Battery Optimization for Crazyflie Flight Time Maximization Using AI Based Battery Management Module

Akshay Raman[1]
Mechanical and AI Lab (MAIL),
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15217 USA
email: akshayra@andrew.cmu.edu

Amir Barati Farimani
Mechanical and AI Lab (MAIL),
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15217 USA
email: barati@cmu.edu

As the use-case of Unmanned Aerial Vehicle (UAV) applications becomes more strenuous, it becomes crucial to optimize current UAV ecosystems to perform more demanding tasks. This study introduces two innovative optimization algorithms for multi-agent UAV control in large scale operation environments with limited battery resources: Error Mitigation Neural Network (EMNN) and Future Battery Life Prediction Network (FBLPN). Deployed within a Crazyflie ecosystem, the algorithms address the critical challenge of both battery life prediction and precise low-battery control to increase time of flight in scarce charging environments. Unlike traditional methods that passively wait for battery state changes, the forward prediction algorithm coupled with the drift error mitigation system allows multi-agent UAVs to optimize their time of flight by anticipating future battery states based on planned movements. The EMNN model, employing a rudimentary multi-layer neural network trained on a custom dataset, learns to minimize drift errors as battery levels drop. Through extensive evaluation, EMNN shows drift errors contained within 1cm even during low battery scenarios. Meanwhile, FBLPN, leveraging a novel transformer module with a convolutional feature selector, learns battery consumption patterns from sequential data to predict battery usage on future sequence data. We evaluate its performance on a Prioritized Planning ecosystem enabling optimized scheduling of three Crazyflies sharing a single wireless charging pad to maximize airtime before battery depletion. Initial results demonstrate a significant increase in operational flight time, highlighting the potential of these algorithms to enhance multi-agent UAV operations.

1 Introduction

Artificial Intelligence (AI) has garnered significant attention in various robotics applications, In particular, AI algorithms have become pivotal in augmenting aerial autonomous systems, facilitating complex tasks such as exploration, search and rescue, surveillance, and other long-term operations. Notably, the application of AI algorithms to enhance current robotic modals allows the ability to perform the said complex tasks. Given the complexity and duration of these tasks, it is crucial for drone ecosystems to employ a synergistic approach between recharging and operational progression to efficiently complete tasks. This requires a dual strategy that not only predicts battery status during flight but also optimizes the path to docking stations for recharging. The process of finding these routes to the docking station is done through path planning. Multi-agent path planning refers to the problem of finding valid low cost paths for each agent while avoiding collisions with static and dynamic obstacles. Hence, the integration of intelligent AI to predict real-time battery status, combined with advanced path planning algorithms, aims to maximize flight duration and operational efficiency.

AI has seldom been used to enhance the throughput of aerial times for drones. Transformer based algorithms were used to predict the remaining useful battery life left for drones [1–4]. While these algorithms did an exceptional job, they only enhanced the preemptive throughput of aerial systems by catching batteries nearing the end of their life rather than per cycle. Reinforcement learning algorithms such as Lifelong Battery Prediction Reinforcement (LBpREd) helped predict battery dissipation over time given a task but had no algorithm in place to increase the aerial throughput based on this knowledge gain [5, 6]. Recently, transformer-based networks have been used to predict battery life but are too computationally expensive to implement online [2, 5]. We propose a novel CNN preprocessing to reduce feature size before encoding the sequence data and sending it into a transformer. This drastically reduces computational power and can predict future battery life left in real time based on past sequential data relaying straight from the drone [7].

Another challenge in aerial drone operations is managing drift errors during low battery conditions, which can degrade motor performance. Traditional control algorithms, like the Kalman filter, address instantaneous errors effectively but do not preemptively adjust motor power based on battery status and navigational needs. During low battery flight, the lack of power translates into accumulated drift [8]. Machine learning techniques can preemptively manage motor power based on the battery level and the distance needed to be travelled to next point, thereby minimizing drift errors and optimizing route adherence.

On the other hand, solely understanding the current and future battery life per cycle will not enhance the performance of the drone ecosystem. Current systems often overlook the need for optimized flight paths in low battery conditions, leading to inefficient routing and potential delays in recharging. A subsystem needs to be implemented that optimizes flying in low battery scenarios and accurately predicts quick trajectories to nearby docking stations. While decentralized solvers (reinforcement learning algorithms) struggle with communication and optimal pathfinding [9–11], multi-agent solvers like Conflict-based Search (CBS) and Enhanced CBS (EECBS) are limited by online processing time and prioritization issues [12]. Priority based planning algorithms can quickly find paths for each agent online and prioritize low battery agents to find optimal paths but fail to solve deadlock situations and scenarios where lower priority drones get trapped against higher priority drones [13, 14]. Solis implements an optimal solving al-

[1]Corresponding Author.
Version 1.18, April 30, 2024

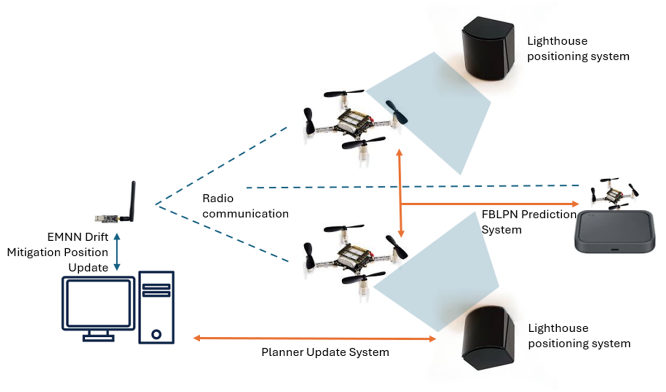

Lighthouse positioning system
Radio communication
FBLPN Prediction System
EMNN Drift Mitigation Position Update
Lighthouse positioning system
Planner Update System

Fig. 1 The Crazyflie environment with labelled subsystems. The computer communicates with the crazyradio shown above the computer. This radio relays new position to drone with the updated EMNN drift position. Crazyflie seequentially flies to all waypoint coordinates before requesting next set of coordinates.

gorithm such as CBS within a niche bounding box area where collisions do occur and expands this box until the collision has been resolved. In this paper, the prioritized planning approach is modified to look for situations where deadlock might occur and implement a conflict resolution method within that local area to sub-optimally solve paths, a strategy we refer to as Priority Planning Plus Local Solve (PP+LS) [10].

1.1 Main Contributions Contained Within This Paper. This paper introduces a harmonized integration of three innovative algorithms to extend drone flight times for long-term projects requiring multiple charging cycles. A central greedy optimizer coordinates these algorithms, assigning drones to specific charging pads efficiently. Specifically, this paper introduces:

- A novel transformer-based network with a CNN feature selector that predicts the remaining distance a drone can cover before needing to recharge (Future Battery Life Prediction Network, FBLPN).
- An error mitigation neural network (EMNN) that predicts and adjusts for drift errors based on battery life stages, enhancing control accuracy.
- A prioritized planning algorithm with an integrated deadlock resolver, ensuring high-priority routing for low battery drones to charging stations.
- A central optimization algorithm that synergistically manages all the aforementioned components, maximizing drone operational time (Flight Time Maximization Module, FTMM).
- A comprehensive study demonstrating the collective efficacy of these algorithms within the Bitcraze Crazyflie ecosystem.

