“PATH PLANNING OF DRONE FOR DATA COLLECTION IN A SMART FARM”

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING
(Specialisation in Internet of Things)

Submitted by
Sri Pranati Neelam (VU21CSEN0600013)
Vamsi V (VU21CSEN0600106)
K Dhanush (VU21CSEN0600121)
M.Ram Sagar (VU21CSEN0600003)

Under the esteemed guidance of
Dr. Sapna Jha
Assistant Professor, Computer Science Engineering, GST, VSP

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE
GITAM SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)
VISAKHAPATNAM
2024

DECLARATION

I hereby declare that the project report entitled **"Path planning of drone for data collection in a smart farm"** is an original work done in the Department of Artificial Intelligence and Data Science, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering (IoT). The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 22-10-2024

Registration No(s)	Name(s)	Signature
VU21CSEN0600013	Sri Pranati Neelam	
VU21CSEN0600106	Vamsi V	
VU21CSEN0600121	K Dhanush	
VU21CSEN0600003	M Ram Sagar	

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled “**Path planning of drone for data collection in a smart farm**” is a bonafide record of work carried out by Student Name (Regd. No.), Student Name(Regd. No.), Student Name(Regd. No.), Student Name(Regd. No.) students submitted in partial fulfillment of the requirement for the award of the degree of Bachelors of Technology in Computer Science and Engineering (IoT).

Date: 22-10-2024

Project Guide	Head of the Department
Dr. Sapna Jha	Dr. Kuppili Naveen Kumar

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to everyone who contributed to successfully completing our final year project on "Path Planning of Drones for Data Collection in Smart Farms."

Our deepest thanks go to the GITAM School of Technology and the Department of Artificial Intelligence and Data Science for providing the resources, support and platform to work on this project. The opportunity to work on technologies like IoT and AI in agriculture has been insightful.

We thank Prof. Kuppili Naveen Kumar, Associate Professor and Head of the Department of Artificial Intelligence and Data Science, for his encouragement and support for this venture. His confidence in our abilities has infused a lot of motivation in us.

We are especially grateful to our guide, Dr. Sapna Jha, Assistant Professor, Department of CSE, for her constant guidance, feedback and patience. Her expert insights in the areas of wireless networks and UAVs were critical to the successful completion of this project.

We especially thank Prof. Nagendra Prasad, Director and Prof. S Arun Kumar, Dean-CSE, for their splendid support and motivation towards completing this project. We appreciate the guidance and leadership they provided.

Ultimately, we thank our team members, friends and family for standing by our side and providing continuous moral support, encouragement and help.

This experience has been an enriching journey and the knowledge and skills gained will significantly enhance our professional growth in the future.

TABLE OF CONTENTS

ABSTRACT

The objective of the project is to implement an algorithm using reinforcement learning for Unmanned Aerial Vehicles (UAVs) in smart farming that will be able to determine the most suitable areas and times for the UAVs to stop for sojourns. UAVs are used in smart farming systems across different applications for the implementation of a broad scope of data gathering and monitoring activities ranging from the evaluation of crop status to moisture content in soil. This project offers a solution to the challenge of dynamic path planning through sensor node deployment within the farm with the aim of creating a sensor node network topology that is recurrently routed to monitor environmental change in real-time.

Raspberry Pis attached to drones are installed to perform edge computing and sensor data collecting, thus empowering the drones with the capacity to make better decisions autonomously without the need to be connected to the server for an extended duration. Latency presents challenges to UAV operations and this problem can be solved by implementing edge computing. With the help of Reinforcement Learning (RL), the aerial vehicles understand and neural networks optimise a set of routes over time for each UAV based on seasonal, harvest, and weather conditions. This approach increases the efficiency of data generation while minimising the cost of flying time and the energy used.

Updating flight paths with real-time data and learning experiences optimises resource usage and scaling. Then, the proposed method would offer an efficient solution to farm operations management and reduce dependency on fixed infrastructure; hence, it is ideal for large-scale and distributed farm environments where conditions are not favourable.

INTRODUCTION

The principle of smart farming, or precision agriculture, is reorienting the conventional agricultural paradigm. The use and integration of emerging technology, including artificial intelligence (AI), machine learning (ML), automated systems, and the Internet of Things (IoT), is enabling agriculture to be more efficient and sustainable. It allows farmers to grow more food, save valuable resources such as water and fertilisers, and enables environmentally friendly practices for future sustainability of agriculture. Smart farming uses several technologies to monitor, manage, and optimize agronomic production. Data analytics, predictive modeling, and automation transform raw data from the farm into actionable insights. For instance, sensors can give readings on soil moisture and nutrient content and drones and satellite imagery can give real-time data regarding crop health. Such technologies enable the potential deployment of resources in a more accurate way, possibly with much less waste and much better efficiency.

Drones enter as essential precision tools for farming. In various ways, drones amplify these advantages and have now entered importantly to enhance farm management effectiveness. These unmanned aerial systems are able to gather vast farm-specific data with their payload devices, such as high-resolution cameras or multispectral sensors over vast areas rapidly. Above all, the precise information about crop health, soil conditions, pest infestations, or variations in localised environmental factors like temperatures and humidity levels emerge with drones. Such information helps farmers respond quickly to any problem and lead to healthier crops with better yields.

Some of the specific roles of drones in smart farming include the following:

- **Crop Health Monitoring:** Drones can detect stress in plants that may not be visible to the naked eye by capturing multispectral and thermal images.
- **Soil Health Monitoring:** Drones can evaluate the health of the soil, detecting nutrient deficiencies or poor moisture levels in specific areas.
- **Automated Irrigation and Fertilisation:** Several drones work in tandem with the actuators on farms in order to regulate automatic systems of irrigation and fertilisers.
- **Data Collection and Processing:** Drones are installed with cameras and sensors that collect massive amounts of data from sensor nodes placed in the field. Advanced drones even use edge computing where onboard processing units, like the Raspberry Pi, process data in real time. This approach limits the volume of data transmission, thereby making the system more efficient and allowing real-time actions based on findings.

Importance of Path Planning in Drones for Smart Farming

Path planning is among the most important areas for deploying drones in precision agriculture because it defines the way in which a drone is able to gather data over the span of a farm. Optimal flight path maximizes the data gathered by the drone while saving energy, which becomes a highly important factor in the case of large-scale farms wherein the mission of the drone will have to cover a lot of distance and frequently need to halt at sensor nodes.

The drones must be flown along perfectly scheduled routes in such a manner that complete and true data is being collected. Poorly planned routes lead to issues such as these:

- **Energy Consumption**: Wasted battery energy on unnecessary backtracking or redundant travels through the same area numerous times. This actually decreases the time the drone spends in the air and leads to the need to recharge the battery more often.

- **Data Redundancy**: In the recurrence of flying over places, they collect redundant data that is unnecessary, hence prolonging the time taken and resource-intensive processing.
- **Loss of Data Collection**: Here the issue lies with the inefficient designing of the routes; this opens the most important areas unscanned or sometimes ignores crucial data in inquiry. It lowers the overall quality of data collected and can even produce a wrong estimate.

With effective path planning, a drone will travel through all of the sensor nodes with minimal energy expenditure for high-quality, non-duplicate data collection. Path planning enables it to avoid obstacles, avoid hazardous zones, and execute a mission in the shortest amount of time. Additionally, path planning becomes essential with sensor nodes in low power mode in order to conserve their batteries; minimizing a path reduces the time spent by each sensor node being active, conserving more power.

Challenges in Path Planning for Drones in Large-Scale Farms

Large farms pose a unique challenge to drones in data gathering. Large farms are extremely difficult due to their size, heterogeneity in topography, and potential for obstacles. The path planning must deal with the following.

- **Obstacle Avoidance:** The drone should be capable of sensing and navigating at physical obstacles like trees, equipment, and buildings. On smart farms, the path should be well planned to reduce collision hazards and maintain continuous data collection without any disruption.
- **Dynamic Environment:** Dynamic data requirements may vary based on the environment, crop condition, and development stages. Path planning must adapt accordingly; e.g., the drone must return to key areas or go to a particular point under specific conditions.
- **Energy Limitations:** The batteries in the drones are limited in capacities. Their recharge and therefore their lifetimes to perform gigantic missions must be carefully managed. Such optimal routes for the mission ensure that it takes minimal energy usage so that downtime is kept as low as possible as well as to improve its capabilities during the drone operations.

This project will employ reinforcement learning with deep Q-network algorithms to have the drone learn how to accommodate the conditions on the farm in order to select the most suitable stop points and stop times to gather basic data without consuming much energy. This cutting-edge technology makes it possible for drones to work with large farm operations and flexible demands for data in order to achieve precision agriculture.

