2024 IEEE International Conference on Mobility, Operations, Services and Technologies (MOST) | 979-8-3503-0773-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/MOST60774.2024.00025

Hazardous Area Aware Path-Planning for Drone Swarms

1st Vinh Quach
Department of Computer Science and Engineering
University of North Texas
Denton, Texas
vinh.quach@unt.edu

2nd Burak Tufekci
Department of Computer Science and Engineering
University of North Texas
Denton, Texas
burak.tufekci@unt.edu

3rd Cihan Tunc
Department of Computer Science and Engineering
University of North Texas
Denton, Texas
cihan.tunc@unt.edu

4th Ram Dantu
Department of Computer Science and Engineering
University of North Texas
Denton, Texas
ram.dantu@unt.edu

***Abstract*—Path-planning for drones in areas with complex and unknown environments, constrained by various obstacles, presents a significant challenge in drone operations. This problem extends beyond merely finding an appropriate path from the starting point to the destination; it also involves selecting the ideal path among all available options based on given conditions. In this paper, we propose a novel smart path planning algorithm based on the Breadth-First Search (BFS) algorithm, taking into account both swarm energy and task completion. Performance metrics include the percentage of tasks completed, unachievable, and incomplete. Our novel algorithm demonstrates a significant improvement over traditional methods, outperforming them by an average of 10-15% in task completion. In extreme cases, this margin increases to nearly 20%. Analysis of unachievable tasks reveals that our method greatly reduces their occurrence. This research underscores the potential of our novel algorithm in enhancing operational performance for drone-based tasks, especially in hazardous contexts.**

***Index Terms*—Drone, Unmanned Aerial Vehicle, UAV, path-planning, hazard, task management**

I. Introduction

Unmanned aerial vehicles (UAVs), or called drones, have become very attractive in recent years for various operations such as delivery, search and rescue, surveillance, traffic monitoring, smart agriculture [1]–[3]. Overall drone market was estimated by ReportLinker to be around $27.4 billion in 2021 and is projected to reach $58.4 billion by 2026 [4]. Drones have some incomparable advantages compared to any other vehicle because drones require little to no physical infrastructure, can traverse areas regardless of most terrain, are highly flexible, and can reduce costs [5], [6]. Regardless of their advantages, drones also have limitations and concerns such as very limited payload capability and limited flight time [2], [3], [7], [8]. As an alternative, drone swarms offer better solutions than a single drone in terms of single-point failure, mission handling, and performance [1], [2], [9]. For large regions, drone swarms can spread and execute the mission cohesively. They can execute the tasks simultaneously to reduce the execution time with better fault tolerance by eliminating the single point of failure.

The most attractive application scenario and also a major issue for drone swarm is to execute tasks autonomously in harsh environments. With the increasing popularity and use of drones, the concept of "hash environments" has expanded beyond just physical hazards to include cyber-threats [10]. Among those challenging problems, anti-failure robustness becomes the most challenging problem [9]. Therefore, a high level of robustness and intelligence is needed from drone swarms, which can be defined as the ability to complete missions despite losing some drones and dynamically adapting to real-time circumstantial changes. Moreover, airspace may contain drones from other organizations or obstacles, and it might be governed by government authorities or corridors. Drones operating in the wild have the potential to encounter environments that were not anticipated by their operators. This is where algorithmic path planning becomes crucial.

Path planning for drones in areas with complex and unknown environments, which are restricted by various obstacles, is one of the most significant challenges facing drone operations [11]. This problem is not only limited to searching for an appropriate path from a starting point to a destination but also related to how to choose an ideal path among all available paths based on given conditions. Path planning for drones must be dynamic and adapt in real-time to circumstantial changes to ensure effective distributed behavior. The first step toward the deployment of such complex systems in real-world scenarios is simulation because drone deployment can pose several safety challenges and would be highly expensive and time-consuming [12], [13]. However, the greater majority of existing simulators utilize a 3D engine, require expensive systems and expertise to set up, and can even have a licensing price. These simulators also require high computational complexity and users who want to dive into the research field of drone swarms often need to interface with multiple programming

979-8-3503-0773-3/24/$31.00 ©2024 IEEE
DOI 10.1109/MOST60774.2024.00025

languages, which makes it impractical for abstract-level studies and experiments to create a drone-based environment.

In this paper, we propose a path-planning approach for drone swarms with hazardous area awareness. Our approach considers battery level, number of drones, environmental conditions, and uncompleted missions. Before planning a path, the drones communicate with each other to collect their information in terms of battery, pending tasks, and distance from the destination to determine which path to choose. We are using the best-effort approach so every aspect of the mission is being considered. Energy constraints greatly influence all the parameters of the drone system, as well as the mission itself. Hence the energy and mission burden are being shared among the members of the swarm to ensure mission succession. To demonstrate our work, we used a previously in-house developed simulator and even further improved it with additional capabilities.

The rest of the paper is organized as follows. In Section II, we discuss related work. In Section III, we describe our problem statement, provide definitions, present our algorithms. In Section IV, we present our experimental study and discuss them. Finally, we conclude with our remarks and discuss possible applications of our novel algorithm in Section V.

II. Related Work

Swarm intelligence and smart path planning are established concepts in the realm of drone technology. The existing body of literature extensively explores various facets of swarm intelligence, particularly in the context of path planning [8], [14]. However, a noticeable gap in this literature is the consideration of the drone swarm as a cohesive, singular entity. There appears to be a lack of studies where drones are programmed to assess their collective situation with queries like: 'What if a drone in the swarm fails? Will the remaining drones have sufficient battery life to not only complete their assigned tasks but also the failed drone's task? How would this impact the overall completion of the assigned packages or missions?' Adopting a 'best-effort approach' within this perspective is critical for enhancing the efficiency and reliability of drone swarms, especially in complex, unknown environments and in the event of drone failures. This approach ensures that the remaining drones strive to maximize their operational capacity and task completion. In this section, we provide related works in the domain of drone and drone swarm route planning.