2 Problem Setup

2.1 Environment. Consider a 3-D environment represented within the world as $W \subseteq \mathbb{R}^3$ with size H*W*L. The environment boundaries are discretized into 1cm sections represented by a numpy array. The array delineates the environmental layout, classifying regions as either obstacles or free space, facilitating the navigation of drones. The map is initialized with these obstacles in known locations before the flight optimization problem begins.

Static obstacles within this environment are denoted as $\mathcal{C}_s = \{s_1, \ldots, s_{N_s}\}$, where $s_i \subset W$ denotes the i-th static obstacle. The free navigable space, $W \setminus \mathcal{C}_s$, is depicted by a roadmap graph $G = (C_f, E)$. Here, $C_f = \{c_1, \ldots, c_{N_f}\}$ represents the set of free cells and $e_{ij} = (c_i, c_j) \subset E$ indicates the set of free cells, and each edge c_i and c_j symbolizes a traversable path between two free cells that avoids any obstructive elements. Prior to the deployment of AI models aimed at enhancing flight time efficiency, a Probabilistic Roadmap (PRM) graph is constructed to establish connectivity between traversable areas. Based on this world map setup, a multi-robot motion planning (MAPF) problem is defined by (E, R, Q), where:

- E characterizes the environment.
- R is the set of drones operating within the environment.
- Q encompasses a series of queries, each specifying a start and a goal position for a drone. Given the perpetual nature of the flight time optimization challenge, these queries are generated ad infinitum as required, facilitating continuous drone operation from one designated position to another.

The EMNN (Error Mitigation Neural Network) and FBLPN (Future Battery Life Prediction Network) modules are initially trained within this environment absent any planning modules. Training involves the collection of data through random direct inputs of coordinates, simulating single-drone flights. Although training occurs under single-agent conditions, these AI models are designed to scale and integrate seamlessly across multiple agents, operating independently for each drone.

2.1.1 Experimental Setup. The experimental infrastructure utilizes the Crazyflie ecosystem, as illustrated in Figure 1. This setup was employed consistently across all data collection phases for model training, although only one drone was deployed at a time to ensure the accuracy and integrity of the data gathered. Figure 1 highlights the main hardware used as well as the how the various modules integrated with the hardware setup. The Crazyflie drone by Bitcraze was selected for its robust built-in API, which seamlessly interfaces with both the controller and the autonomous flight controller. This integration allowed our team to concentrate on developing an external optimization software, the overarching Flight Time Maximization Module (FTMM) [15]. For our experiments, up to three Crazyflie drones were utilized to evaluate the multi-agent capabilities of our optimization software (FTMM). Communication between the host computer and the drones was facilitated by a Crazyradio, also provided by Bitcraze. The positioning system, also provided by bitcraze, was used to detect the precise location of each drone within 1cm. The only drawback was the world boundaries for the problem were confined to the region within the lighthouse positioning system. To demonstrate the practical application of our concepts, a standard off-the-shelf wireless charging pad was used. Table 1 outlines the comprehensive list of hardware components utilized in our experiments.

Table 1 Bitcraze Crazyflie Components Used in Experimental Setup

Part Name	Description	Quantity Used
Crazyflie 2.0	The drone used	3
Lighthouse Position System	A VR laser based positioning system to precisely locate positions of each drone	2
Crazyradio	Radio communication tool to communicate between Crazyflie and host computer	1
Central Computer	Main computer to power all algorithms	1
Charging Pad	Wireless charging pad capable of charging a drone if landed anywhere within charging pad region	1

2.1.2 Simulation Setup. To rigorously evaluate the effectiveness of the planning module, we employed the Crazyswarm's ROS2 RViz simulator. RViz serves as a 3D visualization tool within the Robot Operating System (ROS), offering a dynamic interface for developers to monitor and interpret sensor data and state information from robots [15]. RViz enables the visualization of Crazyflie drones and allows for control through the same API used in the real-world experimental setup, replicating multi-agent behavior for more intricate planning scenarios not achievable in the actual experimental setup.

2.2 Data Collection. Data collection forms the backbone of our research, particularly for training the drift error reduction module (EMNN) and the remaining battery life prediction module (FBLPN).

2.2.1 EMNN Data Collection. For the EMNN, data was collected using a single Crazyflie drone. During flight, a range of parameters were recorded at varying frequencies between 10 and 100 Hertz, depending on their relevance to drift error prediction. A total of 26 key parameters were identified and grouped based on their data relay frequency, chosen for their potential in enhancing drift error predictions. Over 120,000 data points were collected and preprocessed. The raw data is fed into the preprocessing module of the EMNN network to remove outliers, bad data, or NaN values. This is discussed in more detail within the EMNN section. If a anypoint deemed not fit had to be discarded, the whole datarow had to be discarded for that timestep. Some critical parameters include:

xState: State of the drone on the x-axis (e.g., 0.1445).

yState: State of the drone on the y-axis (e.g., 0.0124).

2.2.2 FBLPN Data Collection. The data collection approach for the FBLPN was similar to that of the EMNN but differed slightly in the parameters collected and the duration of data collection sessions. For the FBLPN module, a data collection run lasted until the drone completely ran out of battery. The motivation behind this was to encapsulate the amount a drone can fly before complete battery exhaustion. Data was collected under varied conditions to ensure a diverse set of data points:

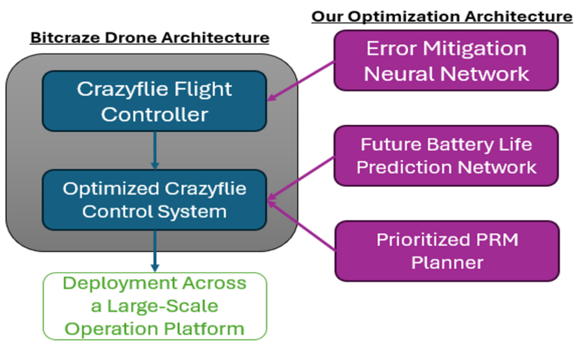

Our Optimization Architecture
Bitcraze Drone Architecture
Crazyflie Flight Controller
Error Mitigation Neural Network
Future Battery Life Prediction Network
Optimized Crazyflie Control System
Prioritized PRM Planner
Deployment Across a Large-Scale Operation Platform

Fig. 2 The overall connection architecture for every module with respect to built in Crazyflie interface

- Randomized location, speed, and acceleration during flights to introduce maximum variability.
- Data collection from different starting battery levels to understand performance variations across the battery life spectrum.

This allowed the transformer network to learn more holistic latent representations rather than niche relationships associated to the current training set. Table 2 shows the percent of total data that was collected at various battery charges. We purposefully collected more data during high and low battery levels to capture the variance in flight behavior at these stages. Per data collection run, the battery level was noted as an integer from 0-10 representing the charge at start of run. During preprocessing, the start charge and the final distance flown for that charge are conditionally inputted into the transformer module to allow the model learn the nuances between total distance flown at a given start charge and the movement patterns throughout the current run. Key parameters for FBLPN include:

Table 2 Percentage of Trials Collected at Various Battery Levels

Battery Level at Start	Percent of Trials
Level 8-10	35%
Level 6-7	11%
Level 4-5	13%
Level 3	6%
Level 1-2	32%
Level 0	3%

Total Flight Time: Time the drone was in flight (e.g., 258 sec).

