Online Path Planning Framework for UAV in Rural Areas

GREGORIUS Airlangga\textsuperscript{1,2} AND ALAN LIU\textsuperscript{2}, (Member, IEEE)

\textsuperscript{1}Information System Department, Atma Jaya Catholic University of Indonesia, Jakarta 12930, Indonesia
\textsuperscript{2}Electrical Engineering Department, National Chung Cheng University, Chiayi 621301, Taiwan

Corresponding author: Gregorius Airlangga (gregorius.airlangga@atmajaya.ac.id)

This work was supported in part by Atma Jaya Catholic University of Indonesia, and in part by MOST, Taiwan.

\textbf{ABSTRACT} A motion strategy plays an important role in supporting autonomous Unmanned Aerial Vehicle (UAV) movement. Many studies have been conducted to improve the motion frameworks in terms of its robustness, safety and performance. Most of them worked on the prior known maps scenario where the area information was collected by Global Positioning System (GPS) and satellite cameras. Even though the scheme can provide high quality map, the computation of motion planning remains dependent on the communication signal. In the rural areas such as forests and mountains, where communication signal does not perform well, unclear and noisy terrain maps can be generated and lead to mission failure. Therefore, it is significant that an alternative framework to enhance autonomous UAV motion performance in these certain conditions should be developed. Our work focuses on developing a high performance path planner for autonomous UAV motion when communication signal does not work well in rural areas. The search mission problem in forest terrain has been implemented in 3D simulation as an evaluation. By conducting a simulation process repeatedly with different test cases for positions, time constraints, flight speed (3-11 m/s) and flight range, our path planning framework can achieve completeness between 90-100\% and better performance compared to others.

\textbf{INDEX TERMS} Online path planning, motion planning, UAV, rural areas.

\section{I. INTRODUCTION}

In recent years, the Unmanned Aerial Vehicle (UAV) technology has received more attention in the community due to its potential and reasonable prices. There are four physical UAV variants: multi-rotor, fixed wings, single-rotor helicopter and fixed-wing hybrid Vertical Take Off and Landing (VTOL) [1]. The different body shapes of UAVs contribute to the payload, battery life and computational abilities. This also indirectly affects the mission capability [2], [3]. The more complex a UAV’s shape is, the more complex the mission can become. Path planning plays the central role in mission accomplishment on all UAV variants [4]. The strategy to establish high performance and safety attributes is mandatory in the UAV world for any situation and circumstances. Therefore, many studies were conducted in two main categories: online and offline strategies [5].

An online strategy tends to be sensitive to environmental changes regarding the dynamic obstacles, terrain and weather. The generated path is decided in reactive manners [6] and deliberative manners [7]. A reactive approach is the automatic path generator extracted rapidly according to the real-time obstacle detector. By utilizing the data of obstacle position, the mechanism to do obstacle avoidance can be achieved immediately as far as the obstacles are located within detector range. On the other hand, a deliberative method can work as an automatic path generator by considering several factors such as safety, energy saving, the shortest path, etc. Commonly, all variables, maps and path candidates are identified before the UAV takes off.

Using this mechanism, a UAV can perform obstacle avoidance if the obstacle’s position and supported variables are not different from the last known data. When facing a dynamic obstacle problem, an updated path mechanism is implemented by collecting real-time variables and environment maps, and this process is time-consuming due to...
preprocessing delay and hardware constraints. In addition, if the preprocessing is not stable due to the Internet connection and hardware fault, then the UAV may fail to avoid the obstacles. Unlike deliberative strategies, a reactive online path planner can fully work without Internet connection and prior map knowledge. However, consequently, it takes heavy local computation process to map the environment and the generated path planner is less accurate. [8].

An offline strategy is hard to adapt in a dynamic environment, and all generated paths are usually predetermined by a third-party platform [9]. Primarily, an offline strategy is implemented by allowing human operators to monitor UAV movement. If there is an environment change, the human operators control the UAV manually or generate a new path using the automatic path planning feature [10]. This strategy can be implemented by using a GPS and cloud computing [11]. However, in the rural areas where the network signal is not stable and available, UAV cannot depend on external platforms such as the cloud, telecommunication networks and other UAVs to accomplish a mission task [12]. Therefore, the online strategy is chosen over the offline strategy to generate vehicle paths in rural areas.

Several online path planners that are commonly used include Neural Network (NN), Stochastic models, Probabilistic models and Graph. NN-based approaches are chosen due to their hardware advancement and big data availability such as an improved Q-learning based method for reactive obstacle avoidance [13], a deep reinforcement learning approach in assisted edge computing networks [14], and a deep reinforcement learning technique to calculate online path planning [15]. Those papers claimed that generated online path is smooth, fast and fuel efficient. However, from the practical point of view, the implementation is tedious in collecting data and preprocessing. It may delay the engineering process to deploy and the approach cannot guarantee a similar result in a different map.

Stochastic approaches may include work like [16]–[19]. The main drawback of this kind of approaches is that it needs appropriate parameter tuning to guarantee path quality. In addition, the process to achieve optimal conditions can be endless due to the enormous local optima trap within the discretized map, and this condition will consume a heavy computation resource and unstable UAV movement.

Probabilistic and graph based techniques are also considered as UAV motion strategies. Even though both methods provide a reasonable result, the comparison study has reported that a graph-based approach offers more optimal path length with high performance [20]. Graph-based approaches have developed with many variants. However, the computation time and space complexity will increase along with the increasing of graph dimension. In the real environment scenario where the generated graph can be large and the goal coordinate is far from the current UAV position, the path calculation can be slow, leading to unstable UAV motion and failing to achieve the mission. In addition, most experiments are implemented in situations where the obstacles are not in the dense formation such as forest, mountain, farm, ocean, etc. and have not considered UAV velocity in terms of evaluation and comparison measurement. In several missions such as search and rescue in rural areas, completeness and velocity are essential in the mission achievement [21]. Therefore, further studies need to be conducted to examine and understand the effect of path planning and velocity performance.

