Mateusz Przychodzki[1]
ORCID: 0009-0000-2668-4875

Artur Niewiadomski[2]
ORCID: 0000-0002-9652-5092

University of Siedlce
Faculty of Exact and Natural Sciences
Institute of Computer Science
ul. 3 Maja 54, 08-110 Siedlce, Poland

{[1]mateusz.przychodzki,[2]artur.niewiadomski}@uws.edu.pl

Application of artificial neural networks and NEAT algorithm to control a quadcopter

DOI: 10.34739/si.2023.29.03

Abstract. We present a framework for building Artificial Neural Networks (ANNs) able to control a quadcopter and perform basic maneuvers, like hover or following waypoints. In this approach, we make use of the Neuroevolution of Augmenting Topologies (NEAT) algorithm which is aimed at creating the network structure and the weights in result of evolutionary computations. In order to evaluate fitness of individuals, we use physics based, realistic simulation engine Gazebo, where each individual controls a drone in a simulated environment. Our approach is aimed at using one of existing, popular protocols used to remotely control drones, and train ANNs able to imitate signals received from a radio controller operated by a human pilot. Thus, contrary to the most of other approaches, our autonomous controller cooperates with standard drone software. Our ultimate goal is to train ANNs able to control a real-world quadcopter and perform advanced tasks autonomously. Not only such ANNs should be able to perform the maneuvers correctly, but they should be small enough to transfer them into

a quadcopter's limited memory. In this paper we report the first stage of our project - a successful development and deployment of the ANNs distributed training framework, and choosing the activation function for further research.

Keywords: Artificial Neural Networks, Unmanned Aerial Vehicles, NEAT Algorithm, Unity, Gazebo

1 Introduction

In the modern world, unmanned aerial vehicles (UAVs) take various forms, from models of monoplane aircraft to helicopters and multi-rotor aircraft. The growing popularity of UAVs in the consumer and commercial markets calls for new technologies, control systems, and autonomous flight features. Applications of multi-rotor drones, besides hobby users and cinematography, are, for example, search and rescue operations [11], construction [10], and agriculture [9]. When performing these kinds of missions, functions like avoiding obstacles, object tracking, or hovering are used, allowing the operator to focus on a given task without the effort of maintaining a multi-rotor UAV in the air. Autonomous behaviors of drones are implemented algorithmically or using artificial intelligence methods, like Artificial Neural Networks (ANNs) [8].

In this work, we focus on training ANNs able to control a quadcopter and perform one of the basic autonomous maneuvers - hover. To this aim, we exploit a neuroevolutionary algorithm called Neuroevolution of Augmenting Topologies (NEAT) [21] and Gazebo [37] - a robust and accurate physics-based drone simulation environment. Our idea is to use input data for ANN solely from onboard sensors and train it to generate output imitating radio signals received by a drone controlled by a human pilot.

The main contribution of this paper is the design of a framework that allows simultaneous training of many simulated quadcopters using containerization technology [12].
We also report on validation and testing of our implementation of the framework. To this aim, we performed multiple experiments designed to test the correctness of the implemented system and reveal the most suitable activation function for ANN. Our solution enables real-time monitoring and visualization of the training progress using a three-dimensional graphical environment Unity 3D [43]. Alternatively, the tool allows running the training process without visualization, enabling the use of this software in server or cloud environments. Moreover, the extensive parametrization possibilities and exploitation of a popular MultiWii Serial Protocol [20] used widely to control drones, allow for modeling a broad spectrum of quadcopters and training well-suited, tailor-made ANNs, which can then be transferred directly to physical quadcopters.

The rest of the paper is structured as follows. First, we introduce basic facts about quadcopters' structure, flight mechanics, and control. Then, we recall fundamental concepts behind ANNs and the NEAT algorithm, and we compare other approaches with ours. The next section introduces our approach and gives some insight into implementation of the framework components. Finally, we present the preliminary experimental results, followed by conclusion and future work.

1.1 Quadcopter flight mechanics and fundamental structure

Nowadays, UAVs have become increasingly complex and advanced machines with a multitude of components and subsystems responsible for particular functions. However, there is a minimum specification for flight that serves as the foundation for all UAVs.
This minimum specification includes a Frame, a Flight Controller (FC), an Electronic Speed Control (ESC), Brushless motors, Propellers, a Battery, and a Transceiver [15].

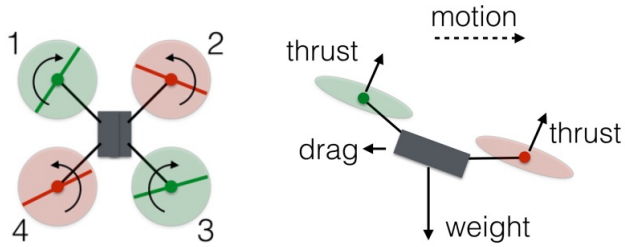

Figure 1. Typical drone motor orientation and resultant thrust forces. Source:[46]

The Flight Controller is a mainboard responsible for housing the drone's CPU, memory, flash storage, and various sensors, such as the Inertial Measurement Unit (IMU) [17, 16]. The IMU reads triaxial linear acceleration and triaxial gyroscope values, which are used alongside pilot directives received from the operator's radio, to generate appropriate motor speeds for each engine. The process of generating motor speeds is based on a proportional–integral–derivative controller (PID) loop [4, 5]. Based on the pilot directives and IMU readouts, the PID loop generates signals, which are then converted by ESC to appropriate values of three-phase alternating current to control each motor individually.

For a drone to hover in the air, the thrust generated by motors must equal the gravitational pull experienced by the drone body [19]. However, if all the motors were rotating in the same direction, the total angular momentum generated by the motors would be uncontrollably rotating the drone body. To counteract this force, quadcopters typically pair their engines into two groups rotating clockwise and counterclockwise (see Fig. 1).

