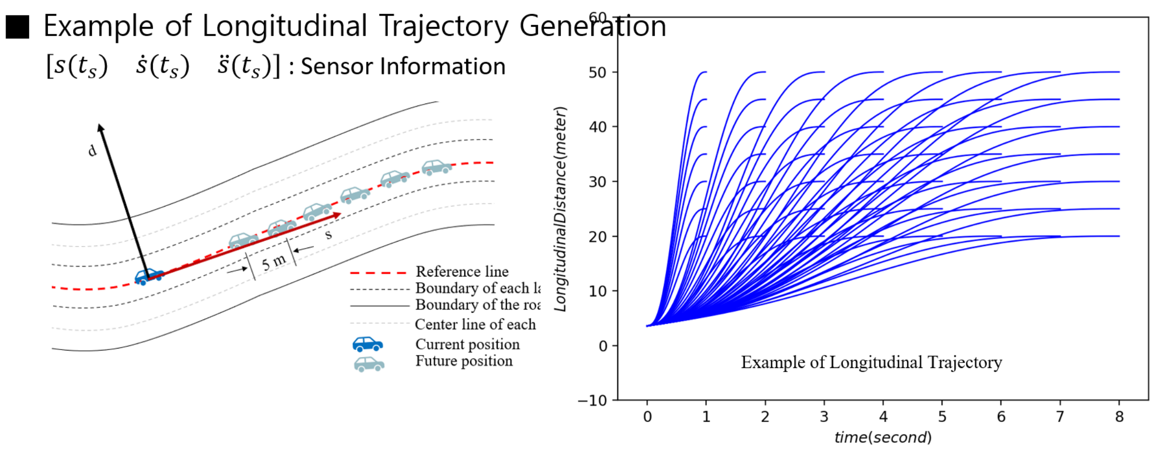


Fig. 5.7 Example of Longitudinal Trajectory Generation

According to different end states, a series of longitudinal trajectories can also be generated, which is shown in Fig. 5.7. Longitudinal trajectory planning usually needs to face four kinds of maneuvers, which are merging, following, velocity keeping and stopping. These four maneuvers all need to sample the final state, but the constraints are different. In the previous section, the coordinate of vehicles is represented as (s,d) in the Frenet frame. The s represents the distance along the reference line with AS-D* lite and the d represents the offset from the reference line respectively. In the present chapter, detailed formulas and methods are omitted because the detailed strategies of the trajectory planning of the detailed Frenet frame follow the strategies of the representative Frenet frame trajectory algorithm. However, the solution to the issues caused by applying the Frenet frame coordinate system, which is the key point of this chapter, is described in the next chapter.

5.2.3 Frenet Frame Trajectory planning with Kalman Filter

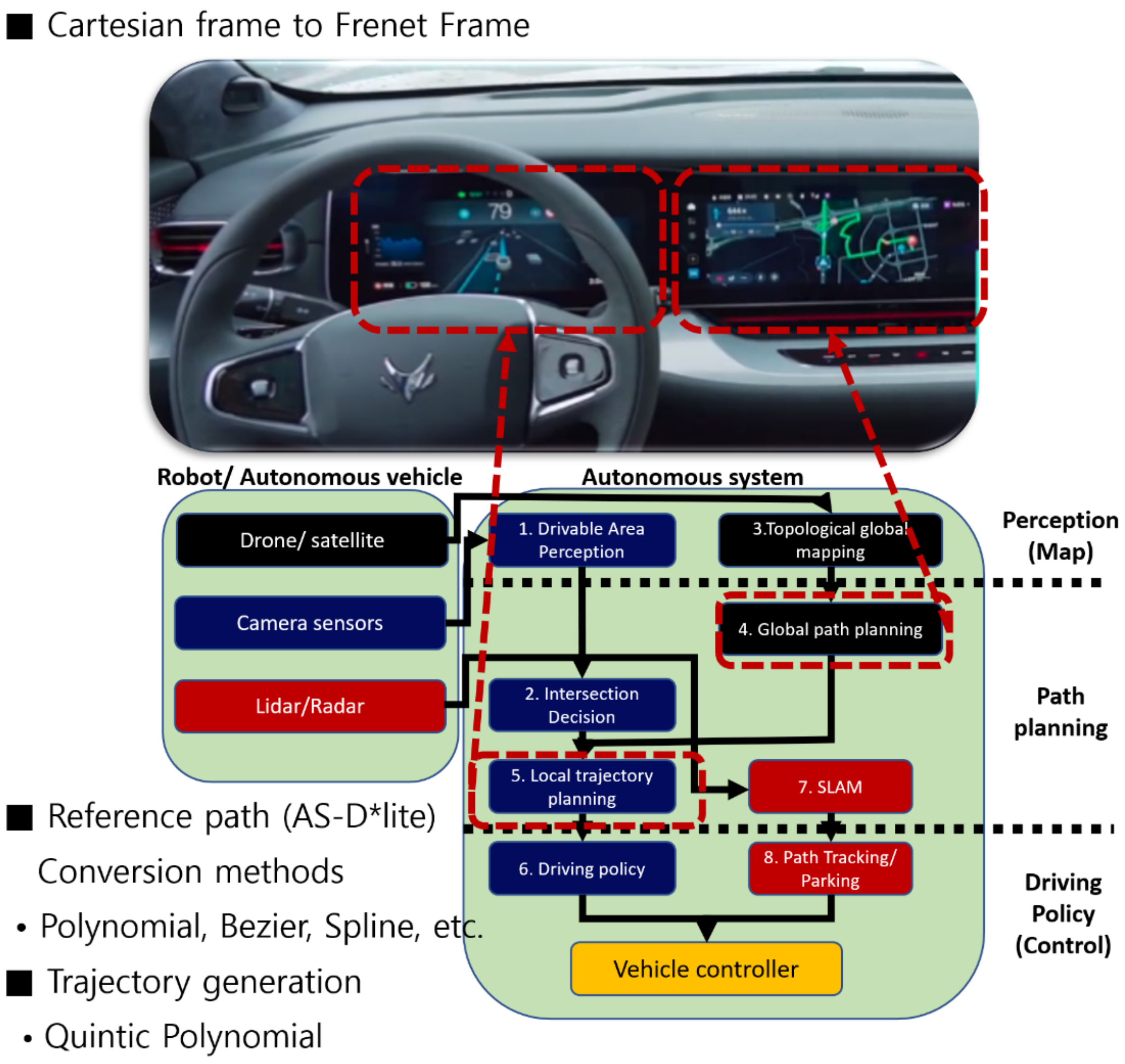


Fig. 5.8 Structure of Frenet frame with AS-D* lite.

The local trajectory planning follows the path of the global path planning. Therefore, the reference path is the global path, and then the coordinate system is changed to a Frenet frame such as 5.4.