In this paper, we propose the Online Motion Framework that focuses on rural areas with several problems such as lost-link, battery and resource limitation, time mission limitation, unstable Internet connectivity and uncertain obstacles. The proposed framework consists of different sensors, computation platforms, and associated data. The framework can host different path planning algorithms and provide test cases to conduct experiments in simulation as presented on Fig. 1.

Our approach focuses on the performance and robustness quality attributes. There are three main tasks of UAV motion framework, first is sensing that is executed by utilizing LIDAR to gather information about the environment. Due to energy-saving cost and performance, we prioritize the use of LIDAR to collect terrain information within a certain range per microsecond instead of a camera that dominantly needs heavy computation in the cloud. Secondly, we build a rapid and lightweight graph map generator containing information of obstacles, free paths, distance and partial goal estimation.

Lastly, by utilizing the microkernel approach, the framework can integrate any path planning algorithms as long as the path planning only needs goal coordinate as an input parameter and generates a high quality path instantly to reduce heavy computation. We have modified the A* search as one of the graph-based variants. We show our framework’s robustness by using simulation and measuring path planning performance quality in terms of time and space factors. Our paper consists of seven sections. Section I is the introduction, and Section II discusses the related work and the state of the art. Section III describes the problem formulation while Section IV presents our framework and the modified A* algorithm. Section V gives the explanation of experiment design, Section VI discusses the experiment results and analysis focusing on performance and completeness quality attributes, and the last section provides the conclusion and future work.

The summary of contributions is given below:

- Generating a rapid and lightweight informative graph maps using LIDAR to collect a real-time information of the environment,
- Developing an end-to-end pipeline for autonomous UAV motion from sensing to landing,
- Proposing a modified A* algorithm, partial goal estimation and automatic pattern extraction methods to improve performance and robustness in terms of time and space measurement,
- Building a microkernel-based module to implement an automatic path planner switching mechanism,
II. RELATED WORK

This section provides a summary of existing work related to the motion framework. For keywords related to the autonomous vehicle and UAV motion, we follow the terminology from [22] that describes similar definition between motion planning, path planning, motion strategy, trajectory planning and motion framework. Therefore, we use those keywords for related work investigations. This section aims to help readers understand more about the state of the art, especially in different domains.

In online motion planning, several studies have been done in 3DoF (Three Degrees of Freedom), such as [23]–[25] The motion is implemented after generating map and calculating the trajectory based on observed obstacles and free paths using vehicle sensors. Even though the concept is most well-known in robotics, it cannot be implemented directly on UAV due to 6DoF, hardware constraints, payload capacity and flight limitation.

Several results related to the autonomous UAV have been reported [26]–[28]. However, performance evaluation in network connection is not considered. They are unable to guarantee fast real time calculation due to exponential time complexity in order to generate maps and paths. In order to guarantee high completeness in static and dynamic areas, an algorithm to integrate graph and neural network variants is also reported by [29]. However, a reinforcement learning algorithm takes a long time for preparation and needs prior map data as much as possible to generate a model.

Other methods based on stochastic approaches [30], [31] and probabilistic approaches [32], [33] have been proposed. Though the results are promising, the computation time tends to be long, and path planning calculations must be recalculated to the final goal. This becomes a serious problem especially when the destination path is very far from the current UAV position. Instead of considering the final goal to calculate the path, we estimate a partial goal by reducing the search graph environment. Our solution has two benefits. First the computation to search a path can always be done within microseconds, and secondly, the memory usage can be reduced.

In addition to propose a framework that can integrate multiple path/motion planning algorithms, we are interested in modifying the A* algorithm as one of the variants of the graph-based method. The algorithm has proven to work well...
in several works and performed better than probabilistic-based method [20]. Our modified A* algorithm is different from others, and we improve the A* method by maximizing the confidence rate of the visited node in an actual direction. At each iteration, the node candidate predicted as the actual direction will be initiated with a minimal cost value; otherwise, it will be filled with a large value. This solution is advantageous in rare u-turn type and maze-like obstacles problems in real-world environment. It will limit obstacle avoidance to cause only small variance errors while keeping the shortest distance between the start and goal positions. In the best scenario, the time complexity will be equal to the length of the shortest path, and memory usage is constant due to heap data structure implementation. The details of our approach are explained more in the section IV.

III. PROBLEM FORMULATION
This study considers a motion planning problem for UAV in a two-dimensional (2D or $R^2$) or three-dimensional (3D or $R^3$) environment through stationary polygonal obstacles, $O = \{O_1, O_2, \ldots, O_n\}$, from a designated starting point $P_{\text{start}}$ to target point $P_{\text{target}}$. The environment is assumed to be a well-built rural area and obstacles are solid, rectangle-shaped objects. It is also assumed that the knowledge of the entire or part of the environment such as the geometries, dimensions and locations of obstacles is known a priori and generated at each microsecond depending on the sensors like LIDAR, cameras, etc. The resultant path has to be collision-free and consists of waypoints $W = \{W_0, \ldots, W_n\}$, which is defined by positions $W_i = \{x_i, y_i\}$ in $R^2$ or $W_i = \{x_i, y_i, z_i\}$ in $R^3$ where $i = 0, \ldots, n$. Note that $W_0$ and $W_n$ are the $P_{\text{current}}$ and $P_{\text{target}}$, respectively. The piece-wise linear segments connect two consecutive waypoints from $P_{\text{current}}$ to $P_{\text{target}}$. Once the UAV starts its mission by traversing along the planned 2D/3D path, the environment may change where pop-ups and/or the previously unknown obstacles might appear on the path. The UAV is equipped with sensors of limited range to collect information about the environment, such as pop-ups and/or the previously unknown obstacles. Using the collected information from the sensors, a new motion has to be replanned in real-time to avoid any collision with the surrounding obstacles.

