AI algorithm for predicting and optimizing trajectory of massive UAV swarm

Amit Raj, Kapil Ahuja, Yann Busnel

AI Algorithm for Predicting and Optimizing Trajectory of Massive UAV Swarm

Amit Raj[a], Kapil Ahuja[a,*], Yann Busnel[b]

[a]*Math of Data Science & Simulation (MODSS) Lab, Computer Science & Engineering, IIT Indore, Indore, 453552, Madhya Pradesh, India*
[b]*IMT Nord Europe – Campus de Recherche, Douai, F-59508, Nord, France*

Abstract

This paper explores the application of Artificial Intelligence (AI) techniques for generating the trajectories of massive swarm of Unmanned Aerial Vehicles (UAVs). The two main challenges addressed include accurately predicting the trajectories of UAVs and efficiently avoiding collisions between them, which we discuss in the two paragraphs below, respectively.

In previous works that did trajectory predictions, a Neural Network (NN) was used. The activation functions used were all standard like Sigmoid, Tanh, and ReLU, which resulted in low trajectory prediction accuracy. In this work, we apply application-oriented activation functions of Swish and Elliott that are known to be resilient to noisy data, which is common in UAV trajectory prediction. We also propose our new activation function, AdaptoSwelliGauss that is fusion of Swish, Elliott and a scaled and shifted Gaussian. This combination better captures the complexities of UAV trajectory prediction (noisy data as well as non-linear trajectory). The trajectory prediction accuracy obtained with our new activation function is three to four orders-of-magnitude better than that obtained from the standard activation functions.

In the UAV context, collision detection and avoidance of UAVs is of utmost importance. While there is a common standard for collision detection, collision avoidance can be done by multiple methods. The first method is by changing their trajectories and the second method is by changing their start times (called batching). When we try to smoothly change the trajectories, then the algorithm gets complicated and runs into an infinite loop. On other

*Corresponding author:
Email address: kahuja@iiti.ac.in (Kapil Ahuja)

hand, when we apply the batching method to our setup, then the number of batches is large delaying the launch of all UAVs. Therefore, in this paper we propose a novel collision avoidance strategy that combines a new trajectory change method with the batching method. This results in smooth, simple, and finite trajectory changes in the first method, and reduction-by-half in the number of batches in the second method.

Keywords: UAV Swarm, Trajectory Prediction, Feed Forward Neural Network, Collision Avoidance, Batching.

1. Introduction

UAVs have become increasingly popular in recent years due to their versatility and potential for a wide range of applications, from surveillance and monitoring to delivery and transportation. However, safe and efficient operation of UAVs in complex environments remains a significant challenge Hasan et al. (2023), particularly when multiple UAVs are involved. A key issue is the need to optimize the trajectories of the UAVs to achieve various objectives, such as minimizing travel time, avoiding collisions, and maximizing coverage. Traditional methods for trajectory planning and control are often limited in their ability to handle the complexity and uncertainty of real-world scenarios and may not be scalable to large swarm of UAVs.

Prior research, exemplified by Qiu and Duan (2020), Reda et al. (2024) and Xu et al. (2024) has demonstrated the efficacy of leveraging non-linear optimization techniques to solve the problem of **UAV trajectory prediction**. When quick trajectory changes are required, the optimization routines are too slow and not adaptive. Artificial Intelligence (AI) techniques, particularly those based upon Neural Networks (NNs), have shown great promise in addressing these challenges.

Almost all of the NNs based previous works have used standard activation functions such as Sigmoid, Tanh, and ReLU. Although these NNs perform well, their trajectory prediction accuracy needs improvement Xue (2017), Xiao et al. (2019), Lai (2020), Sarkar et al. (2020), Jeong et al. (2021), Jiang et al. (2022a). We focus on this aspect here, and have the below contributions.

- We systematically apply both standard and application-oriented activation functions to a NN to predict the trajectories of UAVs. The standard activation function used are Sigmoid, Tanh, and ReLU. The

application-oriented activation functions used are Swish and Elliot, which are known for their resilience to noisy data common in UAVs.

- We combine the application-oriented activation functions with a Gaussian function to better capture the application's behavior (non-linear trajectory). We term our new activation function as AdaptoSwelliGauss.

- Our AdaptoSwelliGauss based NN gives three to four orders-of-magnitude better trajectory prediction accuracy than the best standard activation function (ReLU) based NN.

In the current context, **collision detection and avoidance of UAVs** is also very important. There exists a common standard for collision detection but not for collision avoidance, which can be can be performed by multiple methods. The *first method* is by changing their trajectories Park et al. (2008), Tang et al. (2014), Elmkaiel and Serebrenny (2019), Wan et al. (2019), Ho et al. (2019), Huang et al. (2024). All these approaches work well, however, they have a few drawbacks. While trying to achieve smooth trajectory changes, these algorithms get fairly involved. Further, any alteration in the trajectory of one UAV may inadvertently create collisions with other UAVs, leading to a challenging situation (infinite loops).

The *second method* to avoid collisions between UAVs is to change their start times. Sastre et al. (2022a) and Sastre et al. (2022b) have proposed such a popular approach. They have employed a batching mechanism, creating groups of UAVs with non-colliding trajectories to facilitate safe flight. However, the creation of multiple batches is a time-consuming process, which delays the overall launch of the UAV swarm.

Here, we introduce a novel collision avoidance algorithm that overcomes the drawbacks of both the above methods, and have contributions as below.

- The UAV simulator that we use provides the positions of the UAVs at different time instances (e.g., every 1 second or 0.1 second etc.) Park et al. (2008). Hence, we propose a trajectory change approach that perturbs the position of one of the colliding UAVs at the point of collision. Further, we slowly increase the perturbation before the collision point, reach the maximum perturbation at the collision point, and then slowly decrease the perturbation after the collision point.

- We integrate the trajectory change method with the batching method, leading to our final algorithm.
- Our proposed algorithm leads to smooth, simple, and finite trajectory adjustments for the first method, and a reduction-by-half in the number of batches for the second method.

The remainder of the paper is organized as follows: Section 2 reviews the literature; Section 3 describes our proposed neural network that is based upon different activation functions including our new proposed one; Section 4 discusses our new collision detection and avoidance technique; Section 5 presents numerical results; and Section 6 concludes the paper and suggests directions for future work.

