Università Politecnica delle Marche

**FACOLTÀ DI INGEGNERIA**
**Corso di Laurea in Ingegneria Informatica e dell'Automazione**

# DESIGN AND DEVELOPMENT OF A NEUROEVOLUTIONARY CONTROLLER FOR AUTONOMOUS FLIGHT QUADCOPTER SYSTEMS

# PROGETTAZIONE E SVILUPPO DI UN CONTROLLORE NEUROEVOLUTIVO PER SISTEMI DI VOLO AUTONOMO QUADRIROTORE

**Relatore:**
**Chiar.mo Prof.**
**SIMONE FIORI**

**Tesi di laurea di:**
**MANUEL MARIANI**

**Anno Accademico: 2019/20**

**Abstract**

The problem addressed in the present thesis document is the development of a controller based on a neural network for autonomous flight in quadrotor systems. The controller's objective is to govern the quadcoper such that its center of mass position reaches a specific point, remaining as stable as possible both during flight and upon reaching its target. Given the complex and unstable nature of quadcopters, an appropriate neural network architecture and training algorithm had to be designed. The controller has been implemented with a single multi layer perceptron. Starting from the quadcopter's current state, the neural network produces the correct rotors' speed values that the quadcopter needs to optimally reach the target, both in terms of stability and flight speed. The neural network's training has been achieved using a custom genetic algorithm, alongside a numerical simulation, where particular emphasis is put in the cost function's definition. Lastly, the neural controller has been employed and tested to autonomously follow a complex path. The results are promising: the neural controllers manage to effortlessly follow several types of paths with adequate precision and optimal stability levels while maintaing low travel times.

**Sommario**

In questa tesi viene affrontato il problema dello sviluppo di un controllore per sistemi di volo autonomo quadrirotore basato su una rete neurale. Il controllore ha come obiettivo il far raggiungere al centro di massa del drone un preciso punto, rimanendo il più stabile possibile sia durante il volo sia una volta raggiunto il bersaglio. Per via della natura complessa e instabile dei quadricotteri, è stato necessario progettare un'appropriata strategia di controllo. Il controllore è stato realizzato tramite un singolo percettrone multistrato che, partendo dallo stato attuale del drone in ogni istante, produce i corretti valori di velocità che i quattro rotori devono assumere per raggiungere un bersaglio in modo ottimale, sia in termini di stabilità che di velocità. Per addestrare la rete neurale è stato implementato un particolare algoritmo neuro evolutivo, affiancato da una simulazione numerica, ponendo particolare enfasi nella definizione della funzione di costo. Infine, il controllore neurale è stato utilizzato e testato per seguire autonomamente un percorso complesso. I risultati sono promettenti: i controllori neurali riescono a svolgere efficacemente svariati tipi di percorsi con adeguata precisione e con ottimi livelli di stabilità, mantenendo bassi tempi di percorrenza.

# 1 Introduction

In the recent years quadcopters, also known as drones, have seen an increase in interest and popularity among consumers, professional users and enterprises [4, 7, 12, 15, 24], due to their relatively low manufacturing cost and the continual improvements in the performance and reliability of the controllers applied to them. Also, with the resurgence of artificial intelligence of the past 20 years [19], the topic of neural networks gained growing interest in the scientific community, and its potentialities are starting to gain traction in a multitude of different environments. By applying neural networks to the problem of quadcopter control as an alternative to standard industrial controllers, particularly for autonomous flight systems, a more powerful and flexible solution could be attained.

To assess whether neural network controllers are a suitable type of controllers, a type of neuro-evolutionary controller for autonomous flight in quadcopter systems has been designed, implemented and tested in a simulation, as described in this thesis document. The chosen neural network's topology was a multi-layer perceptron, because of its ease of implementation and low computational cost, thus being a good choice for implementation on dedicated hardware in real applications. To train the network, a genetic algorithm was developed, alongside a 3D environment for qualitatively checking the state of training in real time. The neural network was trained to stably reach a target point in space and, once it reaches the target, hover in that exact position; even so, the resulting trained neural controller can be used to have the quadcopter follow a specified path by sub-dividing the path in multiple segments, achieving a proof-of-concept for autonomous flight. The outcomes of this training algorithm have shown really promising results: it takes a relatively short amount of time for the evolutionary algorithm to produce adequate neural controllers (around 2 hours on a laptop [Intel i5 8250U, 16 GB RAM]) and with more training time and finer settings, the resulting controllers' behavior exhibits a very stable behavior and high response speed.

This thesis document presents the design process employed in the conception of the neural controller's structure and the neuro evolutionary algorithm in Section 3. The software methods used in their development and implementation are described in Section 4, and results of numerical simulations are shown in Section 5. Section 6 concludes this thesis document.