IV. PROPOSED FRAMEWORK
The goal of the framework is to handle online path planning problems for UAV in rural areas. We assume there is a LIDAR sensor to estimate the environment. The collected data are used for modeling UAV positions and dynamic environment conditions. Detailed computation explanation is described in this section.

A. OVERVIEW
Using this framework, a sequence of five main processes takes place for caring out a mission of a UAV as shown in Fig. 2. Activate UAV and Get Sensor Data are related to the UAV hardware capability in terms of sensors and actuator API (Application Programming Interface) activation. After ensuring the equipment status, LIDAR can be used to collect environment data during flight. Those data are collected in real-time and processed within microseconds, so that it is virtually an immediate feedback.

Compute Motion and Move Actuator processes are dominated by computation tasks such as map preprocessing, partial goal estimation, path planning, pattern extraction, automatic speed and altitude configuration. A path planning algorithm plays the main role by utilizing map discretization along with an automatic decision maker impacting performance metrics such as completeness and computation time. In order to reduce heavy load of computation and memory, we apply dynamic transformation from real environment to 2D. Each 2D map node represents a space in which there may include an obstacle; each node will be represented as a cube in the 3D environment. The map will also be updated within microsecond to adapt dynamic environment in real-time. It provides a flexibility of changing different path planning algorithms to achieve a high degree of completeness and short computation time. In this framework, we can use plug and play mechanism to implement other algorithms that are mostly proposed in the literature, such as Dijsktra’s, Genetic Algorithm, PSO, etc. The algorithm module extends our base module through the default constructor with two parameters: Current Coordinate and Goal Coordinate.

After a path candidate is successfully generated, the automatic configuration of speed and altitude is executed for preparing actuator movement and avoiding obstacles. The actuator should move into the partial goal coordinate which is automatically decided when the final goal coordinate is not in the current UAV sight. Whenever the UAV moves, obstacles and free path positions in discretized map are also updated; however the UAV position is still set to the (0,0) coordinate to maintain stability and reduce the searching space and time complexity in path planning computation. Finally, when the Final Goal coordinate is detected, the UAV will be landed to...
safe environment. Details will be explained in the following sections.

**B. UAV MOTION COMPUTATION OVERVIEW**

The motion computation model takes the goal coordinate as an input parameter, and the output is the UAV movement towards the goal position. The goal coordinate data type is limited on the two-dimensional cartesian map, in which the \( x \)-axis represents the east-west relative position from the UAV and the \( y \)-axis the north-south position. The concept of motion computation is shown in Fig. 3 and the description is given below.

In order to compute path planning, first, the current UAV position should be located at \((0, 0)\) position of the discretized map, and the altitude data are also stored in a hash map data structure with a key-value pair mechanism, in which the key is the node position of the obstacle and UAV, calculated using (1). The value is the altitude data collected from the LIDAR sensor for obstacle and the current UAV altitude configuration.

\[
\text{NodeKeyIndex} = mr \times x + y \tag{1}
\]

The variable, \( mr \), is the maximum width of the discretized map that also represents the maximum LIDAR range, and \( x \) and \( y \) represent the \( x \)-axis and \( y \)-axis coordinate of obstacles.

We assume the world coordinate as the local north, east, down (NED) coordinates since it is commonly used in UAV and airplane coordinate systems. After the three-dimensional position of the UAV is extracted, then they will be used as input parameters to get the surrounding obstacles in the range of the LIDAR sensor, and the LIDAR sensor will collect a set of obstacle coordinates in the NED coordinate data format.

A discretized map is a two-dimensional generated map that visualizes the UAV world from a partial point of view. The map does not portray full obstacle and path coordinates from the 3D environment. Map generation is performed in every iteration by utilizing sensor data. A map comprises a set of grids, where a grid value has three states: 0, 1, and 99 to indicate paths, obstacle, and UAV, respectively. In our approach, the UAV position is always in the center; \((0, 0)\). Every time the UAV moves toward the goal position, the goal and obstacle coordinate will adapt relatively according to the UAV direction. This mechanism can handle storage limitation problems, reduce computation cost and increase the efficiency of UAV movement.

When the map generation process is executed, a path planning algorithm calculates the optimal path from the current UAV position to the final/partial goal estimation. After the path is generated, the extraction pattern mechanism is performed in order to smooth UAV movement while the moving UAV command is executed dynamically every second. The extracted pattern function will classify and detect all similar directions towards the final/partial goal coordinate by updating the direction state and distance. By doing this process, moving UAV command can decide how much distance the UAV has to take in a certain direction.

Using discovery and distance storing mechanisms, the flexibility of UAV to configure altitude and speed improves. Also, since the UAV path planning system supports the safe distance toward direction, the speed can be configured more precisely. For example, if the north distance has the safe value of 10 meters, then it can be mapped to five nodes, where each node represents a 2-meter cube in 3D world terrain. Therefore, if the default speed configuration is executed (1 node per time iteration), moving the UAV 10 meters ahead requires five-time iterations. By implementing automatic speed configuration with a reduced map size and a partial goal target, the speed value can be adjusted within a microsecond.

