Drones to the Rescue: Leveraging UAVs for Disaster Relief

Sai Sri Harsha Kotti
Department of Electrical and Computer Engineering
Northeastern University
Boston, Massachusetts, USA
kotti.s@northeastern.edu

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—In the aftermath of natural disasters, rapid restoration of communication networks is crucial for effective relief operations. This paper presents an AI-driven framework for deploying autonomous drones equipped with LTE relays to provide emergency network coverage and assist in locating survivors. Focusing on the software intelligence layer, we implement a Q-learning model to dynamically determine optimal drone placements based on coverage area. To enable efficient navigation within complex or partially obstructed environments, an A-star path finding algorithm helps guide the emergency vehicle to its target location while avoiding obstacles. Simulation results demonstrate that this integrated approach significantly improves coverage area and provides a quick path to the target position. By emphasizing adaptive decision-making and efficient routing, the proposed system offers a scalable solution for intelligent, drone-based disaster response.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Unmanned Aerial Vehicles (UAVs), commonly known as drones, have demonstrated significant utility across a wide array of domains, including aerial photography, environmental monitoring, scientific exploration, logistics, and defense. Their increasing adoption is largely attributed to their ability to traverse difficult terrains, carry payloads, and operate either autonomously or under remote supervision. These attributes make drones particularly advantageous in disaster scenarios, where traditional infrastructure may be compromised or entirely unavailable.

In such contexts, drones can play a critical role in supporting emergency response efforts—delivering supplies, gathering situational data, and re-establishing communication networks. A particularly promising application involves the use of drones as airborne relay nodes to temporarily restore mobile communication using LTE/LoRA technology. However, the effectiveness of this approach relies not only on the hardware capabilities of the drones but also on intelligent decision-making algorithms that can adapt to dynamic and uncertain environments.

This study focuses on the development of an AI-driven control framework for the autonomous deployment of drones in disaster-stricken areas, emphasizing adaptive positioning and efficient navigation. Specifically, we employ reinforcement learning, a type of machine learning where an agent learns optimal behavior by interacting with its environment and receiving feedback in the form of rewards. Within this paradigm, we utilize Q-learning, a model-free reinforcement learning algorithm that enables agents to learn action policies based on expected cumulative rewards. In our system, Q-learning is used to train drones to identify optimal locations for LTE coverage based on real-time feedback such as signal strength and mobile device presence.

To facilitate movement from one location to another in complex or partially obstructed environments, we incorporate the A* (A-star) pathfinding algorithm. A* is a heuristic-based search algorithm widely used in robotics and video games, known for its ability to find the shortest and most efficient path between two points while avoiding obstacles.

By integrating Q-learning for intelligent decision-making with A* for precise path planning, the proposed system aims to provide a robust and scalable solution for autonomous drone deployment in disaster relief operations. This paper primarily focuses on the AI models underlying this framework, offering a software-centric approach that can be extended to physical implementations in future work.

II. Environment modeling

The disaster zone is represented as a two-dimensional grid, where each cell corresponds to a position that a drone can potentially move to. As the drone learns about the environment, it assigns values to each cell. These values help the system decide whether a cell is usable or blocked. For example, if a cell represents an area that is covered in debris or unsafe, it will be marked as blocked and avoided.

Since drones fly at a certain height, most ground-level obstacles do not affect them. However, tall buildings or high-rise structures could interfere with their flight paths. Handling such obstacles is outside the scope of this paper and is suggested for future work.

Each drone has a fixed communication range. It can cover a minimum area of 3×3 cells and a maximum of 5×5 cells, depending on its altitude and power. The total grid size can vary based on the scenario. In this study, we use a fixed-size grid for simplicity. The point (0, 0) is used as the starting location, representing the emergency response vehicle or command center. A random target location is selected within the grid, and the drone must navigate to this point while avoiding any blocked areas.

III. Optimal Placement Using Q-Learning

To determine the best locations for placing drones in a disaster zone, we use **Q-learning**, a reinforcement learning algorithm that enables an agent to learn optimal actions through trial and error. In this context, the agent is responsible for deploying and managing drones to maximize coverage over a grid-based environment.