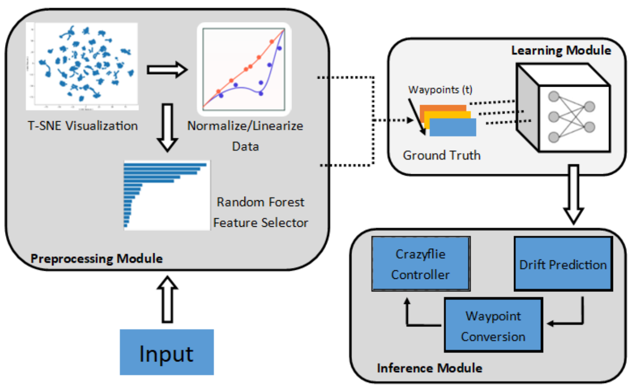
Total Distance Flown: Total distance the drone has flown (e.g., 12.6 m).

Battery Start State: Initial state of the drone's battery (e.g., 6).

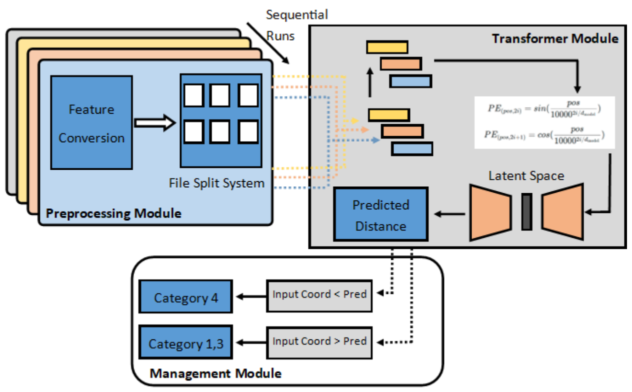
3 Methods

We divide the Flight Time Maximization Module (FTMM) into the three separate algorithms that we implement before conjoining them back into the drone ecosystem. Within each module, the overall system structure is first highlighted before going over the subtle nuances niche to each model.

3.1 EMNN Module. As illustrated in the system architecture (see Figure 2), the EMNN module directly interfaces with the built in Crazyflie controller. Post learning, the forward propagation output is fed directly to into the Crazyflie controller to predict the set of modified waypoints to fly to. Figure 1 shows where the EMNN module integrates with the overall ecosystem.



(*a*)



(*b*)

Fig. 3 Left is the EMNN Architecture where the preprocessing module processes input data before sending it into ANN network. Finally, the inference model calculates the the modified waypoints. Right is the FBLPN network where the preprocessing module process single runs into seperate files before feeding each sequential data into transformer module. The output is sent to management module where one of two outcomes is selected.

3.1.1 System Architecture. Figure 3(*a*) illustrates the overall structure for the EMNN module. As illustrated in the system architecture (see Figure), the EMNN-specific dataset undergoes initial processing where data is transformed to highlight relevant features. The preprocessing includes normalization or standardization and feature selection to enhance model training efficiency and accuracy. The key steps within the preprocessing include:

- **Feature Extraction and Selection:** Features that do not contribute effectively to drift prediction, such as current velocity and position, are dropped. These features, while providing absolute values per timestep, do not offer predictive value for the model and could potentially hinder learning by introducing noise and straying the model from converging on a local minima.

- **Feature Linearization:** Non-linear features are identified using a T-SNE plot (see Figure 3(*a*)), which visually represents how features correlate with each other. These non-linear features are then linearized to ensure better model training performance.

- **Important Feature Selector:** The final section within the preprocessing module selects the top features. A random forest regressor is employed to determine the most critical features for drift error prediction. The analysis helps in focusing the model on the most informative attributes.

The ground truth is calculated as the drift in the drone per given timestep using the formula below.

$$drift_e = current_pos[t] - intended_pos[t] \tag{1}$$

t is given as the absolute time since drone start. This clock helps consolidate like timesteps together. The preprocessed data, now optimized with selected and linearized features, is fed into an Artificial Neural Network (ANN). The ANN architecture, detailed on the right side of Figure 3(*a*), processes these inputs to learn meaningful relationships between the input data and the ground truth drift error. Most of the model architecture is standard except for the loss function which will be discussed in further detail in subsequent subsection. The final learned weights of this module is sent to the final interface module as shown in Figure 3(*a*). The interface module changes waypoint coordinates to account for drone drift at different battery levels.

3.1.2 Loss Function. The EMNN module aims to predict and minimize the drift of an aerial vehicle over varying battery levels and travel distances. Initial models, which used pure Euclidean distance for error calculation, were used to mitigate drift of various robotic agents [16]. However, these approaches utilizing pure Euclidean distance were found inadequate for capturing the nuanced dynamics of drift across individual axes. The revised loss function incorporates the Euclidean distance along with Manhattan distances across each axis to provide a more granular understanding of drift. The formula is given by:

$$\begin{aligned} Loss =& W_1 \cdot |curr_x - des_x| + W_2 \cdot |curr_y - des_y| + \\ & W_3 \cdot |curr_z - des_z| + 0.4 \cdot \\ & \sqrt{(curr_x - des_x)^2 + (curr_y - des_y)^2 + (curr_z - des_z)^2} \end{aligned} \tag{2}$$

Additional weight terms are added to adjust the importance of each axis based on the learning outcomes and the directional errors observed. These weights evolve over time, adapting to the predominant trends of drift errors identified during training. The first set of weight terms encapsulates direction by computing the relative difference between true and desired value across that axis. It is '-' if non absolute calculation is negative. The numerical portion of the weight is the severity of the deviation which is just a normalization with respect to all axis. The second weight term, originally initialized to 1/3, changes based on which axis is deviating more over time as the model learns. Similar to a long term scheduler, the weight changes based on an overall pattern of drone drift behavior [17]. Equation below shows the breakdown for W_1 which can be translated across the other three axis.

$$W_1 = \text{dir\_term} \cdot \text{sched\_weight} \tag{3}$$

$$\text{dir\_term} = \frac{\text{drift\_x}}{\text{drift\_x} + \text{drift\_y} + \text{drift\_z}} \tag{4}$$

$$\text{sched\_weight} = \frac{1}{2}\left(1 + \cos\left(\pi \cdot \frac{\text{drift\_axis}}{\text{total\_drift}}\right)\right) \tag{5}$$

3.2 FBLPN Module. Referring back to Figure 1, the Future Battery Life Prediction Network (FBLPN) module, while controlled by the central computer, operates independently from the

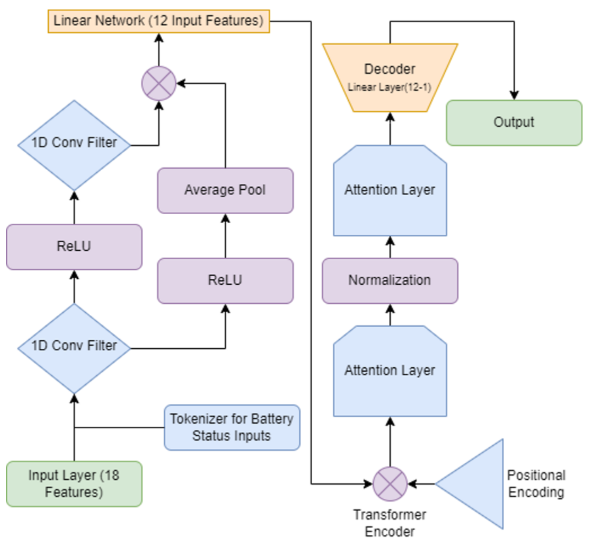


Fig. 4 FBLPN transformer architecture highlighting the various modules/components used to predict remaining flight distance from a masked sequence.