Hamdi et al. [14] proposed an uncertainty-aware route-planning algorithm that factors in weather conditions and individual drone capabilities within predefined sky-paths, using the A* and greedy Dijkstra's algorithm. However, while their approach plans routes based solely on these factors, it fails to account for the conditions or capabilities of other drones in the swarm, as well as the contingency plans for drone failures. This oversight is critical because the algorithm does not consider the collective capability of the entire swarm, especially in the face of uncertain weather predictions. For instance, a '50% chance of rain' in forecasts indicates a 50% probability of rain occurring at any point in the forecast area, which does not guarantee rain over half the area or half the time. Therefore, planning routes without considering the potential risk of choosing a path with even a low chance of adverse weather can lead to suboptimal path selection. It is equally important to recognize that a forecast of a 50% chance of hazardous weather also implies a 50% chance of favorable conditions. This duality highlights the importance of weighing both risk and reward when selecting routes. In such scenarios, taking a path with a forecasted risk of adverse weather could potentially offer significant advantages if the weather remains favorable. This underscores the necessity of incorporating the swarm's overall status and accounting for the relative uncertainties in route planning, as highlighted in our proposed approach. Balancing the potential risks with the potential rewards is crucial for optimal decision-making, particularly in scenarios marked by uncertain weather conditions and other unpredictable factors.

In contrast to Hamdi et al.'s approach in [14], Alkouz et al. [8] proposed a time-constrained algorithm where the swarm is required to perform tasks as quickly as possible, under the assumption of a deterministic environment. They assumed constant operating conditions for the drones and stable weather over time. This assumption represents a major drawback of their design, as the most efficient or 'optimal' path cannot truly be deemed optimal without considering the surrounding environmental variables. While they acknowledged this limitation and have expressed intentions to address it in future work, it's crucial to note that a truly comprehensive approach should include the consideration of uncertainties. In our case, these are classified as hazardous zones, which could be due to a variety of factors such as adverse weather conditions, mechanical breakdowns, physical attacks, cyberattacks, etc.

Iliyan [3] conducted a comparative evaluation between Dijkstra's algorithm and Breadth-First Search (BFS), but their experimental environment lacked any obstacles or uncertainties. As a result, both algorithms appeared to compute identical paths, likely due to the simplicity of the setup. Husain et al. [15] proposed an algorithm tailored for Search and Rescue (SAR) scenarios, which successfully demonstrated the algorithm's advantages and its capability to prevent over-utilization of resources. Their design employed Dijkstra's algorithm to determine the shortest path, testing it within a maze-like environment. However, they neglected to incorporate energy calculations or to use them as constraints in their evaluation.

The authors in [16] acknowledged the presence of dangers in the environment and highlighted how other researchers often assume a constant risk level from the starting point to the endpoint in their analyses. However, their approach involves either completely avoiding these risks or circumventing them by moving far away, without considering the potential benefits of taking calculated risks. As previously mentioned, this method effectively removes consideration of both hazardous and favorable conditions in a somewhat indiscriminate manner. Majd et al. [17] proposed a swarm system designed to prevent collisions between drones and objects in the environment, while also ensuring that each drone has sufficient resources

to complete its assigned tasks. They successfully integrated safety as a key element of the fitness function, but the stringent safety requirement has become a constraint in planning. This overly cautious stance does not take into account the potential benefits of accepting certain risks, which might, in some cases, lead to more optimal solutions.

In summary, our method considers both the operational condition of the swarm and the constantly changing environmental conditions, along with the task queue. While avoiding hazardous zones is beneficial, it is not always the optimal strategy, particularly when drones are far from their base and unable to recharge. Navigating through hazardous zones entails certain risks. Therefore, it is crucial to weigh these risks carefully before planning the path.

III. Methodology

A. Problem statement

In addressing the path planning for drones in complex and unknown environments, which as impeded by various obstacles and include hazardous zones, this paper goes beyond and also tackles the challenge of selecting the ideal path from all available options under specific conditions. Even though the current approaches (e.g., [14], [16]) involve smart drones completely avoiding hazardous zones, as depicted in Figure 1, this approach becomes cumbersome when multiple factors are in play simultaneously. Often, the only viable path may be excessively long, rendering it impractical to fulfill multiple tasks or missions. Hazardous zones need to be taken into account for drone swarms' path-planning. However, rather than completely blocking out a large area, we can consider a weakening effect for weather conditions or even cyber-attacks. For instance, signal propagation (for the case of cyber-attacks) weakens as it distances from the broadcasting device, akin to the diminishing intensity of a storm or rain as it moves from its epicenter. It is also possible that the very destination lies within a hazardous zone. Therefore, this study proposes the use of drone swarms for collaborative learning, evolution, and autonomous decision-making to determine the best action course. As shown in Figure 1, Path 1 and Path 2 for a single drone (for simplicity purposes), though the shortest, cross the hazardous zone's center, entailing higher or absolute risks. Excluding these, *Path 3* emerges as the next shortest alternative, only marginally touching the hazardous zone's outer edge. While the likelihood of an attack is not definite, and the signal weakens with distance, similar to variable weather probabilities. Traditional methods would favor Path 4 as it avoids hazardous zones entirely, despite being significantly longer. However, this approach raises a critical question: what if the destination is within a hazardous zone? Complete avoidance of hazardous zones could render the mission undeliverable or unachievable.