Typically, pilot directives are collected by a radio receiver housed on FC board. In our work, we used an artificial neural network to generate those directives, and we exploited the MultiWii Serial Protocol (MSP) [20] for communication. MSP is a binary message-based protocol for sending information, such as control channels, telemetry, and sensor data. MSP provides fast and reliable communication between drone components, enabling UAV control via a simulated radio transmitter. We send generated raw RC channel values representing throttle, yaw, pitch, and roll axes, and AUX1 channel, allowing arming of the simulated drone. This approach is quite original for simulated drone control, and enables cooperation with a wide range of embedded FC software.

1.2 Neuroevolution of Augmenting Topologies

Neuroevolution of Augmenting Topologies (NEAT) is a powerful approach in the field of evolutionary computation and artificial intelligence. Developed by Kenneth O. Stanley and

Risto Miikkulainen in 2002 [21], NEAT is designed to evolve Artificial Neural Networks with focus on complex and diverse topologies. Unlike traditional methods of ANNs training that focus on optimizing weights of fixed architectures, NEAT usually starts with a population of small, simple, and random selection of networks and evolves them over generations in a process known as environmental evaluation by introducing new neurons, connections, and weight mutations.

NEAT uses many concepts from genetic algorithms [22] to evolve the neural networks. Each ANN in the population is represented as a genome that encodes the structure of the network. At the end of each generation NEAT employs genetic operators, such as mutation and crossover, to create new offspring with different topologies or connection weights from the parent networks. The algorithm assigns a fitness score to each network based on its performance in the given task. The fittest networks (presumably the best adapted to achieving a given task) are more likely to pass their genes to the next generation, encouraging the evolution of more effective architectures. By starting with simple structures and gradually augmenting them over generations, NEAT can explore a diverse range of architectures and adapt to the problem's complexity. Additionally, NEAT promotes the preservation of innovation (Speciation) throughout the evolutionary process, ensuring that promising features and connections are not prematurely discarded [21].

2 Related Work

In this section, we compare our approach to others but, to our best knowledge, only a few approaches come vaguely close to ours. Many publications indicate the successful use of NEAT algorithms in training ANN in various control tasks [33]. Typically, NEAT is used to create and optimize ANN in a virtual environment but, like in [34, 35], UAV flight mechanics is oversimplified. Usually, autonomous UAVs operating as standalone platforms in GPS-less environments rely on vision, combining optical flow sensors and visual-inertial odometry (VIO) with sonar and LIDAR for distance measurement [27, 23, 28]. There are very few attempts to control UAVs by artificial neural networks solely using inertial measurement unit reading.

The approach [29], which compares traditional, widely used PID controllers with custom neural network models, seems to be similar to ours. However, the ultimate goal of this approach was to completely replace the PID controller with an ANN-based one, which receives the IMU sensor data as input and generates a direct PWM signal representing the desired motor speeds. The paper reports promising results in generating accurate motor speed commands with lower oscillation (instability) compared to the PID controller. However, this solution could not be deployed to a physical platform due to flash memory size constraints. In our approach, a trained neural network mimics a pilot with the radio controller, generating the appropriate signals. Thus, our solution works with the PID built into the FC, allowing it to generate signals for the motors, and so far, the resulting models are small enough to fit into the onboard memory.

Numerous studies have also focused on simulating quadcopter flight mechanics, using tools like Matlab [6], Microsoft AirSim [7], and Gazebo [30–32] to enhance the development and testing of autonomous drone systems. The authors of [13] exploited the MatLab Simulink environment to develop a real-time Hardware-In-the-Loop flight simulation system that allows control systems design, testing, and UAV model validation.

Furthermore, Microsoft AirSim has emerged as a powerful platform for simulating quadcopter flight in realistic environments. The work [14] presents a simulation platform designed for efficient algorithm prototyping and machine learning research in the complex field of autonomous drone racing. In our solution, we used Gazebo as our simulation platform. Works [30–32] presents promising approaches in accurate UAV simulation. However, none of them are capable of integration with genuine flight control software.

3 The Proposed Method Overview

In this work, we have developed a system capable of training Artificial Neural Networks for the task of piloting realistically simulated drones. Our approach uses the NEAT algorithm to train ANNs and monitor their progress in a 3D graphical environment. Our resulting framework allows users to control the training process and to create various experimental scenarios. These scenarios enable the observation of UAV behavior and the exploration of how different network topologies and activation functions impact the training process.
To facilitate this, we have adapted the physics-based drone simulation environment, Gazebo [37], which can accommodate many drone configurations.

Additionally, we have integrated Betaflight SITL [36] as the Flight Controller software, although any other controller supporting MultiWii Serial Protocol could also be use. To ensure real-time visualization and communication between simulated UAVs and the evaluation environment, we use Unity 3D [43]. To address the need for parallel training and evaluation of multiple simulated quadcopters, we have leveraged Docker containerization [12] of our Gazebo simulation instances.

3.1 Physics-based Drone Simulation using Gazebo

Gazebo [37] is a widely used open-source simulation environment providing a multi-purpose platform for developing and testing robots and autonomous systems. It provides a realistic 3D simulation environment, allowing researchers and developers to accurately model and visualize complex scenarios. A key feature that makes Gazebo powerful is its integration with the Open Dynamics Engine (ODE) [38] physics engine. ODE is responsible for providing robust and efficient physics simulation, enabling realistic interactions between objects and their environment within the simulation. The combination of Gazebo and ODE provides a comprehensive toolset for simulating a wide range of robotic tasks, from simple mechanical motions to complex multi-agent systems.