The Frenet frame still has lower accuracy for localization even if the Frenet frame trajectory planning is realized the fast dynamic local trajectory planning. Thus, the main purpose of a proposed idea using the Frenet frame trajectory planning with the Kalman filter is to improve performance by increasing the accuracy of localization of target vehicles and surrounding vehicles.5.8 shows how the local trajectory planning is combined with the global path planning. After that, the coordinate system is converted once again to improve the accuracy between the target vehicles and the other vehicles using the Kalman filter. We designed the prediction part is used for the state of other vehicles. A state vector is defined as $X = [x \quad y \quad v_x \quad v_y]^T$; prior estimation state of other vehicles is $X-$; posterior estimation state of other vehicles is X+; state-transition matrix represents F_k; process noise matrix is Q_k; observation noise matrix is R_k; observation matrix is H_k; Kalman gain matrix is K_k

$$P_k^- = F_k P_{k-1}{}^+ F_k^T + Q_k \tag{5.8}$$

$$\hat{X}^-(k) = \begin{bmatrix} x(k) \\ y(k) \\ v_x(k) \\ v_y(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(k-1) \\ y(k-1) \\ v_x(k-1) \\ v_y(k-1) \end{bmatrix} \tag{5.9}$$

$$K_k = P_k^- H_k^T \left(H_k P_k^- H_k^\top + R_k \right)^{-1} \tag{5.10}$$

$$\hat{X}^+(k) = F_k \hat{X}^-(k-1) + K_k \left(Z_k - H_k F_k \hat{X}^-(k) \right) \tag{5.11}$$

$$P_k^+ = (1 - K_k H_k) P_k^- \tag{5.12}$$

The Kalman filter predicts the positional state of another vehicle running around the target vehicle. The Kalman filter reduces the error that occurs when the Frenet frame trajectory planning selects an optimal trajectory which is enabling safe and accurate local trajectory planning. In here, a process noise is assumed $N(0,1)$, Velocity noise $v = N(0,0.5)$. Then, the $z_k = H_k X_k + v$ and H is assumed $H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$.

5.3 Simulations

5.3.1 Simulation of Global Path Planning

Simulation is divided into two parts: global and local. This simulation is carried out on a city map for a larger environment near Fukuoka city in Japan. 4096×4096[pixel] taken at a height of 3[km] is selected. The actual map has an axial distance of 2.4[km] and a map size of 160,000[node] is constructed. The size of each node is 6[m]×6[m]. The size of one node was calculated based on the area of the two-lane road in Japan. The width of the lane on the Japanese road is a general two-lane road which is commonly seen, and it is approximately 3.0[m] wide per lane, 3.25[m] on the main road, and 3.5 – 3.75[m] on the highway. Red spot is a starting point and Green is a goal point. The simulation scenario is shown in 5.9 and a final path with the divided map is shown in 5.10. As shown in 5.10, the *k*-value is selected as *k*=9 when using the AS-D* lite in global path planning. Compared to the traditional D*lite algorithm, the proposed AS-D*lite algorithm shows 25% reduction of the expanded nodes and 42% reduction of the updated nodes simulated in the city map case.

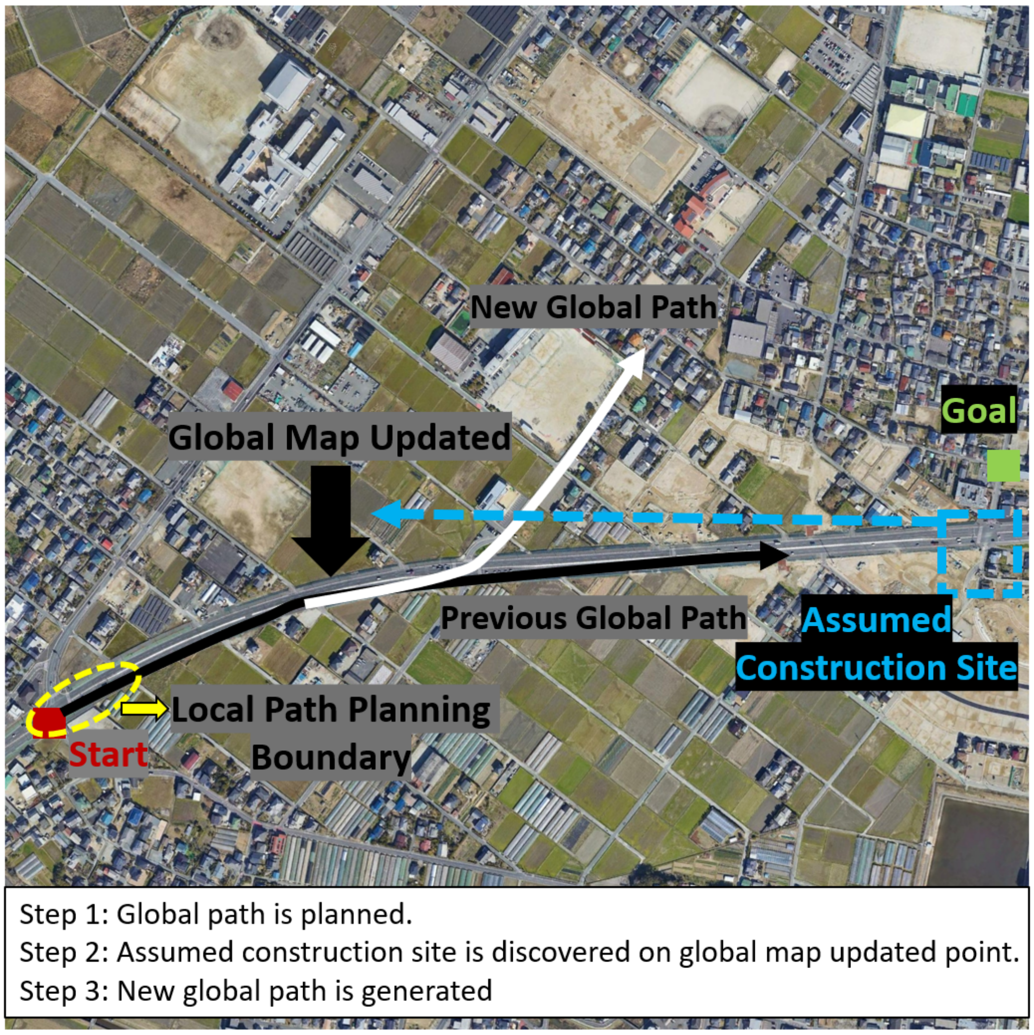

New Global Path
Global Map Updated
Goal
Previous Global Path
Assumed
Construction Site
Local Path Planning
Boundary
Start
Step 1: Global path is planned.
Step 2: Assumed construction site is discovered on global map updated point.
Step 3: New global path is generated

Fig. 5.9 Simulation Scenario

In addition, the path was changed while the vehicle was moving with planning a new path even if a part of the map was updated. The data of the planned path is designed to be sent for local path planning in real-time.