B. Drones and Their Operations

1) Drones: We define a drone as a self-governing vehicle that navigates through a virtual 2D/3D environment. It perceives its surroundings, completes assigned tasks, and communicates with other drones. A real-world drone comprises various elements, such as motors with propellers, a power source like a battery, computational units including a flight controller and planner, an array of sensors, and actuators, for instance, servo actuators. These components, along with the drone's initial position and velocity, are defined in our simulation environment using a JSON file. A sample definition is shown in Listing 1 where a drone is defined with an ID: 3, with a maximum battery capacity of 10,000 mAh, and 10 mAh consumption per tick (i.e., smallest time unit of our simulation). The drone has a speed of 1 cell per tick with an initial position ("init_pos") of (2, 1, 0) in the simulation space (i.e., `X` for lateral, `Y` for longitudinal, and `Z` for vertical movements – for simplicity, we have not used `Z` in our work).

We have set the battery cut-off level to 30%. This simulates real-world conditions where batteries shouldn't be overly discharged. It is also crucial for path planning, as the drone needs to consider its neighbor's battery level to determine whether to risk traversing a hazardous zone or to opt for a safer, longer route. To further emphasize the strength of our path-planning algorithm, we do not allow drones to recharge their batteries. This ensures they must consider battery levels.

```
"drones":
[
    {
        "id":3,
        "battery_max": 10000,
        "battery_move_cost": 10,
        "speed": 1,
        "init_pos": [2, 1, 0],
        "sensors": {}
    }
]
```

Listing 1: Example of a section of a scenario JSON containing a drones list and one drone item.

2) Drone Communication: We use a network class to simulate communication between drones and ground controllers, where the class manages a list of messages with functions to add, remove, and search for them. Each message is an object containing a type, sender and receiver device identifiers, and dynamic contents based on the message type. This network ensures devices and drones receive messages intended for their specific device ID, facilitating coordinated autonomous or human-controlled operations.

3) Tasks and Instructions: A task for a drone in our definition is an ordered list of instructions using a task identifier, a controller identifier, a drone identifier, and a Boolean definition if it is tracked. In this context, 'tasks' can be interpreted as various types of missions. These might include package delivery scenarios, surveying assignments, or even military operations. In the simulator, these tasks are represented by the letter 'T' (we will illustrate these in Figure 3 and Figure 4 in Section IV).

C. Simulation World

The simulation world is the representation of the real world in the simulation, which contains a 2D map (grid of cells)

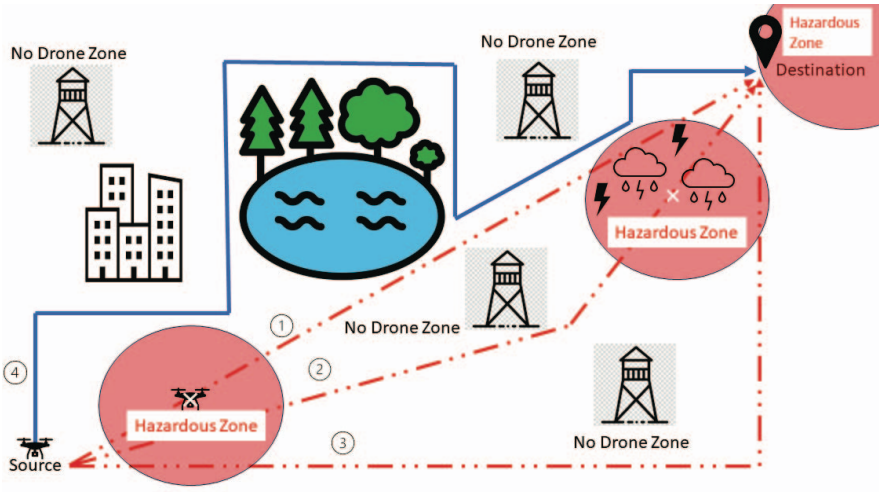

Fig. 1: A demonstration of a scenario showing a drone's source and destination.

and interacts with it. We define the smallest building block of a map as a *cell* to guide the drone (using it to show where the drone is located) and to contain the information about characteristics of that area such as possible obstacles (blocking the cell completely), hazardous cells, hazardous zones, etc.

1) Blocked Cells: In real-world environments, there are various obstacles that drone operators or autonomous drones need to navigate. These include tall trees, power lines, big lakes, buildings, controlled airspace, and no-drone zones. To replicate these challenges in our simulator, we introduced 'blocked cells'. Blocked cells are areas that cannot be traversed, and drones must avoid them. Our path-planning approach recognizes these cells and assigns them predefined values, forcing drones to take alternative routes around them. In the simulator, these cells are marked with the letter 'X'.

2) Hazardous Cells: Our living world is full of hazardous events, especially for flying objects like drones. These hazards can include strong winds, thunderstorms, or heavy rain. They can also be surface-to-air missiles, physical attacks, or cyber-attacks by humans. These cells are not blocked, but yet a drone should not be flying through them. In our system, hazardous cells are predefined in JSON environment files and represented by the letter 'H'. Before any encounter with drones, they remain labeled as 'H'. This is because, prior to an encounter, they are considered normal cells, further implementing the idea of an 'unknown environment'. Only after an encounter that results in a drone being removed, these cells will be marked as 'blocked cells'. At this point, the representation changes from 'H' to 'X', indicating their new status and that a drone has failed at this location.

If a drone encounters a hazardous cell, it is permanently removed from the simulator. This simulates the real-life consequences of a drone being physically damaged or cyber-attacked, hindering its ability to participate in future missions. When a drone encounters a hazardous cell, it becomes incapable of broadcasting or alerting about its compromised state. However, the drone network can detect such issues by the absence of updates from that drone. The last known location of the drone is then marked as a blocked cell. Concurrently, any tasks that were assigned to the compromised drone are reassigned back to the task queue. Additionally, the area surrounding the hazardous cell is then designated as a hazardous zone to alert other drones in that network.