In our solution to accurately model and simulate UAVs, we use Gazebo (see. 2) as the simulation platform. Gazebo allows the creation of sophisticated simulation environments using its SDF world file format [39]. In our approach integrate and adapt other software that plays an essential role in the task of quadcopter simulation. One of these is Betaflight SITL [36] - a full-featured flight controller software responsible for generating motor speed commands via a PID loop.

Other key components of our framework are modified elements from the Vehicle Gateway project [40], which also incorporates some parts of the ArduPilot Gazebo Plugin [41]. The main module, called Gazebo Communication Plugin (see Fig. 2), acts as a bridge between

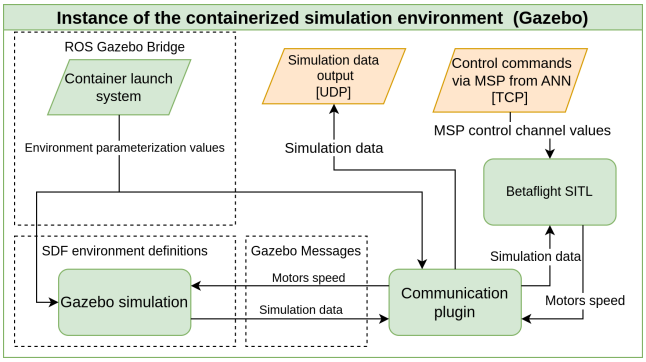

Figure 2. Simulation architecture and data flow diagram. Source: original work.

Gazebo simulation, Betaflight Software-In-The-Loop (SITL) [36], and the NEAT-based training application, which delivers RC radio commands generated by ANN based on simulated drone state computed by Gazebo.

3.2 Using an Artificial Neural Network to Control a Drone

Our solution utilises an artificial neural network to generate raw RC channel values for the throttle, yaw, pitch, and roll axes (MSP Data). These RC values are then transmitted via an MSP communication link directly to the Gazebo drone simulation (refer to Fig 3, left), enabling the flight controller software to perform calculations. The simulation engine computes a new state of the drone based on the updated motor speed values (EPWM).

In Our solution, the input layer of ANN is fed with raw, unfiltered data directly from a simulated drone's onboard IMU sensor. This sensor generates triaxial linear acceleration and triaxial gyroscope values (IMU Data) in the UAV body frame. Thus, the input layer consists of six neurons, one for each axis of these IMU sensors.

The output layer consists of four neurons representing the RC channel values of throttle, yaw, pitch, and roll axis. The output values are mapped from the range $[0, 1]$ to $[900, 1900]$, in accordance with the MSP protocol regarding the raw values of the RC control channels. This method of presenting control signals to FC software ensures compatibility with any other MSP-compatible flight controller. It is important to note that hidden layers are not specified, as they will evolve alongside new connections and weight values during the neuroevolution process.

3.3 Implementation of the Framework

We present a configurable UAV simulation environment that allows the modeling of many different drones using various FC software and the parallel evaluation of Agents. To this aim, we use the Unity 3D game engine [43] to create a 3D application hosting Active Containers Manager (ACM), Agent Evaluation Environment (AEE), and the NEAT algorithm implemented using UnitySharpNEAT library [44, 45] (see Fig 3 right, Fig. 4). ACM is responsible for managing the pool of simulation containers [12, 1] handling isolated instances of Gazebo

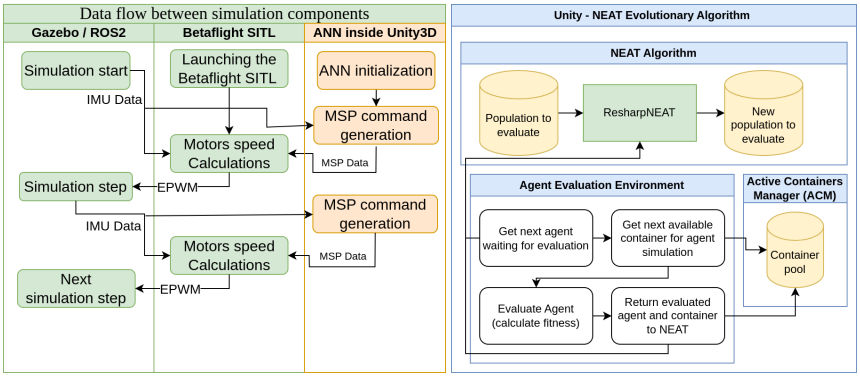

Figure 3. Left: ANN in the drone flight control process, right: architecture of the training application. Original work

simulation. The SharpNEAT library is responsible for generating ANNs by following the steps of the NEAT algorithm. AEE subsystem is responsible for evaluating agent performance in a simulated environment. Together with the ACM, they perform agent evaluation and simulation container management. They also allow users to configure various available options and settings[1]. Some of them are covered in the next subsection.