2. Literature Review

In this section, we *first* highlight the previous works that use NNs for UAV trajectory prediction. These works are summarized in Table 1. As evident from this table, they address two different scenarios; trajectory prediction under effects of wind and to minimize the flight time. Most of these works have used a Feed Forward Neural Network (FFNN) Laudani et al. (2015) as the underlying AI architecture. Some works have used advanced neural networks (e.g., Deep Neural Networks or DNNs Al-Khazraji et al. (2022)) as well. The activation functions used by them are standard (Sigmoid, Tanh, and ReLU).

We propose use of a FFNN not only because it is more popular, but also because it works better than advanced neural networks in this context. This comparison is given in Appendix A. When we apply a FFNN with standard activation functions to our data[1], we get low trajectory prediction accuracy. We propose use of FFNN with application-oriented activation functions Swish, Elliot, and our new AdaptoSwelliGauss leading to substantially improved accuracy (by three to four orders-of-magnitude).

Second, we review the literature on collision detection and avoidance. Detecting collisions between UAVs is standard, where UAV positions or velocities or both have been used Park et al. (2008), Tang et al. (2014),

[1]It is important to note that the data used to train and test the NN is different for each work.

Table 1: Use of NNs for UAV Trajectory Prediction.

Previous Works	**Focus of Trajectory Prediction**	**AI Arch.**	**Activation Function**	**Order of MSE**
Xue (2017)	Under Effects of Wind	FFNN	Tanh	10^{-3}
Xiao et al. (2019)	Under Effects of Wind	Recurrent NN	Tanh	10^{-3}
Lai (2020)	Minimize Flight Time	FFNN	Tanh	10^{-2}
Sarkar et al. (2020)	Minimize Flight Time	FFNN	Tanh	10^{-4}
Jeong et al. (2021)	Under Effects of Wind	Deep NN	ReLU	10^{-6}
Jiang et al. (2022a)	Under Effects of Wind	FFNN	Sigmoid	10^{-2}

Elmkaiel and Serebrenny (2019), Wan et al. (2019), Ho et al. (2019), Huang et al. (2024).

The basic approach is to use positions of UAVs to detect collisions. We use this because focus on this work is less on collision detection and more on collision avoidance. More advanced approaches that use velocities of UAVs can also be used.

There are two main methods to avoid collisions between UAVs: by changing their trajectories or by changing their start times (batching). Previous works on the trajectory change method are summarized in Table 2. As evident from this table, in these works the trajectories of UAVs have been altered at the collision point either by changing their position, direction, speed or by adding a force to them. When we try to achieve smooth trajectory changes using these approaches, they get complicated (run into an infinite loop). The reason for this is that these works have been designed to work for a small set of UAVs (from a couple of UAVs to few tens of UAVs) while we work with massive UAVs (hundreds of them).

The UAV simulator setup that we use helps achieve smooth trajectory changes simply by slowly perturbing the position of one the colliding UAVs (without accessing its direction, speed, or acceleration). We integrate this method with the below discussed batching method, which leads to finite loops

as well.

Only two works have proposed the batching method Sastre et al. (2022a), Sastre et al. (2022b), and one is extension of the other. We integrate this method with the above trajectory change method leading to faster launch of all UAVs.

Table 2: Avoidance Approaches Applied to Colliding UAVs.

Previous Works	**Avoidance Approaches for Colliding UAVs**
Park et al. (2008)	Change their Direction
Tang et al. (2014)	Change their Direction
Elmkaiel and Serebrenny (2019)	Apply Repulsive Force to them
Wan et al. (2019)	Change their Direction
Ho et al. (2019)	Change their Velocities
Huang et al. (2024)	Change their Positions

3. Proposed Neural Network using Different Activation Functions

A FFNN comprises of an input layer, hidden layers, and an output layer. The input layer is structured to incorporate the initial, intermediate, and final states of the UAV across all three spatial coordinates: X, Y, and Z. These states collectively form a multidimensional input vector that encapsulates the complete trajectory information of the UAV.

Hidden layers play a pivotal role in capturing and representing complex patterns within the input data. Based upon previous works, we use only one hidden layer. Comprising 15 neurons, this layer employs various activation functions to nonlinearly transform the input, facilitating the extraction of relevant features and patterns essential for accurate prediction. This is the layer where we experiment with different types of activation functions and propose a new activation function as well, which is discussed below.

Responsible for producing the final predictions, the output layer is meticulously designed with 201 output neurons, which comprise linear polynomials. The deliberate choice of linear activation functions in this layer aligns with the nature of regression tasks, where the objective is to predict continuous

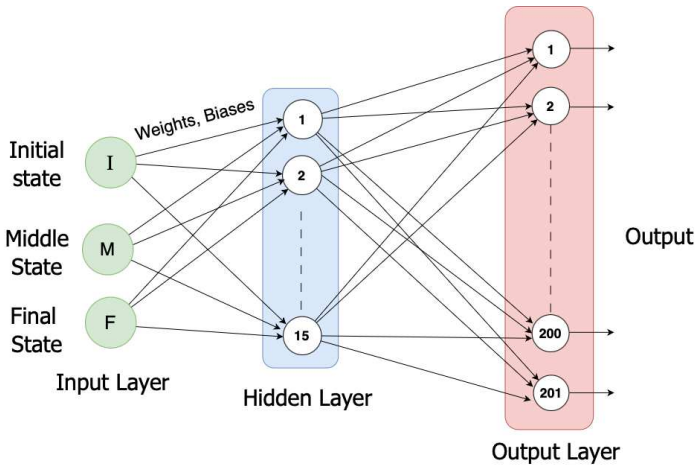


Figure 1: FFNN Architecture.

numerical values. We don't change activation functions in this layer. The diagrammatic representation of this network is illustrated in Figure 1.

To effectively train our neural network architecture, we employ the Levenberg – Marquardt (LM) back propagation algorithm from Hagan and Menhaj (1994). This optimization technique blends the advantages of both gradient descent and the Gauss-Newton method and updates old network's weights w_{old} to new weights w_{new} by solving

$$w_{\text{new}} = w_{\text{old}} - (J^T J + \lambda I)^{-1} J^T e, \tag{1}$$

where J is the Jacobian matrix of the network's error function, λ is the damping parameter that controls the transition between the gradient descent and Gauss-Newton approaches, I is the identity matrix, and e represents the error vector. When λ is large, the update rule resembles gradient descent, while for smaller values of λ, it approaches the Gauss-Newton method.