3) Hazardous Zones: In real life, signal propagation weakens as it distances from the broadcasting device, similar to how intensity of a storm or rain diminishes as it moves away from its epicenter. To mirror this, we introduce the concept of hazardous zones, which are the areas in the vicinity of hazardous cells (the 8 cells forming a square around a hazardous cell). These areas carry certain risks, but not all of them are hazardous cells. In our simulator, we represent the hazardous cells within these zones by the letter 'E', indicating the eliminating potential of traversing through these cells. Once encountered, 'E' will change to 'X' indicating a newly blocked cell. This is illustrated in Figure 2a (before) and Figure 2b (after). This further restricts available paths, forcing drones to navigate more intelligently through these dense obstacles, hazardous cells, and hazardous zones.

```
7  -  -  h  H  h  -
8  -  -  h  h  h  -
9  -  -  E  H  h  -
10 X  X  h  h  h  X
11 -  X  -  -  -  -
12 -  -  -  X  -  -
```

(a) BEFORE

```
7  -  -  h  H  h  -
8  -  -  X  h  h  -
9  -  -  X  X  h  -
10 X  X  h  h  h  X
11 -  X  -  -  -  -
12 -  -  -  X  -  -
```

(b) AFTER

Fig. 2: Hazardous cells and zones BEFORE and AFTER the encounter.

The non-destructive hazardous zones in our system are represented by the letter 'h'. It is important to note that these hazardous cells and zones can be customized to fit the specific environment users wish to simulate. In our case, we choose to set the number of hazardous cells equal to the number of drones. This approach ensures that, in the worst-case scenario, all drones could be removed, yet it also provides them with the opportunity to complete their assignments. Setting this number too high would prematurely terminate the simulation, which would not be sensible as it would not give the drones even a chance to compete for their success.

D. Algorithms

In this section, we describe the algorithms used in this study. The primary algorithms are Breadth First Search (BFS), Smart BFS, and Dijkstra's algorithms. Additionally, there is an intermediate algorithm that bridges BFS and Smart BFS algorithms, providing an alternative path when BFS cannot populate the path.

1) Breadth First Search (BFS) algorithm: BFS is a graph traversal algorithm that starts from the starting point and explores its neighbors first before moving to the next level of nodes. It uses a queue to keep track of the nodes to be visited, ensuring that each node is explored in the order it was discovered [18]. The steps for generating the path using this algorithm are given in Algorithm 1. The Costmap_Builder function, as indicated in line 4, constructs a costmap beginning at the starting point, assigning a value of 100 to hazardous zones to simulate the conventional approach of avoiding these areas. As indicated in line 6, the algorithm will avoid zones marked with this value, which can sometimes lead to negative effects, such as being unable to find a path or failing to reach a destination that lies within a hazardous zone.

2) Smart BFS algorithm: This algorithm utilizes the classic BFS, but makes significant revisions in the costmap and path selection. It takes into account all major aspects of drone operation; e.g., battery level, pending tasks, uncompleted tasks, the number of drones, and the capabilities of neighboring drones in completing tasks if the current drone fails. For instance, as stated in line 4 of Algorithm 2, when there are fewer than two drones available, the destination is not in a hazardous zone, and there are still tasks pending, the algorithm will select the safest path, which is the standard BFS approach.

However, if this conservative path cannot be traversed for any reason, it can switch to an alternative method referred to as 'Try BFS', which is detailed in Algorithm 3. When using 'Try BFS', the drone attempts to plan its path using BFS.

In worst-case scenarios, where a path cannot be found, it will revert to a smart path-planning approach by calling function Risk_Aware_Path_Builder, summarized in Algorithm 4. In this mode, it accepts associated risks and makes its best effort to complete the task. Another difference of 'Smart BFS' lies in the construction of the costmap. Since it thoroughly analyzes the situation before selecting a path, the Risk_Aware_Costmap_Builder treats hazardous zones similarly to normal cells, recognizing that there is a chance of success. The level of risk can be adjusted based on the criticality of the assignments. In critical scenarios, such as life-saving operations, drones can tolerate higher risks. Conversely, in more routine scenarios where it's acceptable to forego certain tasks, the drones can operate at lower risk levels.

Algorithm 1 Breadth First Search Algorithm

```
1:  Input: drone_id, pos_a, pos_b
2:  Output: path from pos_a to pos_b
3:  function BFS
4:      point_queue ← Costmap_Builder(grid width, grid length, pos_a, pos_b)
5:      path.insert(0, point_queue[0])
6:      while cost_map[path[0].x][path[0].y] != 0 and cost_map[path[0].x][path[0].y] < 100 do
7:          surrounding ← getSurroundingPositions(path[0])
8:          last_cell ← path[0]
9:          best_cell ← last_cell
10:         best_cost ← cost_map[path[0].x][path[0].y]
11:         for cell in surrounding do
12:             if cost_map[cell.x][cell.y] != -1 and cost_map[cell.x][cell.y] < best_cost then
13:                 best_cell ← cell
14:                 best_cost ← cost_map[cell.x][cell.y]
15:             end if
16:         end for
17:         if last_cell == best_cell then
18:             break
19:         end if
20:         path.insert(0, best_cell)
21:     end while
22:     path.pop(0)
23:     return path
24: end function
```

Algorithm 2 also addresses situations where a neighboring drone is no longer communicating its location, or it has completed the task and is returning home, even when the instructions do not specify the destination for completed tasks. It is important to emphasize that there are times when a destination falls within a hazardous zone, and in such cases, the drone must definitely go there to complete the task.

3) Dijkstra's algorithm: Dijkstra's algorithm is a method for finding the shortest path between nodes in a graph. It starts by assigning a tentative distance to every node: zero for the initial node and infinity for all others. The algorithm repeatedly selects the node with the smallest tentative distance, calculates and updates the distances of its neighbors, and marks the node as visited. This process continues until all nodes are visited or the shortest paths to all remaining nodes are determined [19]. The aforementioned process can be summarized in Algorithm 5. This algorithm also avoids hazardous zones.