**C. MAPPING ENVIRONMENT USING LIDAR**

A discretized map is the main component of UAV motion planning, and the component represents a partial view of environmental conditions. The map is generated in real-time by using a LIDAR sensor. As depicted in Fig. 4, initially, the LIDAR data will be generated based on the \( \text{getLidarData} \) function. There are four two-dimensional arrays that should be initiated with the same width and height dimension. The second parameter is the UAV position in the discretized map, which is always in center coordinates. The center coordinate of variables \( \text{realCoorMapX} \), \( \text{realCoorMapY} \) and \( \text{realCoorMapZ} \) represents the NED position value of UAV that is located in the real environment.

After initialization processes have been completed, all processed data from LIDAR will be used to map obstacle positions in the discretized map. Therefore, each LIDAR...
D. PARTIAL GOAL ESTIMATION

In order to reduce computation for path planning, we propose a partial goal coordinate estimation method. Besides the computation benefit, this algorithm can reduce processing storage while pertaining the quality of path planning. The path planning algorithm can still be executed even if the map dimension remains a partial view and the goal coordinate is located far outside the UAV discretized map. The concept is shown in Alg. 2 and is described below.

Suppose our discretized map only has the dimension of 60 units of width and height, meaning the sensors can only be effective to catch the environment condition within 120 meters in range with a node representing a two-meter square in 3D terrain. If the goal is located within a 120-meter range, the path planning computation can directly estimate the optimal path. However, the problem occurs when the target goal coordinate is outside of 120 meters. Suppose the goal coordinate is located at the (200, 200) coordinate, and the instruction is to go 400 meters north and then 400 meters west from the UAV starting position. The UAV needs to move 800 meters of total flight length to achieve the goal.

The target position is outside of discretized map constraints based on the above scenario. Then, the partial goal map position to find optimal path needs to be estimated carefully. We propose a direction-rectangular method to find the partial goal. The first direction to goal is decided as mentioned in Alg. 4. Secondly, the node in each octagonal line related to the direction position is extracted. Third, every node is examined as a decided octagonal obstacle. The picked node will push to heap memory. After that, the top node will be picked based on the shortest and safest distance to the goal position.

Algorithm 1 Map Discretization

Require: uavPos, lidData, mapWidth, mapHeight, obsInd, mW
Ensure: disMap, rCoMaps

1: x, y, z ← uavPos
2: lidX, lidY, lidZ ← lidData
3: obsIndX, obsIndY, obsIndZ ← obsInd
4: uavFlag ← 99
5: obsFlag ← 1
6: disMap ← zeroArray()
7: rCoMapX, rCoMapY, rCoMapZ ← zeroArray()
8: coCenterX ← ceil(mapWidth/2)
9: coCenterY ← ceil(mapHeight/2)
10: NDKIndex ← coCenterX*mW+coCenterY
11: disMap(NDKIndex) ← [uavFlag, z]
12: rCoMap(coCenterX, coCenterY) ← x
13: rCoMap(coCenterX, coCenterY) ← y
14: rCoMap(coCenterX, coCenterY) ← z
15: i ← 0
16: while i < length(lidData) do
17:   oX ← ceil(lidX(i) + (mapWidth/2) + ϵ)
18:   oY ← ceil(lidY(i) + (mapHeight/2) + ϵ)
19:   NRKIndex ← oX*mW+oY
20:   disMap(NRKIndex) ← {obsFlag, obsIndZ}
21:   rCoMapX(oX, oY) ← (lidX(i) + x) * obsIndX
22:   rCoMapY(oX, oY) ← (lidY(i) + y) * obsIndY
23:   rCoMapZ(oX, oY) ← lidZ(i) * obsIndZ
24:   i ← i + 1
25: end while
26: rCoMaps ← {rCoMapX, rCoMapY, rCoMapZ}

Algorithm 2 Estimate Partial Goal

Require: currCoor, goalCoor, topPrevStack, dirPattern
Ensure: parGoal, selParGoal

1: direction ← findDirection(currCoor)
2: for i ← 0 to length(direction) do
3:   dis ← euclidean(topPrevStack, goalCoor)
4:   if direction(i) == True then
5:     parGoal(i) ← currCoor + dirPattern(i) + dis
6:     check ← checkObs(parGoal(i))
7:     parGoal(i) ← check * parGoal(i)
8:     parGoal(i) ← euclidean(parGoal(i), goalCoor)
9: end if
10: end for
11: selParGoal ← minimum(parGoal)
E. MICROKERNEL

The role of the microkernel system in this framework is divided into two aspects: development and integration. From the development point of view, we build several base functions to support the plugin maker. There are four base function varieties: map processor, standardization, monitoring, and computation.

Path planner plugins are created based on supported base functions. When a unique function is not supported in the base function, a developer can add it into one of the base function varieties or add it inside path planner plugin class. After the path planner plugin is created, it is registered into a database and can be configured to perform in UAV within several scenarios: scheduled, static, dynamic, competitive, and cooperative.

The scheduled scenario allows the developer manually to set the appropriate time slots for the module to execute an algorithm. The static scenario means that a certain algorithm is set manually by the developer to perform; this scenario only executes one algorithm for UAV in any circumstance.

In the dynamic scenario, several algorithms are set based on adapting to situations and conditions. If the UAV is in a low-energy situation, it will automatically pick one path planner plugin with the smallest energy consumption even if the optimal path should be sacrificed. In the competitive scenario, all path planning algorithms are available. Then, based on the computation time, the UAV system will pick the fastest path planner to resolve the mission.

The cooperative scenario consists of multiple path planners executing concurrently, and the generated path results will be compared to each other. If there is a different path maneuver, each node that has similar estimated transverse time will be compared based on the shortest distance and true direction towards goal position. Then, the shortest nodes from each different path planning plugin will be assembled and producing a new transverse path.