LITERATURE REVIEW

The existing algorithms for Path Planning of UAVs[1] can be broadly classified into:

- Algorithmic-level Path Planning:
    - Traditional Algorithms
    - Intelligent Algorithms
    - RL Based Algorithms
    - Hybrid Algorithms
- Functional-level Path Planning:
    - Space-Based Algorithms
    - Task-Based Algorithms

Table 1: Traditional Algorithms

Reference Number	Title	Method	Limitation
[2]	Bi-Directional Adaptive A Algorithm Toward Optimal Path Planning for Large-Scale UAV Under Multi-Constraints	The Bi-Directional Adaptive A* Algorithm improves pathfinding with directional search, adaptive step size, evaluation functions, and a re-wiring process to remove duplicate nodes.	It is difficult to ensure node expansion efficiency and optimally avoid obstacles, especially in a large, complex environment.
[3]	Mutual Information-Based Multi-AUV Path Planning for Scalar Field Sampling Using Multidimensional RRT	The proposed approach implements the selective basis function Kalman filter for scalar field estimation. It utilises the multidimensional RRT* algorithm to find informative sampling locations that maximise mutual information between the model and sensor observations.	The more Gaussian functions used in this methodology, the more complex it is, and thus, its computational costs in sampling and path planning are very high.
[4]	Analysis of Probabilistic Roadmaps for Path Planning	It discusses the probabilistic roadmap method of connecting configurations with estimates of failure probability based on path length, distance from obstacles, and number of nodes. After that, it proposes a	It assumes the path and properties, making it practical for only a few applications because failure probabilities depend on the path and minimum

		simplified probabilistic roadmap, namely s-PRM.	distance from obstacles.
[5]	Control of Multiple UAVs for Persistent Surveillance: Algorithm and Results of Flight Tests	A semi-heuristic control policy that is designed with a reactive multi-agent policy coordinated for UAVs. Simulation and flight tests are used to evaluate the performance of UAVs under endurance and dynamic constraints.	The simplified dynamic model does not focus on heterogeneous coordination.

Table 2: Swam Intelligence Path-Planning Algorithms

Reference Number	Title	Method	Limitation
[6]	Research on path planning Algorithm for Multi-UAV Maritime Targets Search Based on Genetic Algorithm.	Development of a sparse CBS-D* algorithm for cooperative dynamic path planning that has rapid re-planning in case of sudden obstacles.	Need help with optimal pathfinding through highly complex or rapidly changing environments.
[7]	Safety-enhanced UAV path planning with spherical vector-based particle swarm optimization	Spherical Vector-based Particle Swarm Optimisation - a generic variant of particle swarm optimisation to optimise UAV paths about safety and feasibility constraints.	In most cases, SPSO suffers from premature convergence that results in suboptimal solutions.

Table 3: AI Path-Planning Algorithms

Reference Number	Title	Method	Limitation
[8]	Remote UAV Online path planning via Neural Network-Based Opportunistic Control	Neural network (NN) aids remote UAV control algorithm solution of the Hamilton-Jacobi-Bellman equation in real time.	Poor performance if the channel is adverse or the UAV is very far from the base station.

[9]	Unmanned Aerial Vehicles path planning for area monitoring.	The proposed two-step approach: First, cells should be assigned to UAVs via cluster formation or coalition game; second, UAV paths would be optimised using heuristics.	Coalition games tend to become computationally expensive as the area size increases.
[10]	A Deep Learning Trained by Genetic Algorithm to Improve the Efficiency of Path Planning for Data Collection with Multi-UAV	Deep Learning Genetic Algorithm (DL-GA): genetic algorithm along with deep learning for optimised rapid path generation	Performances vary with the changing environment as well as number of nodes available.

Table 4: Hybrid Path-Planning Algorithms

Reference Number	Title	Method	Limitation
[11]	An improved UAV path planning method based on RRT-APF hybrid strategy	The paper presents a hybrid UAV path planning method using RRT for path exploration and APF for obstacle avoidance. It addresses issues like RRT's randomness and APF's local minima.	The challenges include real-time obstacle detection and fine-tuning APF to avoid oscillations.
[12]	A Hybrid Path Planning Method in Unmanned Air/Ground Vehicle (UAV/UGV) Cooperative Systems	An adaptive path planning method for the UAV/UGV systems is developed. The UAV includes denoising, correction, and obstacle recognition features in most calculations. The genetic algorithm handles global path planning, with the LRO optimising it further based on updated environmental data.	Major challenges in adopting this or similar methods are scalability in dynamic environments, the weather's influence on image recognition, and delay due to the life cycle of obstacle recognition and mapping algorithms.

Table 5: Task-Based Algorithms

Reference	Title	Method	Limitation

Num ber			
[15]	Travelling salesman problem for UAV path planning.	Utilisation of Improved Genetic Algorithm (IGA) and Particle-Swarm-Optimisation-based Ant Colony Optimisation (PSO-ACO) to find optimal paths for UAVs.	Proposed the Weighted Targets Sweep Coverage (WTSC) algorithm to optimise task time and coverage rate by considering the target.
[16]	A Path Planning Method for Sweep Coverage with Multiple UAVs	Proposed the Weighted Targets Sweep Coverage (WTSC) algorithm to optimise task time and coverage rate by considering target weights.	The algorithm's performance may decrease with an extremely high number of targets due to UAV flight time constraints.

Table 6: RL Path-Planning Algorithms

Reference Number	Title	Method	Limitation
[17]	Deep Reinforcement Learning for UAV Navigation Through Massive MIMO Technique	Uses Deep Q-Network (DQN) for UAV navigation optimisation via massive MIMO technology. - DQN selects optimal UAV locations based on received signal strengths to maximise coverage and reduce energy consumption.	Performance may degrade in real-world environments with dynamic obstacles and environmental changes, primarily indoors.
[18]	Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach	Deep Reinforcement Learning (DRL) framework using a modified actor-critic method to solve the Partially Observable Markov Decision Process (POMDP) for UAV navigation.	Heavy reliance on fine-tuning the reward function. Performance could degrade in environments with more dynamic or densely packed obstacles.
[19]	Path Planning for UAV Ground Target Tracking via Deep Reinforcement	Uses an improved Deep Deterministic Policy Gradient (DDPG) algorithm for UAV	Performance depends on the careful design of the

	Learning.	path planning, combined with Long Short-Term Memory (LSTM) networks for state approximation. The system includes a reward function based on line of sight and artificial potential field, with penalty terms for action to smooth the trajectory.	reward function, and model effectiveness is limited by partially observable environments.

Research Gap:

- Due to the highly dynamic environment, Reinforcement Learning (RL)-based algorithms are well-suited for path planning.
- However, existing research must address stop time and optimal locations for drones at sensor nodes.
- Current algorithms focus on static path planning, lacking adaptability to dynamic sensor node topologies.
- The current RL-based algorithms failed in truly dynamic environments, as they fell short in determining optimal sojourn locations and stop times at each sensor node.
- The dynamic sensor node topologies are with respect to the changes in node energy levels and data available in the nodes that are present in the nodes.
- Without careful consideration of stop times at sensor nodes, data collection may be inefficient, leading to missed data or unnecessary energy consumption.
- Heavy reliance on external servers for data processing introduces latency and limits real-time decision-making.

PROBLEM IDENTIFICATION AND OBJECTIVES

In the smart farming context, UAVs are extensively used as data collectors to get farm insights. The efficiency of data collected by the drone depends on the flight path and the sojourn locations. Current path-planning techniques often focus on navigating drones efficiently through static or predetermined paths. However, these methods must be revised to adapt to the dynamic nature of farms where environmental conditions, sensor node status, and data collection needs can change in real time.

A drone operating without a considered path would face several challenges, including high energy consumption, reduced flight time and inaccurate data. With an optimised path, the drones can avoid getting damaged, stopping at sensor nodes for inappropriate time durations, missing critical areas and even gathering redundant data by flying over the same area multiple times.

Therefore, a strategy is required to enable the drones to collect high-quality data from sensor nodes across the farm with minimal resource usage and in the shortest time possible to save energy of the sensor nodes and drones.

The project aims to identify the optimal path for drones in a smart farm; specifically, the drone's stop time and sojourn location will be identified.

By implementing optimal path planning,

- **Data Accuracy:** Relevant and updated data is gathered without redundancy.
- **Resource Efficiency:** Minimising energy consumption for sensor nodes and drones.
- **Time Efficiency:** Reducing the time required for drones to complete data collection.
- **Enhanced Crop Management:** Timely and accurate data of the farm can improve farm management and decision-making.

OBJECTIVES:

The primary goal of this project is to develop an optimal path-planning algorithm for drones in smart farming that dynamically determines sojourn locations and stop durations based on the data to be collected.

1. **Develop a Reinforcement Learning-based Path Planning Algorithm:** Use reinforcement learning (RL) to enable drones to learn and adapt their flight paths based on real-time conditions in the farm.

2. **Determine Optimal Sojourn Locations and Durations:** The drone should be able to dynamically select the sojourn locations and manage the stop duration to maximise data collection from sensor nodes.

3. **Improve Resource Efficiency:** The drone should be able to reduce flight time to get good quality data and save energy and time for the drone and sensor nodes.

EXISTING SYSTEM

The A* algorithm is a widely used pathfinding and graph traversal method known for its efficiency in finding the shortest path between nodes. Developed to improve upon the Dijkstra and Greedy Best-First search algorithms, A* combines the benefits of both approaches, balancing the cost from the starting node and the estimated cost to the target node.