Algorithm 2 Smart BFS Algorithm

```
Input: drone_id, pos_a, pos_b
Output: path from pos_a to pos_b
actual_count ← num_drones
if actual_count < 2 and not (cells[pos_b.x][pos_b.y].is_hazardous_zone) and (tasks_ready > 0) then
    path ← BFS(drone_id, pos_a, pos_b)
    if path is empty then
        path ← Try_BFS(drone_id, pos_a, pos_b)
    end if
    return path
end if
num_tasks_ready ← length(tasks_ready)
for each drone_object_id in drones_info do
    if drone_object_id =! drone_id then ▷ Analyze neighbor drone
        Obtain battery level, init-pos, final-pos, instructions
        if drone_object_id in list_of_active_drones then ▷ Obtain position of neighbor drone
            if position is not empty then
                Save position
            else
                Select appropriate BFS strategy
                return path
            end if
        end if
        if position conditions met then
            Calculate battery requirement and select path
        else
            Select appropriate BFS strategy
        end if
    end if
end for
Update battery counts based on predefined percentages
if destination is in hazardous zone then
    path ← Risk_Aware_Path_Builder(drone_id, pos_a, pos_b)
    return path
else
    Choose path based on number of drones, task counts, and battery levels
end if
```

Algorithm 3 Try Breadth First Search Algorithm

```
Input: drone_id, pos_a, pos_b
Output: path from pos_a to pos_b
function TRY_BFS
    point_queue ← Costmap_Builder(grid width, grid length, pos_a, pos_b)
    path.insert(0, point_queue[0])
    while cost_map[path[0].x][path[0].y] != 0 and cost_map[path[0].x][path[0].y] < 100 do
        surrounding ← getSurroundingPositions(path[0])
        last_cell ← path[0]
        best_cell ← last_cell
        best_cost ← cost_map[path[0].x][path[0].y]
        for cell in surrounding do
            if cost_map[cell.x][cell.y] != -1 and cost_map[cell.x][cell.y] < best_cost then
                best_cell ← cell
                best_cost ← cost_map[cell.x][cell.y]
            end if
        end for
        if last_cell == best_cell then
            break
        end if
        path.insert(0, best_cell)
    end while
    path.pop(0)
    if path is empty then
        path ← Risk_Aware_Path_Builder(drone_id, pos_a, pos_b)
        return path
    end if
end function
```

IV. Experimental Study

A. Experimental Setup

In order to demonstrate our drone swarm path-finding approaches, we have used our in-house drone simulator, which was explained in [20]. In this work, we further enhanced the simulator by introducing the ability to modify environmental factors. These enhancements include adding blocked cells, incorporating wind effects, designating hazardous cells, managing drone numbers, and reintegrating pending tasks into the queue, along with the addition of new tasks. In our experimental simulations, we used various configurations including number of drones, blocked cells, and hazardous environments. The following simulations were run on a 12th Gen Intel(R) Core(TM) i7-1255U processor with 16GB RAM. Our experiments generally took no significant resource utilization and execution time.

B. Experimental Results

In order to evaluate the performance of our proposed hazard-aware path-finding algorithms for a drone swarm, we simulated a task completion scenario. The initial stage, shown in Figure 3, features a simulation space with nine tasks at corresponding locations for four drones. Recharging is not allowed, forcing the drones to conserve battery power and make informed decisions based on battery level. All assignments begin from the center of the grid and spread out to nine locations, distributed near the outer edges of the grid to maximize travel distance and increase the likelihood of encountering hazardous cells. Figure 4 shows the final positions of the tasks and the ideal scenario where all four drones return safely. The percentage of blocked cells ranges

Algorithm 4 Risk Aware Path Builder

```
Input: drone_id, pos_a, pos_b
Output: path from pos_a to pos_b
function RISK_AWARE_PATH_BUILDER
    point_queue ← Risk_Aware_Costmap_Builder(grid width, grid length, pos_a, pos_b)
    path.insert(0, point_queue[0])
    while cost_map[path[0].x][path[0].y] != 0 do
        surrounding ← getSurroundingPositions(path[0])
        last_cell ← path[0]
        best_cell ← last_cell
        best_cost ← cost_map[path[0].x][path[0].y]
        for cell in surrounding do
            if cost_map[cell.x][cell.y] != -1 and cost_map[cell.x][cell.y] < best_cost then
                best_cell ← cell
                best_cost ← cost_map[cell.x][cell.y]
            end if
        end for
        if last_cell == best_cell then
            break
        end if
        path.insert(0, best_cell)
    end while
    path.pop(0)
    return path
end function
```

Algorithm 5 Dijkstra's Algorithm

```
Input: drone_id, pos_a, pos_b
Output: path from pos_a to pos_b
current_cell ← pos_b
last_cell ← None
path ← empty list
while current_cell ≠ pos_a do
    path.insert (0, current_cell)
    surrounding ← getSurroundingPositions(current_cell)
    min_cost ← ∞
    next_cell ← None
    for all neighbor in surrounding do
        if 0 ≤ neighbor.x < width and 0 ≤ neighbor.y < length and not cells[neighbor.x][neighbor.y].is_hazardous_zone then
            if cost_map[neighbor.x][neighbor.y] < min_cost then
                min_cost ← cost_map[neighbor.x][neighbor.y]
                next_cell ← neighbor
            end if
        end if
    end for
    last_cell ← current_cell
    current_cell ← next_cell
end while
    return path
```

from 15%, 20%, to 25% randomly placed, with the number of hazardous cells equal to the number of drones. 10% of the hazardous zones have an eliminating effect. These parameters are customizable to suit specific user scenarios. We found this to be a balanced scenario, as it allows the drones to perform their tasks while imposing significant restrictions. In our experiments, at most, 1 out of every 4 cells is a blocked cell (25%) to compel drones to critically think and analyze the environment. This forces the drones not only to conserve battery but also to accept appropriate risks while employing a best-effort approach to complete as many tasks as possible within the constraints of the battery level.