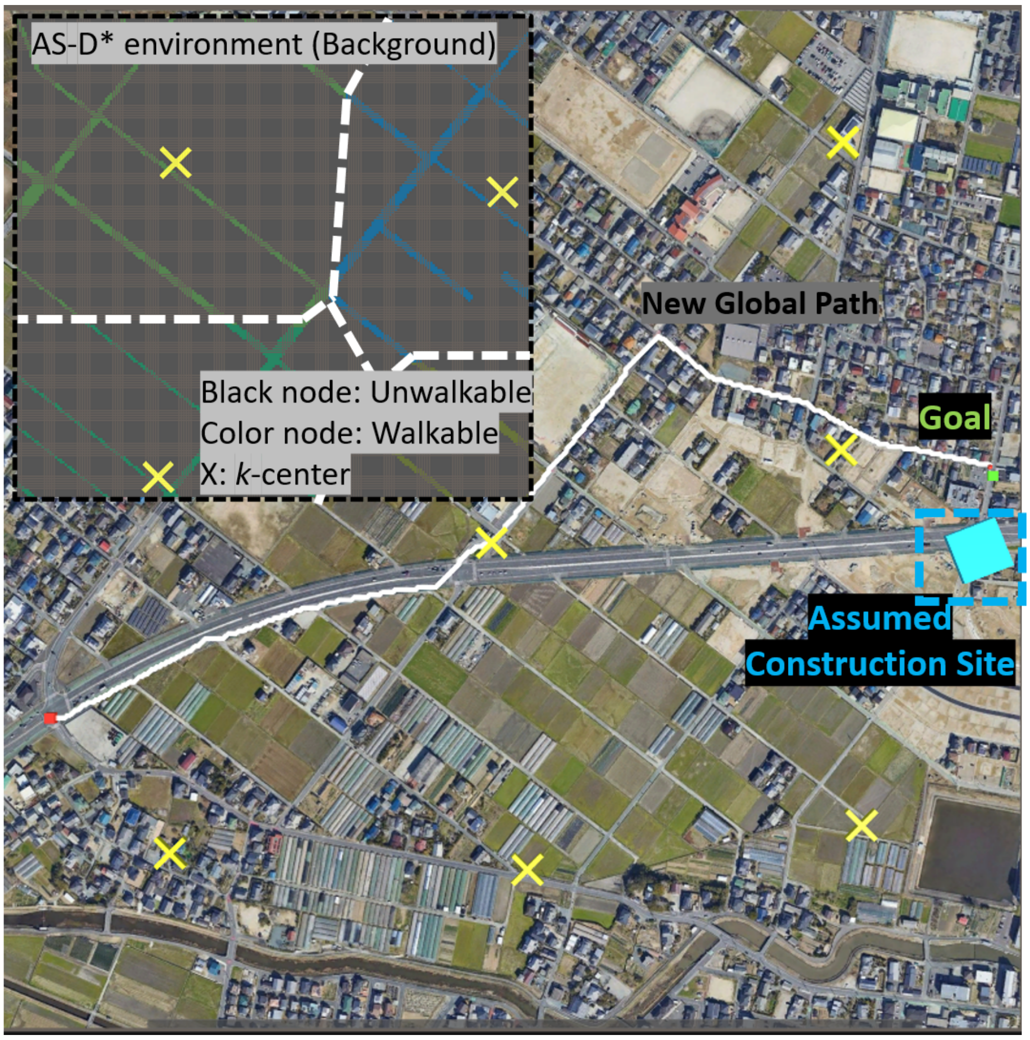
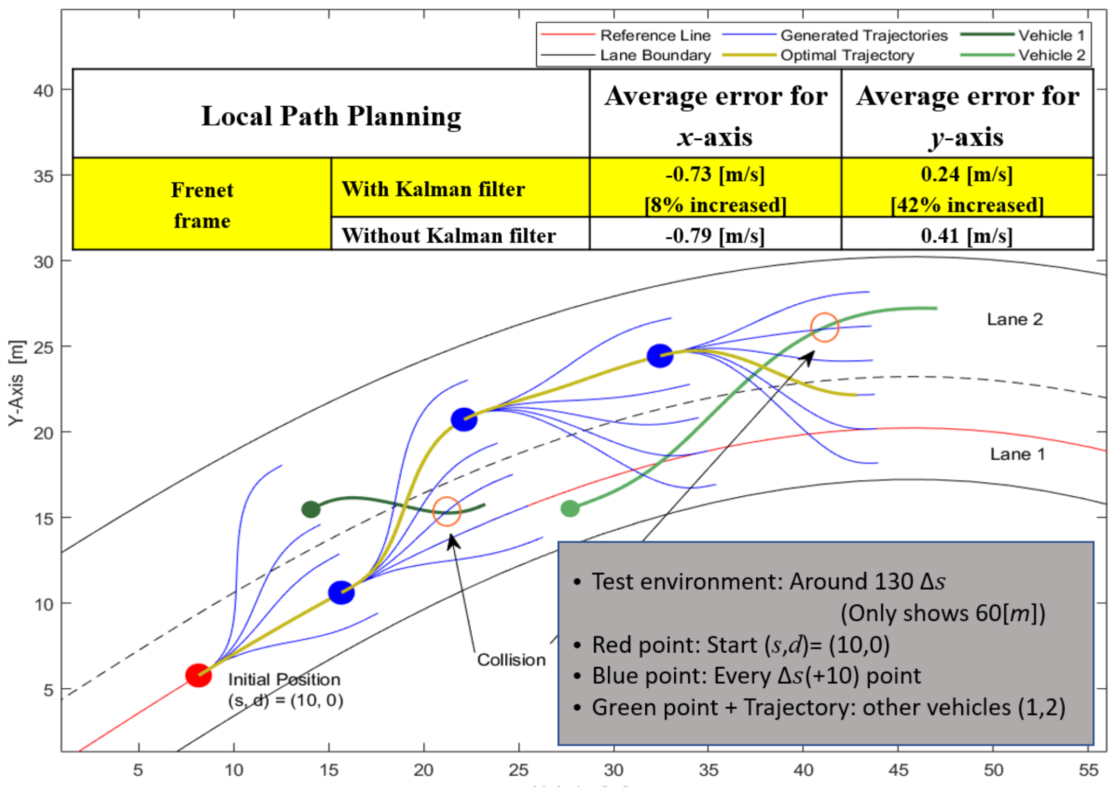


Fig. 5.10 Global Path Planning using AS-D* lite

5.3.2 Simulation of Local Trajectory Planning

The second simulation is the local trajectory planning part. The simulation shows a short trajectory of 10-20[m] in the peripheral radius of the target vehicle by Frenet frame trajectory planning using location data after global path planning (reference data). In the scenario, two nearby vehicles were set to change their course in front of the traveling direction of the target vehicle on a two-lane road. First, the target vehicle is following the global path using the AS-D* lite path. Then, vehicle 1 is interrupting the trajectory from lane 2 to lane 1. This point is named collision point 1 and vehicle 2 is escaping the trajectory from Lane 2 to Lane 1. This point is named collision point 2.