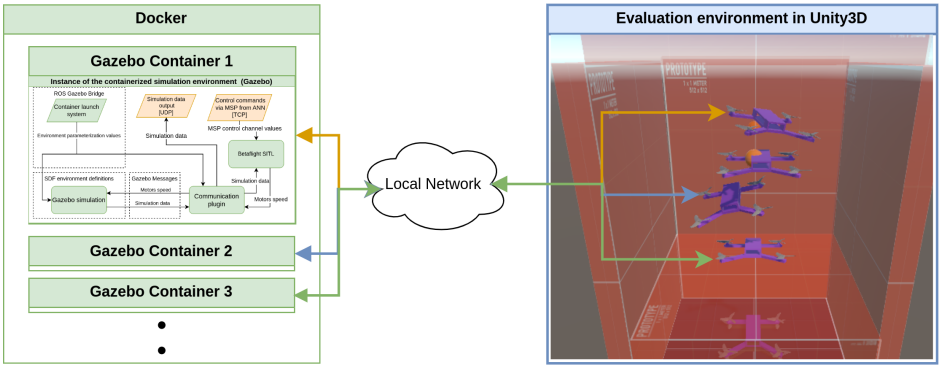

Figure 4. Unity training application during population evaluation. Original work

3.4 Fitness Function

As NEAT is a genetic algorithm variant, the fitness function is crucial for its proper operation. The fitness value indicates how well the agent is adapted to a given task. Our goal is to create and train an artificial neural network (ANN) capable of remaining airborne and performing the hover maneuver. To achieve this goal, we consider several factors when calculating the

[1] A detailed description of all available parameters is beyond the scope of this paper. For more details see [1].

fitness value, including the total time spent in the air, the distance from the designated point, the ability to remain close to the target, and the avoidance of undesirable behavior such as sharp and sudden maneuvers, flipping, or flying away.

Table 1. Variables and constraints used to calculate the fitness values. Symbol '∼' denotes values calculated during computation. Original work

Parameter Name	Description	Value
TA_v	Total time the agent is alive	∼
$AvgD_v$	Average distance to target	∼
$MinD_v$	Minimal distance to target	∼
TH_v	Total time hovering near target	∼
$MaxTgD_c$	Maximum allowed distance to target	4 [m]
$TminH_c$	Minimal time required in hover near target	1 [s]
$MaxTA_c$	Expected time of an agent in the air	45 [s]
$AlEx_e$	Counter increased when required alignment angle with forward axis is exceeded	∼
$GdCol_e$	Counter increased when agent crashed to the ground	∼
$DeEx_e$	Counter increased when agent exceed maximum allowed deflection angle	∼
$BBCol_e$	Counter increased when agent collided with simulation bounding box	∼

To calculate the fitness value, several variables and constants are introduced. Table 1 presents values collected during simulations (denoted by the index $_v$), constraints expressing our expectations towards agents (marked with $_c$), and event counters (index $_e$). Table 2 presents the reward and penalty values, denoted by $_r$ and $_p$, respectively. The parameter values used and the final fitness function presented below were developed through preliminary experiments and observations.

Table 2. Reward and penalty values used to calculate agents' fitness. Original work

Parameter name	Description	Value
TA_r	Time alive multiplier, the reward for each second of flight	15
$AvgD_r$	Maximal reward for average distance	200
$MinD_r$	Maximal reward for minimal distance	50
TH_r	Maximal reward for total time in hover	300
$MinH_r$	Minimal reward for total time in hover	30
SV_r	Reward for spending an expected time in the air	200
$DeOk_r$	Reward for not exceeding maximum allowed angles	35
$NoGdCol_r$	Reward for not crashing to the ground (but collided with the boundary box)	10
$NoCols_r$	Reward for avoiding all collision types	60
$AlEx_p$	Penalty for exceeding required alignment angle with forward axis	-120
$DeEx_p$	Penalty for exceeding maximum allowed angles	-100
$GdCol_p$	Penalty for collision with the ground	-80
$BBCol_p$	Penalty for collision with the simulation boundary	-130

The fitness function that evaluates whether the drone is hovering steadily near the target is the sum of 11 values. Each value rewards the agent for correct behavior, increasing the fitness value, or punishes the agent for performing wrong maneuvers, decreasing the fitness value. The first value $\boldsymbol{Ft_1}$ (see Eq. 1) maximizes the total lifespan of agents. This is crucial to promote

individuals with longer lifetimes, particularly at the beginning of a new population when most agents quickly crash. The values $\boldsymbol{Ft_2}$ and $\boldsymbol{Ft_3}$ work in similar way rewarding shorter average distance and minimal distance to the target achieved during a flight, respectively.

$$\boldsymbol{Ft_1} = TA_v \cdot TA_r;\ \boldsymbol{Ft_2} = AvgD_r \cdot \left(1 - \frac{AvgD_v}{MaxTgD_c}\right);\ \boldsymbol{Ft_3} = MinD_r \cdot \left(1 - \frac{MinD_v}{MaxTgD_c}\right) \quad (1)$$

The value $\boldsymbol{Ft_4}$ awards more points to agents that remain in close proximity to the target. This encourages agents to stay close to the target without unnecessary movement. The ultimate goal of all agents is to stay alive for the entire duration of the $MaxTA_c$ evaluation and to hover as close to the target as possible. The $TminH_c$ condition also allows us to discard agents that fly through the target position and crash immediately, resulting in a higher score because their total lifespan was short. We can discourage fly-through behaviour by requiring agents to stay very close to the target for a minimum amount of time.

$$\boldsymbol{Ft_4} = \begin{cases} \left(\frac{TH_v}{TA_v} \cdot 100\right) \cdot TH_r, & \text{if } TH_v \geq TminH_c \\ MinH_r & \text{otherwise} \end{cases} \quad \boldsymbol{Ft_5} = \begin{cases} SV_r, & \text{if } TA_v \geq MaxTA_c \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Equations 2 and 3 present expressions that conditionally award or punish agents for registered positive and negative events. $\boldsymbol{Ft_5}$ awards agents for long time spent in the air, while $\boldsymbol{Ft_6}$ and $\boldsymbol{Ft_7}$ give penalties when agents try to flip or fly aside suddenly.
Those behaviors are undesirable because a sudden flip or side maneuver will result in a spinning, fly away, or crash scenario.