F. MODIFIED A*

1) DEFINITION

Our framework can host different algorithms as long as each algorithm can be implemented in the two-dimensional map and only need the goal coordinate as the target. However, based on our experiments, other algorithms including A* variants provide inferior results compared to our modified A*. The criteria for comparison include the computation speed, storage size and mission completeness. Most algorithms cannot perform well when UAV have acceleration up to 7 m/s. In contrast, our modified A* algorithm can perform better up to 11 m/s.

As shown in Alg. 3, the modified A* begins with the initialization of eight direction relative position from the start to goal coordinates. There are two arrays used in the algorithm: FlagOfDirection and DirectionCost. The first array represents eight boolean direction states: North (N), South (S), East (E), West (W), North-East (NE), North-West (NW), South-East (SE) and South-West (SW). The boolean value will change to 1 if findDirection function returns true. If boolean value direction is 1 then the directionCost will be set to very small value; otherwise, the directionCost will be set to very high value. This mechanism will make the node with smaller directionCost have more priority for a visit compared to larger directionCost. In addition, we use a minimum heap that always sets the smallest value in the top of memory space. It makes the algorithm have the $O(\log n)$ time complexity and the constant space complexity. The algorithm is presented as Alg. 3.

2) ANALYSIS

In this section, we present some of the theoretical properties of the proposed modified A* method. In the theorems, we use $g*$ to denote the cost of an optimal path from $currCoor$ or $s_{(start)}$ to $s_{(goal)}$. There are three notes to make in this section.

First, all the line numbers of the pseudocode mentioned in this section come from Alg. 3. Second, we assume a greedy algorithm as a comparison by defining a greedy path from $s_{(start)}$ to $s_{(goal)}$, according to the best next node which is chosen from the possible neighboring node $s(n)$, we pick a node $s(n) = \arg \min_{s(n) \in Succ(s)}(g'(s(n)) + cost(s(n), s))$, where $s$ is the current node and $s'$ means the successor state. Lastly, we use the repeated-node term as the condition when the visited node will be reused as the current node $s$. Thus, on the next iteration, the total candidate number of successor node $s(n)$ will be reduced since the previously picked successor node $s(n)$ will be defined as an obstacle to prevent going back to the previous state.

**Theorem 1:** If $fValue(s(n))$ for any node $s(n)$ with $fValue(s(goal))$ exits, then $fValue(s(goal)) \leq \min OPEN(fValue(s(n)))$, and we have newCost $\leq g_{(s)}$ and the iteration will be terminated.

**Proof:** The content value of $g$ can only change in line 35 when the condition satisfies newCost $< g_{(s)}$. After that, the content value of $g$ will be used to decide the $fValue(s(n))$. When $s = s_{(goal)}$, the statement in line 38 will enter the state(s) into OPEN. Therefore, for the next iteration, the value of $fValue(s(goal))$ will not be $\infty$ anymore but $0 \leq fValue(s(goal)) \leq fValue(s)$. After that, when the node $s_{(goal)}$ is achieved, the iteration will be terminated and satisfy newCost $\leq g_{(s)}$.

**Lemma 2:** The PARENT values can only contain at most a number of numberOfPossibleDirections pairs between parent and children states.

**Proof:** Given a node $s$, a number of possible directions is 8. Assume the adjacent nodes of the node $s$ are all obstacle free. Therefore, according to lines 27 and 29, all newCost of the successor nodes can be considered as a candidate node. Furthermore, the instruction on line 38 will push the pair information between parent node $s$ and $s(n)$ as children into the PARENT.

**Theorem 3:** The value of successor node ($s'$) after repeated-node is always different with the previous value and satisfies current $g_{(s(n))} \neq$ previous $g_{(s(n))}$.
Algorithm 3 Modified A*

Require: currCoor, goalCoor, disMap, tinyCost, bigCost
Ensure: pathCandidate, visNode

1: flagDir ← zeroArr(8D)
2: dirCost ← zeroArr(8D)
3: OPEN ← emptyHeap()
4: PARENT ← initializeTable()
5: g ← initializeTable()
6: fValue ← initializeTable()
7: PARENT(currCoor) ← currCoor
8: g(currCoor) ← 0
9: g(goalCoor) ← ∞
10: heapPush(OPEN, fValue(currCoor), currCoor)
11: while OPEN is NonEmpty do
12: $s$ ← heapPop(OPEN)
13: if $s ==$ goalCoor then
14: break
15: end if
16: flagDir ← findDirection(s, goalCoor)
17: for $i$ ← 1 to length(flagDir) do
18: if flagDir$_i$ == 1 then
19: dirCost$_i$ ← tinyCost
20: else
21: dirCost$_i$ ← bigCost
22: end if
23: end for
24: tempIndex ← 0
25: for each $s(n)$ in getNeighbor(s) do
26: if isCollision(s, s(n)) == True then
27: newCost ← $g(s) + ∞$
28: else
29: newCost ← $g(s) + dirCost(tempIndex)$
30: end if
31: if $s(n)$ not in g then
32: $g(s(n)) = ∞$
33: end if
34: if newCost + Euclid(s(n), goalCoor) < $g(s) + Euclid(s, goalCoor)$ then
35: $g(s(n))$ ← newCost
36: fValue$_{(s(n))}$ ← $g(s(n)) + Euclid(s, goalCoor)$
37: PARENT$_{(s(n))}$ ← $s$
38: heapPush(OPEN, fValue$_{(s(n))}$, s(n))
39: end if
40: tempIndex ← tempIndex + 1
41: end for
42: pathCandidate ← extractPath(PARENT)
43: end while

Theorem 4: At line 8, the cost of the node always starts with 0 implying $g(s_{(n)}) = 0$ and for other node $s$ will satisfy $∀s \neq s_{(start)}, g(s) = min_{s \in pred(s)}(g(s) + dirCost)$.