Local Path Planning		Average error for *x*-axis	Average error for *y*-axis
Frenet frame	With Kalman filter	-0.73 [m/s] [8% increased]	0.24 [m/s] [42% increased]
	Without Kalman filter	-0.79 [m/s]	0.41 [m/s]

- Test environment: Around 130 Δs (Only shows 60[m])
- Red point: Start (s,d)= (10,0)
- Blue point: Every Δs(+10) point
- Green point + Trajectory: other vehicles (1,2)

Fig. 5.11 Local Trajectory Planning using Frenet Frame

5.11 is the trajectory of the applied Frenet frame trajectory planning, and 5.1 is a comparison of accuracy. There is a significant difference in the local trajectory planning accuracy. The calculation time was almost the same in this case, because the simulation environment was simple.

Table 5.1 Local Trajectory Planning Accuracy

Local Trajectory Planning		Average error (x-axis)	Average error(y-axis)
Cartesian frame	Kalman filter	-0.87 [m/s]	-36.76 [m/s]
	Kalman filter (X)	-1.08 [m/s]	-36.75 [m/s]
Frenet frame	Kalman filter	-0.73 [m/s]	0.24 [m/s]
	Kalman filter (X)	-0.79 [m/s]	0.41 [m/s]

5.4 Conclusions

Two issues are introduced in this chapter. The first issue is that it is difficult to compute the number of candidate paths created when planning a local trajectory as a reference path for existing AS-D*lite as Cartesian frames. After re-creating the Local path in the Frenet frame, the offset path of various paths was finally regenerated. One path selected from this set of paths is called the trajectory, and it is confirmed that easily determined by the s and d values of the Frenet Frame. Additionally, the second issues for the localization is attempted to reduce the error of the d value determined by the velocity factor by finally applying the Kalman filter. Accordingly, it was confirmed that the error in the vehicle traveling direction was reduced by 8% and the error in the vehicle rotating direction was reduced by 42%. Finally, the proposed AS-D* lite solved the slow global path planning issues in the large road networks. Also, the Frenet frame with Kalman filter algorithm was realized the actual movement trajectory for the dynamic local environment, and had high localization accuracy compared to the existing methods.

Chapter 6

Conclusion

This study proposed several solutions with a method for solving the issues of the existing global path planning methods and three studies focus on solving the problems of local trajectory planning. The first research focus is to propose a new algorithm to solve the problem of global path planning in a static environment, solving problems of the existing algorithms. As an idea for this, we searched for the peripheral nodes of the current node searching in the grid map, and we proposed a diagonal path planning method that forcibly extends only four diagonal nodes instead of eight surrounding nodes. The resulting zig-zag problem reduced the smoothing time as well as the expansion speed of the node by converting the path smoothing algorithm into a quaternion space and smoothing. The algorithm used for the quaternion transformation was applied by modifying Shoemake's scheme, and finally the calculation time of global path planning in the static known area was reduced by 45% in the 51[grid]$\times$51[grid] to 49% in the 1601[grid]$\times$1601[grid].

The second research focus is to reduce expanded nodes in dynamic and partially known environments, especially in large maps. Many unnecessary node extensions occur, especially when the map is wide in the dynamic areas, and the proposed method to solve this issue is the Auto-splitting D*lite. The proposed algorithm operates only in a split map to reduce the expanded node in a wide map so that the expanded node is not re-generated even if the observation information outside the split map is updated, and an auto-clustering method with the same size is applied. As a result, it was confirmed that the calculation time decreased at a rate close to the number of divided maps, even in a dynamic environment. To prove such computational reduction rates, we also propose the Number for the Expected value of replanned node (ERPN) method that quantifies the performance of the algorithm using the expected value of the probability principle from which the expanded node occurs. Then, the proposed Auto-splitting D* lite algorithm solves the issue of the unnecessary area by ceasing the unnecessary calculation of the traditional D* lite methods and alleviating the over-calculation issue of dynamic path planners in a large area. The Auto-splitting D* lite removes nodes of the unnecessary area when a new path is updated (Updated node) in 58% on the city map and 46% on the rural map.

The third research focus was the application of local trajectory planning in a more complex dynamic and partially known environment. The main issue is the difficulty of generating candidate paths (Sets of path) in the Cartesian frame when planning a local trajectory as a reference path for existing AS-D*lite as Cartesian frames. After re-creating the Local path in the Frenet frame, the offset path of various paths was finally regenerated. One path selected from this set of paths is called the trajectory, and it is confirmed that easily determined by the s and d values of the Frenet Frame. Additionally, the second issue for the localization is attempted to reduce the error of the d value determined by the velocity factor by finally applying the Kalman filter. Accordingly, it was confirmed that the error in the vehicle traveling direction was reduced by 8% and the error in the vehicle rotating direction was reduced by 42%. Finally, the proposed AS-D* lite solved the slow global path planning issues in the large road networks. Also, the Frenet frame with the Kalman filter algorithm was realized the actual movement trajectory for the dynamic local environment.

Acknowledgements

I am honored to be able to complete my PhD at Graduate School of Information, Production and Systems, Waseda University. This thesis, like all scientific and engineering endeavours, could not have been completed without the support of many people.

Firstly, I would like to express my sincere gratitude to my advisor Prof. Hee-Hyol LEE for the continuous support of my PhD study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my PhD study.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Shigeyuki TATENO, Prof. Eiichiro TANAKA and Prof. Kenji UEDA, for their insightful comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives.

I am grateful to all the members of the System Control Laboratory. I thank my fellow labmates in for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last seven years. In particular, Mr. Dong-wook KIM, who was remembered as the first member after forming a team. Thank you for Mr. Jiahua YU, Ms. Shengyu LU and Ms. Shi GUO for co-studying the basis of team research. Since then, I would also like to thank Mr. Erxiang XU and Mr. Jiaheng CHEN, who have discussed the challenges together numerous issues. In addition, I would like to express my gratitude to many other team members who have been through a long time and those who have struggled to follow me as a team leader.

One more impartant person, I would like to thank Prof. Jang-myung LEE who is now the deceased. He was participated in various academic societies with me and gave generous advice since my master's degree.

Finally, I would like to thank the most important people in my life, my parents and family, who showed me a lot of courage, support, and generous faith. Also, I thank my girlfriend who gave me most powerfull courage whenever I had a hard time in a foreign coutnry. This thesis is dedicated to them. Thank you.

Bibliography

[1] Masato Noto and Hiroaki Sato. A method for the shortest path search by extended dijkstra algorithm. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 3, pages 2316–2320. IEEE, 2000.

[2] David R Forsey and Richard H Bartels. Hierarchical b-spline refinement. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 205–212, 1988.

[3] Sky McKinley and Megan Levine. Cubic spline interpolation. *College of the Redwoods*, 45(1):1049–1060, 1998.