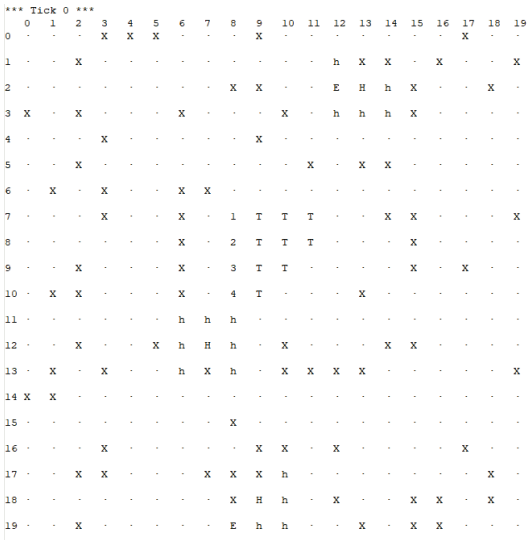

Fig. 3: The initial stage of the simulation where all drones and tasks are located in the center. This scenario shows 20% of blocked cells.

We evaluate the performance by comparing the percentages of completed tasks, unachievable tasks, and incomplete tasks between the three algorithms. The results for these two algorithms are somewhat comparable, similar to what is reported in [3]. Our novel algorithm outperformed both, on average, in terms of the percentage of completed tasks, ranging from 10-15% better. In some extreme cases, the difference can be nearly 20%, while their best performer is still 8% worse than our worst performer. The bar graph in Figure 5 shows the percentage of completed tasks in three different scenarios for the three algorithms. Our algorithm consistently completed more than 91% of the tasks in all cases.

Both unachievable tasks and incomplete tasks for the BFS and Dijkstra's algorithms are close to each other but are significantly higher than for our novel approach in all cases, as shown in Figures 6 and Figure 7, respectively. Unachievable tasks can occur in a few different scenarios: the most obvious is when the destination is within a hazardous zone, a drone is trapped in areas surrounding hazardous zones, or the initial location of the task is within hazardous zones. Note that the percentage of unachievable tasks is directly caused by

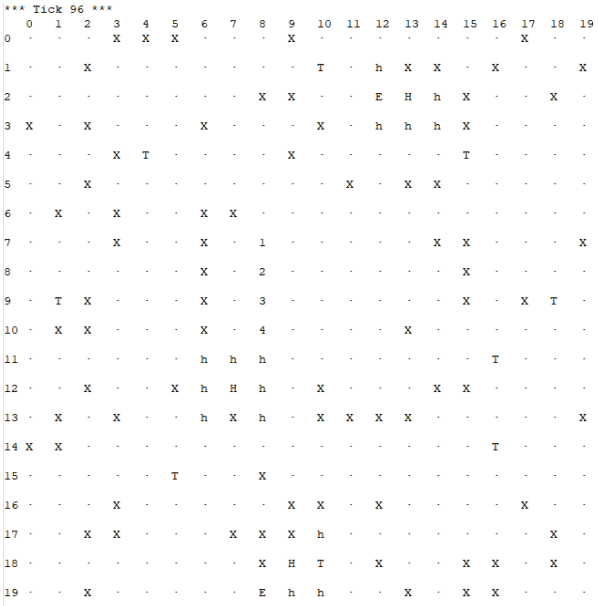

Fig. 4: The final stage of the simulation where all drones come back to the initial locations and all tasks have been completed. This scenario shows 20% of blocked cells.

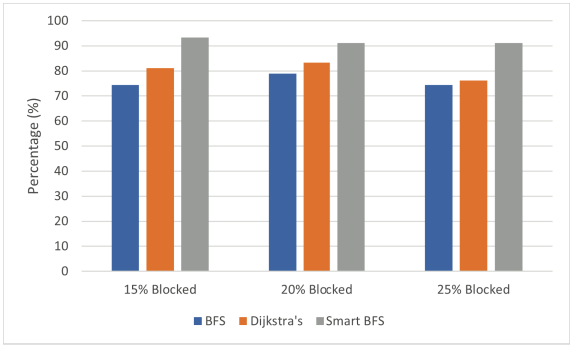

Fig. 5: Percentage of completed tasks

selecting the traditional BFS algorithm when only one drone is left, further proving that our novel approach is superior. Incomplete tasks can be caused by drones running out of battery before reaching the task or by drones changing their status to 'HOME' before the assignment is added back to the queue. In either case, our algorithm outperforms both algorithms.

Figure 8 and Figures 9 show the cost maps for the BFS and Dijkstra's algorithms, respectively. When examining Figure 8, it becomes evident that the center of the image is filled with numbers valued at 100 or greater, indicating that these cells are non-traversable when using the BFS algorithm. This implies two constraints: entities within these areas cannot exit, and those outside cannot enter, significantly limiting the available paths for drones. Sometimes, this can render certain tasks unachievable. Similarly, upon examining Figure 9, we