Crazyflie's direct control systems. Instead, it interfaces directly with the drones to manage their remaining flight times and orchestrate when a drone should return for charging. This is crucial for rotating drones efficiently and ensuring continuous operation without direct intervention in their movement plans. The structural and integration details of the FBLPN are depicted in Figure 3(*b*).

3.2.1 System Architecture. The architecture of the FBLPN, shown in Figure 3(*b*), includes a preprocessing module very similar to the EMNN module. A significant difference is the absence of filter representations that are prominent in the EMNN preprocessing steps. Moreover, the features selected for training the FBLPN are specifically tailored to address the nuances of battery life prediction, as detailed in the data collection section and summarized in Table 2. Since sequence is very important here, the preprocessing module has a file process section which properly handles the collected data as shown in Figure 3(*b*). Each data run is meticulously catalogued with the battery level at the start and the data points collected during that flight. This data is then organized into individual CSV files, each file annotated with the starting battery level and the total distance traveled during that run [6]. This systematic organization is critical for the encoder within the transformer module, allowing it to recognize the type of run and the aggregate distance traveled, thereby aiding in the learning of accurate latent representations [18].
The processed csv files are all saved and set as inputs for the main transformer learning module. The choice of a transformer architecture is due to its proven effectiveness in learning latent representations of sequences, particularly those of variable lengths, although, for training consistency, sequence lengths were standardized [19]. Figure 4 delineates the structure of the FBLPN learning module, which is divided into three principal sections, each addressing different aspects of sequence handling and learning:

(1) **Convolutional Filter:** The initial stage involves 1D convolutional filters which processes 18 input features, learning crucial feature representations over a fixed token size [20]. The implementation follows the methodology discussed in paper by Han, utilizing convolutional filters to preprocess inputs and enhance the transformer's ability to learn from these features [21, 22]. This approach not only improves the quality of feature representation but also optimizes the learning process by reducing the number of input features to the positional encoder, thus decreasing the training parameters from 11 million to under 9 million.
(2) **Multi-Head Encoder:** A tokenizer model, as described in the referenced paper, is employed to convert text-based modules of battery status into numerical representations suitable for processing in the latent space [23]. Input data sequences are positionally encoded using sine and cosine functions, which embed each sequence point effectively [19]. The encoded sequences are then processed through a multi-head self-attention mechanism, utilizing two attention heads to capture and model complex dependencies and relationships within the data, particularly between the battery's initial state and subsequent flight patterns.
(3) **Decoder and Output:** The final component involves decoding the next state's distance and time tokens to predict the remaining flight distance. This is achieved by inputting the trained attention variables along with a masked sequence into the decoder. The masked sequence contains the full sequence as per before but the remaining distance is masked [19]. Based on recent behaviors and accumulated flight data, the decoder is used to predict the masked remaining distance and output the final distance a drone can fly before the battery is completely exhausted.

The predicted remaining distance is then conveyed to the management module, depicted in Figure 3(*b*), which determines the next necessary action for the drone. The management module evaluates whether the drone can reach the next waypoint or requires redirection to the nearest available charging station. If the remaining flight distance exceeds the distance to the next point within a predetermined safety margin, the drone is cleared to continue its flight plan. Conversely, if the remaining distance falls short of this margin, the FBLPN signals the Priority Planning Plus Local Solve (PP+LS) module to route the drone to an appropriate charging station.

3.3 Prioritized Planning Cycle.

The final module is unique because it contains the main MAPF based planner in addition to an integration module that connects the previously mentioned modules to increase flight time. Though the individual modules above are tested within the real crazyflie setup, robust testing of this module is purely conducted on Crazyswarm2 inbuilt RViz simulation [15].

3.3.1 System Initialization. At the heart of our system is a novel planning approach, initially implemented with a random planner as a proof of concept. The planner operates under the assumption that the points visited by the drones are part of an infinite sequence, allowing each drone to continuously travel to new valid random points until a recharge is necessary. Essentially, the lifelong planner is 'designed' (not tested) to function perpetually until the ecosystem is manually shut down. The planner architecture is depicted in Figure 5, which illustrates the initialization process where all obstacles within a predefined map boundary are identified. For simulation, a ROS node reads the obstacle data, translating it into a 3D numpy array where each cell represents a discrete subregion. Subregions with more than 50% of their area covered by obstacles are marked as occupied (denoted by a '1'), while free spaces are denoted by a '0'. This mapping approach is also mirrored in real-life setups, with obstacle locations hard-coded within the bounds of the lighthouse system to prevent signal interference between the lighthouse module and the drones. The generated numpy map is then relayed to the PRM module to create a traversable road map.

3.3.2 PRM Roadmap Generation. As illustrated in Figure 5, subsequent to the occupancy map initialization, a 'semi-dynamic' (capability of the system to expand the initial roadmap by extending the occupancy map's range) roadmap is generated. The Probabilistic Roadmap (PRM) method works by generating random points that are verified against the occupancy map for validity. The number of random points generated is proportionally sized to the size of

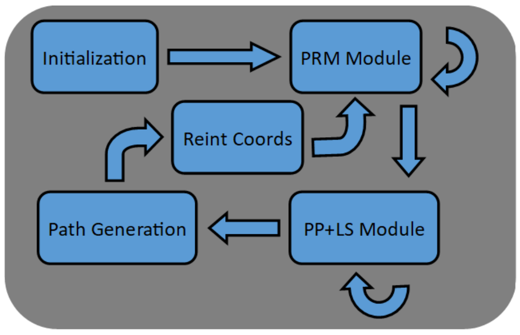


Fig. 5 The full prioritized path planning cycle.

the map. Connectivity between points is established based on their proximity within a predefined neighborhood range and the absence of obstacles along the direct Euclidean path between them. This process involves checking the two points' locations on the numpy occupancy array and verifying that there is no obstacle (non-zero value) blocking the path [8]. Once a connection is deemed valid, it is bidirectional, forming a comprehensive grid network that facilitates traversal from any point on the map. When a new path is required, the start and goal positions are temporarily integrated into this grid, allowing for the computation of paths for all agents involved [24].

3.3.3 Prioritized Planning. As illustrated in Figure 5, PP+LS module is the MAPF solver that calculates optimal paths for each agent. A priority-based solver is utilized, ensuring that drones with the least remaining battery receive the most optimal routes to the charging pads [14]. This prioritization is governed by a 'priority schema', detailed in Table 3. If paths for top-priority agents (categories 1-3) are successfully established but those for lower-priority agents (category 4) are not, the system reinitiates the prioritized planning (PP) process with a reshuffled order for category 4 agents, while maintaining the established paths for higher-priority agents [13]. If no viable path is found after reshuffling, or if a high-priority path fails, the system activates local search (LS).

Table 3 Agent Priority Planning Categories.

Category	Description
1	The agent is out of battery and needs to get the highest priority to the nearest viable charging pad whether empty or full.
2	The agent is charging and is being replaced by a category 1 agent to charge.
3	The agent has less than 20 percent charge and can be pathed to any open charging pad not occupied by another agent.
4	The agent has sufficient battery and is just being pathed to the next random goal point.

3.3.4 PP+LS (Local Search). LS is looped until a solution is found as depicted in Figure 6. The steps below shows how local search resolves conflicts [10].