The environment is structured as a two-dimensional grid. At each time step, the agent can choose from several actions:

- `NOOP` (do nothing)
- `SPAWN_RANDOM` (deploy a drone of size 3 or 5 at a random location)
- `ACT` (either maintain or remove an existing drone)

The primary objective is to achieve near-complete coverage using the fewest drones, while minimizing overlap between their coverage areas. To guide the learning process, a reward function is defined to:

- Reward new coverage
- Penalize overlapping coverage
- Slightly penalize excessive drone usage

State Representation

Each state is represented as a list of all currently deployed drones, including their (x, y) positions, sizes, and activity status. This state is then converted into a canonical string form to serve as an index in the Q-table.

Action Selection

The agent follows an ϵ-greedy strategy for exploration and exploitation:

- With probability ϵ: a random action is chosen (exploration).
- With probability $1 - \epsilon$: the agent selects the action with the highest estimated Q-value (exploitation).

The exploration probability ϵ decays gradually with each episode to promote learning early and refine strategies later.

Q-Value Update

After executing an action and observing the resulting state and reward, the Q-value is updated using the Bellman equation: $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ where:

- s and a are the current state and action
- r is the received reward
- s' is the next state
- α is the learning rate
- γ is the discount factor for future rewards

Training and Evaluation

The best-performing Q-table (i.e., the one that achieves the highest coverage) is saved and later used for evaluation. During evaluation, the agent acts greedily, always selecting the best-known action. This allows us to measure how well the learned policy performs without further learning.

This Q-learning framework enables the agent to place drones adaptively and efficiently, ensuring maximum coverage with minimal redundancy—an essential capability for real-time disaster response systems.

A. Psuedo-code

Algorithm 1: Q-Learning for Adaptive Drone Placement

```
Input: Environment with grid, max episodes $N$,
       learning rate $\alpha$, discount factor $\gamma$, exploration
       rate $\epsilon$
Output: Optimized Q-table for drone placement policy
1  Initialize Q-table $Q(s,a) \leftarrow 0$ for all states $s$ and
   actions $a$
2  for episode $= 1$ to $N$ do
3      Reset environment; observe initial state $s$
4      done $\leftarrow$ False, reward $\leftarrow 0$
5      while not done do
6          Get all valid actions $A$ for state $s$
7          if random number $< \epsilon$ then
8              Choose random action $a \in A$ ;
                // Exploration
9          else
10             Choose action $a = \arg\max_{a'} Q(s,a')$ ;
                // Exploitation
11         Execute action $a$, observe next state $s'$, reward
            $r$, and done flag
           // Compute max future Q-value
12         $Q_{\text{max}} \leftarrow \max_{a'} Q(s',a')$
           // Update Q-table using Bellman
              equation
13         $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot Q_{\text{max}} - Q(s,a)]$
14         $s \leftarrow s'$
15     Decay $\epsilon$: $\epsilon \leftarrow \max(\epsilon \cdot \epsilon_{\text{decay}}, \epsilon_{\min})$
16 return Q-table
```

IV. Path Planning Using A* Algorithm

Once drones have scanned the environment and stitched together a global map of safe and blocked cells, the system uses the **A* (A-star) algorithm** to compute an efficient path from the starting point to a selected emergency location. A* is a widely used graph traversal and path search algorithm that combines the strengths of Dijkstra's algorithm and greedy best-first search.

Environment Representation

The disaster zone is modeled as a 2D grid where each cell is either:

- `UNEXPLORED` (-1): Areas not yet scanned by any drone.
- `OBSTACLE` (0): Blocked cells that cannot be traversed.
- `SAFE` (1): Cells confirmed as safe for travel.

As drones perform scans over the environment, their local data is stitched into a global map, updating previously unexplored regions. If conflicting data arises, `SAFE` cells are prioritized to ensure pathfinding flexibility.

Emergency Location and Goal Selection

After a sufficient portion of the grid has been scanned (e.g., more than 70% of the environment is known), the system selects an emergency location. This point is chosen from among the farthest `SAFE` cells from the origin (0, 0), based on Manhattan distance, to simulate a realistic search and rescue target.

A Pathfinding Mechanism*

A* searches for the shortest path by maintaining two cost values:

- $g(n)$: The actual cost from the start node to the current node n.
- $h(n)$: The heuristic estimate of the cost to reach the goal from n.

The total cost function is:

$$f(n) = g(n) + h(n)$$

In our implementation, the heuristic function $h(n)$ is the Manhattan distance:

$$h(n) = |x_1 - x_2| + |y_1 - y_2|$$

The algorithm uses a priority queue to explore the lowest $f(n)$-scoring node at each step. It avoids revisiting already evaluated cells and considers only valid neighboring cells marked as `SAFE`. If a path is found, it is returned in reverse by tracing back through a `came_from` map that stores each node's predecessor.