The combination of a diverse set of activation functions in the hidden layer and linear activation functions in the output layer forms a comprehensive framework for our research, allowing us to address the specific complexities and intricacies of our dataset. Here, we initially describe both types of existing activation functions, i.e., standard (Sigmoid, Tanh, ReLU) and application-oriented (Swish and Elliot) (in Section 3.1). Further, we describe our novel AdaptoSwelliGauss activation function (in Section 3.2).

3.1. Standard and Application-Oriented Activation Functions

In a neural network, an activation function plays a crucial role by applying a mathematical operation to the weighted sum of inputs. This introduces non-linearity, which is essential for the network to comprehend intricate patterns and connections within the data. Essentially, the activation function decides whether a neuron should fire or remain inactive, thereby impacting the flow of information throughout the network. Different types of activation functions are enumerated below. Here, x represents the input to each activation function.

1. **Sigmoid:** The Sigmoid function maps input values into a probability range between 0 and 1, which is ideal for binary classification outputs Ding et al. (2018). The formula for this is

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2}$$

It's major drawback is the vanishing gradient problem.

2. **Tanh:** Tanh extends the Sigmoid activation function, mapping inputs to a range between -1 and 1, which is zero-centered. It typically results in faster convergence than Sigmoid Orr and Müller (2003). The formula for this is

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{3}$$

Like Sigmoid, it suffers from vanishing gradients, affecting its utility in deep networks.

3. **ReLU:** ReLU computes the maximum value between zero and input. It addresses some of the critical issues of earlier activation functions (prevents vanishing gradients Apicella et al. (2021)) and is computationally cheaper as well. The formula for this is

$$f(x) = \max(0, x). \tag{4}$$

Nonetheless, ReLU is susceptible to the "dying ReLU" problem.

4. **Swish:** Swish is a self-gated activation function that blends input and Sigmoid output. It deals with the vanishing Gradient problem Ramachandran et al. (2017). The formula for this is

$$f(x) = x \cdot \sigma(\beta x), \tag{5}$$

where σ is the Sigmoid function and β is a trainable parameter. It requires careful tuning of β, which is computationally expensive Dash (2021).

5. **Elliot:** Elliot is computed via ratio of input and its absolute value. It is designed to provide a simpler and computationally efficient alternative to the Tanh activation function with the same property Dennis et al. (2020). The formula for this is

$$f(x) = \frac{x}{1 + |x|}. \tag{6}$$

It suffers from longer convergence time for large values.

3.2. Novel Activation Function of AdaptoSwelliGauss

The UAV data contains a lot of noise due to the inherent jerky nature of the automatic control of UAVs Shi et al. (2024). Trajectories of UAVs are non-linear as well Singh et al. (2022). The activation function that behaves well under both these condition (noisy data as well as non-linearity) is the Swish Nikhade (2023). However, the Swish activation function is highly sensitive for large values of input. Hence, we combine it with the Elliot activation function Salmeron and Ruiz-Celma (2019), which besides being less sensitive for large values of input, is also considered good for noisy data. Since Elliot does not have non-linear component, we multiply it with a scaled and shifted Gaussian Contributor (2024). Hence, our new activation function has the following formula:

$$\text{AdaptoSwelliGauss}(x) = \begin{cases} \text{Swish}(x), & \text{if } x \leq \alpha \\ \text{Elliott}(x) \times \text{Scaled Shifted Gaussian}(x), & \text{otherwise} \end{cases} \tag{7}$$

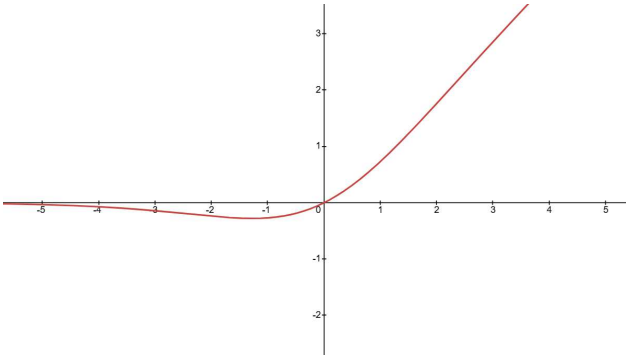
In this formulation,

- x represents the input to the activation function,
- $\text{Swish}(x)$ is the output when the Swish activation is applied to input x,
- $\text{Elliott}(x)$ is the output when the Elliott activation is applied to input x,
- $\text{Scaled Shifted Gaussian}(x)$ is the output of the function

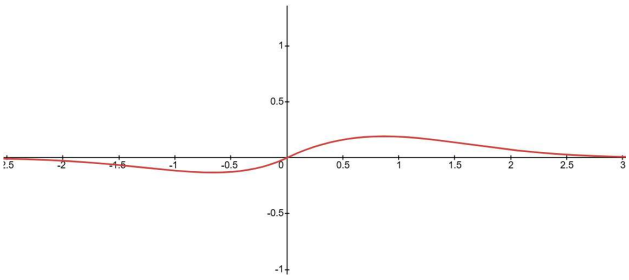
$$Scale \cdot e^{-\frac{(x-Shift)^2}{2}}.$$

- α and β (of Swish) are the hyperparameter adjusted based on experimentation and learning.

The plot of this activation function, which is dependent upon the relationship of α with input, is given in Figure 2. Some properties that we can read from this figure are that the function is bounded below, it is non-monotonic, and it is also not differentiable everywhere (because of Elliot).



(a) Behaves like Swish(x) when $x \leq \alpha$.



(b) Behaves like Elliot(x) $\times$ Scaled Shifted Gaussian(x) when $x > \alpha$.

Figure 2: Behavior of AdaptoSwelliGauss.