(1) **Conflict Localization:** Local search starts by localizing the conflict area given by a set minimum radius. LS extracts the occupancy grid and the PRM roadmap that confine within this localized area.
(2) **Path Replanning:** The algorithm then attempts to replan a path for the lower-priority agent. This involves the agent temporarily waiting at a point on the periphery of the defined radius, allowing the higher-priority agent to pass without conflict. It involves sampling random nodes near the outside radius of the graph. Since this process is tedious, only ten attempts are made before a trial is considered unsuccessful. If successful, the system rechecks for new conflicts created and continues to resolve next conflict.
(3) **Conflict Resolution with CBS:** If a straightforward solution is not found, Conflict-Based Search (CBS) is employed within the local search radius to find an optimal solution. If no solution emerges, the search radius is gradually expanded until a viable path is found.

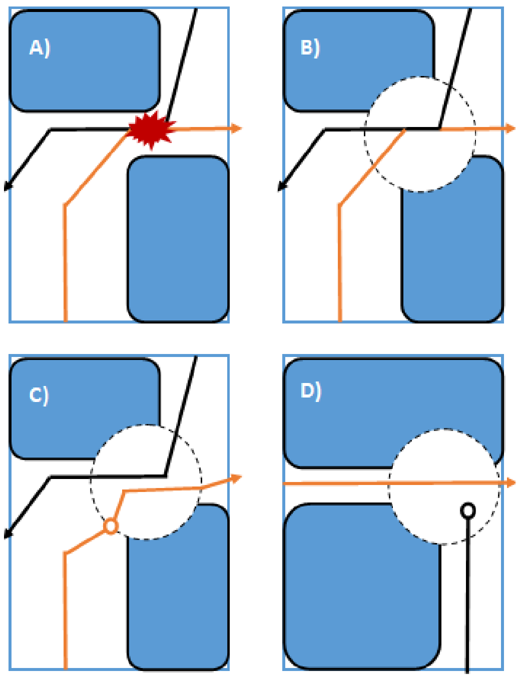


Fig. 6 A simplified overview of resolution method for local search. Step A) locates the conflict and step B) draws the boundary. In step C), the lower priority orange line waits at the edge of the radius highlighted by the orange circle till the high priority black line crosses. In step D) CBS needs the low priority orange agent to leave corridor before high priority can pass. High priority waits at the black dot.

Following the resolution of a conflict, the prioritized planning process is reinitiated. If additional conflicts are detected, local search continues iteratively until all conflicts are resolved. While this integration of local search does not necessarily expedite the computational process, it ensures that higher-priority agents maintain precedence in their routing, particularly in conflict situations. The pseudocode for this algorithm is shown in Algorithm 1 [10].

3.4 Integration. The three modules are integrated as shown earlier in Figure 2. The integration module interfaces directly with the crazyflie cfb api. The api is used to command the drone to complete different fly to actions. After PP+LS computes a path of coordinates, it is sent to the EMNN module which outputs a set of modified waypoints that mitigate drift when flying. Upon reaching its goal coordinate, FBLPN is used to compute if the next randomly generated point is feasible and classifies it according to the established category priority schema for path planning. After assigning paths for all active agents, the assigned path is sent back to EMNN network and the cycle continues. Note, a significant limitation of the current integration method is the coordination of drone movements. Once a drone reaches its designated goal, it must hover and wait for all other drones to similarly achieve their respective positions before the next set of movements can be calculated. This is a major limitation and can lead to inefficiencies.

Algorithm 1 Prioritized Planning with Local Search (PP+LS)

```
1:  Input: Set map boundary, Obstacle information, Drone states
2:  Output: Lifelong paths for drones
3:  Initialize 3D numpy array for occupancy grid
4:  Categorize occupied subregions and free space
5:  Initialize PRM roadmap based on occupancy map
6:  Assign initial priority order using ‘priority schema‘
7:  Initialize PP+LS module
8:  while system not shutdown do
9:      Generate random valid points for PRM
10:     Connect points if within neighborhood range and no obstacles in between
11:     Compute paths for all agents with PP using priority order
12:     if path for top 3 category agents found then
13:         Re-run PP with different ordering for category 4 agents
14:     else
15:         Enable LS for failed top 3 category agents
16:         Resolve conflicts with LS
17:     end if
18:     Check for new conflicts
19:     Run PP again if new conflicts found
20: end while
21: Interface with crazyflie cfb API for flight actions
22: Modify waypoints with EMNN module
23: Use FBLPN to check feasibility of next random points
24: Assign paths and continue the cycle
        return Lifelong paths for drones
```

4 Implementation

In this section, we introduce the network parameters and describe our training and testing strategies.

4.1 Model Parameters. The implementation utilizes an A* based priority planner within the PRM to search the roadmap and generate viable paths. The associated hyperparameters for PRM generation and the Local Search (LS) module, which resolves collisions, are varied and empirically determined. These parameters primarily define the search radius for connecting generated nodes (for roadmap generation) and the radius used within the LS module to resolve conflicts (PP+LS).

The EMNN consists of a simple Artificial Neural Network (ANN) with three hidden layers and a final output layer designed for linear regression, assessing custom loss against the ground truth as discussed in the methods section. The architecture of the network is as follows, starting from the input layer: 26-64-256-128-24-1. Detailed model parameters can be found in Table 4. For the FBLPN, the input layer accepts 18 features, processed through two 1D sequential filters with a stride of 1 and a kernel size of 4. These convolutional filters are applied in parallel across all sequence input points before transitioning to the transformer module, which features two multihead attention layers. Interlayer batch normalization is performed in the latent space, followed by a decoder that feeds into a feed-forward neural layer scaling from 64 to 1 [25, 26]. Parameters for the FBLPN model are also listed in Table 4.

4.2 Training and Testing. We train the model on NVIDIA RTX 4060 with 8gb of memory using PyTorch libraries. The EMNN model was trained with a learning rate of 0.008, while the FBLPN transformer module used a learning rate of 0.01, adjusted by a cosine annealing scheduler [27]. Both models utilized the ADAM optimizer and were trained over 100 epochs [17]. In training, the dataset was randomly divided into training (70%), validation (10%), and testing (20%) segments, with data shuffled in each iteration to ensure robustness. During testing, we sample random start and end points from our collected dataset and compute the drift the drone encountered during data collection. This value is compared against model output to evaluate performance.

Table 4 Hyperparameters for FBLPN and EMNN Models

FBLPN Model Hyperparameters

Hyperparameter	Description	Value
Learning Rate	Learning rate for the optimizer	0.008
Batch Size	Number of samples per batch	32
Epochs	Number of training cycles	100
Dropout Rate	Rate for dropout regularization	0.3
Optimizer	Optimization algorithm	Adam

EMNN Model Hyperparameters

Hyperparameter	Description	Value
Learning Rate	Learning rate for the optimizer	0.01
Hidden Layers	Number of hidden neural layers	3
Neurons per Layer	Number of neurons in each layer	128
Activation Function	Activation function for neurons	ReLU
Loss Function	Loss function for optimization	MSE

As for the FBLPN testing, random sequences from test runs were extracted while preserving their sequential integrity. The autoregressive module of the FBLPN then predicted the remaining travel distance from masked sequential inputs processed by the decoder. Testing was conducted using an Intel i9-14900HX CPU.