Proof: From line 8, it can be seen that at the beginning, $g(s_{(start)}) = 0$ which implies $g(s_{(start)}) = 0$, while the rest of the $g$ values are non-zero. The only place where $g$ values change is only line 35. If $g(s)$ is changed, then $g$-values of its successors will decrease. The test of line 34 checks this and updates the $g$-values if necessary. Since any cost is positive and never changes, $g(s_{(start)})$ can never be changed, thus is always 0.

Theorem 5: In the worst-case scenario when repeated-node occurs many times, the cost of the proposed modified A* method will be greater than the greedy path $g(s_{(modified\ A*)}) > g(s_{(greedy)}) ≥ 0$.

Proof: As mentioned in Theorem 3, the values of current $g(s_{(s)})$ will be different from the previous. The candidate for the next node $s$ can be derived from $g(s_{(s)}) = g(s) + dirCost$. Since the value of bigCost is predefined by the developers, it provides much greater values compared to the default cost used by the greedy path algorithm. Therefore, the total cost from $s_{(start)}$ to $s_{(goal)}$ will always be greater than the total cost generated by the greedy path algorithm.

Theorem 6: In the best-case scenario when repeated-node does not occur, the cost of the modified A* method will be much smaller than the greedy path $g(s_{(modified\ A*)}) > g(s_{(greedy)}) > g(s_{(modified\ A*)}) ≥ 0$.

Proof: Since the picked node $s$ is decided by line 12 based on the minimum fValue. If repeated-node condition never occurs, then all the picked nodes will be $g(s) = min_{s \in pred(s)}(g(s) + tinyCost)$, in which the tinyCost value has a much smaller value compared to the default cost used by the greedy path algorithm. Therefore, the total cost from $s_{(start)}$ to $s_{(goal)}$ will always be smaller than the total cost generated by the greedy path algorithm.

G. PATTERN

In order to reduce the computation cost for UAV movement, the generated path from path planning will be used as an input.
for `patternExtractor` algorithm. Without a pattern extraction mechanism, the system executes the preplanned UAV commands whenever the UAV moves to an adjacent location, and this makes the UAV unable to have a flexible maneuver and smooth flight to the goal position. In addition, without a pattern extraction algorithm, the UAV may crash if it flies at high speed in order to move from one node to another node which is closely located.

Our pattern extraction begins with the initiation of 4 arrays: `countPattern`, `directionPattern`, `directionPatternLabel`, and `patternSequence`. `countPattern` is initialized with eight number elements with zero value, and this array has a function to store a number of maneuver detected in the pattern sequence searching. The direction pattern contains eight patterns that are related to the eight-direction element. Each pattern is determined based on the observation of difference between adjacent two nodes. `directionPatternLabel` comprises eight direction labels: North (N), South (S), East (E), West (W), North East (NE), South East (SE), North West (NW), and South West (SW). Lastly, `patternSequence` is the empty array that will be used to store extracted pattern.

There are three looping processes to implement pattern extraction as shown in Alg. 5. The first loop is used to extract patterns for each element by subtracting the current searched node with the next node from the generated path. After the subtraction process, the direction value is matched with the `directionPatternLabel` that is already initialized in the first algorithm. The extracted label is appended into the `patternSequence`. The output of the first loop is the `patternSequence` that contains the label name of direction with the size length equal to the path length.

The second loop is used to reduce the number of path candidates by eliminating similar directions in sequence into a single value of direction. In addition, the process also extracts a number of similar directions. When the searching process finds a maneuver that is portrayed by the difference between adjacent node label in pattern sequences, then it will be a trigger to append all previous labels into `extractedPatternSequence` and `extractedPatternDistance` elements.

The third loop is used to make a sequence of NED coordinates. The extracted distance will be added to the previous element of extracted pattern sequence. After those computations have been calculated, the function will return `extractedPatternDistance`, `extractedPatternSequenceLabel`, `realCoordinateX` and `realCoordinateY` containing NED x-axis coordinate and NED y-axis coordinate.

### H. AUTO SPEED AND ALTITUDE CONFIGURATION

The auto speed and altitude configuration function is executed before UAV movement to set a specific coordinate. This module will be triggered by the extraction pattern function. When the extraction pattern function return values such as maneuver distance and direction, both values will be utilized to decide whether altitude and speed alteration will take place as shown in Fig. 5 and 6.

---

**Algorithm 5 Pattern Extraction**

**Require:** `pathCand`

**Ensure:** `excPatDis, excPatSeqLab, rNodeX, rNodeY`

```plaintext
1: countPat ← zeroArr(8D)
2: dN ← (1, 0)
3: dS ← (−1, 0)
4: dE ← (0, 1)
5: dW ← (0, −1)
6: dNE ← (1, −1)
7: dSE ← (1, 1)
8: dNW ← (−1, −1)
9: dSW ← (−1, 1)
10: dP ← [dN, dS, dE, dW, dNE, dSE, dNW, dSW]
11: dPL ← ["N", "S", "E", "W", "NE", "SE", "NW", "SW"]
12: patSeq ← emptyArr()
13: for i ← 1 to length(pathCand) − 1 do
14:  nxp ← pathCand(i+1)
15:  cP ← pathCand(i)
16:  deP ← ((nxp(x) − cP(x)), (nyp(y) − cP(y)))
17:  rl ← match(deP, dP)
18:  patSeq.append(dPL(rl))
19: end for
20: counterManeuver ← 0
21: excPatDis ← emptyArr()
22: excPatSeqLab ← emptyArr()
23: for j ← 0 to length(pathSeq) do
24:  counterManeuver ← counterManeuver + 1;
25:  if pathSeq(j) != pathSeq(j+1) then
26:    excPatSeqLab.append(pathSeq(j))
27:    rl ← match(pathSeq(j), dPL)
28:    cP(rl) ← counterManeuver
29:    excPatDis.append(cP(rl))
30:  end if
31: end if
32: end for
33: rNodeX ← emptyArr()
34: rNodeY ← emptyArr()
35: for k ← 0 to length(pathSeq) do
36:  rl ← match(pathSeq(k), dPL)
37:  if length(rNodeX) == 0 then
38:    rNodeX.append(dP(rl)) + excPatDis(rl)
39:  end if
40:  else
41:    rNodeX.append(rNodeX(k−1) + dP(rl)) * excPatDis(rl)
42:    rNodeY.append(rNodeY(k−1) + dP(rl)) * excPatDis(rl)
43: end if
44: end for
```