4. Integrated Collision Detection and Avoidance by Trajectory and Start Time Changes (ICDATS)

As discussed earlier, this section presents our system designed for efficient collision detection (in Section 4.1) and avoidance (in Section 4.2) between the different UAVs in a massive UAV swarm simulation, which is the focus of this work. A more advanced scenario is where obstacles (static or dynamic) also appear in the trajectory of the UAVs. For the sake of completeness, we present brief experiments with this advanced aspect in Appendix B. More

extensive experimentation and integration with the current work can be done and is part of future work.

4.1. Collision Detection

For collision detection, we use the basic UAV position based approach from Elmkaiel and Serebrenny (2019), Wan et al. (2019), and Ho et al. (2019). As practically common, we assume that all UAVs take off and fly simultaneously, following predefined trajectories between their respective pickup and delivery points. In this collision detection methodology, we employ a three-dimensional virtual sphere around each UAV referred to as the "collision sphere" at every time instance. This sphere is defined by a parameter "R" that denotes its radius (See Figure 3). On a broader level, a collision is defined when any UAV's collision sphere intersects with the collision sphere of another UAV at a particular time instance. Next, we make this statement precise.

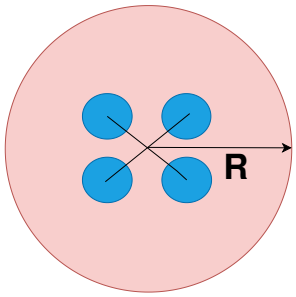


Figure 3: Collision sphere around the UAV.

Let us be given a list of UAVs and their respective trajectories in terms of X, Y, and Z coordinates at different time instances, also called waypoints. The granularity of time instances can be adapted based upon user need. Without loss of generality for a pair of UAVs, the standard algorithm calculates the Euclidean distance between their positions at each waypoint, and checks if these set of distances falls below a user-defined distance threshold.

Let the position vector of the i^{th} UAV at the w^{th} waypoint is given by $\mathbf{p}_i(w)$. The Euclidean distance between the i^{th} and j^{th} UAVs at this waypoint is

$$d_{ij}(w) = \|\mathbf{p}_i(w) - \mathbf{p}_j(w)\|. \tag{8}$$

A collision is detected, if a distance condition is satisfied, i.e., if $d_{ij}(w) < 2R + \delta$, where δ is a user-defined threshold.

If a collision is detected, then instead of the standard procedure of building a list of colliding pairs of UAVs, we apply the collision avoidance strategy discussed below. Finally, this procedure is repeated for all pairs.

4.2. Collision Avoidance

There are two methods for collision avoidance: one by changing trajectories of UAVs and the other by adjusting start times of UAVs. The novelty of our algorithm lies in integrating these two methods. We define the system as efficient when performs smooth, simple, and finite trajectory changes to a UAV's path, and substantially reduces the number of batches required to dispatch all UAVs using start time adjustments. We propose such an efficient system that is discussed in-detail below.

4.2.1. Collision Avoidance by Trajectory Changes

To avoid collisions, we propose a simpler variant of the trajectory change approaches from Park et al. (2008), Tang et al. (2014), Wan et al. (2019) (using positional perturbation). Without loss of generality, we assume UAV1 and UAV2 are two UAVs that would collide. Without loss of generality again, we change the trajectory of UAV1.

Assuming that two UAVs are colliding at waypoint w_c, which as above is a three-dimensional position array at the c^{th} time instance. We have the option of changing the position of UAV1 in any of the X,Y, and Z directions. Without loss of generality, we change the position of UAV1 in the X direction, i.e., we add (2R+`safe distance`) to $(w_c)_{\text{X}}$. Here, R is the collision sphere discussed in the detection section and `safe distance` is the extra distance added between the "collision sphere" of the two UAVs. This approach moves the UAV1 to a different position at the c^{th} time instance.

There might be a jerk by this sudden change in the position of UAV1. Hence, to make this update smoother, we start deviating from the original trajectory k time instances before the c^{th} time instance, reaching the changed position at the c^{th} time instance, and merging back to the original trajectory k time instances after the c^{th} time instance. This process is mathematically demonstrated below.

$$\begin{pmatrix} w_{c-k} \\ \vdots \\ w_{c-1} \\ w_c \\ w_{c+1} \\ \vdots \\ w_{c+k} \end{pmatrix} + \begin{pmatrix} \left(\frac{2R+\text{safe distance}}{k}\right) \times 1 \\ \vdots \\ \left(\frac{2R+\text{safe distance}}{k}\right) \times k-1 \\ 2R+\text{safe distance} \\ \left(\frac{2R+\text{safe distance}}{k}\right) \times k-1 \\ \vdots \\ \left(\frac{2R+\text{safe distance}}{k}\right) \times 1 \end{pmatrix}$$

We now introduce our contribution involving integration with a batching mechanism, which is discussed in the next sub-section below. The trajectory adjustments are recorded in a `tracking array` for each UAV, ensuring that adjustments do not exceed a predefined limit, preventing infinite adjustment loops (as seen in standard approaches). If this limit is reached for a UAV, the trajectory of the other UAV in the colliding pair is examined. If the adjustment for the second UAV also exceeds this limit, both UAVs are moved to a `batching list` as separate entries.

4.2.2. Collision Avoidance by Start Time Change (Batching)

The batching technique, as outlined in Sastre et al. (2022a) and Sastre et al. (2022b), is adopted here. The goal of the batch generation algorithm is to form collision-free batches, which can be managed as a single unit. In the standard algorithm, input is the list of colliding pairs of UAVs. Here, the input is the `batching list` from the above method. This reduces the number of batches required to ensure collision free flight. The UAVs that are not on the `batching list` are free from any collision and outputted as one batch.

Next, the 1^{st} UAV from the batching list is picked for the next batch. This 1^{st} UAV is checked for a collision with all the subsequent UAVs in the list (2^{nd} UAV onwards). Without loss of generality, let us assume that the i^{th} UAV does not collide with the 1^{st} UAV, then 1^{st} and i^{th} UAVs are added to this next batch, and both are further checked for collision from $(i+1)^{th}$ UAV in the batching list. Without loss of generality, let us assume that now the j^{th} UAV does not collide with both the 1^{st} and i^{th} UAV. Then, j^{th} UAV is also added to the next batch. Further, all three are checked for collision from $(j+1)^{th}$ UAV in the batching list. This process is recursively done to finalize the next batch. This process is repeated to get the subsequent batches.