4.3 Performance Metrics. The following metrics are used for performance evaluation:

- Drift Reduction:

$$\text{num\_valid} = \sum_{i=1}^{N} \left[|\text{p\_drift}_i - \text{a\_drift}_i| < 2 \text{ cm} \right] \tag{6}$$

$$\%corrected = \text{num\_valid}/\text{N} * 100$$

  This metric evaluates the EMNN model's accuracy in predicting drift errors. It is quantified by the percentage of test points where the predicted drift (p_drift) deviates from the actual observed drift (a_drift) by less than 2 cm. Here, N represents the total number of points evaluated.

- Future Battery Life Remaining

$$\text{RMSE} = \sqrt{\frac{1}{N} * \sum_{i=1}^{N} [p\_dist - a\_dist]^2} \tag{7}$$

  The primary metric for evaluating the FBLPN model is the Root Mean Square Error (RMSE). In addition to RMSE, we also assess R^2 and MAE to further evaluate the variation in predictions and the average error in the distances predicted by the FBLPN model. Here, p_dist represents the predicted remaining distance to travel based on the model's output and a_dist is the actual distance the drone travelled before complete battery exhaustion.

5 Results

To demonstrate the efficacy of our integrated system, we initially evaluated the three algorithms—EMNN, FBLPN, and PP+LS—individually. This separate assessment allows us to establish a baseline performance for each before combining them to verify their collective impact on extending total flight time through optimized battery management. The three algorithms are integrated and total percent average increase in time of flight is evaluated against a naive modulator.

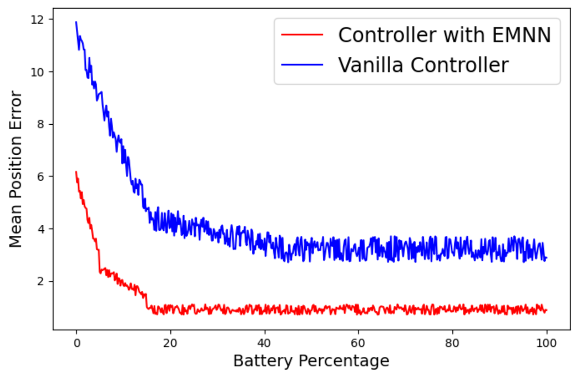


Fig. 7 EMNN performance compared to vanilla controller.

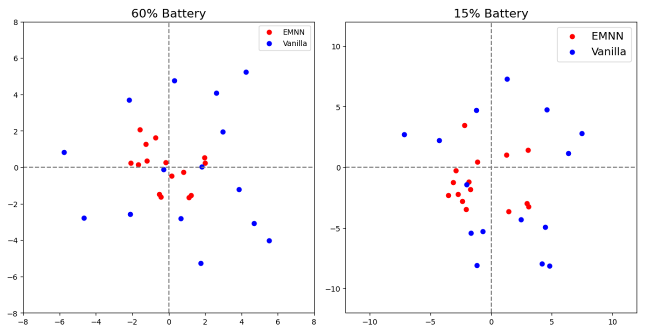


Fig. 8 Landing Accuracy for EMMN controller and vanilla controller at 60% battery (left) and 15% battery (right).

5.1 Individual EMNN Algorithm Results. As mentioned before, the EMNN model is trained on real collected crazyflie data based on random point generation. The evaluation compared the performance of a baseline controller, which inputs raw drone positions directly to the Crazyflie, against the EMNN-enhanced controller. For the controller incorporating EMNN, a forward pass through the tuned neural network was performed to derive the optimal set of coordinates for the drone, factoring in current speed, desired travel points, and battery state. These optimized coordinates were then communicated to the drone via the Crazyflie API. The performance is evaluated in the real setup mentioned in the implementation section. For EMNN testing, the drones followed a preplanned path that exceeded their battery life to fully assess the EMNN's effectiveness throughout the battery cycle. The model's accuracy was gauged by measuring the drift error—defined as the 3D Euclidean distance between the drone's actual position and the ground truth coordinates. The number of valid EMNN adjustments were counted using the formula in performance metrics section.

Figure 7 shows the performance of the controller with EMNN versus the vanilla controller. We evaluate the performance over 22 different ramdom generated cycles and average the drift error over all 22 cycles from full battery to complete battery depletion. The EMNN model significantly reduced the average drift error to about 1cm, compared to 3-4cm observed with the baseline controller at higher battery levels. Notably, as the battery level decreased below 40%, the error in the baseline controller progressively increased, whereas the EMNN model maintained an error close to 1cm. When battery levels dropped below 15%, both models exhibited increased errors due to the drone entering a low power mode. However, the EMNN model consistently outperformed the baseline, only exceeding a 10cm error threshold when the battery fell below 8%.

Another test performed evaluated the drones' landing accuracy on a charging pad during varying battery conditions. Figure 8 shows the accuracy of the drone landing on the center of a charging pad during 60% battery and 15% battery. There were 15

Table 5 Average land accuracy on charging pad for both controllers at different battery levels.

Controller Type	60% Charge	15% Charge
EMNN Controller	**2.1cm**	**3.9cm**
Vanilla Controller	6.1cm	8.3cm

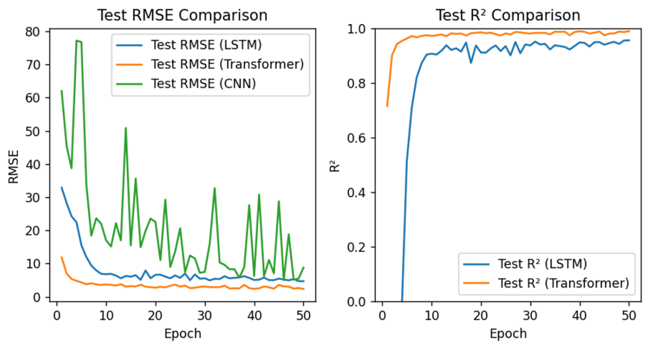


Fig. 9 Training Performance of CNN, LSTM, Transformers over epochs. The RMSE of Transformers is the least while LSTM is slightly higher. CNN shows no correlation to seuquence prediction as shown by its r2 value.

trials run for each type of controller. From Table 5, the EMNN model maintained an average landing precision within 1.5cm of the target (60% battery), improving to within 3.2cm at 15% battery. In contrast, the baseline controller generally missed the charging pad (8cm wide), with an average error of approximately 8.3cm. These results underscore the substantial improvements offered by the EMNN in navigation precision and operational reliability, particularly under low battery conditions.

5.2 Individual FBLPN Algorithm Results. The FBLPN model, leveraging a transformer-based architecture, was evaluated using real Crazyflie data collected from random preplanned precollected flight data. The model is evaluated against 2 other baseline models: LSTM and CNNs. These baseline models are considered theoretically less capable for our specific application due to their inherent limitations in handling sequential data effectively. Although CNNs are proficient in feature extraction, they lack mechanisms to retain sequential information over time. This deficiency is evident as shown in Figure 9, where CNNs demonstrate significantly poorer capability in learning latent representations compared to LSTM and Transformers [20, 28, 29]. Known for their ability to learn sequential data, LSTMs nevertheless struggle with long-term dependencies in extended sequences. This challenge often results in exploding or vanishing gradients, which compromise their effectiveness. Despite these issues, LSTMs displayed a comparable root mean square error (RMSE) to Transformers during tests involving shorter sequences as seen on Figure 9 [30]. During testing, Figure 9 shows transformers perform the best with a root mean square error (RMSE) of less than 1mm, indicating its robust capability in predicting the remaining flight distance of the drone accurately. This level of precision allows for reliable battery life estimation in scenarios where onboard battery prediction falls short, crucial for operational planning in real-time. The RMSE of LSTM is around the same as transformers. This test was conducted with a sequence length of 20 and average over 10 random path sequences from different initial battery percentages.