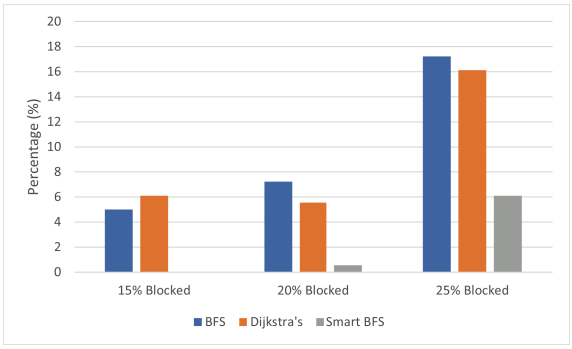

Fig. 6: Percentage of unachievable tasks

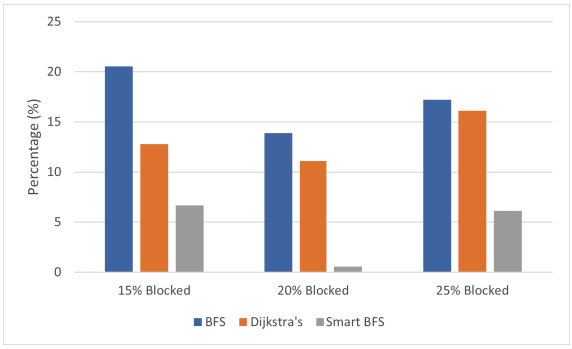

Fig. 7: Percentage of incomplete tasks

observe comparable scenarios. Certain areas are encircled by 'inf' values, indicating that if a final destination lies within these zones, the tasks become unachievable. Contrary to these approaches, our costmap, as illustrated in Figure 10, treats the hazardous zone as normal cells after careful consideration of all other factors. This gives the drones in our case more available paths, thereby enhancing the completion rate.

```
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 11 12 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 11 10 11 12 -1 12
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 11 10  9 10 11 12 11
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 11 10  9  8 -1 10 11 10
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  7  8  9 10  9
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 11 10  9  8  7  6  7 -1 -1  8
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 11 10  9 -1  7  6  5  6  5  6  7
-1 -1 -1 -1 -1 -1 -1 -1 -1 11 10  9  8  9 -1  5  4 -1  4  5  6
-1 -1 -1 -1 -1 -1 -1 -1 11 10  9  8  7 -1  5  4  3  2  3  4  5
-1 -1 -1 -1 -1 -1 -1 -1 -1 11 10 -1  6  5  4  3  2  1  2  3  4
-1 -1 -1 -1 -1 -1 -1103 12 11100100100 -1  2  1  0 -1  4  5
-1 -1 -1 -1 -1 -1 -1 -1103 12 -1 -1100  6 -1  2  1 -1  5 -1
-1 -1 -1 -1 -1 -1 -1 -1104 -1100100100  5  4  3  2  3  4 -1
-1 -1 -1 -1 -1 -1 -1104103102101 -1  7  6  5  4  3  4 -1  8
-1 -1 -1 -1 -1 -1 -1 -1 -1103102 -1  8  7  6  5 -1  5  6  7
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 10  9  8  7  6  7 -1  7 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 10  9  8 -1  8  9  8  9
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 10  9 10  9 10  9 10
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 11 10 -1 10 11 10 -1
```

Fig. 8: Costmap when using BFS algorithm for path planning

```
infinfinfinfinf 14 13 12 11 10 11 12 13inf 13 12 13infinfinf
infinfinfinf 14 13 12 11inf  9 10 11 12inf 12 11 12 13 14inf
infinfinf 14 13 12 11 10  9  8  9 10 11 12 11 10 11 12inf 14
infinf 14 13 12inf 12infinf  7  8  9 10 11 10  9 10 11 12 13
inf 14 13 12 11 10inf  8  7  6  7  8  9 10  9  8inf 10 11 12
inf 13 12 11inf  9  8  7  6  5infinfinfinfinf  7  8  9 10 11
 13 12 11 10  9inf  7infinf  4  3  2  3  4  5  6  7infinf 12
infinfinfinf  8  7  6  5  4  3  2  1inf  5  6  7  8  9 10 11
 13 12 11 10  9  8inf  6  5  4inf  0  1inf  7  8inf 10 11 12
inf 13 12inf 10  9inf  5  4  3  2  1inf  5  6  7  8  9 10 11
inf 14 13 14infinf  7  6  5  4inf  2  3  4  5  6  7  8  9 10
infinfinfinfinfinf  8  7  6  5infinfinfinf  6  7  8inf 10 11
infinfinfinf 13 12infinf  7  6infinfinf 12inf  8  9inf 11inf
infinf 14 13 12 11 10  9  8infinfinfinf 11 10  9 10 11 12inf
infinfinf 14 13inf 11 10  9 10 11inf 13 12 11 10 11 12infinf
infinfinfinf 14 13 12 11inf 11 12inf 14 13 12 11inf 13 14inf
infinfinfinfinfinf 13infinfinfinfinfinf 14 13 12 13infinfinf
infinfinfinfinfinf 14infinfinfinfinfinfinf 14inf 14infinfinf
infinfinfinfinfinfinfinfinfinfinfinfinfinfinfinfinfinfinfinf
infinfinfinfinfinfinfinfinfinfinfinfinfinfinfinfinfinfinfinf
```