5. Results

Each UAV's output trajectory is generated with a total of 201 waypoints, providing a rich and detailed dataset conducive to robust algorithm development. This data is generated using a UAV simulator implemented with the UAV toolbox of Simulink in MATLAB. The simulation is run on a PC equipped with an Intel Core i7 12th generation processor, a 64-bit operating system, and 8 GB of RAM.

For the input data, the coordinates for initial, middle, and destination positions for each UAV are taken from Lai (2020), which provides guidance on realistic ranges. The X and Y coordinates for the initial position are in the range $[10 - 50]$, with a fixed Z-coordinates of 0. The X and Y coordinates for the destination position are in the range $[200 - 300]$, with a fixed Z-coordinates of 0 again. The X and Y coordinates for the middle position are taken as the average of the initial and destination position coordinates, with Z-coordinates in the range $[30 - 40]$. A pictorial representation of all the UAVs is shown in Figure 4.

The 500 UAV dataset is partitioned into three distinct segments: 70% is allocated for training, 15% for validation, and the remaining 15% for testing. The computational process iterates over 1000 epochs to optimize outcomes.

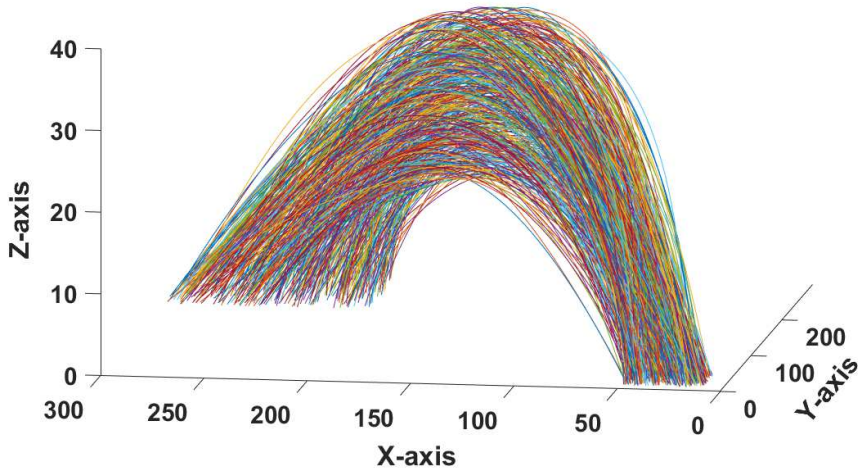


Figure 4: Dataset of 500 UAVs.

The choice of a loss function is crucial in neural networks since it measures the difference between the predicted and the actual values. For UAV trajectory prediction, Mean Squared Error (MSE) has been widely used in literature Xue (2017), Sarkar et al. (2020), Jiang et al. (2022b), and hence, we also adopt it here.

The rest of this section has two parts. In Section 5.1, we present MSE values when using standard, application-oriented, and our Novel AdaptoSwelliGauss activation functions. In Section 5.2, we give experimental results for our ICDATS techniques.

5.1. MSE Values Using Different Activations

For the Swish activation function, based upon experience, the value of hyperparameter β is taken as 0.14. For AdaptoSwelliGauss activation function, to ensure that Swish and Elliot-times-Gaussian are used an equal number of times, the value of hyperparameter α is taken as the median of the inputs to the activation function. Again, based upon experience, the value of other hyperparameters *scale* and $shift$ are taken as 0.5 and 0.25, respectively.

We calculate the average error across 201 waypoints, derived from 15% of 500 UAVs (i.e., test data consisting of 75 UAVs). The results for the optimal epoch for the X, Y, and Z-Coordinates are given in Tables 3, 4, and 5, respectively.

Table 3: Error for the X-Coordinates.

No.	**Category**	**Activation Function**	**MSE**
1-3	Standard	Sigmoid	9.200×10^{-5}
		Tanh	1.406×10^{-7}
		ReLU	1.733×10^{-10}
4-6	Application-Oriented	Swish	1.638×10^{-9}
		Elliot	3.605×10^{-11}
		AdaptoSwelliGauss	3.059×10^{-14}

The best performing standard activation function is ReLU, which gives error in the order of 10^{-10}, 10^{-8}, and 10^{-9} for X,Y, and Z-Coordinates, respectively. On the contrary, our novel AdaptoSwelliGauss activation function

Table 4: Error for the Y-Coordinates.

No.	Category	Activation Function	MSE
1-3	Standard	Sigmoid	1.822×10^{-8}
		Tanh	8.664×10^{-7}
		ReLU	1.793×10^{-8}
4-6	Application-Oriented	Swish	6.805×10^{-10}
		Elliot	3.780×10^{-9}
		AdaptoSwelliGauss	5.127×10^{-11}

Table 5: Error for the Z-Coordinates.

No.	Category	Activation Function	MSE
1-3	Standard	Sigmoid	4.588×10^{-11}
		Tanh	8.031×10^{-7}
		ReLU	5.851×10^{-9}
4-6	Application-Oriented	Swish	2.182×10^{-13}
		Elliot	4.851×10^{-8}
		AdaptoSwelliGauss	1.739×10^{-13}

gives error in the order of 10^{-14}, 10^{-11}, and 10^{-13} in X, Y, and Z-Coordinates, respectively. Thus, by using our new activation function, we see three to four orders-of-magnitude improvement in accuracy.

5.2. Results On ICDATS

For collision detection, based upon experience, the value of "R" is taken as 0.5. Based upon a previous work Park et al. (2008), the granularity of the time instances is taken as 1 second. Application of this results in 262 collisions. Next, when we apply the *original* collision avoidance with trajectory change method, the algorithm although does smooth changes it gets complicated and leads to an infinite loop. Thus, we apply our *proposed* collision avoidance with trajectory change method. Based upon experience, the settings used in this are as follows: `safe distance` is taken is 0.5, k used for smoothing is taken as 5, and the number of trajectory manipulations allowed is taken as 10. Now, our algorithm converges resulting in only 41 collisions. This highlights the superiority of our trajectory change method.

Apply batching brings the above mentioned collisions down to 0. This highlights the gain in trajectory change method, obtained because of integration with batching.