An extensive evaluation across multiple sequence lengths, ranging from 5 to 60 in increments of 5, revealed interesting dynamics between the LSTM and Transformer models. As shown in Figure 10, surprisingly, LSTM outperformed the Transformer in shorter sequences (5 to 15). However, as sequence lengths extended beyond 20, the Transformer model consistently achieved lower RMSE, underlining its improvement to handle longer data sequences effectively.

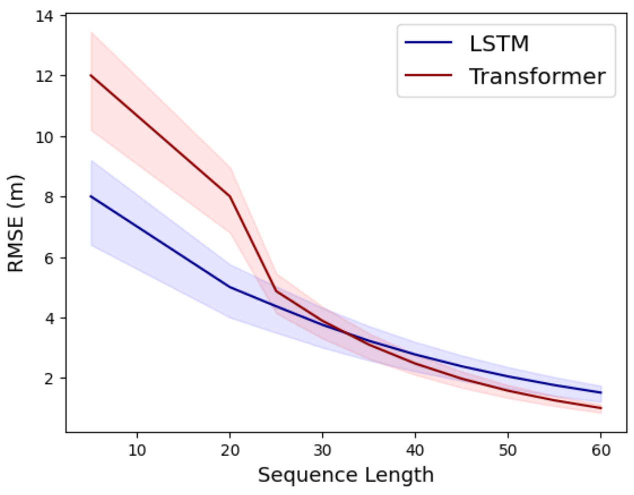


Fig. 10 The figure shows the performance of both LSTM and Transformer as sequence length increases from 5 to 60. At around 30 sequence length, transformer has a better performance and decreases to about 0.8m in remaining distance accuracy predictions.

5.3 Path Planning Results.

5.3.1 Simulation Environment. Unlike the other algorithms, the performance for the PP+LS was evaluated on a simulation environment. The simulation environment is explained in the Problem Setup environment. To test the algorithm, 25 random maps were generated featuring obstacles of three different shapes ranging in size from 0.5 to 1.5 meters. The number of obstacles within a map ranged from 3-8 at random. These maps were randomly generated of a simple publish marker Ros2 node script. In this simulated setting, the performance of PP+LS was benchmarked against two baseline algorithms: regular Priority Planning (PP) and simple Joint A*. The max solve time given was 4 seconds to be realistic during real time deployment and path finding. Figure 11 shows a theoretical path for PP+LS algorithm. Although Joint A* does not provide prioritization for low battery drones, it was still used as a baseline as it provided optimal solutions if a solution was found. Both naive planners were leveraged from online sources.

5.3.2 Performance Evaluation. Table 6 shows the evaluation metric for each algorithm. PP+LS had a 90% solve rate, with only a marginally lower solve rate compared to PP. This is due to slight increase in time is attributed to its localized optimal solving strategy, which contrasts with broader optimal solve algorithms. Hence, no solution was found within 4 seconds for some maps. Joint A*, while providing the highest quality solutions with the shortest average path lengths, achieved a solve rate of 67% with three drones and only 12% with five drones, indicating scalability issues. PP had a much quicker solve time and had 100% solve percentage for both cases but there could be unsolvable instances for PP if the maps were designed better. Additionally, PP based algorithms can prioritize lower battery drones for more optimal paths. PP+LS was the superior algorithm with the ability to solve most scenarios where is is only limited by solve time. Improving algorithm efficiency of PP+LS could allow it to solve all scenarios. Also specialized maps will show examples where PP+LS solves scenarios where PP cannot.

5.4 Overall System Results. The comprehensive performance of the integrated Flight Time Maximization Module

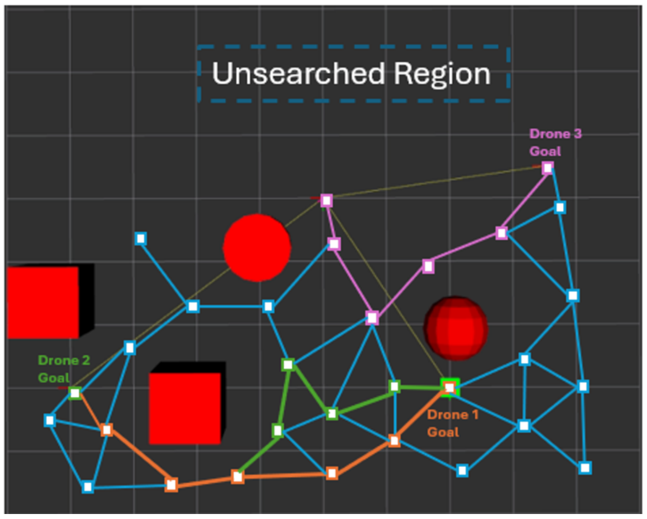


Fig. 11 A sample path PP+LS plans out for all three agents. The orange drone one gets the most priority as that is the low battery drone followed by the green drone two and finally drone three which is just going from one assigned point to another.

(FTMM) was assessed in two distinct scenarios to validate its efficacy in extending operational flight times. Both scenarios employed identical maps featuring two one-meter square foam boxes located at the origin and at (2m, 2m), respectively. A wireless charging pad was strategically placed between these boxes at (1.2m, 1.5m), creating a bottleneck that tested the coordination of drone takeoffs and landings. The initial conditions varied only in the drones' starting points: in each scenario, two drones began at full battery capacity while one started at 30% battery. The system operated continuously, directing drones to random points until the cumulative flight time reached 30 minutes. To benchmark the FTMM's performance, a baseline algorithm was employed where three drones flew randomly until their battery levels dropped to 25%, at which point they would land to recharge. This cycle was repeated until the combined flight time of all three drones reached 30 minutes. The effectiveness of both the FTMM and the baseline approach was measured by the total time taken to log 30 minutes of flight. Table 7 depicts the FTMM results compared to the naive baseline model.
The FTMM facilitated a slight improvement in the efficiency of flight time accumulation, enabling the drones to remain airborne approximately 6% longer than the baseline, naive method. Though there was an increase, the increase was very slight due to the FBLPN inconsistency in truly detecting remaining battery life. Another inaccuracy could be the The planning module sometimes failed to accurately account for the drones' energy consumption during flight between waypoints, leading to inefficiencies. Hence, shown in Table 7, the number of time the drone completely drained battery and failed to reach the charging pad was significantly higher than the naive conservative planner.

6 Conclusion

This research demonstrated significant advancements in UAV operational efficiency through the integration of the Flight Time Maximization Module (FTMM), incorporating two AI-driven algorithms and an innovative planning module: Error Mitigation Neural Network (EMNN), Future Battery Life Prediction Network (FBLPN), and Prioritized Planning with Local Search (PP+LS). The EMNN effectively minimized navigation errors to within 1 cm under battery conditions >40%, markedly improving over traditional control methods. The FBLPN utilized a transformer-based architecture to accurately predict battery depletion, outperforming

Table 6 Performance of all 3 algorithms over various categories for 3 agents

Algorithm Type	Success Rate (%)	Average Solution Cost (m)	Average Solve Time (s)
Joint A*	67	4.3	5.2
Vanilla PP	100	5.8	0.8
PP+LS	90	5.4	2.6

Table 7 FTMM Performance compared to Naive Conservative Lifelong Algorithm. Total time for each Algorithm to reach a total of 30 minutes of flight time.