A* operates by exploring a graph or grid, where each position is considered a "node" with connections or "edges" to neighbouring nodes. By calculating the actual cost to reach a node and an estimated cost from the node to the target, A* can make informed choices that lead to optimal and efficient pathfinding. This balance is achieved through the algorithm's evaluation function, which makes A* especially suited for static environments with defined start and end points.

The A* algorithm uses an evaluation function(cost function), f(n) , defined as:

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

where: $g(n)$ is the actual cost from the start node to the current node n.

$h(n)$ is the heuristic estimate of the cost from n to the goal node.

MAIN ELEMENTS OF A* ALGORITHM

- **Graph Representation:** The environment is structured as a grid or graph where nodes represent possible positions, such as waypoints for an Unmanned Aerial Vehicle (UAV). Connections between nodes (edges) represent paths that can be taken between these positions, including cost and distance factors.
- **Open and Closed Lists:**
  - Open List: This list contains nodes that are yet to be evaluated. Nodes with the lowest $f(n)$ value are prioritised, helping explore the most promising paths.
  - Closed List: Once a node is thoroughly evaluated, it is added to the closed list, ensuring that nodes are not revisited.
- **Node Evaluation Process:** Nodes are evaluated based on $f(n)$ values. Selecting nodes with the lowest $f(n)$ values ensures that the promising nodes are explored first. This involves calculating $g(n)$ and $h(n)$ for each neighbouring node and updating its parent if a lower-cost path is found.
- **Path Reconstruction:** When the goal node is reached, it reconstructs the path by backtracking through the parent pointers from the goal to the start node and ensuring the shortest, most efficient path based on the combined cost function.
- **Handling Constraints**: The A* algorithm can be modified to include specific constraints, such as altitude for UAVs or proximity to obstacles.

LIMITATIONS OF A* ALGORITHM

- **Static Assumptions:** A* assumes a static environment where obstacles and elements remain constant. With moving barriers, A* struggles as it requires re-evaluating the entire path.

- **Re-computation Overhead:** When an obstacle appears in an already computed path, A* needs to start over or re-compute the path, which is time-intensive and computationally expensive.
- **Heuristic Limitations:** The effectiveness of A* depends heavily on the heuristic function. An accurate heuristic can lead to efficient paths and sufficient computation.
- **Limited Adaptability:** A* is designed to find a single optimal path rather than adaptively responding to environmental changes. It lacks the mechanisms to alter the path dynamically without restarting.
- In expansive or highly detailed environments, the number of nodes and potential connections increase significantly, leading to higher memory usage and computation time.

PROPOSED SYSTEM

REINFORCEMENT LEARNING

Reinforcement Learning is a type of machine learning where an agent (drone) learns to make decisions by interacting with an environment to maximise cumulative rewards. RL does not require labelled data. Instead, the agent learns from the outcomes of its actions over time, adjusting its behaviour to optimise its rewards.

MAIN ELEMENTS:

- **Agent:** The drone takes action in the environment.
- **Environment:** The system the agent interacts with, including sensor nodes, the base station and energy constraints.
- **State:** The current status of the environment as perceived by the agent. This includes the drone's position, battery level and amount of data at each sensor node.
- **Actions:** The agent can make possible decisions at any state, such as moving to a node or hovering for data collection.
- **Reward:** Feedback received by the agent after taking an action. Positive rewards encourage desirable actions (e.g., data collection) and penalties discourage undesirable actions (e.g., unnecessary movement or battery wastage).
- **Policy:** The strategy that the agent develops, mapping states to actions to maximise the expected reward over time.

PROXIMAL POLICY OPTIMIZATION (PPO) FOR UAV NAVIGATION

Proximal Policy Optimization (PPO) is the RL algorithm used in this project. PPO is well-suited for continuous environments and efficiently handles path optimization problems.

PPO Process in UAV Path Planning:

- **Policy Network:** A neural network that maps the UAV's current state to an optimal action.
- **Exploration & Experience Collection:** The UAV explores different paths, records data, and updates its policy based on rewards.
- **Advantage Estimation:** The algorithm determines how much better an action is compared to the expected baseline.
- **Policy Update:** PPO updates the policy iteratively to improve decision-making while avoiding drastic changes that can destabilize learning.

Unlike traditional path-planning algorithms, PPO dynamically adjusts the UAV's route based on real-time constraints such as battery levels and changing sensor node locations.

TRAINING THE RL AGENT (DRONE)

The UAV undergoes a training phase where it learns optimal movement strategies:

- **Initialisation:** The UAV starts with a random policy and explores different flight paths.
- **Data Collection:** The UAV collects data by visiting cluster stops determined by DBSCAN.
- **Reward Optimisation:** The policy is updated to favour paths with minimal flight time and optimal data collection.
- **Continuous Learning:** The UAV refines its strategy through multiple iterations, ensuring efficient path planning under varying conditions.

ACTIONS:

- Move to a cluster stop location.
- Collect data from sensor nodes.
- Return to the base station.

REWARDS:

- Positive Rewards:
    - Efficiently visiting cluster stops.
    - Collecting maximum data with minimal hover time.
    - Conserving battery while completing tasks.
- Negative Rewards (Penalties):
    - Unnecessary movement or revisiting the same stop.
    - Hovering longer than needed at a stop.
    - Draining the battery below the threshold before reaching the base.

STATE SPACE:

- UAV Position: The current coordinates of the UAV in the 30x30 grid.
- Battery Level: Remaining energy, affecting travel speed when low.
- Data at Sensor Nodes: The amount of data still available at each node.
- Distance to Cluster Stops: The UAV's proximity to the following stop location.

METHODOLOGY

A simulation code based on advanced planning and methodology utilises optimisation algorithms that enhance the design of flight routes and decrease the energy demand of the UAV while collecting data from distributed nodes in the desired environment. These techniques constitute the essential components that will follow when executing the code later

1. Reinforcement Learning (RL) using Proximal Policy Optimization (PPO)Algorithm Utilised:

Proximal Policy Optimization (PPO) in Stable-Baselines3 Target Establish autonomous operational capability for the UAV which will facilitate path planning and decision making for data collection while supplanting energy.

Steps:1 Environment Setup: Open AI Gym permits the development of an environment. This environment replicates all the UAV operation, including flight patterns to replace energy requirements to reach target nodes.The observation space of the UAV has its current position information, along with the locations of sensor nodes and the amount of data being returned as well as the battery level remaining.The actions available to the UAV within that environment are the actual movements to specify the next place or direction to stop as the UAV is flying.

2. Training the PPO model: The training of PPO uses the following defined hyperparameters:
• Learning rate: 0.0001 (controls the optimization step size).

• Discount factor (gamma): 0.99 (the relative weight between immediate and future rewards).Each update uses a sample size of 256 per the batch size.

• Total timesteps: 5,000,000 (the total interactions with the environment).

3. Inference and Path Planning: The PPO model codes cognizant actions for the UAV during each step of the simulation for planned navigation while collecting data in a battery efficient manner after the

IMPLEMENTATION OF DBSCAN:

To help the UAV visit essential stops for data collection efficiently, the research aims to cluster groups of sensor nodes.

1. **Clustering Sensor Nodes:** Based on the following parameters, DBSCAN will give clustering to the sensor nodes.The eps parameter refers to the UAV communication range (self.uav_range), which is used to cluster the nearby nodes.Min samples set to the value of 1 allows clusters with a single node only.The technique clusters nodes based on proximity in space and thus reduces the stops for the UAV for collection of data
2. **Weighted Centroid Computation:** This method computes the weighted centroid of the cluster according to the total amount of data collected in each node. The operator stops the UAV close to the nodes with more data so that the most data can be collected. The UAV operator will stop the UAV at the centroid nodes that are accessible to the

UAV's operational range.laces stops at the centroid nodes that the UAV can access in its operational range.

3. **Optimized Travelling Salesman Problem (TSP):** The TSP algorithm is applied to optimize the flight path of the UAV. The TSP algorithm determines the shortest path that goes through all the sensor nodes and returns to the base station, minimizing the distance and energy loss of the UAV.
4. **Step-1:Distance Matrix Calculation:** Distance matrix calculation:To create a distance matrix, we use the function scipy.spatial.distance.cdist(), which calculates the euclidean distance between the base station and all other stops, including all pairs of stops.

STEP 2: Path Planning:

An optimized TSP solver such as Google OR-Tools or concorde is applied to determine the most efficient route.The UAV starts at the base station, follows the computed optimal path to visit each stop exactly once, and then returns to the base. By utilizing TSP solver (which is globally optimum), you will get better paths to the base station in comparison to Nearest Neighbor (NN) approach. In fact, at higher density of nodes, the routes are better in terms of battery consumption as well as travelling time.

4.Battery and Flight Time Management :

BATTERY CONSUMPTION:

- **TRAVEL BATTERY CONSUMPTION:** The battery disposes directly in proportion to distance travelled (password_player_distance).
- **Stop Battery Consumption:** The battery power reduces during periods when the UAV stops for data transfer operations (battery_per_time).
- **Low Battery consumption:** A reduction in speed occurs when the battery reaches low_battery_threshold to allow energy conservation and safe base station return.