# 2 Literature review and motivation

To perform the task of quadcopter control, a numerous quantity of controllers and techniques have been designed and implemented in the past. The most commonly used type is the proportional-derivative (PD) [10] or proportional-derivative-integral (PID) controller [6, 16, 27, 32] and there exist several different ways to implement these types of controller. The most used structures are a multitude of PID controllers in series (or cascading) [16], in parallel [27] or a mix of these configurations [32] and each controller is specialized on controlling the position or the attitude. Other control techniques have also been implemented for quadcopter flight, such as backstepping [26], linear quadratic regulators (LQR) [1] and sliding mode controllers [5]. These controllers' implementations provide more than adequate results, both in stability and performance, but they mainly focus on stabilization or remote control. The paper [29] deals with the trajectory tracking problem for a quadrotor unmanned aerial vehicle. A flight controller with a hierarchical structure is designed, whereby the complete closed-loop system is divided into two blocks. The system has an inner block for attitude control and an outer block for position stabilization based on PD/PID controllers.

To address the problem of autonomous quadcopter flight, several techniques have been designed, implemented and tested. The article [11] presents a control system that allows a small sized quadcopter to achieve autonomous flight in indoor environments, without relying on GPS, with both PD and PID controllers and high level control modules for path planning and collision avoidance. The paper [25] implements PID, backstepping and fuzzy control techniques, overviewed by a high level task planning module, for autonomous flight and qualitatively evaluates each technique's performance, both in indoor and outdoor scenarios, discussing each control technique advantages and disadvantages. The paper [22] presents a hybrid robust control strategy to solve the trajectory tracking function of a quadcopter with time-varying mass. The article [3] describes the design, fabrication, and flight test evaluation of a morphing geometry quadcopter.

Neural networks have also been utilized as quadcopter controllers, particularly in autonomous flight settings. In the paper [13], an autonomous vision based neural network control system, for governing a drone in dynamic racing environments, is implemented using a convolutional neural network (CNN) to convert the raw images in waypoints and desired speeds which are then converted to the appropriate rotors speeds. To train the network, supervised learning was used. This is a viable approach but requires the acquisition of a conspicuous amount of data for the training. Another approach that does not involve the gathering of large training data sets or the tuning of PID parameters is reinforcement learning, as it has been proven in the paper [14], where low-level control of a quadrotor is achieved using a model-based reinforcement learning (MBRL) technique. In the paper [9], a novel framework for leader-follower formation control is developed for the control of multiple quadrotor unmanned aerial vehicles (UAVs). A novel neural network control law for the dynamical system is introduced to learn the complete dynamics of the UAV including unmodeled dynamics like aerodynamic friction. Also, the paper [21] addresses formation tracking for multiple low-cost underwater drones, by implementing distributed adaptive neural network control (DANNC), on the basis of a leader-follower architecture to operate in hazardous environments.

Genetic algorithms, a particular unsupervised learning technique, have also been proven capable of training neural controllers for autonomous flight in quadrotor systems. In the paper [20], the authors introduced the use of the neuro evolution of augmented topologies' (NEAT) algorithm [31] which evolves a neural network control structure for optimal dynamic soaring flight trajectories. NEAT is also used in [28] where a hierarchical controller composed of multiple neural networks, each controlling a quadcopter's variable (roll, pitch, yaw and elevation) are used in a simulation to follow a path. The advantage of NEAT is that allows neural networks to evolve both their weights and topology, without having to heuristically specify the desired network's topology.

The aim of the present research endeavor is to prove that a simpler controller, structured as a single neural network, is a viable solution for the problem of autonomous flight in quadcopter systems, and also that complex training methods, such as algorithms that require considerable amount of data or algorithms that require complex implementations, can be replaced by simpler and more elegant solutions.

# 3 Drone control by a neuro-evolutionary algorithm

Quadcopters are flying vehicles composed of a rigid frame with four propellers equally spaced from the center of the frame, which is also the center of mass of the system. The variation of rotors' speeds generates torque and thrust, thus allowing the quadcopter to vary its position and attitude relative to the ground. Therefore, a quadcopter has six degrees of freedom, but only four rotors that can control its attitude and altitude. This makes quadcopters an interesting challenge to control because they are under-actuated non-linear systems.