Algorithm Type	Time to 30min	Num of Failures
Naive	61min	0
Scenario 2	58min	3
Scenario 1	**55min**	2

conventional LSTM and CNN models. Collectively, these novel models led to a 6% increase in UAV flight time in field tests, show the potential of integrated AI strategies to enhance current UAV platforms.

Looking ahead, our future research will focus on enhancing testing capabilities to prove FTMM performance over other naive methods. The current model lacks the ability to profoundly gauge the remained battery life left per charge given a future set of actions. Additionally, the current planning module lacks a robust method to find potential points within local search which is dramatically increasing the solve time. These changes will allow for a more streamlined overarching module to better predict future drone actions based on current battery usage and status to take more educated risks on remaining longer in the air without compromising complete battery loss.

Acknowledgment

I would like to acknowledge the MAIL lab as well as Professor Amir Barati Farimani.

References

[1] Liu, X., Yang, C., Meng, Y., Zhu, J., and Duan, Y., 2023, "Capacity estimation of Li-ion battery based on transformer-adversarial discriminative domain adaptation," AIP Advances, **13**(7), p. 075113.

[2] Cai, Y., Li, W., Zahid, T., Zheng, C., Zhang, Q., and Xu, K., 2023, "Early prediction of remaining useful life for lithium-ion batteries based on CEEMDAN-transformer-DNN hybrid model," Heliyon, **9**(7), p. e17754.

[3] Chen, D., Hong, W., and Zhou, X., 2022, "Transformer Network for Remaining Useful Life Prediction of Lithium-Ion Batteries," IEEE Access, **10**, pp. 19621–19628.

[4] Zhang, W., Jia, J., Pang, X., Wen, J., Shi, Y., and Zeng, J., 2024, "An Improved Transformer Model for Remaining Useful Life Prediction of Lithium-Ion Batteries under Random Charging and Discharging," Electronics, **13**(8).

[5] Ren, D., Lu, L., Shen, P., Feng, X., Han, X., and Ouyang, M., 2019, "Battery remaining discharge energy estimation based on prediction of future operating conditions," Journal of Energy Storage, **25**, p. 100836.

[6] Hsu, C.-W., Xiong, R., Chen, N.-Y., Li, J., and Tsou, N.-T., 2022, "Deep neural network battery life and voltage prediction by using data of one cycle only," Applied Energy, **306**, p. 118134.

[7] Meneghetti, L., Demo, N., and Rozza, G., 2021, "A Dimensionality Reduction Approach for Convolutional Neural Networks," CoRR, **abs/2110.09163**.

[8] Mohsan, S. A. H., Othman, N. Q. H., Li, Y., Alsharif, M., and Khan, M., 2023, "Unmanned Aerial Vehicles (UAVs): Practical aspects, applications, open challenges, security issues, and future trends," Intelligent Service Robotics.

[9] Li, W., Chen, H., Jin, B., Tan, W., Zha, H., and Wang, X., 2022, "Multi-Agent Path Finding with Prioritized Communication Learning," *2022 International Conference on Robotics and Automation (ICRA)*, pp. 10695–10701, doi: 10.1109/ICRA46639.2022.9811643.

[10] Solis, I., Motes, J., Qin, M., Morales, M., and Amato, N. M., 2023, "Adaptive Robot Coordination: A Subproblem-based Approach for Hybrid Multi-Robot Motion Planning," 2312.08554

[11] Wang, B., Liu, Z., Li, Q., and Prorok, A., 2020, "Mobile Robot Path Planning in Dynamic Environments through Globally Guided Reinforcement Learning," 2005.05420

[12] Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R., 2015, "Conflict-based search for optimal multi-agent pathfinding," Artificial Intelligence, **219**, pp. 40–66.

[13] Zhang, S., Li, J., Huang, T., Koenig, S., and Dilkina, B., 2022, "Learning a Priority Ordering for Prioritized Planning in Multi-Agent Path Finding," *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pp. 208–216.

[14] Čáp, M., Novák, P., Kleiner, A., and Selecký, M., 2015, "Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots," IEEE Transactions on Automation Science and Engineering, **12**(3), pp. 835–849.

[15] Pichierri, L., Testa, A., and Notarstefano, G., 2023, "CrazyChoir: Flying Swarms of Crazyflie Quadrotors in ROS 2," 2302.00716

[16] Nam, Y.-W., Cho, H.-Y., Kim, D.-Y., Moon, S.-H., and Kim, Y.-H., 2020, "An Improvement on Estimated Drifter Tracking through Machine Learning and Evolutionary Search," Applied Sciences, **10**(22).

[17] Kingma, D. and Ba, J., 2015, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA.

[18] Aitken, K., Ramasesh, V. V., Cao, Y., and Maheswaranathan, N., 2021, "Understanding How Encoder-Decoder Architectures Attend," CoRR, **abs/2110.15253**.

[19] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I., 2017, "Attention is All you Need," *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., Vol. 30, Curran Associates, Inc., https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[20] O'Shea, K. and Nash, R., 2015, "An Introduction to Convolutional Neural Networks," 1511.08458

[21] Han, Y., Li, C., Zheng, L., Lei, G., and Li, L., 2023, "Remaining Useful Life Prediction of Lithium-Ion Batteries by Using a Denoising Transformer-Based Neural Network," Energies, **16**(17).

[22] Song, W., Wu, D., Shen, W., and Boulet, B., 2023, "A Remaining Useful Life Prediction Method for Lithium-ion Battery Based on Temporal Transformer Network," Procedia Computer Science, **217**, pp. 1830–1838, 4th International Conference on Industry 4.0 and Smart Manufacturing.

[23] Zhou, A. and Farimani, A. B., 2024, "FaultFormer: Pretraining Transformers for Adaptable Bearing Fault Classification," 2312.02380

[24] Alarabi, S., Luo, C., and Santora, M., 2022, "A PRM Approach to Path Planning with Obstacle Avoidance of an Autonomous Robot," *2022 8th International Conference on Automation, Robotics and Applications (ICARA)*, pp. 76–80, doi: 10.1109/ICARA55094.2022.9738559.

[25] Xu, J., Sun, X., Zhang, Z., Zhao, G., and Lin, J., 2019, "Understanding and Improving Layer Normalization," *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds., Vol. 32, Curran Associates, Inc., https://proceedings.neurips.cc/paper_files/paper/2019/file/2f4fe03d77724a7217006e5d16728874-Paper.pdf

[26] Ioffe, S. and Szegedy, C., 2015, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," CoRR, **abs/1502.03167**.

[27] Loshchilov, I. and Hutter, F., 2016, "SGDR: Stochastic Gradient Descent with Restarts," CoRR, **abs/1608.03983**.

[28] Krizhevsky, A., Sutskever, I., and Hinton, G. E., 2012, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds., Vol. 25, Curran Associates, Inc., https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[29] Wang, S., Jin, S., Bai, D., Fan, Y., Shi, H., and Fernandez, C., 2021, "A critical review of improved deep learning methods for the remaining useful life prediction of lithium-ion batteries," Energy Reports, **7**, pp. 5562–5574.

[30] Hochreiter, S. and Schmidhuber, J., 1997, "Long Short-term Memory," Neural computation, **9**, pp. 1735–80.

List of Figures

List of Tables