Fig. 9: Costmap when using Dijkstra's algorithm for path planning

```
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1  7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1  6  7  8 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1  7 -1  5  6  7  8 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1  6  5  4  5  6  7 -1 -1  8 -1 -1 -1 -1 -1 -1
-1 -1 -1  7 -1  5  4  3 -1  7  8 -1 -1  7  8 -1 -1 -1 -1 -1
-1 -1  7  6  5 -1 -1  2  1 -1 -1 -1  5  6  7  8 -1 -1 -1 -1
-1 -1 -1 -1  4  3  2  1  0  1  2  3  4  5  6  7  8 -1 -1 -1
-1 -1 -1 -1 -1 -1  3  2  1  2 -1 -1  5  6  7  8 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1  4  3  2 -1 -1  7  6  7  8 -1 -1 -1 -1 -1
-1 -1 -1  8  7  6  5  4 -1 -1 -1  8  7  8 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1  7  6  5  6 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1  8  7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1  8 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Fig. 10: Costmap when using our Smart BFS algorithm for path planning

V. Conclusion

In this paper, we presented an approach for drone swarms' path-finding for hazardous environments. We provided a novel presentation of the hazardous areas to avoid once a problem is detected. To avoid such environments and possible obstacles, we provided multiple algorithms, including Breadth First Search (BFS) and Dijkstra's algorithms as well as an improved version of BFS that we call Smart BFS. The algorithm presented in this paper demonstrates significant potential in enhancing the efficiency and adaptability of drone swarm operations in various challenging environments. Through its dynamic analysis of drone status, task completion, and environmental factors, our approach offers customizable solutions tailored to the demands and critical nature of diverse missions. Our experimental results demonstrate the superiority of Smart BFS compared to the other algorithms by 20% higher task completion ratio. Particularly in contexts such as military operations, active disaster search and rescue, and aid delivery to isolated regions post-disaster, our algorithm shows promising applicability. This research not only contributes to the current body of knowledge but also opens avenues for future exploration in optimizing drone swarms for critical and time-sensitive tasks.

Acknowledgment

This research was partially supported by the National Science Foundation (NSF) Award Number 2149969 and the National Centers of Academic Excellence in Cybersecurity, housed in the Division of Cybersecurity Education, Innovation and Outreach, at the National Security Agency (NSA) grants H98230-20-1-0329, H98230-20-1-0414, H98230-21-1-0262, H98230-21-1-0262, and H98230-22-1-0329.

References

[1] M. Campion, P. Ranganathan, and S. Faruque, "Uav swarm communication and control architectures: A review," *Journal of Unmanned Vehicle Systems*, vol. 7, no. 2, pp. 93–106, June 2019.

[2] Y. Aydin, G. K. Kurt, E. Ozdemir, and H. Yanikomeroglu, "Authentication and handover challenges and methods for drone swarms," *IEEE Journal of Radio Frequency Identification*, vol. 6, pp. 220–228, 2022.

[3] I. Z. Boychev, "Research algorithms to optimize the drone route used for security," in *2018 IEEE XXVII International Scientific Conference Electronics - ET*, Sep. 2018, pp. 1–4.

[4] "The global UAV market is estimated to be USD 27.4 billion in 2021 and is projected to reach USD 58.4 billion by 2026, at a CAGR of 16.4%," Jun 2021. [Online]. Available: https://www.globenewswire.com/news-release/2021/06/18/2249504/0/en/The-global-UAV-market-is-estimated-to-be-USD-27-4-billion-in-2021-and-is-projected-to-reach-USD-58-4-billion-by-2026-at-a-CAGR-of-16-4.html

[5] E. Yağdereli, C. Gemci, and A. Z. Aktaş, "A study on cyber-security of autonomous and unmanned vehicles," *The Journal of Defense Modeling and Simulation*, vol. 12, no. 4, pp. 369–381, 2015.

[6] A. Nayyar, B.-L. Nguyen, and N. G. Nguyen, "The Internet of Drone Things (IoDT): future envision of smart drones," in *First International Conference on Sustainable Technologies for Computational Intelligence: Proceedings of ICTSCI 2019*. Springer, 2020, pp. 563–580.

[7] G. Raja, S. Anbalagan, A. Ganapathisubramaniyan, M. S. Selvakumar, A. K. Bashir, and S. Mumtaz, "Efficient and secured swarm pattern multi-uav communication," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 7, pp. 7050–7058, 2021.

[8] B. Alkouz, A. Bouguettaya, and S. Mistry, "Swarm-based drone-as-a-service (sdaas) for delivery," in *2020 IEEE International Conference on Web Services (ICWS)*, 2020, pp. 441–448.

[9] W. Chen, J. Liu, H. Guo, and N. Kato, "Toward robust and intelligent drone swarm: Challenges and future directions," *IEEE Network*, vol. 34, no. 4, pp. 278–283, July 2020.

[10] M. Krichen, W. Y. H. Adoni, A. Mihoub, M. Y. Alzahrani, and T. Nahhal, "Security challenges for drone communications: Possible threats, attacks and countermeasures," in *2022 2nd International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*, Riyadh, Saudi Arabia, 2022, pp. 184–189.

[11] K. Bezas, G. Tsoumanis, and K. Oikonomou, "A coverage path planning algorithm for self-organizing drone swarms," in *2021 International Balkan Conference on Communications and Networking (BalkanCom)*, 2021, pp. 122–126.

[12] S. Baidya, Z. Shaikh, and M. Levorato, "FlyNetSim: An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. Montreal QC Canada: ACM, Oct 2018, p. 37–45. [Online]. Available: https://dl.acm.org/doi/10.1145/3242102.3242118

[13] P. S. Andrews, S. Stepney, and J. Timmis, "Simulation as a scientific instrument," in *Proceedings of the 2012 workshop on complex systems modelling and simulation, Orleans, France*. Citeseer, 2012, pp. 1–10.

[14] A. Hamdi, F. D. Salim, D. Y. Kim, A. G. Neiat, and A. Bouguettaya, "Drone-as-a-service composition under uncertainty," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2685–2698, 2022.

[15] H. Z, A. Z. A, H. H, S. F, R. D, and I. AF, "Search and rescue in a maze-like environment with ant and dijkstra algorithms," *Drones*, vol. 6, no. 10, p. 273, 2022.

[16] J. Li and Y. Shi, "Auv path planning for environment changes over time," in *2018 2nd International Conference on Electronic Information Technology and Computer Engineering (EITCE 2018)*, ser. MATEC Web of Conferences, vol. 232, no. 04020, EITCE. EDP Sciences, 2018, p. 04020, published online: 2018-11-19. [Online]. Available: https://doi.org/10.1051/matecconf/201823204020

[17] A. Majd, M. Loni, G. Sahebi, and M. Daneshtalab, "Improving motion safety and efficiency of intelligent autonomous swarm of drones," *Drones*, vol. 4, no. 3, p. 48, 2020. [Online]. Available: https://doi.org/10.3390/drones4030048

[18] S. Beamer, K. Asanovic, and D. Patterson, "Direction-optimizing breadth-first search," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–10.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs:(numerische mathematik, 1 (1959), p 269-271)," 1959.

[20] E. B. Putnam, D. N. Senarath, G. R. Urquijo, C. Tunc, and R. Bryce, "A lightweight drone simulator," in *2023 Tenth International Conference on Software Defined Systems (SDS)*. IEEE, 2023, pp. 52–59.