Algorithm 6 shows the use of the threshold value of altitude and acceleration unit, which are initially determined by system operator. Both initial values are useful for preventing the UAV to collide with obstacles and crash. After that, the
Algorithm 6 Auto UAV Movement

Require: uavPos, disThres, disMap, rNodeX, rNodeY, ePL, dPL, dP, curSp, curAl, acU, alU, sDis

Ensure: uavMovement

\[\text{speed} \leftarrow \text{currSp}\]
\[\text{altitude} \leftarrow \text{currAl}\]
\[\text{finishStateX}, \text{finishStateY} \leftarrow 0\]

if \(\text{disThres} < (sDis + \text{disThres})\)

\[\text{speed} \leftarrow \text{speed} + \text{acU}\]
\[\text{speed} \leftarrow \text{speed} + \text{acU}\]

end if

\[\text{resDis} \leftarrow \text{match}(\text{ePL}.\text{first}(), \text{dPL})\]

if \(\text{rNodeX} + \text{disThres} > \text{uavPos}(x) > \text{rNodeX} - \text{disThres}\)

\[\text{finishStateX} \leftarrow 1\]

end if

if \(\text{rNodeY} + \text{disThres} > \text{uavPos}(y) > \text{rNodeY} - \text{disThres}\)

\[\text{finishStateY} \leftarrow 1\]

end if

if \(\text{finishStateX} == 0\) or \(\text{finishStateY} == 0\)

\[\text{speedX} \leftarrow \text{speed} \ast \text{dP}(\text{resDis}(x))\]
\[\text{speedY} \leftarrow \text{speed} \ast \text{dP}(\text{resDis}(y))\]
\[\text{speedZ} \leftarrow \text{altitude}\]

\text{moveUAVPosition(speedX, speedY, speedZ)}

end if
TABLE 1. Average completeness.

| Average Speed (m/s) | Searching Radius (km) | Dijkstra’s A* (s) | A* (s) | Modified A* (s) | RRT* (s) | PSO (s) | GA (s) | D* (s) | Potential Field (s) |
|---------------------|-----------------------|------------------|--------|----------------|----------|--------|--------|--------|---------------------|
| 3                   | 1                     | 90               | 92     | 100            | 89       | 86     | 84     | 94     | 91                  |
| 5                   | 1                     | 87               | 89     | 98             | 86       | 84     | 80     | 92     | 89                  |
| 7                   | 1                     | 85               | 86     | 97             | 83       | 82     | 78     | 88     | 86                  |
| 9                   | 1                     | 82               | 84     | 96             | 80       | 79     | 75     | 86     | 83                  |
| 11                  | 1                     | 78               | 80     | 95             | 75       | 74     | 72     | 82     | 80                  |
| 3                   | 2                     | 88               | 90     | 99             | 86       | 84     | 82     | 92     | 89                  |
| 5                   | 2                     | 85               | 87     | 97             | 84       | 82     | 80     | 89     | 86                  |
| 7                   | 2                     | 83               | 85     | 96             | 82       | 79     | 77     | 86     | 84                  |
| 9                   | 2                     | 81               | 83     | 94             | 81       | 78     | 76     | 84     | 82                  |
| 11                  | 2                     | 75               | 77     | 92             | 75       | 74     | 72     | 81     | 78                  |
| 3                   | 3                     | 83               | 85     | 98             | 84       | 83     | 80     | 90     | 87                  |
| 5                   | 3                     | 80               | 81     | 96             | 79       | 77     | 75     | 87     | 84                  |
| 7                   | 3                     | 77               | 79     | 94             | 75       | 72     | 70     | 85     | 82                  |
| 9                   | 3                     | 73               | 75     | 92             | 74       | 70     | 68     | 82     | 78                  |
| 11                  | 3                     | 70               | 72     | 90             | 71       | 69     | 64     | 79     | 75                  |

TABLE 2. Average computation time per second.

| Dijkstra’s A* (s) | A* (s) | Modified A* (s) | RRT* (s) | PSO (s) | GA (s) | D* (s) | Potential Field (s) |
|-------------------|--------|-----------------|----------|--------|--------|--------|---------------------|
| 0.245             | 0.109  | 0.082           | 0.389    | 1.21   | 1.03   | 0.091  | 1.93                |

FIGURE 7. Experiment design.

since we just want to get a result from one path planner variant for each measurement. Experiment design and rescue mission simulation are given in Fig. 7 and Fig. 8 respectively.