$$\boldsymbol{Ft_6} = \begin{cases} AlEx_p, & \text{if } AlEx_e \geq 1 \\ 0, & \text{otherwise} \end{cases} \quad \boldsymbol{Ft_7} = \begin{cases} DeEx_p, & \text{if } DeEx_e \geq 1 \\ DeOk_r, & \text{otherwise} \end{cases} \quad (3)$$

The next two components (Eq. 4) are focused on heavily penalizing crashing to the ground or colliding with the simulation boundary. Next, $\boldsymbol{Fit_{10}}$ and $\boldsymbol{Ft_{11}}$ are special condition rewards promoting collision avoidance.

$$\boldsymbol{Ft_8} = \begin{cases} GdCol_p, & \text{if } GdCol_e \geq 1 \\ 0, & \text{otherwise} \end{cases} \quad \boldsymbol{Ft_9} = \begin{cases} BBCol_p, & \text{if } BBCol_e \geq 1 \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

$$\boldsymbol{Ft_{10}} = \begin{cases} NoCols_r, \text{ if } \begin{cases} BBCol_e \leq 0 \\ \wedge\ GdCol_e \leq 0 \end{cases} \\ 0, \text{otherwise} \end{cases} \quad \boldsymbol{Ft_{11}} = \begin{cases} NoGdCol_r, \text{ if } \begin{cases} GdCol_e \leq 0 \\ \wedge BBCol_e \geq 1 \end{cases} \\ 0, \text{otherwise} \end{cases} \quad (5)$$

Finally, the fitness value is calculated as the sum:

$$\boldsymbol{Fitness} = \sum_{i=1}^{11} \boldsymbol{Ft_i} \quad (6)$$

4 Preliminary Experimental Results

This section reports the preliminary results of three series of experiments that demonstrate the impact of three different activation functions on the training efficiency of ANNs: Steepened Sigmoid, Plain Sigmoid, and Gaussian. As NEAT, like any other genetic algorithm, is non-deterministic, the results presented are the average values of several runs for every activation function, as obtaining the same result is only possible once. Subsequent experiments were conducted in the fixed environment[2], and their results are shown below. The collected statistics were analyzed, including the average and best fitness values in subsequent generations. and the number of quadcopters that were able to stay in the air.

4.1 Experiment 1: Steepened Sigmoid

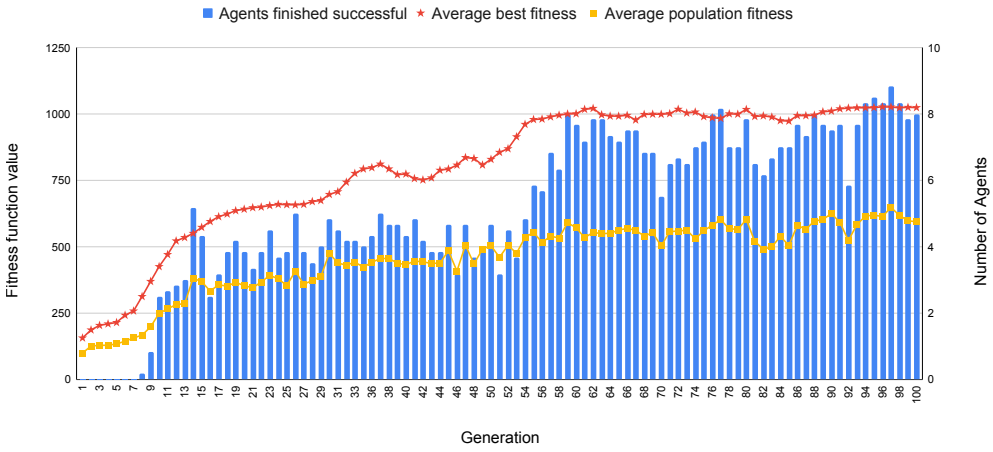

Figure 5. Exp. 1 - Basic statistics of hover training using steepened sigmoid activation function. Original work

The first experiment utilized a steepened sigmoid activation function (defined in Eq. 7) with effective input and output range of $[-1, 1]$ and $[0, 1]$, respectively. The experiment consisted of five runs, each with 100 generations. The population size was set to 40, and overall 20000 individuals were evaluated during the experiment. The total evaluation time of one run was 10 hours and 32 minutes on average.

$$f(x) = \frac{1}{1 + e^{(-4.9x)}} \tag{7}$$

Fig. 5 presents the average statistics collected during the experiment. The x-axis indicates consecutive generations. The fitness function values are displayed on the left Y-axis. The red

[2]The experiments were performed on a server equipped with two Intel Xeon Gold 6234 @ 3.30GHz CPUs and 64GB of RAM running Rocky Linux 8.8.

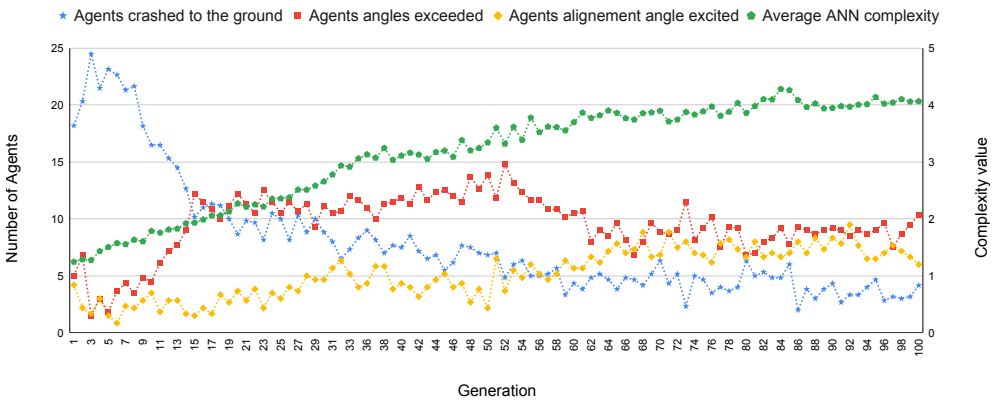

Figure 6. Average event count through generations - Steepened Sigmoid activation function, Original work

line (with stars) shows the average fitness function value of the best individual computed over all runs in subsequent generations, while the yellow line (with squares) represents the average fitness function value for the entire population. The blue bars represent the average number of agents that successfully completed the task by hovering steadily and long enough in the target area. The number of individuals is indicated on the right Y-axis. Upon analyzing Fig. 5, it is evident that Experiment 1 was successful. The methodical increase of all mentioned indicators leads to the conclusion that selecting the steepened sigmoid as activation function allows to train agents able to hover steadily.