5.**Gymnasium-based UAV Enviroment for simulation:**

Technologies:

The OpenAI Gym framework allows users to create custom reinforcement learning settings within which developers create custom reinforcement learning settings. The UAVPathPlanningEnv class extends Gym's Env class to simulate the UAV's movement, data collection, and energy consumption process. The language used to develop the logic structure made up of observation room and action option, and rewards determination is Python.

6. **Data Tracking and Visualization:**

Technologies

The Python library Matplotlib enables users to generate both static and interactive visualizations as well as animated charts through its functionality. The 2D grid draws the UAV flight path along with its stopping points and sensor nodes as through its plotting features.

Pandas:A library for data manipulation and analysis is pandas. This tool helps to monitor the performance of the simulations by logging the UAV flight path, visited stopping points, data collected, time taken and power used.

SYSTEM ARCHITECTURE

1. KEY COMPONENTS:

- **Sensor Nodes:** Deployed in a 30x30 grid, each node stores environmental data and transmits it when required.
- **UAV (RL Agent):** Acts as an autonomous data collector, navigating using PPO reinforcement learning and optimising its path dynamically.
- **Clustering & Stop Selection (DBSCAN):** Groups sensor nodes and assigns optimal stop locations at cluster centroids.
- **Path Optimization (PPO):** Trains the UAV to minimise flight time and energy consumption while ensuring complete data collection.

2. DATA FLOW AND CONTROL:

- The UAV scans the farm, identifying sensor nodes that have stored data.
- DBSCAN clustering is applied to group nearby nodes, reducing unnecessary stops.
- A PPO-based reinforcement learning model determines the most efficient stop sequence, optimising travel time and battery consumption.
- The UAV adjusts its path based on real-time battery levels and remaining unvisited nodes.
- System feedback loops monitor collected data, flight time, and energy usage to improve future path planning continuously.

3. SYSTEM WORKFLOW:

- **Initialisation:** The UAV starts at the base station (0,0), aware of sensor node locations.
- **Clustering & Stop Selection:** DBSCAN groups nodes into clusters, assigning a single stop per cluster.
- **Path Planning & Data Collection:** The UAV follows the optimised path, stops at clusters, and collects data efficiently.
- **Energy-Aware Navigation:** If the battery < 40%, UAV slows down; if battery < 20%, it immediately returns to base.
- **Return to Base:** After visiting all required stops, UAV returns to base and offloads collected data.

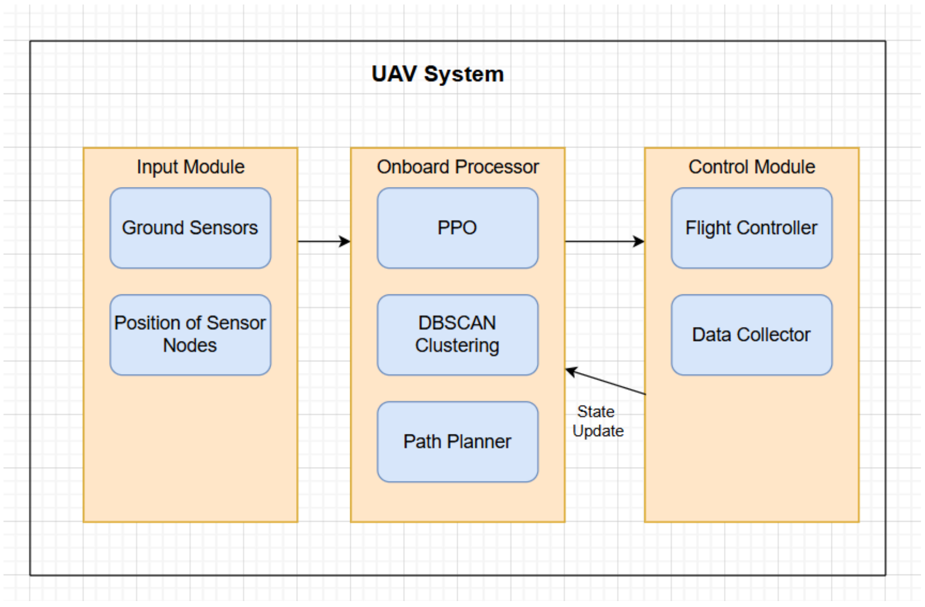


Image 2: Block Diagram

TOOLS AND TECHNOLOGY USED

1. **Programming Language**
- **Python 3.12**
  The code project is written and executed in Python.
2. **Development Environment**
- **Visual Studio Code (VS Code)**
  The entire project development environment, coding, and debugging were all done in Visual Studio Code (VS Code).
3. **ML and Reinforcement Learning Framework**
- **Stable-Baselines3 (SB3)**
  - Stable-Baselines3 (SB3), a suite of state-of-the-art RL algorithms written in Python based on PyTorch, were used in this project, specifically the Proximal Policy Optimization (PPO) implementation in SB3.
  - SB3 simplifies training and evaluating RL models while providing optimised implementations of popular algorithms.
- **Gym**
  - OpenAI Gym was used to create a custom UAV environment.
  - Gym provides a standard framework for training RL models and allows integration with Stable-Baselines3.
  - The UAV environment was designed as a custom Gym environment, defining the state space, action space, and rewards.
4. **Clustering & Optimization Tools**
- **Scikit-learn (DBSCAN)**
  - DBSCAN (Density-Based Spatial Clustering of Applications with Noise) groups sensor nodes into clusters.
  - It is a machine-learning algorithm that identifies natural groupings without specifying the number of clusters beforehand.
  - In this project, DBSCAN helped determine UAV stop locations by clustering sensor nodes based on proximity.
- **SciPy & NumPy**
  - SciPy (Spatial Distance Module) computed the shortest path between stop locations.
  - NumPy was used for efficient array and matrix operations throughout the project.
5. **Hardware Configuration**
- Lenovo Ideapad Pro 5 (32GB RAM)
  - The project was developed and trained on a Lenovo Ideapad Pro 5 laptop with 32GB RAM.
  - The large RAM capacity enabled efficient training of the RL model and handling of large computation tasks, such as clustering and path optimisation.
6. **Visualization & Data Analysis**
- **Matplotlib**
  - Used for plotting the UAV path, sensor node positions, and clusters in a 2D grid.
- **Pandas**
  - Used for managing and analyzing sensor node data and logs related to UAV movements.

IMPLEMENTATION

1. UAV Environment(uav_env.py)

This script defines the UAV's environment using OpenAI Gym and simulates the UAV's interaction with sensor nodes.

- **Sensor Node Deployment:** 20 sensor nodes are placed randomly in a 30x30 grid smart farm, each node storing a predefined amount of data.
- **Clustering & Stop Selection:** Uses DBSCAN clustering to form groups of nearby nodes and assigns an optimal stop location for each cluster, ensuring efficient data collection.
- **Path Optimisation:** Computes the shortest path, visiting all stop locations and returning to the base using a TSP (Traveling Salesman Problem) heuristic.
- **State & Action Space:** The UAV's state includes position, battery level, visited stops, and remaining data at nodes. The action space is discrete, representing movement between stop locations.
- **Rewards:** Encourages the UAV to minimize flight time and battery consumption while penalizing unnecessary movements and inefficient data collection.

Key Functions:

- **compute_cluster_stops():** Groups sensor nodes into clusters and assigns a stop location at each cluster centroid based on the node distribution.
- **get_optimized_path():** Computes the optimal stop sequence using a TSP approximation, minimizing total travel distance.
- **step():** Moves the UAV to the next stop, collects data from nodes in range, updates battery consumption, and calculates the reward for the current step.

This environment models a realistic UAV mission, incorporating dynamic battery consumption, node data transfer rates, and optimized navigation.

2. Training PPO Model (train_ppo.py)

This script trains the UAV policy using Proximal Policy Optimization (PPO), a reinforcement learning algorithm that optimizes path selection.

Key Steps:

- **Initialise Environment:** Creates multiple instances of the UAV environment using vectorized environments, enabling parallel learning.
- **Define PPO Model:** Configures hyperparameters such as learning rate (0.0001), batch size (256), discount factor ($\gamma = 0.99$), and entropy coefficient, ensuring stable training.
- **Train the Model:** The PPO agent interacts with the environment and learns the optimal path over 5 million timesteps, gradually improving efficiency in selecting stop locations.
- **Save the Model:** The trained policy is stored and can be reused for evaluation, reducing computation time when deploying the UAV in real-world scenarios.

The model learns to adaptively choose efficient paths, reducing travel time and battery drain while ensuring full data collection.

3. Evaluation (evaluate_ppo.py): This script tests the trained UAV model on new sensor node distributions and visualizes its performance metrics.

Key Steps:

- **Load the trained model and environment:** The saved PPO policy is loaded and deployed in the UAV environment.
- **Simulate UAV movement:** The UAV follows the optimized stop sequence while tracking its energy usage, flight time, and data collection efficiency.
- **Track performance metrics:** The system records total flight time, battery consumption per movement, time spent at each stop, and overall data collected.
- **Plot results:** The UAV path, visited stop locations, and sensor node positions are visualized to validate the path planning strategy.

The evaluation phase ensures the trained model performs efficiently across different node distributions and meets the design objectives.

TEST CASES

Testing Case 1: 10 Nodes in a 20×20 Grid

Objective: Verify that the UAV visits all sensor nodes efficiently.