The results of batching when it is integrated with the trajectory change method are given in Table 6. As evident, the number of batches required is reduced from 7 to 5, while the maximum number of UAVs per batch increased from 38 to 315. This highlights the gain to batching, obtained because of integration with the trajectory change method.

Table 6: Result before and after integrating batching with trajectory change technique.

Metric	**Before Trajectory Change Integration**	**After Trajectory Change Integration**
Number of Batches	7	5
Maximum Number of UAVs per Batch	38	315

5.2.1. Sensitivity Analysis with respect to Safe Radius

Next, we show the importance of carefully calibrating the safe radius in our ICDATS algorithm. The number of collisions before trajectory change

integration remain 262. As shown in Table 7, increasing the safe radius initially results in a decrease in the number of colliding UAVs, decrease in number of batches, and increase in the maximum number of UAVs per batch, achieving optimal results at a safe radius of 2.3. At this radius, the number of colliding UAVs is reduced to 33, the number of batches is minimized to 4, and the maximum number of UAVs per batch increases to 396.

However, beyond this optimal point, further increases in the safe radius lead to a reverse trend, where both the number of colliding UAVs and the number of batches begin to increase again, while the maximum number of UAVs per batch decreases. These results are subject to the characteristics and dynamics of the specific dataset under consideration, and different datasets may yield different optimal safe radius values. However, there is a scientific reason for this observed trend.

Table 7: ICDATS Outcomes with Different Safe Radius Values.

Safe Radius	# of Collisions after Trajectory Change Integration	Number of Batches	Max Number of UAVs per Batch
0.5	47	5	315
0.7	46	6	321
1.9	39	7	326
2.1	35	4	378
2.3	33	4	396
2.5	42	6	335
2.7	61	8	305
2.9	65	10	278

This is because of the unintended consequences of over-adjusting UAV trajectories at a larger safe radius (see Figure 5). As in the given figure, when the trajectory of UAV1 is significantly altered to avoid UAV2, it may inadvertently collide with UAV3, which was previously not on a collision course with UAV1. This has a cascading effect on all the UAVs.

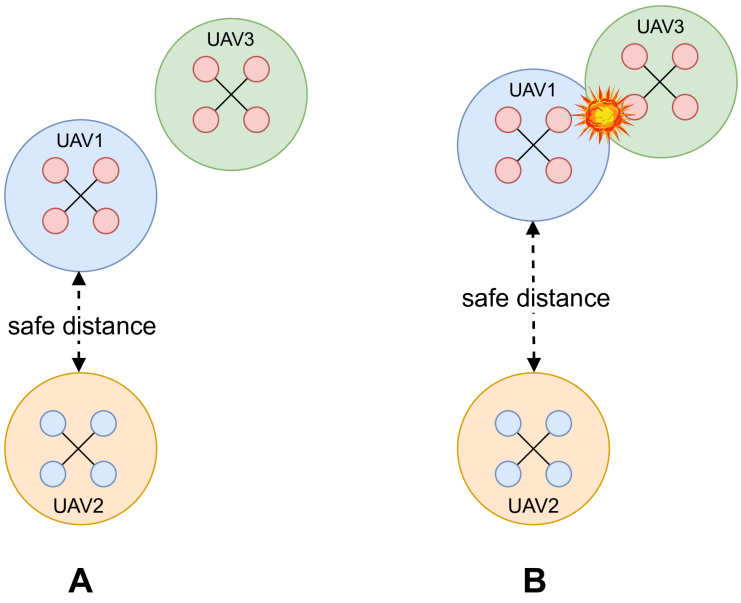
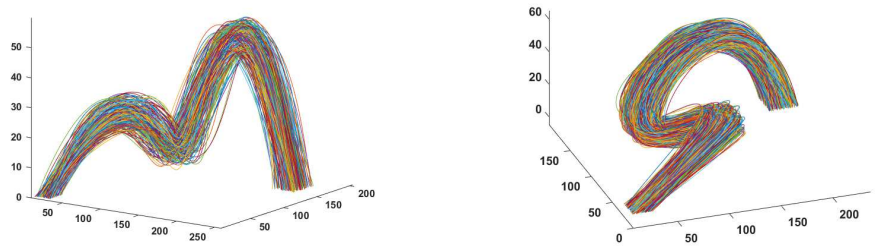


Figure 5: **A:** ICDATS with small safe radius, **B:** ICDATS with large safe radius.

5.2.2. Sensitivity Analysis with respect to Trajectories and Swarm Sizes

To demonstrate the scalability and effectiveness of the proposed ICDATS algorithm, we evaluate its performance on UAV datasets of varying trajectories and swarm sizes. In addition to the 500 UAVs with parabolic trajectories considered in the main experiments (Figure 4), we also apply the ICDATS algorithm to 250 UAVs and 750 UAVs with more nonlinear trajectories as depicted in Figure 6.

The results for this are presented in Table 8. As evident, use of our trajectory change algorithm for 250 UAVs and 750 UAVs leads to reduction in number of colliding UAVs by a factor of 6 to 3, respectively (eventually becoming 0 with its integration with batching). Again, for 250 UAVs and 750 UAVs, the number of required batches reduce from 7 to 4 and from 31 to 14, respectively, with maximum number of UAVs per batch increase from 56 to 180 and 55 to 426, respectively.



(a) 250 UAVs dataset (b) 750 UAVs dataset

Figure 6: Different datasets of UAVs

Table 8: ICDATS analysis with different datasets

Trajectory Change Integration	Number of UAVs	Number of Collisions	Batch Size	Maximum Number of UAVs Per Batch
Before	250	195	7	56
After		32	4	180
Before	750	695	31	55
After		274	14	426

6. Conclusion

In this paper, we focus on development of an intelligent framework for predicting and optimizing trajectories of large swarm of UAVs. Traditional methods have limitations in accurately predicting trajectories as well as efficiently avoiding collisions.

In the past, people have mostly used FFNNs with standard activation functions (Sigmoid, Tanh, ReLU) resulting in low trajectory prediction accuracy. We use application-oriented activation functions of Swish and Elliot, and also combine these with a Gaussian function, leading to our new activation function AdaptoSwelliGauss. This combination better captures the properties of trajectory path prediction (noisy data and non-linearity) leading to improvement in accuracy by three to four orders-of-magnitude.