These conclusions are supported by further observations summarized in Fig. 6.
It presents three statistics related to the average number of agents that crashed to the ground (the blue line with stars), exceeded the maximum allowed deflection angle (the red line with squares), and alignment angle (the yellow line with diamonds) of the drone. The values are marked on the left Y-axis. The line in green with pentagons represents the average complexity[3] of the ANN individual, and the correponding values are marked on the right Y-axis. It is evident that as the complexity of ANN increases, the number of drones that behave incorrectly decreases. At the start of a run, when brand new ANNs are randomly created without a proper structure to handle the input signals, most agents fall to the ground. However, as the complexity of the ANN increases in subsequent generations, agents are able to respond to IMU values and gradually generate proper RC channel values. This process is illustrated in Fig.5 by the crossing of the blue and red lines (marked with stars and squares, respectively) starting in generation 13.

Experiment 1 demonstrates that the steepened sigmoid activation function is a suitable choice for this type of training. Additionally, we have confirmed that the implemented framework, which incorporates the NEAT algorithm and simulation environment, is functioning properly. The NEAT algorithm rapidly evolved solutions capable of hovering near the target for the required duration. Based on this observation, the authors concluded that running exper-

[3]The function assessing complexity of ANN has been defined originally in [21].

iments beyond 60 generations is unnecessary, as only minor improvements in agent behavior occurred after that point.

4.2 Plain Sigmoid

$$f(x) = \frac{1}{1 + e^{(-x)}} \tag{8}$$

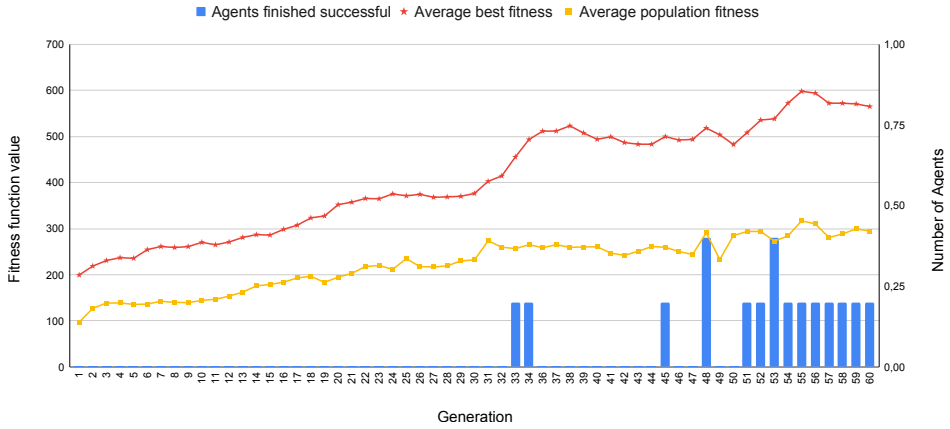

Figure 7. Experiment 2 - basic statistics of hover training using plain sigmoid activation function, Original work

In the second experiment, we utilized the plain sigmoid activation function defined in Eq. 8 with effective input and output range of $[-5, 5]$ and $[0, 1]$, respectively.
The experiment included 4 runs, each with 60 generations. The population size was set to 40, and a total of 9600 individuals were evaluated during the experiment. On average, one run took 4 hours and 52 minutes to complete.

Fig. 7 presents average statistics collected during the experiment, similar to the previous case. From the analysis, it is clear that the simple sigmoid activation function performed significantly worse than the steepened sigmoid activation function. Fewer agents were able to complete the goal, and the best and average fitness values were much lower. For instance, the maximal fitness value for a plain sigmoid was approximately 600, whereas for a steepened sigmoid, it exceeded 1000.

Fig. 8 shows that over time, an increasing number of agents were able to create the necessary structure to analyze IMU input signals and fly, as evidenced by the systematic decrease in crashes (the blue line). However, the agents tended to perform rapid maneuvers that exceeded the allowed angles, resulting in a lower total score.

4.3 Gaussian

$$f(x) = e^{-(2.5x)^2} \tag{9}$$

The third experiment utilized the Gaussian activation function, as defined in Eq. 9, with the effective input range of $[-1, 1]$ and the output range of $[0, 1]$. The experiment consisted