Input:

10 sensor nodes randomly deployed in a 20x20 grid.

UAV starts at (0,0), clusters are computed using DBSCAN and PPO optimizes the route.

Expected Output:

- UAV visits all clusters.

- Completes data collection with minimal flight time.

- Returns to base before battery depletion.

Actual Output:

- All nodes were covered successfully.

- Battery usage remained within limits.

- UAV followed the shortest optimized path.

Result: PASS

Testing Case 2: Failed Case – 20 Nodes in a 20×20 Grid, One Node Missing

Objective: Identify a failure scenario where a node is left unvisited.

Input:

- 20 sensor nodes randomly placed in a 20x20 grid.

- UAV follows DBSCAN-based clustering and PPO-based path optimization.

Expected Output:

- UAV visits all nodes and completes data collection.

Actual Output:

- One node was left unvisited due to improper clustering.

- UAV returned to base without collecting data from that node.

Result: FAIL

Cause of Failure:

- DBSCAN misclassified the node, placing it outside a cluster.

- Possible Fix: Increase the DBSCAN epsilon (eps) or manually check cluster assignments.

Testing Case 3: Unexpected Node Deployment in a 40×40 Grid

Objective: Test the UAV's ability to adjust to unevenly spaced nodes.

Input:

- 30 sensor nodes randomly placed, some very close, others far apart.

- UAV follows DBSCAN clustering and PPO path optimization.

Expected Output:

- UAV efficiently adjusts cluster stops.

- Completes data collection with optimal flight path.

Actual Output:

- UAV adapted well to varying node distances.

- Successfully covered all nodes without unnecessary detours.

Result: PASS

Final Observations & Fixes:

- The algorithm works efficiently for most cases, but clustering can fail if nodes are isolated.
- Possible Fix: Dynamically adjust DBSCAN eps based on node density.
- Battery-aware decisions help in returning safely, even in large grid sizes.

RESULTS AND DISCUSSION

1. Evaluation Metrics

For the evaluation of the performance of the UAV model trained, we look at the following significant metrics:

1. **Total Flight Time** – It calculates the entire time that the UAV will take to reach all clusters, collect data, and return to the base.
2. **Battery Consumption** – It tracks battery drain over time with consideration of travel distance and duration at every site.
3. **Efficiency of Data Collection** – Ensures that the UAV collects 100% of data from all the sensor nodes.
4. **Path Optimization** – Examines how efficiently the UAV moves between the clusters and whether it is along an optimal stop path.
5. **Stop Time at Every Location** – Monitors how long the UAV spends at every cluster, offloading data.
6. **Impact of RL Training** – Compares the RL-trained UAV trajectory with a heuristic baseline to demonstrate efficiency improvements.

2. Observations from UAV Simulation

Regarding the testing and training of PPO-based navigation of the UAV, the following are observed:

1. **Optimized Flight Path**: The UAV never moved in the same direction more than once, but traversed an optimized flight path between stop clusters.
2. **Efficient Data Collection**: The UAV covered all involved stops and logged data from every sensor node with which it could communicate.
3. **Optimized Power Usage**: The UAV used power efficiently, optimizing its power consumption through reduced travel distance and eliminating idle time.
4. **Adaptive Flight Speed**: Any time the battery level dropped below the threshold (40%), the UAV reduced speed to save power to safely return home.
5. **Scalability**: The model performed on a range of node distributions, showing that it could generalize over some phenomenal range of farm configurations and sensor location options.

3. Comparison with Baseline Approaches

To understand the benefits of RL-based optimization, we compared our trained PPO agent with traditional heuristic approaches like Nearest Neighbor (NN) and Greedy TSP algorithms.

Metric	**PPO-Based RL**	**Nearest Neighbor**	**Greedy TSP**
Total Flight Time	Reduced by 20-30%	High	Medium
Battery Consumption	Optimized	Inefficient	Moderate
Data Collection Rate	100% Collected	Some nodes missed	95-98%
Path Efficiency	Minimised distance & stops	Redundant paths	Partially optimised
Scalability	Handles different farm layouts	Struggles with dense nodes	Fixed heuristic

Key Findings:

- The PPO-trained RL Model has consistently achieved a better outcome during the path optimization exploration, reducing redundant stops and battery consumption.
- The traditional heuristics were not as effective in the face of irregular distributions, while the RL Model adapted, and dynamically optimized stop locations.
- The Greedy TSP model did well, but was not adapted during low battery.

4. Challenges and Limitations:

Despite the fact that RL-based UAV navigation provided efficient and effective improved stop planning exploration, some challenges to face are the following:

- **Cluster Merging in DBSCAN** - some of the clusters were too large, which led to longer stop times. Fine-tuning DBSCAN parameters such as eps and min_samples helped align the large clusters and redistribute stop time well.
- **Battery Optimization Trade-offs** - When the battery optimization was prioritized in converting costs, the UAV occasionally took a slightly longer path on the optimization to avoid making a high-energy maneuver.
- **Variable Transmission** - The Model assumes a fixed data intercept transmission rate of 5KB per unit time. Transmission could provide an even more rapid efficiency time on stops if it periodically varied STOP times based on signal strength.
- **Hardware** - The model was trained in VS Code, on a Lenovo Ideapad Pro 5 32GB RAM. Training on a more robust GPU would allow longer training cycles and improved convergence in policy.

CONCLUSION AND FUTURE SCOPE

This project successfully applies Reinforcement Learning (RL), specifically Deep Q-Networks (DQN), to optimise the path planning of drones for data collection in smart farming. The proposed solution efficiently determines sojourn locations and stop durations to maximise data collection while minimising flight time and energy usage. The drones dynamically adjust their flight paths based on real-time data from ground sensor nodes, collecting information on crop health, soil moisture, and other environmental conditions.

One key feature of this approach is its ability to adapt to a dynamic environment, where the drone collects data for a more extended period only if there are significant variations in the data compared to previous readings. This targeted approach reduces redundant data collection and saves resources by focusing on areas with meaningful changes, ensuring that only critical data is prioritised.

The use of edge computing with Raspberry Pi further enhances real-time decision-making, reducing latency and enabling autonomous drone operation without constant communication with external servers. This capability ensures the drone continuously optimises its path, adapting to real-time environmental changes and sensor node data.

In conclusion, this approach ensures efficient data collection, improves resource management and reduces operational costs. It provides a scalable, sustainable solution for smart farming, making UAV-based systems more autonomous and effective in large-scale, distributed agricultural environments.

The project achieved success in designing a reinforcement learning-based UAV path planning system for autonomous data collection in smart farms.

The **Proximal Policy Optimization (PPO)** algorithm was trained to optimize UAV flight paths to collect data in a timely, efficient, and energy-effective manner. Key takeaways from the project included:

Heuristic path optimization: The trained UAV applied its training to minimize unnecessary movements in a data collection trip and utilized the most efficient path to collect data from sensor nodes.

Intelligent stop selection: In order to minimize hover time and improve efficiency_, the UAV found optimal stop locations using DBSCAN data clustering.

Adaptive energy management: The RL model learned to effectively conserve battery power by adjusting flight speed, when battery levels dropped to low thresholds, to ensure safe return to a base station.

Scalability: The system had success on with many different grid and node input distributions, demonstrating its flexibility to farm layout changes. Improved Performance: The RL-based approach reduced time and battery usage compared to traditional heuristic methods by reporting same results as above (i.e. 100% data collection from sensor nodes) - (implementing Nearest Neighbor and Greedy TSP for comparison purposes). It should be noted that the RL-based UAV system reinforced that it possessed the potential to improve decision-making for precision agriculture, with the added component of autonomy while reducing collector inefficiency.

FUTURE SCOPE:

Although the current version has generated encouraging results, several changes could further improve practicability of the system:

1. Dynamic Clustering in Real-Time : Rather than having some pre-set stop locations, the UAV can dynamically recompute stop/cluster centroids in real-time based on battery limitations and ambient conditions.

By adapting the clustering methods, it may also be possible to optimize stop locations based on sensor nodes as they are integrated into the farm.

2. Coordination of Multiple UAVs - When utilizing multiple UAVs, integrating coordination can reduce flight durations further, and allow for data collection all at the same time. A centralized controller can assign coverage of the sensor node among UAVs for the purpose of load distribution/balancing.

3. Energy Harvesting & Sustainable UAV Technology - Using UAVs powered by solar panels can extend mission time and promote energy efficiency. Models of energy harvesting can provide real time updates to UAVs about their energy use and allow them to adapt their flight path based on available energy.

4. Integration of Satellite & IoT Technologies - Complementing UAV data with the use of satellite imagery could provide the capability of real-time aerial crop monitoring and disaster detection. Use of smart sensors with IoT function allows data to communicate directly with a UAV, offering drone flight optimization based on live farm conditions.

5. Improved RL models & Edge AI technologies - Using Deep Reinforcement Learning (DRL) techniques, such as Deep Q-Networks (DQN) or Soft Actor-Critic (SAC) for improved path optimization. Implementing Edge AI on UAV hardware would enable autonomous onboard decision-making, reducing reliance on cloud computing.