Algorithm 2: A* Pathfinding Algorithm

```
Input: Start cell $s$, Goal cell $g$, Grid with SAFE and OBSTACLE labels
Output: Path from $s$ to $g$, or failure if no path exists
1  Initialize open set as a priority queue with $s$
2  Initialize closed set as empty
3  Set $g(s) \leftarrow 0$
4  Set $f(s) \leftarrow h(s, g)$ ; // Heuristic (Manhattan distance)
5  while open set is not empty do
6      $current \leftarrow$ node in open set with lowest $f$ value
7      if $current = g$ then
8          return reconstructed path from $s$ to $g$
9      Remove $current$ from open set
10     Add $current$ to closed set
11     foreach neighbor of current do
12         if neighbor is not in bounds or not SAFE or in closed set then
13             continue
14         $tentative\_g \leftarrow g(current) + 1$ ;   // Cost of 1 per move
15         if neighbor not in open set or $tentative\_g < g(neighbor)$ then
16             $came\_from(neighbor) \leftarrow current$
17             $g(neighbor) \leftarrow tentative\_g$
18             $f(neighbor) \leftarrow g(neighbor) + h(neighbor, g)$
19             Add neighbor to open set if not already in it
20 return failure (no path found)
```

Outcome

Once the path is successfully computed, it is displayed over the scanned map. The system highlights the origin, the emergency location, and the computed path. If no viable path exists (e.g., due to disconnected safe regions), the algorithm reports failure, though this was rare in tested environments with sufficient scan coverage.

The A* algorithm ensures efficient and accurate navigation, making it a critical component of our drone-based relief system.

V. Implementation

To evaluate the performance of the proposed Q-learning-based drone deployment strategy, we conducted simulations across a range of grid sizes and configurations. This section outlines the setup used for training, environment parameters, and evaluation procedures.

A. Grid and Simulation Configuration

Experiments were performed on square grids ranging from 7×7 to 20×20. For each grid:

- The maximum number of drones was set as $\max(2 \times \text{grid size}, \text{grid size}^2/4)$.

- Each training run consisted of 20,000 episodes.
- Drone sizes used were 3×3 and 5×5.
- Obstacle percentage was set to 0 during training to focus on placement strategy.

Each training instance produced a Q-table and a JSON file capturing the final drone placements, rewards, and coverage statistics.

B. Environment Behavior

The training environment was implemented using a custom-built simulator called `DroneCoverageEnvAdaptiveLimited`, which models the disaster zone as a two-dimensional grid. The environment supports random spawning of drones during training and optimized placements during evaluation. The system ensures 100% coverage by continuing drone placement until the entire traversable area is covered.

A scalar reward signal guides the agent, incorporating multiple components:

$$\begin{aligned} R = c \cdot \text{Coverage Fraction} \\ - \alpha \cdot \text{Overlap Fraction} \\ - \beta \cdot \text{Uncovered Fraction} \\ - \gamma \cdot \text{Number of Active Drones} \end{aligned} \tag{1}$$

where c, α, β, and γ are configurable constants that control the relative importance of each component.

C. Q-Learning Configuration

The agent's learning configuration was as follows:

- Learning rate $\alpha = 0.1$
- Discount factor $\gamma = 0.95$
- Initial exploration rate $\epsilon = 1.0$
- Minimum exploration rate $\epsilon_{\text{min}} = 0.05$
- Exploration decay factor = 0.995 per episode

The agent chose actions using an ϵ-greedy strategy, balancing exploration of new actions and exploitation of learned policies. Available actions included spawning new drones, maintaining or removing existing ones, or performing no operation.

D. Evaluation Procedure

After training, the Q-table was evaluated in test mode using a purely greedy policy. If full coverage was not achieved, the system automatically added drones to high-value positions until 100% coverage was confirmed. Each evaluation logged the total number of drones used, final reward, and overall coverage percentage.