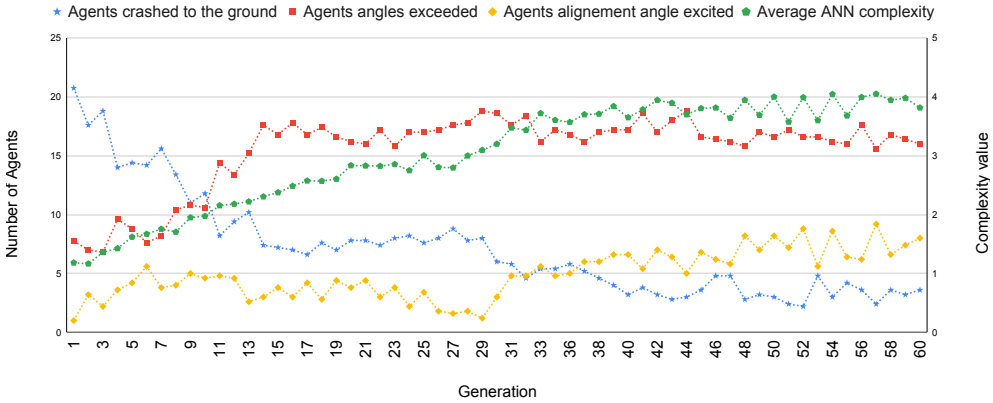

Figure 8. Exp. 2 - Average event count through generations - Plain Sigmoid activation function, Original work

of four runs, each with 60 generations. The population size was set to 40, resulting in a total of 9600 individuals evaluated throughout the experiment. On average, each run took 5 hours and 4 minutes to complete. The results indicate that the Gaussian activation function performed considerably worse than the two previous activation functions.

The goal of achieving a stable hover near the target was not accomplished by any of the agents, and the overall fitness values were the lowest. The best fitness value achieved was approximately 150, while the average fitness did not exceed 100. These low fitness values indicate that the agents had extremely short life spans, as they were either crashing instantly or attempting illegal maneuvers upon taking control of the quadcopter.
Based on this result, it can be concluded that the Gaussian activation function is not suitable for processing IMU input data.

5 Conclusions and Future Work

We have presented the approach for training Artificial Neural Networks to control a quadcopter in the Gazebo environment and perform basic autonomous maneuvers.
To achieve this, we utilized the NEAT algorithm and implemented a distributed framework that enables users to control and observe the training process in real-time. We conducted a series of experiments and obtained encouraging results. The experiments are a validation of the correctness and robustness of the implementation.

As a future work, we plan to explore alternative activation functions and tackle more demanding scenarios, including wind and advanced maneuvers. Nevertheless, our ultimate objective is to transfer the trained ANN onto a physical drone and validate the results in real-world conditions.

References

1. Przychodzki, M. Zastosowanie sztucznych sieci neuronowych do kontroli bezzałogowych statków powietrznych. (Siedlce University of Natural Sciences,2023,9)

2. Rosenblatt, F. Principles of neurodynamics: Perceptions and the theory of brain mechanisms. (Spartan,1962)
3. Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature. **323** pp. 533-536 (1986), https://api.semanticscholar.org/CorpusID:205001834
4. Maaloul, B. & Elloumi, S. Adaptive PID Controller of a Quadrotor. 2023 IEEE International Conference On Advanced Systems And Emergent Technologies. pp. 1-6 (2023)
5. Tayebi, A. & McGilvray, S. Attitude stabilization of a four-rotor aerial robot. 2004 43rd IEEE Conference On Decision And Control (CDC) (IEEE Cat. No.04CH37601). **2** pp. 1216-1221 Vol.2 (2004)
6. MATLAB version 7.10.0 (R2010a). (The MathWorks Inc.,2010)
7. Shah, S., Dey, D., Lovett, C. & Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. Field And Service Robotics. (2017), https://arxiv.org/abs/1705.05065
8. AlMahamid, F. & Grolinger, K. Autonomous unmanned aerial vehicle navigation using reinforcement learning: A systematic review. Engineering Applications Of Artificial Intelligence. **115** pp. 105321 (2022)
9. Duggal, V., Sukhwani, M., Bipin, K., Reddy, G. & Krishna, K. Plantation monitoring and yield estimation using autonomous quadcopter for precision agriculture. 2016 IEEE International Conference On Robotics And Automation (ICRA). pp. 5121-5127 (2016)
10. Patrick, O., Nnadi, E. & Ajaelu, H. Effective use of quadcopter drones for safety and security monitoring in a building construction sites: Case study Enugu Metropolis Nigeria. Journal Of Engineering And Technology Research. **12**, 37-46 (2020)
11. Mohd Daud, S., Mohd Yusof, M., Heo, C., Khoo, L., Chainchel Singh, M., Mahmood, M. & Nawawi, H. Applications of drone in disaster management: A scoping review. Science And Justice. **62**, 30-42 (2022), https://www.sciencedirect.com/science/article/pii/S1355030621001477
12. Merkel, D. Docker: lightweight linux containers for consistent development and deployment. Linux Journal. **2014**, 2 (2014)
13. Lu, P. & Geng, Q. Real-time simulation system for UAV based on Matlab/Simulink. 2011 IEEE 2nd International Conference On Computing, Control And Industrial Engineering. **1** pp. 399-404 (2011)
14. Madaan, R., Gyde, N., Vemprala, S., Brown, M., Nagami, K., Taubner, T., Cristofalo, E., Scaramuzza, D., Schwager, M. & Kapoor, A. AirSim Drone Racing Lab. Proceedings Of The NeurIPS 2019 Competition And Demonstration Track. **123** pp. 177-191 (2020,12,8), https://proceedings.mlr.press/v123/madaan20a.html
15. González Rodríguez, C. Development and assembly of a cinematography drone. (UPC, Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, Departament d'Enginyeria Electrònica,2019,6), http://hdl.handle.net/2117/165496
16. Zhao, H. & Wang, Z. Motion Measurement Using Inertial Sensors, Ultrasonic Sensors, and Magnetometers With Extended Kalman Filter for Data Fusion. IEEE Sensors Journal. **12**, 943-953 (2012)
17. Bezkorovainyi, Y. & Sushchenko, O. Improvement of UAV Positioning by Information of Inertial Sensors. 2018 IEEE 5th International Conference On Methods And Systems Of Navigation And Motion Control (MSNMC). pp. 123-126 (2018)
18. Sun, J. Modeling and analysis of pulse-width modulation. Professional Education Seminar At IEEE Applied Power Electronics Conference, Austin, Texas, USA. (2008)
19. Gopalakrishnan, E. Quadcopter flight mechanics model and control algorithms. Czech Technical University. **69** pp. 8-30 (2017)
20. Member of Multiwii community - Waltr New MultiwiiSerialProtocol-MultiWii. Www.multiwii.com. (2011), $http://www.multiwii.com/wiki/index.php?title=MultiwiiSerialProtocol$