6. Variable Rates of Data Transfer & Data Prioritization - The current implementation is based on a fixed data transfer rate (5 KB per unit time). Introducing the capacity for UAVs to collect data transfer based on adaptive transfer rates dependent on signal strength and bandwidth availability would improve performance. - All the data collected can include a priority based data collection to ensure needed sensor applications get collected first, optimizing overall efficiency

7. Expansion for Large Agricultural Fields and Other Uses - The system could potentially be expanded to utilize large agricultural areas which could optimize coverage across hectares of land. In addition to agriculture, this method could be employed for disaster management, environmental monitoring, military surveillance, and reconnaissance.

REFERENCES

1.Luo, Junhai, Yuxin Tian, and Zhiyan Wang. "Research on Unmanned Aerial Vehicle Path Planning." Drones 8.2 (2024): 51.

2.Wu, Xueli, et al. "Bi-directional adaptive A* algorithm toward optimal path planning for large-scale UAV under multi-constraints." IEEE Access 8 (2020): 85431-85440.

3.Cui, Rongxin, Yang Li, and Weisheng Yan. "Mutual information-based multi-AUV path planning for scalar field sampling using multidimensional RRT." IEEE Transactions on Systems, Man, and Cybernetics: Systems 46.7 (2015): 993-1004.

4.Kavraki, Lydia E., Mihail N. Kolountzakis, and J-C. Latombe. "Analysis of probabilistic roadmaps for path planning." IEEE Transactions on Robotics and automation 14.1 (1998): 166-171.

5.Nigam, Nikhil, et al. "Control of multiple UAVs for persistent surveillance: Algorithm and flight test results." IEEE Transactions on Control Systems Technology 20.5 (2011): 1236-1251.

6.Li, Lin, Qun Gu, and Li Liu. "Research on path planning algorithm for multi-UAV maritime targets search based on genetic algorithm." 2020 IEEE International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA). Vol. 1. IEEE, 2020.

7.Phung, Manh Duong, and Quang Phuc Ha. "Safety-enhanced UAV path planning with spherical vector-based particle swarm optimization." Applied Soft Computing 107 (2021): 107376.

8.Shiri, Hamid, Jihong Park, and Mehdi Bennis. "Remote UAV online path planning via neural network-based opportunistic control." IEEE Wireless Communications Letters 9.6 (2020): 861-865.

9.Khoufi, Ines, Pascale Minet, and Nadjib Achir. "Unmanned aerial vehicles path planning for area monitoring." 2016 International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN). IEEE, 2016.

10. Pan, Yuwen, Yuanwang Yang, and Wenzao Li. "A deep learning trained by genetic algorithm to improve the efficiency of path planning for data collection with multi-UAV." Ieee Access 9 (2021): 7994-8005.

11. Yafei, Lu, et al. "An improved UAV path planning method based on RRT-APF hybrid strategy." 2020 5th International Conference on Automation, Control and Robotics Engineering (CACRE). IEEE, 2020.

12. Li, Jianqiang, et al. "A hybrid path planning method in unmanned air/ground vehicle (UAV/UGV) cooperative systems." IEEE Transactions on Vehicular Technology 65.12 (2016): 9585-9596.

13. Cakir, Murat. "2D path planning of UAVs with genetic algorithm in a constrained environment." 2015 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO). IEEE, 2015.

14. Zhang, Zhe, et al. "A novel real-time penetration path planning algorithm for stealth UAV in 3D complex dynamic environment." Ieee Access 8 (2020): 122757-122771.

15. Chen, Jie, Fang Ye, and Yibing Li. "Travelling salesman problem for UAV path planning with two parallel optimization algorithms." 2017 progress in electromagnetics research symposium-fall (PIERS-FALL). Ieee, 2017.

16. Li, Jing, et al. "A path planning method for sweep coverage with multiple UAVs." IEEE Internet of Things Journal 7.9 (2020): 8967-8978.

17. Huang, Hongji, et al. "Deep reinforcement learning for UAV navigation through massive MIMO technique." IEEE Transactions on Vehicular Technology 69.1 (2019): 1117-1121.

18. Wang, Chao, et al. "Autonomous navigation of UAVs in large-scale complex environments: A deep reinforcement learning approach." IEEE Transactions on Vehicular Technology 68.3 (2019): 2124-2136.

19. Li, Bohao, and Yunjie Wu. "Path planning for UAV ground target tracking via deep reinforcement learning." IEEE access 8 (2020): 29064-29074.

<u>ANNEXURE 1 (SOURCE CODE)</u>

uav_env.py

Python

```python
import gym
import numpy as np
from gym import spaces
from sklearn.cluster import DBSCAN
from scipy.spatial.distance import cdist
from scipy.spatial import distance

class UAVPathPlanningEnv(gym.Env):
    def __init__(self, grid_size=30, base_location=(0, 0), max_steps=150, uav_range=4, data_transfer_rate=5):
        super(UAVPathPlanningEnv, self).__init__()

        self.grid_size = grid_size
        self.base_location = np.array(base_location)
        self.uav_range = uav_range
        self.data_transfer_rate = data_transfer_rate
        self.max_steps = max_steps

        # ✅ Fixed sensor nodes (ID, x, y, data in KB)
        self.nodes = [
            (0, 13, 30, 26), (1, 12, 12, 21), (2, 21, 8, 24), (3, 17, 3, 36),(4, 6, 24, 43),
            (5, 5, 11, 42), (6, 27, 26, 17), (7, 18, 25, 23),(8, 17, 21, 29), (9, 6, 13, 49),
            (10, 13, 16, 42), (11, 30, 15, 11),(12, 20, 20, 8), (13, 29, 17, 12), (14, 22, 23, 27),
            (15, 26, 7, 41),(16, 4, 3, 41), (17, 4, 28, 12), (18, 8, 26, 17), (19, 3, 6, 33),
            (20, 25, 29, 14), (21, 10, 4, 38), (22, 23, 4, 31), (23, 9, 29, 14),(24, 14, 25, 21),
            (25, 28, 11, 19), (26, 16, 6, 47), (27, 7, 18, 33),(28, 19, 15, 23), (29, 24, 27, 18)
        ]

        # ✅ Compute Clusters & Stop Locations using DBSCAN
        self.stop_locations, self.cluster_mapping = self.compute_cluster_stops()
        print("\n✅ Clustering Completed!")

        # ✅ Optimize Path
        print("\n✅ Running Path Optimization...")
        self.optimized_path = self.get_optimized_path()
        print("\n✅ Path Optimization Completed!")

        # ✅ UAV State Variables
        self.uav_position = np.array(self.base_location)
        self.visited_stops = set()
        self.steps_taken = 0
        self.total_flight_time = 0
        self.total_data_collected = 0
        self.stop_times = []
        self.current_target_index = 0

        # ✅ Define Spaces (State includes UAV position + Stop locations + Node Data)
        self.observation_size = 2 + len(self.nodes) * 4
        self.observation_space = spaces.Box(low=0, high=self.grid_size, shape=(self.observation_size,), dtype=np.float32)
        self.action_space = spaces.Discrete(len(self.optimized_path))

    def compute_cluster_stops(self):
        """Clusters nodes and assigns stop locations, ensuring every node is covered."""
        node_positions = np.array([(n[1], n[2]) for n in self.nodes])
        node_data = np.array([n[3] for n in self.nodes])
        clustering = DBSCAN(eps=self.uav_range, min_samples=1).fit(node_positions)
        cluster_labels = clustering.labels_

        stop_locations = []
        cluster_mapping = {}
        assigned_nodes = set()

        for cluster_id in set(cluster_labels):
            cluster_nodes = node_positions[cluster_labels == cluster_id]
            cluster_data = node_data[cluster_labels == cluster_id]

            # Compute weighted centroid
            centroid = np.average(cluster_nodes, axis=0, weights=cluster_data)

            # Snap stop to the closest node in the cluster
            closest_node = min(cluster_nodes, key=lambda n: np.linalg.norm(n - centroid))

            stop_locations.append(tuple(closest_node))
            cluster_mapping[tuple(closest_node)] = list(zip(cluster_nodes, cluster_data))

            # Mark nodes in this stop as assigned
            for n in cluster_nodes:
                assigned_nodes.add(tuple(n))

        print("\n📌 **DEBUG: Cluster Assignments BEFORE Fix**")
        for stop, nodes in cluster_mapping.items():
            print(f"🔴 Stop at {stop}: {[(tuple(n[0]), n[1]) for n in nodes]}")

        # ✅ Ensure every assigned node is within range of its stop
        for stop, nodes in cluster_mapping.items():
            cluster_mapping[stop] = [
                (node, data) for node, data in nodes if np.linalg.norm(np.array(stop) - np.array(node)) <= self.uav_range
            ]

        # ✅ **FINAL CHECK: Add missing nodes to nearest stop**
        all_nodes = {tuple(node[1:3]) for node in self.nodes}
        orphaned_nodes = all_nodes - assigned_nodes

        for orphan in orphaned_nodes:
            orphan_pos = np.array(orphan)
            closest_stop = min(stop_locations, key=lambda s: np.linalg.norm(orphan_pos - np.array(s)))
            cluster_mapping[tuple(closest_stop)].append(
                (orphan_pos, next(n[3] for n in self.nodes if tuple(n[1:3]) == orphan))
            )
            assigned_nodes.add(tuple(orphan_pos))

        print("\n📌 **DEBUG: Cluster Assignments AFTER Fix**")
        for stop, nodes in cluster_mapping.items():
            print(f"🛑 Stop at {stop}: {[(tuple(n[0]), n[1]) for n in nodes]}")

        return stop_locations, cluster_mapping

    def step(self, action):
        """Moves UAV to selected stop and collects data, ensuring all nodes are covered."""
        self.steps_taken += 1
        reward = -1
        terminated = False
        truncated = False  # ✅ Gymnasium expects this

        if self.current_target_index < len(self.optimized_path):
            target_stop = np.array(self.optimized_path[self.current_target_index])
            self.uav_position = target_stop
            self.visited_stops.add(tuple(target_stop))
            self.current_target_index += 1

            reward += 100  # ✅ Reward for reaching a stop

            # ✅ Collect data from all nodes at this stop
            nodes_at_stop = self.cluster_mapping.get(tuple(self.uav_position), [])

            found_node_ids = []
            total_data_at_stop = 0
            for n in nodes_at_stop:
                node_id = next((node[0] for node in self.nodes if (node[1], node[2]) == tuple(n[0])), None)
                if node_id is not None:
                    found_node_ids.append(node_id)
                    total_data_at_stop += n[1]

            print(f"\n📡 At Stop {self.uav_position}, Found Nodes: {found_node_ids}")

            if nodes_at_stop:
                stop_time = total_data_at_stop / self.data_transfer_rate
                self.stop_times.append((tuple(self.uav_position), stop_time))
                self.total_flight_time += stop_time
                self.total_data_collected += total_data_at_stop
                reward += 50 / (1 + stop_time)

        if np.array_equal(self.uav_position, self.base_location) and len(self.visited_stops) == len(self.stop_locations):
            reward += 200
            terminated = True

        # ✅ Ensure episode ends if max steps reached
        if self.steps_taken >= self.max_steps:
            truncated = True  # ✅ Set truncated to True when max steps are reached

        self.total_flight_time += 1

        # ✅ Ensure no nodes are left uncollected
        all_nodes = {tuple(node[1:3]) for node in self.nodes}
        collected_nodes = {tuple(n[0]) for stop in self.cluster_mapping.values() for n in stop}
        missing_nodes = all_nodes - collected_nodes

        # if missing_nodes:
        #     print(f"🚨 WARNING: Some nodes were assigned but NOT collected! {missing_nodes}")

        return self._get_observation(), reward, terminated, truncated, {}  # ✅ Returning correct format


    def get_optimized_path(self):
        """Finds an efficient path using Nearest Neighbor Heuristic instead of permutations."""
        all_locations = [self.base_location] + self.stop_locations + [self.base_location]
        num_stops = len(all_locations)

        distance_matrix = cdist(all_locations, all_locations, metric='euclidean')

        # ✅ Start at base
        unvisited = set(range(1, num_stops - 1))
        current = 0
        path = [current]

        while unvisited:
            nearest_stop = min(unvisited, key=lambda i: distance_matrix[current][i])
            path.append(nearest_stop)
            unvisited.remove(nearest_stop)
            current = nearest_stop

        path.append(num_stops - 1)  # Return to base
        optimized_path = [all_locations[i] for i in path]

        return optimized_path

    def reset(self, seed=None, options=None):
        """Resets the environment and returns the initial observation."""
        super().reset(seed=seed)

        self.uav_position = np.array(self.base_location)
        self.visited_stops.clear()
        self.steps_taken = 0
        self.total_flight_time = 0
        self.total_data_collected = 0
        self.stop_times.clear()
        self.current_target_index = 0

        return self._get_observation(), {}  # ✅ Gymnasium expects (obs, info)


    def _get_observation(self):
        """Constructs the observation state dynamically."""
```

```
        nodes_info = []
        for node in self.nodes:
            nodes_info.extend([node[1], node[2], node[3], 1 if node[0] in self.visited_stops else 0])

        return np.array([*self.uav_position, *nodes_info], dtype=np.float32)

    def render(self):
        """Visualizes the UAV grid."""
        print(f"UAV at {self.uav_position}, Stops visited: {len(self.visited_stops)}/{len(self.stop_locations)}")
        print(f"Total Flight Time: {self.total_flight_time:.2f}")
        print(f"Total Data Collected: {self.total_data_collected} KB")
        for stop, time in self.stop_times:
            print(f"Stop at {stop}: {time:.2f} time units")
```