There are two metrics used: completeness and computation time. Those are related to the performance quality attributes. Our goal is to find the path planning algorithm with the highest completeness and the shortest computation time. Completeness is just a total number of scenarios that can be completed by UAV to reach the missing person position, and computation time is an average of path planner time execution in seconds for each successful mission. We examine 8 algorithms: Dijkstra’s, A*, RRT*, PSO, GA, D*, Potential Field, and our proposed Modified A*. Those approaches are selected due to being commonly used in the robotics community. We also examine the effect of each algorithm with the score of average speed and searching radius. By doing this setting, we can understand how robust our framework is and which path planning algorithm can provide the best performance within limited circumstances.

VI. RESULT AND ANALYSIS

There are two performance measurements for our framework. First is completeness, which is the total success percentage of UAV for landing safely at the location of missing person location. The second is the computation time which is the total time consumed by UAV to discover missing person in the forest. We provide an average result extracted from 100 simulations with the random coordinates.

Based on the tables 1, 2, and 3, we can observe three findings: the qualities of completeness and computation time metrics tend to decline when the searching radius gets wider. In addition, a faster UAV causes both metrics to decline. Finally, our modified A* provides the best result according to the completeness rate and computation time average. The experiments have been made possible by the flexibility of our framework to host different path planning methods.
Based on our analysis, computation time plays the primary role for completeness since faster computation time at each iteration of time can affect the swiftness of maneuver when facing the obstacles. If the path planning method is not quick enough, the maneuver cannot be executed smoothly and potentially will cause the UAV to crash with the obstacle.

Our modified A* can provide the shortest computation time and highest completeness since the direction-finding, cost adjustment and pattern extraction mechanism reduce computation time. Compared to the methods of emitting dynamic probability line such as RRT, exploring all adjacent nodes such as Dijkstra’s, generating evolution agents such as PSO and GA, and finding the global optima from each discretized map generation like potential field method, our modified A* is relatively fast and not have heavy computation. In addition, the environment itself also does not provide u-turn environment and recursive obstacle such as maze problem. Therefore, the find direction function in Alg. 4 and the cost adjustment mechanism becomes efficient in this case. In each discretized map generation, our modified A* can provide a safe and accurate path candidate without revisions and call remote parent nodes from heap.

**VII. CONCLUSION**

Based on our simulations, our path planning approach can find a safe, complete and high-performance path in...
unknown terrain. A challenge in the rural environment where the network connection or GPS cannot be guaranteed has been given a solution, because all data is calculated by the UAV alone. Our framework can also integrate many path planning algorithms as long as those algorithms can work in a discretized map with a goal coordinate as the parameter input. We have also proposed a modified A* algorithm that can work robustly and provided high performance in terms of completeness rate and computation time. The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.

The direction-based, adjustment cost and pattern extraction algorithm that can work robustly and provided high performance.
[28] M. Jun and R. D’Andrea, “Path planning for unmanned aerial vehicles in uncertain and adversarial environments,” in Cooperative Control: Models, Applications and Algorithms. Boston, MA, USA: Springer, 2003, pp. 95–110. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4757-3758-5_6, doi: 10.1007/978-1-4757-3758-5_6.

[29] A. A. Maw, M. Tyan, T. A. Nguyen, and J.-W. Lee, “IADA*-RL: Anytime graph-based path planning with deep reinforcement learning for an autonomous UAV,” Appl. Sci., vol. 11, no. 9, pp. 1–18, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/9/3948, doi: 10.3390/app11093948.

[30] J. da Silva Arantes, M. da Silva Arantes, C. F. M. Toledo, O. T. Júnior, and B. C. Williams, “Heuristic and genetic algorithm approaches for UAV path planning under critical situation,” Int. J. Artif. Intell. Tools, vol. 26, no. 1, 2017, Art. no. 1760008. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0218213017020018, doi: 10.1142/S0218213017020018.

[31] H. Zhou, H.-L. Xiong, Y. Liu, N.-D. Tan, and L. Chen, “Trajectory planning algorithm of UAV based on system positioning accuracy constraints,” Electronics, vol. 9, no. 2, pp. 1–21, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/2/250, doi: 10.3390/electronics9020250.

[32] S. Poudel and S. Moh, “Hybrid path planning for efficient data collection in UAV-aided WSNs for emergency applications,” Sensors, vol. 21, no. 8, pp. 1–23, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/8/2839, doi: 10.3390/s21082839.

[33] M. Kothari and I. Postlethwaite, “A probabilistically robust path planning algorithm for UAVs using rapidly-exploring random trees,” J. Intell. Robot. Syst., vol. 71, pp. 231–253, Dec. 2013. [Online]. Available: https://link.springer.com/article/10.1007/s10846-012-9776-4, doi: 10.1007/s10846-012-9776-4.

GREGORIUS AIRLANGGA received the B.S. degree in information system from the Yos Sudarso Higher School of Computer Science, Purwokerto, Indonesia, in 2014, and the M.Eng. degree in informatics from Atma Jaya Yogyakarta University, Yogyakarta, Indonesia, in 2016. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, National Chung Cheng University, Taiwan. He is also an Assistant Professor with the Department of Information System, Atma Jaya Catholic University of Indonesia, Jakarta, Indonesia. His research interests include artificial intelligence and software engineering include path planning, machine learning, natural language processing, deep learning, software requirements, and architecture.

ALAN LIU (Member, IEEE) received the Ph.D. degree from the Department of Electrical Engineering and Computer Science, University of Illinois Chicago, Chicago, in 1994. He is currently a Professor with the Department of Electrical Engineering, National Chung Cheng University, Taiwan. His research interests include artificial intelligence and software engineering include knowledge acquisition, requirements analysis, intelligent agents, case-based reasoning, service computing, and applications in smart home systems and robotic systems. He is a member of ACM and also the funding member of the Taiwanese Association for Artificial Intelligence, the Software Engineering Association of Taiwan, and the Robotics Society of Taiwan.