[4] Gerald Farin, Gerhard Rein, Nikolas Sapidis, and Andrew J Worsey. Fairing cubic b-spline curves. *Computer Aided Geometric Design*, 4(1-2):91–103, 1987.

[5] Larry Schumaker. *Spline functions: basic theory*. Cambridge University Press, 2007.

[6] Erik Meijering. A chronology of interpolation: from ancient astronomy to modern signal and image processing. *Proceedings of the IEEE*, 90(3):319–342, 2002.

[7] Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, interpolation and animation*, volume 2. Citeseer, 1998.

[8] Carl De Boor and Carl De Boor. *A practical guide to splines*, volume 27. springer-verlag New York, 1978.

[9] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[10] Anthony Stentz. Optimal and efficient path planning for unknown and dynamic environments. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1993.

[11] Sven Koenig and Maxim Likhachev. D^* lite. *Aaai/iaai*, 15, 2002.

[12] Joanne Pransky. Geoff howe, senior vice president, howe and howe, inc., a subsidiary of textron systems; co-pioneer of robotic firefighting technologies, including thermite™ firefighting robots. *Industrial Robot: the international journal of robotics research and application*, 2021.

[13] Robert Bogue. The role of robots in firefighting. *Industrial Robot: the international journal of robotics research and application*, 2021.

[14] Marcel Ceelen, CJ van der Geer, Floris Rouwen, and Stijn Seuren. Fundamentals for an autonomous zebro swarm. 2015.

[15] Gabe Nelson, Aaron Saunders, and Robert Playter. The petman and atlas robots at boston dynamics. *Humanoid Robotics: A Reference*, 169:186, 2019.

[16] Sara Abdallaoui, El-Hassane Aglzim, Ahmed Chaibet, and Ali Kribèche. Thorough review analysis of safe control of autonomous vehicles: Path planning and navigation techniques. *Energies*, 15(4):1358, 2022.

[17] Lin Zhang, Yingjie Zhang, and Yangfan Li. Path planning for indoor mobile robot based on deep learning. *Optik*, 219:165096, 2020.

[18] Puyong Xu, Ning Wang, Shi-Lu Dai, and Lei Zuo. Motion planning for mobile robot with modified bit* and mpc. *Applied Sciences*, 11(1):426, 2021.

[19] Wenda Xu, Jia Pan, Junqing Wei, and John M Dolan. Motion planning under uncertainty for on-road autonomous driving. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2507–2512. IEEE, 2014.

[20] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, 2016, 2016.

[21] Wang Shufeng and Zhang Junxin. A research on overtaking lane planning for intelligent vehicles based on improved artificial potential field method. *Automobile Technology*, 3(2):2–7, 2018.

[22] Li-sang Liu, Jia-feng Lin, Jin-xin Yao, Dong-wei He, Ji-shi Zheng, Jing Huang, and Peng Shi. Path planning for smart car based on dijkstra algorithm and dynamic window approach. *Wireless Communications and Mobile Computing*, 2021, 2021.

[23] Cui Zhi and Shi Xiumin. Research on path planning of mobile robot based on a* algorithm.

[24] Sven Koenig and Maxim Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings 2002 IEEE international conference on robotics and automation (Cat. No. 02CH37292)*, volume 1, pages 968–975. IEEE, 2002.

[25] Alex Nash, Sven Koenig, and Craig Tovey. Lazy theta*: Any-angle path planning and path length analysis in 3d. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 147–154, 2010.

[26] Kairong Li, Qianqian Hu, and Jinpeng Liu. Path planning of mobile robot based on improved multiobjective genetic algorithm. *Wireless Communications and Mobile Computing*, 2021, 2021.

[27] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. A hierarchical global path planning approach for mobile robots based on multi-objective particle swarm optimization. *Applied Soft Computing*, 59:68–76, 2017.

[28] Xun Li, Dandan Wu, Jingjing He, Muhammad Bashir, and Ma Liping. An improved method of particle swarm optimization for path planning of mobile robot. *Journal of Control Science and Engineering*, 2020, 2020.

[29] BK Patle, Anish Pandey, DRK Parhi, A Jagadeesh, et al. A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, 15(4):582–606, 2019.

[30] Chengke Xiong, Danfeng Chen, Di Lu, Zheng Zeng, and Lian Lian. Path planning of multiple autonomous marine vehicles for adaptive sampling using voronoi-based ant colony optimization. *Robotics and Autonomous Systems*, 115:90–103, 2019.

[31] Khaled Akka and Farid Khaber. Mobile robot path planning using an improved ant colony optimization. *International Journal of Advanced Robotic Systems*, 15(3):1729881418774673, 2018.

[32] Zexuan Zhu, Fangxiao Wang, Shan He, and Yiwen Sun. Global path planning of mobile robots using a memetic algorithm. *International Journal of Systems Science*, 46(11):1982–1993, 2015.

[33] Jing Xin, Huan Zhao, Ding Liu, and Minqi Li. Application of deep reinforcement learning in mobile robot path planning. In *2017 Chinese Automation Congress (CAC)*, pages 7112–7116. IEEE, 2017.

[34] Muhammad Umer KHAN. Mobile robot navigation using reinforcement learning in unknown environments. *Balkan Journal of Electrical and Computer Engineering*, 7(3):235–244, 2019.

[35] Feihu Zhang, Can Wang, Chensheng Cheng, Dianyu Yang, and Guang Pan. Reinforcement learning path planning method with error estimation. *Energies*, 15(1):247, 2021.

[36] Hartmut Surmann, Christian Jestel, Robin Marchel, Franziska Musberg, Houssem Elhadj, and Mahbube Ardani. Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments. *arXiv preprint arXiv:2005.13857*, 2020.

[37] Alejandro Hidalgo-Paniagua, Miguel A Vega-Rodríguez, Joaquín Ferruz, and Nieves Pavón. Mosfla-mrpp: multi-objective shuffled frog-leaping algorithm applied to mobile robot path planning. *Engineering Applications of Artificial Intelligence*, 44:123–136, 2015.

[38] Dimia Iberraken, Lounis Adouane, and Dieumet Denis. Reliable risk management for autonomous vehicles based on sequential bayesian decision networks and dynamic inter-vehicular assessment. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2344–2351. IEEE, 2019.

[39] BK Patle, Alok Jha, Anish Pandey, N Gudadhe, and SK Kashyap. The optimized path for a mobile robot using fuzzy decision function. *Materials Today: Proceedings*, 18:3575–3581, 2019.

[40] Daniel Ioan, Ionela Prodan, Sorin Olaru, Florin Stoican, and Silviu-Iulian Niculescu. Mixed-integer programming in motion planning. *Annual Reviews in Control*, 51:65–87, 2021.

[41] Xuemin Hu, Long Chen, Bo Tang, Dongpu Cao, and Haibo He. Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles. *Mechanical systems and signal processing*, 100:482–500, 2018.