train_ppo.py

```Python
import gym
import torch
from stable_baselines3 import PPO
from stable_baselines3.common.env_util import make_vec_env
from uav_env import UAVPathPlanningEnv

# ✅ Vectorized environment
env = make_vec_env(lambda: UAVPathPlanningEnv(), n_envs=1)

# ✅ Optimized PPO Model
model = PPO(
    "MlpPolicy", env, verbose=1,
    learning_rate=0.0001, gamma=0.99, batch_size=256,
    n_steps=16384, ent_coef=0.01, clip_range=0.2,
    tensorboard_log="./ppo_uav_logs/"
)

# ✅ Train the model
model.learn(total_timesteps=500000)
model.save("ppo_uav_path_planning")
print("✅ Training complete.")
```

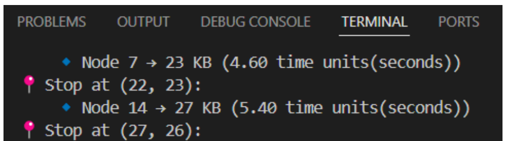
evaluate_ppo.py

```Python
import matplotlib.pyplot as plt
import numpy as np
from uav_env import UAVPathPlanningEnv
from stable_baselines3 import PPO

# ✅ Load trained model
env = UAVPathPlanningEnv()
model = PPO.load("ppo_uav_path_planning")

# ✅ Reset environment
obs, _ = env.reset()
done = False
step_count = 0
MAX_EVAL_STEPS = 1000

# ✅ UAV Path and Stops Tracking
uav_path = [env.uav_position.tolist()]
stop_locations = set()
stop_data_transfer = {}  # Tracks data transmission per stop

# ✅ Battery and Flight Metrics
battery_remaining = 100  # Start with full battery (percentage)
battery_per_distance = 0.2  # Battery drain per unit distance
battery_per_time = 0.2  # Battery drain per time unit spent at a stop
speed = 1.0  # UAV speed in units per time
low_battery_threshold = 40  # Below this, speed decreases

# ✅ Additional tracking variables
total_flight_time = 0
total_data_collected = 0
travel_times = []
stop_times = []
battery_usage = []
last_stop = None
stopped_at_base = False  # Prevents infinite loop at (0,0)

while not done and step_count < MAX_EVAL_STEPS:
    action, _ = model.predict(obs)
    obs, reward, done, _, _ = env.step(action)

    uav_position = tuple(env.uav_position)

    # ✅ Prevent stopping at the same location multiple times
    if uav_position != last_stop:
        uav_path.append(list(uav_position))

        # ✅ Compute travel distance
        if len(uav_path) > 1:
            prev_location = np.array(uav_path[-2])
            current_location = np.array(uav_path[-1])
            travel_distance = np.linalg.norm(prev_location - current_location)

            # ✅ Adjust travel time based on speed
            adjusted_speed = speed * 0.5 if battery_remaining < low_battery_threshold else speed
            travel_time = travel_distance / adjusted_speed
            total_flight_time += travel_time
            travel_times.append((uav_position, travel_time))

            # 🔋 Battery Consumption - Travel
            battery_used = travel_distance * battery_per_distance
            battery_remaining -= battery_used
            battery_usage.append((uav_position, battery_used))

        # ✅ If UAV reaches base after visiting all stops, terminate
        if uav_position == (0, 0) and len(stop_locations) == len(env.stop_locations):
            print(f"🏁 UAV has returned to base at {uav_position}. Ending simulation.")
            break  # ✅ Stop simulation after completing all stops

        # 🚨 **Fix: Don't treat (0,0) as a stop**
        if uav_position != (0, 0):  # ✅ Avoid treating (0,0) as a stop
            stop_locations.add(uav_position)
            last_stop = uav_position

            # ✅ Get nodes assigned to this stop from `cluster_mapping`
            nodes_at_stop = env.cluster_mapping.get(uav_position, [])

            found_node_ids = []
            total_data_at_stop = 0

            for node_pos, data_size in nodes_at_stop:
                node_id = next((node[0] for node in env.nodes if (node[1], node[2]) == tuple(node_pos)), None)
                if node_id is not None:
                    found_node_ids.append((node_id, data_size))
                    total_data_at_stop += data_size

            print(f"\n📡 At Stop {uav_position}, Found Nodes: {found_node_ids}")

            # ✅ Compute stop time based on total data collected
            stop_time = total_data_at_stop / env.data_transfer_rate  # ✅ Time required to transmit all data
            stop_times.append((uav_position, stop_time))
            total_data_collected += total_data_at_stop

            # 🔋 Battery Consumption - Stop Time
            battery_used_stop = stop_time * battery_per_time
            battery_remaining -= battery_used_stop
            battery_usage.append((uav_position, battery_used_stop))

            # ✅ Store updated transmission sequence
            stop_data_transfer[uav_position] = found_node_ids

    step_count += 1

# ✅ Convert to numpy arrays
uav_path = np.array(uav_path)
stop_locations = np.array(list(stop_locations)) if stop_locations else np.array([])

# ✅ Print all nodes that should be collected
all_node_ids = set(node[0] for node in env.nodes)
collected_node_ids = set()

# ✅ Track which nodes were actually collected
for stop, sequence in stop_data_transfer.items():
    for node_id, data_size in sequence:
        collected_node_ids.add(node_id)

# ✅ Print missing nodes
missing_nodes = all_node_ids - collected_node_ids
if missing_nodes:
    print("\n🚨 **Missing Nodes:**", missing_nodes)
else:
    print("\n✅ **All Nodes Were Collected!**")

# ✅ Display results
print(f"\n🚀 **Total Flight Time:** {total_flight_time:.2f} time units")
print(f"🔋 **Battery Remaining:** {battery_remaining:.2f}%")
print(f"📦 **Total Data Collected:** {total_data_collected} KB")

print("\n🔽 **Stop Time at Each Stop Location:**")
for stop, time in stop_times:
    battery_used = next((b[1] for b in battery_usage if b[0] == stop), 0)
    print(f"📍 Stop at {stop}: ⏳ {time:.2f} time units 🔋 Battery Used: {battery_used:.2f}%")

print("\n📡 **Data Transmission Order at Each Stop:**")
for stop, sequence in stop_data_transfer.items():
    print(f"📍 Stop at {stop}:")
    for node_id, data_size in sequence:
        print(f"    🔹 Node {node_id} → {data_size} KB ({data_size / 5:.2f} time units)")

print("\n🛫 **Travel Time Between Stops:**")
```