Collision detection is straightforward while avoidance is challenging, which we improve. We can avoid collisions between UAVs by either changing their trajectories or sending them in batches. Both these methods have drawbacks. The first method has the disadvantage that when trying to achieve smooth trajectory changes, the algorithms get complicated (lead to infinite loops). The second method has the drawback of requiring large number of batches to send all UAVs. We propose a new approach to the first method, and we also uniquely combine the two methods, leading to smooth, simple, and finite trajectory changes in the first and reduction-by-half in the number of batches in the second.

Future work involves improving the underlying mathematical optimization Ahuja et al. (2008); automating model selection and hyperparameter tuning Hadi et al. (2023); using approximate computing in neural networks Gupta et al. (2020); Ullah et al. (2020); leveraging multi-agent reinforcement learning for collaborative decision-making Canese et al. (2021); exploiting implicit relation between different drone trajectories Kim et al. (2005); integrating our with-in UAV swarm collision avoidance algorithm with static & dynamic obstacles collision avoidance algorithm, with latter briefly discussed in Appendix B; and porting & testing our complete set of algorithms on real UAVs.

Acknowledgements
We would like to thank Mr. Patel Nilay Prakashbhai and Ms. Gavhale Pranjali Arun for discussions regarding the different aspects of this project. Thanks to the anonymous reviewers that helped to greatly improve the quality of this manuscript. We would also like to thank the editor handling our manuscript, Prof. Ivan Petrovic (Receiving Editor, Robotics and Autonomous Systems), in giving us the flexibility during revision submissions.

References

Ahuja, K., Watson, L.T., Billups, S.C., 2008. Probability-one homotopy maps for mixed complementarity problems. Computational Optimization and Applications 41, 363 – 375.

Al-Khazraji, H., Nasser, A.R., Hasan, A.M., Al Mhdawi, A.K., Al-Raweshidy, H., Humaidi, A.J., 2022. Aircraft engines remaining useful life prediction based on a hybrid model of autoencoder and deep belief network. IEEE Access 10, 82156–82163.

Apicella, A., Donnarumma, F., Isgrò, F., Prevete, R., 2021. A survey on modern trainable activation functions. Neural Networks 138, 14–32.

Canese, L., Cardarilli, G.C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., Spanò, S., 2021. Multi-agent reinforcement learning: A review of challenges and applications. Applied Sciences 11, 4948.

Contributor, A., 2024. Gaussian activation function. Codecademy URL: https://www.codecademy.com/resources/docs/ai/neural-networks/gaussian-activation

Dash, A.B., 2021. Top 10 activation function's advantages and disadvantages. Linkedin URL: https://www.linkedin.com/pulse/top-10-activation-functions-advantages-disadvanta

Dennis, C., Engelbrecht, A., Ombuki-Berman, B., 2020. An analysis of activation function saturation in particle swarm optimization trained neural networks. Neural Processing Letters 52.

Ding, B., Qian, H., Zhou, J., 2018. Activation functions and their characteristics in deep neural networks, in: 2018 Chinese Control And Decision Conference (CCDC), pp. 1836–1841.

Elmkaiel, G., Serebrenny, V.V., 2019. Collision avoidance algorithm for a quadcopters swarm. AIP Conference Proceedings 2171, 190006.

Gupta, S., Ullah, S., Ahuja, K., Tiwari, A., Kumar, A., 2020. ALigN: A highly accurate adaptive layerwise Log_2_Lead quantization of pretrained neural networks. IEEE Access 8, 118899.

Hadi, R.H., Hady, H.N., Hasan, A.M., Al-Jodah, A., Humaidi, A.J., 2023. Improved fault classification for predictive maintenance in industrial IoT based on AutoML: A case study of ball-bearing faults. Processes 11.

Hagan, M., Menhaj, M., 1994. Training feedforward networks with the marquardt algorithm. IEEE Transactions on Neural Networks 5, 989–993.

Hasan, A.F., Humaidi, A.J., Al-Obaidi, A.S.M., Azar, A.T., Ibraheem, I.K., Al-Dujaili, A.Q., Al-Mhdawi, A.K., Abdulmajeed, F.A., 2023. Fractional order extended state observer enhances the performance of controlled tricopter UAV based on active disturbance rejection control, in: Azar, A.T., Kasim Ibraheem, I., Jaleel Humaidi, A. (Eds.), Mobile Robot: Motion Control and Path Planning. Springer International Publishing, Cham, pp. 439–487.

Ho, F., Geraldes, R., Gonçalves, A., Cavazza, M., Prendinger, H., 2019. Improved conflict detection and resolution for service UAVs in shared airspace. IEEE Transactions on Vehicular Technology 68, 1231–1242.

Huang, S., Zhang, H., Huang, Z., 2024. E^2CoPre: Energy efficient and cooperative collision avoidance for UAV swarms with trajectory prediction. IEEE Transactions on Intelligent Transportation Systems 25, 6951–6963.

Jeong, S., You, K., Seok, D., 2021. Hazardous flight region prediction for a small UAV operated in an urban area using a deep neural network. Aerospace Science and Technology 118, 107060.

Jiang, B., Li, B., Zhou, W., Lo, L.Y., Chen, C.K., Wen, C.Y., 2022a. Neural network based model predictive control for a quadrotor UAV. Aerospace 9, 460.

Jiang, B., Li, B., Zhou, W., Lo, L.Y., Chen, C.K., Wen, C.Y., 2022b. Neural network based model predictive control for a quadrotor UAV. Aerospace 9.

Kim, S., Murthy, U., Ahuja, K., Vasile, S., Fox, E.A., 2005. Effectiveness of implicit rating data on characterizing users in complex information systems, in: Rauber, A., Christodoulakis, S., Tjoa, A.M.e. (Eds.), Research and Advanced Technology for Digital Libraries (ECDL 2005), Lecture Notes in Computer Science. Springer. volume 3652, pp. 186 – 194.

Lai, R., 2020. A machine learning approach to trajectory planning for UAV. M.S. Thesis, Rensselaer Polytechnic Institute.