[42] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[43] Xiaoxun Sun, Sven Koenig, and William Yeoh. Generalized adaptive a. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 469–476, 2008.

[44] Zhiyong Wang and Sisi Zlatanova. Multi-agent based path planning for first responders among moving obstacles. *Computers, Environment and Urban Systems*, 56:48–58, 2016.

[45] Zhiyong Wang, Sisi Zlatanova, and Peter van Oosterom. Path planning for first responders in the presence of moving obstacles with uncertain boundaries. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):2163–2173, 2017.

[46] Jeffrey Delmerico, Elias Mueggler, Julia Nitsch, and Davide Scaramuzza. Active autonomous aerial exploration for ground robot path planning. *IEEE Robotics and Automation Letters*, 2(2):664–671, 2017.

[47] Maxim Likhachev and Sven Koenig. Lifelong planning a* and dynamic a* lite: The proofs. In *Technical report*. 2001.

[48] Xiaoxun Sun, William Yeoh, and Sven Koenig. Moving target d* lite. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 67–74, 2010.

[49] Dave Ferguson and Anthony Stentz. Using interpolation to improve path planning: The field d* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2006.

[50] Joseph Carsten, Dave Ferguson, and Anthony Stentz. 3d field d: Improved path planning and replanning in three dimensions. In *2006 IEEE/RSJ international conference on intelligent robots and systems*, pages 3381–3386. IEEE, 2006.

[51] J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964.

[52] I Smith. Making successful drone maps: Photogrammetry is more art than science. *drone deploy*, 2017.

[53] C Strecha, R Zoller, S Rutishauser, B Brot, K Schneider-Zapp, V Chovancova, M Krull, and L Glassey. Quality assessment of 3d reconstruction using fisheye and perspective sensors. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(3):215, 2015.

[54] Kwangjin Yang and Salah Sukkarieh. An analytical continuous-curvature path-smoothing algorithm. *IEEE Transactions on Robotics*, 26(3):561–568, 2010.

[55] Mohamed Elbanhawi, Milan Simic, and Reza N Jazar. Continuous path smoothing for car-like robots using b-spline curves. *Journal of Intelligent & Robotic Systems*, 80(1):23–56, 2015.

[56] Kwangjin Yang and Salah Sukkarieh. 3d smooth path planning for a uav in cluttered natural environments. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 794–800. IEEE, 2008.

[57] Aurelio Piazzi, C Guarino Lo Bianco, and Massimo Romano. Smooth path generation for wheeled mobile robots using η3-splines. *Motion Control*, pages 75–96, 2010.

[58] Verena Elisabeth Kremer. Quaternions and slerp. *Embots. dfki. de/doc/seminar_ca/Kremer_Quaternions. pdf*, 2008.

[59] John C Hart, George K Francis, and Louis H Kauffman. Visualizing quaternion rotation. *ACM Transactions on Graphics (TOG)*, 13(3):256–276, 1994.

[60] William Rowan Hamilton. Xi. on quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 33(219):58–60, 1848.

[61] Richard E Korf, Michael Reid, and Stefan Edelkamp. Time complexity of iterative-deepening-a*. *Artificial Intelligence*, 129(1-2):199–218, 2001.

[62] Michael Barbehenn. A note on the complexity of dijkstra's algorithm for graphs with weighted vertices. *IEEE transactions on computers*, 47(2):263, 1998.

[63] Alexander Bock, Åsa Svensson, Alexander Kleiner, Jonas Lundberg, and Timo Ropinski. A visualization-based analysis system for urban search & rescue mission planning support. In *Computer Graphics Forum*, volume 36, pages 148–159. Wiley Online Library, 2017.

[64] Ashish Macwan, Julio Vilela, Goldie Nejat, and Beno Benhabib. A multirobot path-planning strategy for autonomous wilderness search and rescue. *IEEE transactions on cybernetics*, 45(9):1784–1797, 2014.

[65] Luca De Filippis, Giorgio Guglieri, and Fulvia Quagliotti. Path planning strategies for uavs in 3d environments. *Journal of Intelligent & Robotic Systems*, 65(1):247–264, 2012.

[66] Pavan Kumar Chitumalla, Douglas Harris, Bhavani Thuraisingham, and Latifur Khan. Emergency response applications: Dynamic plume modeling and real-time routing. *IEEE Internet Computing*, 12(1):38–44, 2008.

[67] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[68] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[69] Xunyu Zhong, Jun Tian, Huosheng Hu, and Xiafu Peng. Hybrid path planning based on safe a* algorithm and adaptive window approach for mobile robot in large-scale dynamic environment. *Journal of Intelligent & Robotic Systems*, 99(1):65–77, 2020.

[70] Rüdiger Ebendt and Rolf Drechsler. Weighted a* search–unifying view and application. *Artificial Intelligence*, 173(14):1310–1342, 2009.

[71] Xiaowu Liu, Riqiang Deng, Jinwen Wang, and Xunzhang Wang. Costar: A d-star lite-based dynamic search algorithm for codon optimization. *Journal of theoretical biology*, 344:19–30, 2014.

[72] Christopher Makoto Wilt and Wheeler Ruml. When does weighted a* fail? In *SOCS*, pages 137–144, 2012.

[73] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[74] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483. IEEE, 2011.

[75] Iram Noreen, Amna Khan, and Zulfiqar Habib. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. *International Journal of Computer Science and Network Security (IJCSNS)*, 16(10):20, 2016.

[76] Jiankun Wang, Wenzheng Chi, Chenming Li, Chaoqun Wang, and Max Q-H Meng. Neural rrt*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, 17(4):1748–1758, 2020.

[77] Yong Koo Hwang, Narendra Ahuja, et al. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.

[78] Jerome Barraquand, Bruno Langlois, and J-C Latombe. Numerical potential field techniques for robot path planning. *IEEE transactions on systems, man, and cybernetics*, 22(2):224–241, 1992.

[79] Yong-bo Chen, Guan-chen Luo, Yue-song Mei, Jian-qiao Yu, and Xiao-long Su. Uav path planning using artificial potential field method updated by optimal control theory. *International Journal of Systems Science*, 47(6):1407–1420, 2016.

[80] Baochang Zhang, Zhili Mao, Wanquan Liu, and Jianzhuang Liu. Geometric reinforcement learning for path planning of uavs. *Journal of Intelligent & Robotic Systems*, 77(2):391–409, 2015.

[81] Ankit A Ravankar, Abhijeet Ravankar, Takanori Emaru, and Yukinori Kobayashi. Hpprm: Hybrid potential based probabilistic roadmap algorithm for improved dynamic path planning of mobile robots. *IEEE Access*, 8:221743–221766, 2020.

[82] Manh Duong Phung, Cong Hoang Quach, Tran Hiep Dinh, and Quang Ha. Enhanced discrete particle swarm optimization path planning for uav vision-based surface inspection. *Automation in Construction*, 81:25–33, 2017.