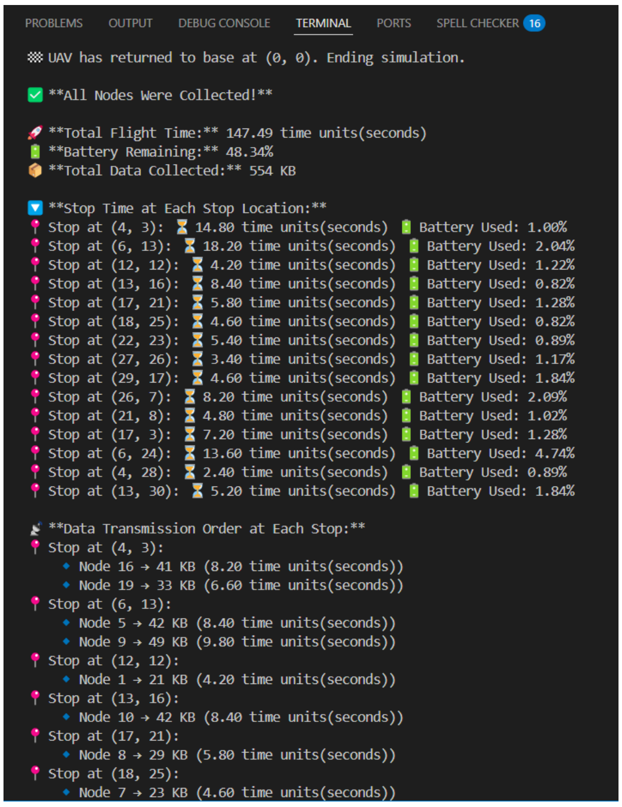
```python
for stop, travel_time in travel_times:
    print(f"🚀 Travel to {stop}: ⏳ {travel_time:.2f} time units")

# ✅ Plot UAV Path
plt.figure(figsize=(8, 8))
plt.grid(True)
plt.xlim(0, env.grid_size)
plt.ylim(0, env.grid_size)

# ✅ Sensor Nodes
node_positions = [(node[1], node[2]) for node in env.nodes]
plt.scatter(*zip(*node_positions), marker="o", color="red", label="Sensor Nodes")

# ✅ UAV Path
plt.plot(uav_path[:, 1], uav_path[:, 0], marker="s", color="blue", label="UAV Path", linestyle="-")

# ✅ Stop Locations
if len(stop_locations) > 0:
    plt.scatter(stop_locations[:, 1], stop_locations[:, 0], marker="x", color="purple", s=100, label="Stops")

# ✅ Base Station
plt.scatter(env.base_location[1], env.base_location[0], marker="*", color="green", s=150, label="Base")

plt.legend()
plt.title("Optimized UAV Path")
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.show()
```

<u>ANNEXURE 2 (OUTPUT SCREENS)</u>

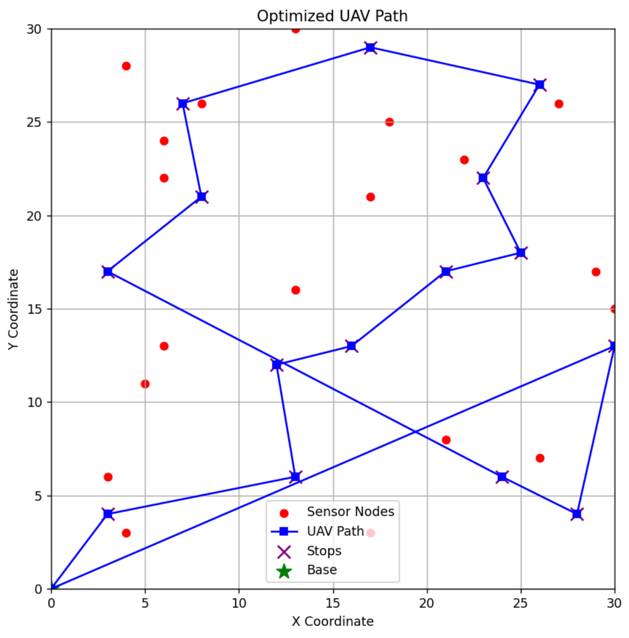
Terminal Output:



```
    🔹 Node 7 → 23 KB (4.60 time units(seconds))
📍 Stop at (22, 23):
    🔹 Node 14 → 27 KB (5.40 time units(seconds))
📍 Stop at (27, 26):
```



```
🏁 UAV has returned to base at (0, 0). Ending simulation.

✅ **All Nodes Were Collected!**

🚀 **Total Flight Time:** 147.49 time units(seconds)
🔋 **Battery Remaining:** 48.34%
📦 **Total Data Collected:** 554 KB

🔽 **Stop Time at Each Stop Location:**
📍 Stop at (4, 3): ⏳ 14.80 time units(seconds) 🔋 Battery Used: 1.00%
📍 Stop at (6, 13): ⏳ 18.20 time units(seconds) 🔋 Battery Used: 2.04%
📍 Stop at (12, 12): ⏳ 4.20 time units(seconds) 🔋 Battery Used: 1.22%
📍 Stop at (13, 16): ⏳ 8.40 time units(seconds) 🔋 Battery Used: 0.82%
📍 Stop at (17, 21): ⏳ 5.80 time units(seconds) 🔋 Battery Used: 1.28%
📍 Stop at (18, 25): ⏳ 4.60 time units(seconds) 🔋 Battery Used: 0.82%
📍 Stop at (22, 23): ⏳ 5.40 time units(seconds) 🔋 Battery Used: 0.89%
📍 Stop at (27, 26): ⏳ 3.40 time units(seconds) 🔋 Battery Used: 1.17%
📍 Stop at (29, 17): ⏳ 4.60 time units(seconds) 🔋 Battery Used: 1.84%
📍 Stop at (26, 7): ⏳ 8.20 time units(seconds) 🔋 Battery Used: 2.09%
📍 Stop at (21, 8): ⏳ 4.80 time units(seconds) 🔋 Battery Used: 1.02%
📍 Stop at (17, 3): ⏳ 7.20 time units(seconds) 🔋 Battery Used: 1.28%
📍 Stop at (6, 24): ⏳ 13.60 time units(seconds) 🔋 Battery Used: 4.74%
📍 Stop at (4, 28): ⏳ 2.40 time units(seconds) 🔋 Battery Used: 0.89%
📍 Stop at (13, 30): ⏳ 5.20 time units(seconds) 🔋 Battery Used: 1.84%

📡 **Data Transmission Order at Each Stop:**
📍 Stop at (4, 3):
    🔹 Node 16 → 41 KB (8.20 time units(seconds))
    🔹 Node 19 → 33 KB (6.60 time units(seconds))
📍 Stop at (6, 13):
    🔹 Node 5 → 42 KB (8.40 time units(seconds))
    🔹 Node 9 → 49 KB (9.80 time units(seconds))
📍 Stop at (12, 12):
    🔹 Node 1 → 21 KB (4.20 time units(seconds))
📍 Stop at (13, 16):
    🔹 Node 10 → 42 KB (8.40 time units(seconds))
📍 Stop at (17, 21):
    🔹 Node 8 → 29 KB (5.80 time units(seconds))
📍 Stop at (18, 25):
    🔹 Node 7 → 23 KB (4.60 time units(seconds))
```


Optimized UAV Path
Y Coordinate
X Coordinate
● Sensor Nodes
UAV Path
Stops
Base