21. Stanley, K. & Miikkulainen, R. Evolving neural networks through augmenting topologies. Evolutionary Computation. **10**, 99-127 (2002)
22. Lambora, A., Gupta, K. & Chopra, K. Genetic Algorithm- A Literature Review. 2019 International Conference On Machine Learning, Big Data, Cloud And Parallel Computing (COMITCon). pp. 380-384 (2019)
23. Lu, Y., Xue, Z., Xia, G. & Zhang, L. A survey on vision-based UAV navigation. Geo-spatial Information Science. **21**, 21-32 (2018)
24. O'Shea, K. & Nash, R. An introduction to convolutional neural networks. ArXiv Preprint ArXiv:1511.08458. (2015)
25. Boden, M. A guide to recurrent neural networks and backpropagation. The Dallas Project. **2**, 1-10 (2002)
26. Svozil, D., Kvasnicka, V. & Pospichal, J. Introduction to multi-layer feed-forward neural networks. Chemometrics And Intelligent Laboratory Systems. **39**, 43-62 (1997)
27. Rady, S., Kandil, A. & Badreddin, E. A hybrid localization approach for UAV in GPS denied areas. 2011 IEEE/SICE International Symposium On System Integration (SII). pp. 1269-1274 (2011)
28. Valenti, F., Giaquinto, D., Musto, L., Zinelli, A., Bertozzi, M. & Broggi, A. Enabling Computer Vision-Based Autonomous Navigation for Unmanned Aerial Vehicles in Cluttered GPS-Denied Environments. 2018 21st International Conference On Intelligent Transportation Systems (ITSC). pp. 3886-3891 (2018)
29. Burman, P. & Others Quadcopter stabilization with neural network. (2016)
30. Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U. & Von Stryk, O. Comprehensive simulation of quadrotor uavs using ros and gazebo. Simulation, Modeling, And Programming For Autonomous Robots: Third International Conference, SIMPAR 2012, Tsukuba, Japan, November 5-8, 2012. Proceedings 3. pp. 400-411 (2012)
31. Zhang, M., Qin, H., Lan, M., Lin, J., Wang, S., Liu, K., Lin, F. & Chen, B. A high fidelity simulator for a quadrotor UAV using ROS and Gazebo. IECON 2015 - 41st Annual Conference Of The IEEE Industrial Electronics Society. pp. 002846-002851 (2015)
32. Sciortino, C. & Fagiolini, A. ROS/Gazebo-Based Simulation of Quadcopter Aircrafts. 2018 IEEE 4th International Forum On Research And Technology For Society And Industry (RTSI). pp. 1-6 (2018)
33. Stanley, K., Clune, J., Lehman, J. & Miikkulainen, R. Designing neural networks through neuroevolution. Nature Machine Intelligence. **1**, 24-35 (2019)
34. Perez, R., Arnal, J. & Jansen, P. Neuro-Evolutionary Control for Optimal Dynamic Soaring. AIAA Scitech 2020 Forum., https://arc.aiaa.org/doi/abs/10.2514/6.2020-1946
35. Kim, E. & Perez, R. Neuroevolutionary Control for Autonomous Soaring. Aerospace. **8** (2021), https://www.mdpi.com/2226-4310/8/9/267
36. Betaflight community Betaflight flight software. Betaflight., https://betaflight.com/docs/wiki, [Accessed: 2023]
37. Gazebo community Gazebo. Gazebo., https://gazebosim.org/docs, [Accessed: 2023]
38. Smith, R. ODE Physic Engine. Open Dynamics Engine., http://ode.org/ode-latest-userguide.html, [Accessed: 2023]
39. Osrf What is SDFormat. SDFormat., http://sdformat.org/, [Accessed: 2023]
40. Osrf Osrf/vehicleGateway: A pluginlib-based C++ library that interfaces with several vehicle SDK's. GitHub., $https://github.com/osrf/vehicle_gateway$, [Accessed: 2023]
41. ArduPilot community ArduPilot/ARDUPILOT GAZEBO: Plugins and models for vehicle simulation in gazebo SIM with Ardupilot SITL controllers. GitHub., $https://github.com/ArduPilot/ardupilot_gazebo$, [Accessed: 2023]
42. Liang, O. RC TX RX Protocols Explained: PWM, PPM, SBUS, DSM2, DSMX, SUMD (2015). Oscarliang.com., https://oscarliang.com/pwm-ppm-sbus-dsm2-dsmx-sumd-difference/, [Accessed: 2023]

43. Technologies, U. Unity 3D. Unity3D., https://unity.com/, [Accessed: 2023]
44. Green, C. ResharpNEAT libary. GitHub., https://github.com/colgreen/sharpneat, [Accessed: 2023]
45. Wolf, F. UnityResharpNEAT libary. GitHub., https://github.com/flo-wolf/UnitySharpNEAT, [Accessed: 2023]
46. ALLAIN, R. How Do Drones Fly? Physics, of Course!. Wired., https://www.wired.com/2017/05/the-physics-of-drones/, [Accessed: 2023]