[83] PK Das and Prabir Kumar Jena. Multi-robot path planning using improved particle swarm optimization algorithm through novel evolutionary operators. *Applied Soft Computing*, 92:106312, 2020.

[84] Zheng Wang, Guangfu Li, and Jia Ren. Dynamic path planning for unmanned surface vehicle in complex offshore areas based on hybrid algorithm. *Computer Communications*, 166:49–56, 2021.

[85] Yang Peng, Sun-Wei Li, and Zhen-Zhong Hu. A self-learning dynamic path planning method for evacuation in large public buildings based on neural networks. *Neurocomputing*, 365:71–85, 2019.

[86] Ryo Yonetani, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, and Asako Kanezaki. Path planning using neural a* search. In *International Conference on Machine Learning*, pages 12029–12039. PMLR, 2021.

[87] Aleksandr I Panov, Konstantin S Yakovlev, and Roman Suvorov. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia computer science*, 123:347–353, 2018.

[88] Daniel Drake, Scott Koziol, and Eugene Chabot. Mobile robot path planning with a moving goal. *IEEE Access*, 6:12800–12814, 2018.

[89] Pi-Ying Cheng and Pin-Jyun Chen. Navigation of mobile robot by using d++ algorithm. *Intelligent Service Robotics*, 5(4):229–243, 2012.

[90] Jur Van Den Berg, Dave Ferguson, and James Kuffner. Anytime path planning and replanning in dynamic environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2366–2371. IEEE, 2006.

[91] Maciej Przybylski and Barbara Putz. D* extra lite: a dynamic a* with search-tree cutting and frontier-gap repairing. *International Journal of Applied Mathematics and Computer Science*, 27, 2017.

[92] Purnima Bholowalia and Arvind Kumar. Ebk-means: A clustering technique based on elbow method and k-means in wsn. *International Journal of Computer Applications*, 105(9), 2014.

[93] Gereon Frahling and Christian Sohler. A fast k-means implementation using coresets. *International Journal of Computational Geometry & Applications*, 18(06):605–625, 2008.

[94] Omur Arslan, Dan P Guralnik, and Daniel E Koditschek. Coordinated robot navigation via hierarchical clustering. *IEEE Transactions on Robotics*, 32(2):352–371, 2016.

[95] JAS Almeida, LMS Barbosa, AACC Pais, and SJ Formosinho. Improving hierarchical cluster analysis: A new method with outlier detection and automatic clustering. *Chemometrics and Intelligent Laboratory Systems*, 87(2):208–217, 2007.

[96] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[97] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.

[98] Matus Telgarsky and Andrea Vattani. Hartigan's method: k-means clustering without voronoi. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 820–827. JMLR Workshop and Conference Proceedings, 2010.

[99] J Senthilnath, Deepak Kumar, JA Benediktsson, and Xiaoyang Zhang. A novel hierarchical clustering technique based on splitting and merging. *International Journal of Image and Data Fusion*, 7(1):19–41, 2016.

[100] Saket Anand, Sushil Mittal, Oncel Tuzel, and Peter Meer. Semi-supervised kernel mean shift clustering. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1201–1215, 2013.

[101] Damodar Reddy, Prasanta K Jana, and IEEE Senior Member. Initialization for k-means clustering using voronoi diagram. *Procedia Technology*, 4:395–400, 2012.

[102] Hong Bo Zhou and Jun Tao Gao. Automatic method for determining cluster number based on silhouette coefficient. In *Advanced Materials Research*, volume 951, pages 227–230. Trans Tech Publ, 2014.

[103] Trupti M Kodinariya and Prashant R Makwana. Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95, 2013.

[104] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[105] Martin Erwig. The graph voronoi diagram with applications. *Networks: An International Journal*, 36(3):156–163, 2000.

[106] Adam Niewola and Leszek Podsedkowski. L* algorithm—a linear computational complexity graph searching algorithm for path planning. *Journal of Intelligent & Robotic Systems*, 91(3):425–444, 2018.

[107] Scott Beamer, Krste Asanovic, and David Patterson. Direction-optimizing breadth-first search. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–10. IEEE, 2012.

[108] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[109] Anil Chaturvedi, Paul E Green, and J Douglas Caroll. K-modes clustering. *Journal of classification*, 18(1):35–55, 2001.

[110] Preeti Arora, Shipra Varshney, et al. Analysis of k-means and k-medoids algorithm for big data. *Procedia Computer Science*, 78:507–512, 2016.

[111] Dorin Comaniciu and Peter Meer. Mean shift analysis and applications. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1197–1203. IEEE, 1999.

[112] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

List of Publications

Journal Papers (Peer Reviewed)

- 1. Shin-nyeong Heo. Jiaheng Chen, Yu-chi Liao and Hee-hyol Lee. “Auto-splitting D* lite path planning for large disaster area.” Intelligent Service Robotics, Vol.15, Issues 3, pp. 289-306 (2022).
  - https://doi.org/10.1007/s11370-022-00416-8,*Hybrid (Transformative) Journal: Traditional Access & Open Access. Published online March 2022.
  - SCIE/SSCI Impact factor: 2.246 (2020)
  - Scopus Impact Factor: 3.149 (2021)
  - Scopus Quartile: Top Q1(25%) in Computational Mechanics/Engineering (miscellaneous)
  - 1861-2784 ISSN 1861-2776 ISSN Print

- 2. Heo, Shin Nyeong, Erxiang Xu, and Hee-Hyol Lee. "Diagonal Node-based Path Planning and Path Smoothing for First Responders and Rescue Robots." IEEJ Transactions on Electronics, Information and Systems, Vol.141, No.2, pp.179-192. February 2021.

International Conference Papers (Peer Reviewed)

- 1. Shin-nyeong Heo, Huimin Sun, Ji-sun Shin and Hee-hyol Lee. “Auto-splitting D* lite with Frenet Frame Path Planning for Large Road Network.” The Twenty-Seventh International Symposium on Artificial Life and Robotics 2022 (AROB 27th 2022), January 25-27, 2022. (Proceeding)