Laudani, A., Lozito, G.M., Riganti Fulginei, F., Salvini, A., 2015. On training efficiency and computational costs of a feed forward neural network: A review. Computational intelligence and neuroscience 2015, 818243.

Nikhade, A., 2023. Swish activation function. Medium URL: https://amitnikhade.medium.com/swish-activation-function-d106fe13930e.

Orr, G., Müller, K., 2003. Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, Springer Berlin Heidelberg.

Park, J.W., Oh, H.D., Tahk, M.J., 2008. UAV collision avoidance based on geometric approach, in: 2008 SICE Annual Conference, pp. 2122–2126.

Qiu, H., Duan, H., 2020. A multi-objective pigeon-inspired optimization approach to UAV distributed flocking among obstacles. Information Sciences 509, 515–529.

Ramachandran, P., Zoph, B., Le, Q.V., 2017. Searching for activation functions. arXiv:1710.05941 .

Reda, M., Onsy, A., Haikal, A.Y., Ghanbari, A., 2024. Path planning algorithms in the autonomous driving system: A comprehensive review. Robotics and Autonomous Systems 174, 104630.

Salmeron, J.L., Ruiz-Celma, A., 2019. Elliot and symmetric elliot extreme learning machines for gaussian noisy industrial thermal modelling. Energies 12.

Sarkar, S., Totaro, M.W., Kumar, A., 2020. An intelligent framework for prediction of a UAV's flight time, in: 2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS), IEEE. pp. 328–332.

Sastre, C., Wubben, J., Calafate, C.T., Cano, J.C., Manzoni, P., 2022a. Collision-free swarm take-off based on trajectory analysis and UAV grouping, in: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), IEEE. pp. 477–482.

Sastre, C., Wubben, J., Calafate, C.T., Cano, J.C., Manzoni, P., 2022b. Safe and efficient take-off of VTOL UAV swarms. Electronics 11, 1128.

Shi, Z., Zhang, J., Shi, G., Ji, L., Wang, D., Wu, Y., 2024. Design of a UAV trajectory prediction system based on multi-flight modes. Drones 8.

Singh, S.K., Sinha, A., Kumar, S.R., 2022. Nonlinear control design for an unmanned aerial vehicle for path following. IFAC-PapersOnLine 55, 592–597. 7th International Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2022.

Tang, J., Fan, L., Lao, S., 2014. Collision avoidance for multi-UAV based on geometric optimization model in 3D airspace. Arabian Journal for Science and Engineering 39, 8409–8416.

Ullah, S., Gupta, S., Ahuja, K., Tiwari, A., Kumar, A., 2020. L2L: A highly accurate Log_2_Lead quantization of pretrained neural networks, in: 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE. pp. 979–982.

Wan, Y., Tang, J., Lao, S., 2019. Distributed conflict-detection and resolution algorithm for UAV swarms based on consensus algorithm and strategy coordination. IEEE Access 7, 100552–100566.

Xiao, K., Zhao, J., He, Y., Yu, S., 2019. Trajectory prediction of UAV in smart city using recurrent neural networks, in: ICC 2019-2019 IEEE International Conference on Communications, IEEE. pp. 1–6.

Xu, X., Xie, C., Luo, Z., Zhang, C., Zhang, T., 2024. A multi-objective evolutionary algorithm based on dimension exploration and discrepancy evolution for UAV path planning problem. Information Sciences 657, 119977.

Xue, M., 2017. UAV trajectory modeling using neural networks, in: 17th AIAA Aviation Technology, Integration, and Operations Conference, ARC. p. 3072.

Appendix A. Comparison of a FFNN with Advanced NNs

Here, we compare the performance our single hidden layer based FFNN with three other advanced NNs. That is, Radial Basis Function Neural Network (RBFNN), Liquid Neural Network (LNN), and 1D Convolutional Neural Network (1D CNN). We use 15 hidden neurons in all of them. RBFNN have their own activation functions while for FFNN, LNN and 1D CNN, without loss of generality, we use Tanh. These results are given in Table A.9. It is evident from this table that FFNN works the best.

Table A.9: Trajectory Prediction Errors using FFNN and Advanced NNs.

FFNN	**RBFNN**	**LNN**	**1D CNN**
10^{-7}	10^{-1} to 10^{-2}	10^{-4} to 10^{-6}	10^{-4} to 10^{-5}

Appendix B. Avoiding Static and Dynamic Obstacles

For avoiding static and dynamic obstacles, the VFH+ (Vector Field Histogram Plus) algorithm along with the LiDar sensor is a standard component. We use this component in our simulation as well.

The UAV starts at $[0, 0, 0]$ and aims to reach $[45, 43, 0]$. Two static obstacles are positioned at coordinates $[20, 15, 0]$ and $[30, 20, 0]$, along with a moving obstacle starting from $[11.2, 16.6, 0]$ and moving toward the UAV. The obstacles have a height of 15 units and dimensions of 4 units in length and width.

Figure B.7 shows the top view of our simulation. The green line illustrates the UAV's original trajectory that passes through all the obstacles, i.e., without use of VFH+ and LiDAR component. The red line illustrates the UAV's updated trajectory that avoids all the obstacles, i.e., with the use of this component.

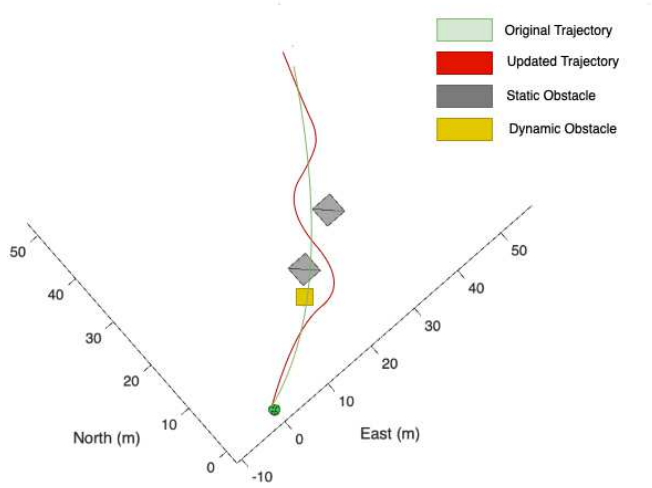


Figure B.7: Top View of UAV Trajectory.