E. Libraries

Our simulation and training system is built in Python and uses several commonly available libraries to handle learning, visualization, and file management. Below is a summary of the key libraries and their roles in our project:

- **NumPy:** Used for handling grid data and doing fast calculations. We store the environment, drone positions, and scanned areas in 2D arrays using NumPy.
- **Random:** Helps add randomness during training. It is used to place drones randomly, choose actions for exploration, and create obstacles on the grid.
- **PyGame:** This library is used to display the grid visually. It shows drones, covered areas, blocked cells, and the path found by the A* algorithm. It also allows interaction, such as pausing or restarting the simulation.
- **Matplotlib:** Used to plot graphs like training reward over time or training time vs. grid size. These graphs help us understand how the system is performing.
- **Pickle and JSON:** `pickle` is used to save the learned Q-values so we can reuse them without training again. `json` is used to save the final positions of the drones and obstacle details for visualization.
- **Time and OS:** `time` is used to measure how long training or evaluation takes. `os` helps manage file paths and organize result files.
- **Heapq:** This library provides a priority queue for the A* pathfinding algorithm. It helps pick the next best cell to move to during pathfinding.
- **Sys:** This allows us to pass inputs like grid size directly from the command line when running the simulation.

Together, these libraries help us build a system that is efficient, easy to visualize, and simple to modify for future improvements.

VI. Output

After running the simulation for 20,000 episodes on each grid size ranging from 7×7 to 20×20, the system generates two output files: a pickle file containing the learned Q-table and a JSON file recording the drone positions that yielded the highest reward for that specific grid. The JSON file is then used to deploy the optimal set of drones onto the grid for evaluation and visualization.

Initial grid, *Fig.1* view showing scanned areas after partial drone placement. Dark-shaded cells represent blockages, while light-shaded cells indicate traversable paths.

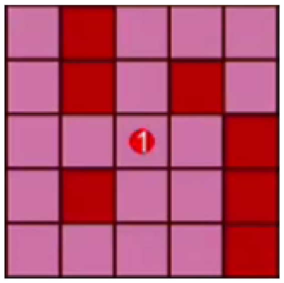

Fig. 1. Zoomed in picture of first drone placement

Final grid, *Fig. 2* shows the state after all drones are placed. The grid shows complete coverage of the safe area and blocked cells, with drones positioned to minimize overlap and maximize exploration.

On observing the training process over varying grid sizes, it was noted that the time required for convergence increases approximately proportionally with the grid dimensions. This

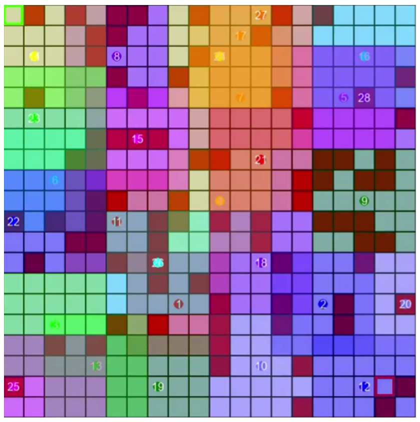

Fig. 2. Final grid after placing all the drones

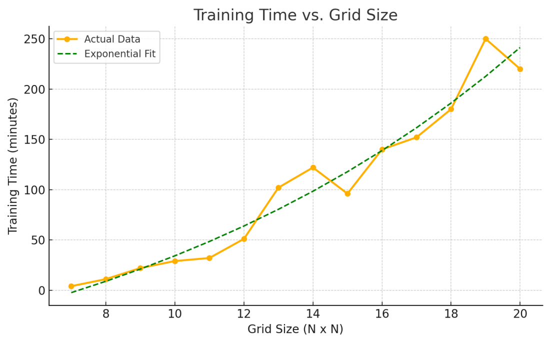

Fig. 3. Training time versus grid size with exponential trend.

increase is primarily due to the exponential growth in the state space and the number of possible drone placements as the grid expands. Larger grids result in more potential positions, greater coverage area to evaluate, and a higher number of drone configurations to explore, all of which contribute to longer training durations and slower convergence of the Q-learning algorithm.

VII. Contributions

VIII. Future Scope

Real-World Constraints: Future environments will model battery limitations, line-of-sight coverage, and environmental dynamics (e.g., wind or shifting obstacles) to reflect operational challenges in disaster zones.

3D Navigation: Extending the grid to multiple altitude layers or full 3D voxel maps will allow for more realistic drone placement and maneuvering strategies.

Multi-Agent Coordination: As drone counts increase, decentralized multi-agent reinforcement learning (MARL) can replace the single-agent setup, allowing each drone to learn a local policy while cooperating toward global coverage.

Multi-Objective Optimization: Incorporating additional objectives such as energy usage, flight time, and communication constraints can help balance coverage efficiency with operational sustainability.

Real Map Integration: Integrating GIS or LiDAR data will allow testing in realistic disaster environments, improving the robustness of learned policies.

Acknowledgment

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4] K. Elissa, "Title of paper if known," unpublished.
[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.