- 2. Shin-nyeong Heo, Jiheng Chen, Youngdal O, Ji-sun Shin and Hee-hyol Lee. “Dynamic Weighted Splitting D* lite for Rescue Robot Path Planning in Large Fire Areas.”, The Twenty-Sixth International Symposium on Artificial Life and Robotics 2021 (AROB 26th 2021), January 21-23, 2021. (Proceeding)
- 3. Jiaheng Chen, Shin-nyeong Heo and Hee-hyol Lee. “Auto-Splitting using k-means with Voronoi Diagram to Divide Large Mapfor Splitting D* Lite.” The Twenty-Sixth International Symposium on Artificial Life and Robotics 2021 (AROB 26th 2021), January 21-23, 2021. (Proceeding)
- 4. Youngdal Oh, Shin-nyeong Heo, Sunhong Park, Jinhae Yae, Sujin Baek, and Hee-hyol Lee. “Real-time Learning Detection of Drowsy Driving using Bus Driver.” The Twenty-Sixth International Symposium on Artificial Life and Robotics 2021 (AROB 26th 2021), January 21-23, 2021. (Proceeding)
- 5. Po-wen Cheng, Shin-nyeong Heo, Xingyu Zhang and Hee-hyol Lee. “PID Neural Network Control of Quadrotor with Path Tracking.” The Twenty-Fifth International Symposium on Artificial Life and Robotics 2020 (AROB 25th 2020), January 22-24, 2020. (Proceeding)
- 6. Shin nyeong Heo, Ji-Sun Shin, and Hee-hyol Lee. “Realistic looking path on Grid-node in Disaster area” The Twenty-Fifth International Symposium on Artificial Life and Robotics 2020 (AROB 25th 2020), January 22-24, 2020. (Proceeding)
- 7. Erxiang Xu, Shin Nyeong Heo, and Hee-Hyol Lee. “Drone-based Parking Space Detection and Path Planning for Vehicles” The Twenty-Fifth International Symposium on Artificial Life and Robotics 2020 (AROB 25th 2020), January 22-24, 2020. (Proceeding)
- 8. Shin nyeong Heo, Shen yu Lu, Shi Guo and Hee-hyol Lee, “Diagonal Path planning for Rescuing Robot and First Responders”, The Twenty-Fourth International Symposium on Artificial Life and Robotics 2019, January 23-25, 2019. (Proceeding)
- 9. Shengyu Lu, Shin nyeong Heo and Hee-hoyl Lee “Optimal path planning with heap optimization and least turn for multiple rescue robots”, The Twenty-Fourth International Symposium on Artificial Life and Robotics 2019, January 23-25, 2019. (Proceeding)

- 10. Guo Shi, Shin nyeong Heo and Hee-hyol Lee “Different Bounded Curvature Methods for Diagonal Path Planning”, The Twenty-Fourth International Symposium on Artificial Life and Robotics 2019, January 23-25, 2019. (Proceeding)

- 11. Shin-nyeong Heo, Jiahua Yu, Ji-Sun Shin, and Hee-hyol Lee, “Improved 3D Node based Optimal Algorithm with Position Estimation for First Responder”, The Twenty-Third International Symposium on Artificial Life and Robotics 2018, pp 606-611, Japan, January, 2018. (Proceeding)

- 12. Jiahua Yu, Shin nyeong Heo, Ji-sun Shin and Hee-hyol Lee “Improved Node Based Optimal Path Planning Algorithm with Box Blur Method”, The Twenty-Third International Symposium on Artificial Life and Robotics 2018, pp 618-621, Japan, January, 2018. (Proceeding)

- 13. Shin-nyeong Heo, Hee-hyol Lee “Path planning in Unknown Terrain using D* lite and Localization with Bayes Theorem combining Kalman Filter”, 22nd International Symposium on Artificial Life and Robotics, pp669-674, Japan, January, 2017. (Proceeding)

- 14. Shin-nyeong Heo, Hee-hyol Lee “Multiple GPS Localization using Bayes Theorem and Kalman Filter”, 21st International Symposium on Artificial Life and Robotics 2016, pp.555-559, Japan, January, 2016. (Proceeding)

International Conference Papers (Oral)

- 1. Shin nyeong Heo, Shi Guo, Ji-sun Shin and Hee-hyol Lee, “Diagonal A* Path planning with Position Estimation”, International Conference on ICT Robotics(ICT-ROBOT), Korea, September, 2018.

- 2. Shin nyeong Heo, Sheng yu Lu, Ji-sun Shin and Hee-hyol Lee, “Multi-Robot-Multi-Target Path Planning and Position Estimation for Disater area”, International Conference on ICT Robotics(ICT-ROBOT), Korea, September, 2018.

- 3. Jiahua Yu, Shin nyeong Heo, Ji-sun Shin and Hee-hyol Lee “Fire Area Detection based on Convolutional Neural Network and Improved A* Path Planning”, International Conference on

ICT Robotics(ICT-ROBOT), Korea, September, 2018.

- 4. Shin-nyeong Heo and Hee-hyol Lee. “Path Planning of Moving Object on Off-Road Environment with Obstacle Avoidance” International Conference on ICT Robotics(ICT-ROBOT), Korea, September, 2016.
- 5. Keon-woo Jeong, Shin-nyeong Heo, Seung-ik Hwang, Han-dong Yoo and Jangmyung Lee, International Conference on Artificial Life and Robotics (ICAROB), “Design of Fuzzy Controller using variable Fuzzy membership factors for Inverse ball drive Mobile Robot”, Japan, January, 2014.
- 6. Shin-nyeong Heo, Hyun-Seop Lim, Seungik Hwang, Jangmyung Lee, “"Object Following Robot Using Vision Camera, Single Curvature Trajectory and Kalman Filters." International Conference on Intelligent Robotics and Applications. Springer, Berlin, Heidelberg, 2013, Proceedings, Part I, Lecture Notes in Computer Science, 2013, pp.562-575, ISSN : 0302-9743.

Symposium and Domestic Journal

- 1. Feng Qiushi, Shin-nyeong Heo and Hee-hyol Lee, “An improved drowsy driving detection method by removing abnormality using iForest with DBSCAN” 15h International collaboration Symposium on Information, Production and Systems (ISIPS), Japan, 2021.
- 2. Shin-Nyeong Heo and Hee-Hyol Lee, “Splitting D* lite Path Planning in Wide Fire Area for Rescue Robots.” 14h International collaboration Symposium on Information, Production and Systems (ISIPS), Japan, 2020.
- 3. Shin Nyeong Heo, Shi Guo, Sehngyu Lu and Hee-Hyol Lee, “Improved A* path planning using new fire area detection based on YOLOv3”, 12th International collaboration Symposium on Information, Production and Systems (ISIPS), Japan, 2018.
- 4. Shengyu Lu, Shin Nyeong Heo and Hee-Hyol Lee, “A fast multi-robot path planning using heap optimization for disaster area”, 12th International collaboration Symposium on Information, Production and Systems (ISIPS), Japan, 2018.

- 5. Shi Guo, Shin Nyeong Heo and Hee-Hyol Lee, “Diagonal A* Path Planning with Bounded Curvature”, 12th International collaboration Symposium on Information, Production and Systems (ISIPS), Japan, 2018.
- 6. Shin nyeong Heo, Jiahua Yu and Hee hyol Lee, “3D path planning method with position estimation for moving target search in partially known terrain” Workshop on Artificial life and Robotics (AROB Workshop 2017), Korea, September, 2017.
- 7. Shin-nyeong Heo and Jang-myung Lee, "Predictive Control of an Efficient Human Following Robot Using Kinect Sensor" , Journal of Institute of Control, Vol.20(9), pp.957-963, September, 2014 (Domestic Journal of Korea)