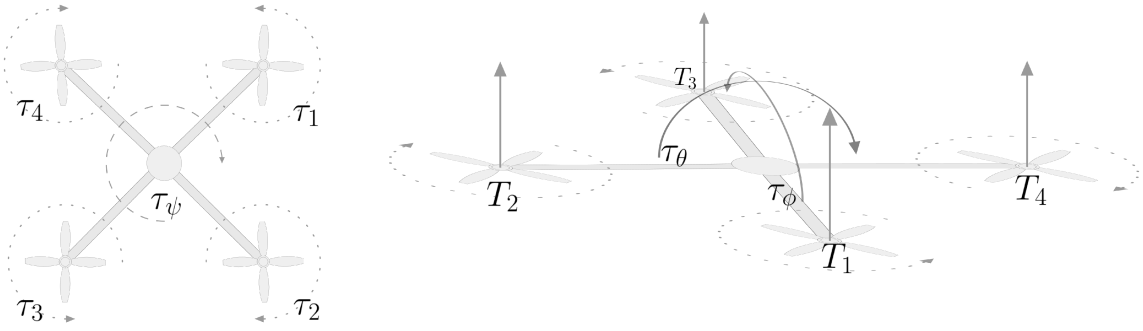



Figure 1: Left-hand panel: overhead view of the quadcopter, showing the rotors' torques $\tau_i$ and the resulting torque on the drone's z axis $\tau_\psi$. Right-hand panel: side view of the quadcopter showing the thrusts $T_i$ generated by the propellers and the torques $\tau$ on the x-y axes.

In a standard four-rotors quadcopter, the rotors are divided in two pairs, one for each of the x and y body's axis, and each pair spins clockwise or counter-clockwise, as illustrated in Figure 1. This makes the torques $\tau$ of each pair of rotors counteract the others, resulting in zero torque

$\tau_\psi$ along the z axis of the drone. To achieve a rotation along the z axis (yawing), the speed of one pair is decreased or increased, so that the net torque along the z axis of the drone's body $\tau_\psi$ is not zero. For example, by increasing only the clockwise rotor's speed, the component of the torque along the z axis of the body will be non-zero and will be directed in the counter-clockwise direction. This method of controlling the torque generated by the rotors, and along with similar others (like coaxial-blades and tandem-rotors), compared to traditional systems (like tail-rotors in helicopters), has the advantage of symmetry, because each blade has the same exact role as the others.

Each rotor $i$ produces a vertical trust $T_i$ perpendicular to the quadcopter body. By controlling each rotor's speed, it is possible to control the rotation of the drone along its body's x and y axis (pitch and roll). To induce a rotation along one of these axes, the thrust generated by the propellers is used: along one axis, if one propeller is faster (or slower) the difference in thrust compared to the thrust generated by the opposite propeller generates a moment of force $\tau_\phi$ or $\tau_\theta$ which induces a rotation in the body. To avoid unwanted yawing, the increase in speed in one of the rotors must be compensated by an equal decrease in the speed of the rotor opposite to it (along the same axis), that results in a zero net torque along the body's z axis.

## 3.1 Mathematical model of the quadcopter

In the used model [17], the quadcopter is defined in two frames of reference: the "earth" frame and the "body" frame. The quadcopter is described in the earth frame by its position $\boldsymbol{\xi}$ and angular position (attitude) $\boldsymbol{\eta}$ vectors:

$$\boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \ \boldsymbol{\eta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \tag{1}$$

where $\phi$ is the roll angle, the rotation around the earth frame x-axis, $\theta$ is the pitch angle, the rotation around the earth frame y-axis, and $\psi$ is the yaw angle, around the earth frame z-axis.

In the body frame are defined the linear velocities $\boldsymbol{V}_B$ and the angular velocities $\boldsymbol{\nu}$ relative to the origin (the center of mass)

$$\boldsymbol{V}_B = \begin{bmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{bmatrix}, \ \boldsymbol{\nu} = \begin{bmatrix} \nu_x \\ \nu_y \\ \nu_z \end{bmatrix}. \tag{2}$$

To transform the angular velocities relative to the body frame $\boldsymbol{\nu}$ to the attitude-rate (the derivative with respect to time $\dot{\boldsymbol{\eta}}$ of the attitude $\boldsymbol{\eta}$) a transformation matrix $\boldsymbol{W}_\eta$ is used:

$$\boldsymbol{W}_\eta^{-1} = \begin{bmatrix} 1 & S_\phi T_\theta & C_\phi T_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi / C_\theta & C_\phi / C_\theta \end{bmatrix}, \tag{3}$$

where $C_x = \cos x$, $S_x = \sin x$ and $T_x = \tan x$. In fact, $\dot{\boldsymbol{\eta}} = W_\eta^{-1} \boldsymbol{\nu}$.

Since the rotors only generate thrust along the z-axis of the body, to obtain the resultant thrust vector relative to the earth frame of reference, the attitude of the quadcopter is accounted by using a rotational matrix $\boldsymbol{R}$:

$$\boldsymbol{R} = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix}. \tag{4}$$

The formulas that relate the input of the system $\boldsymbol{\omega}$ (the rotors' speeds) to both the external state (position $\boldsymbol{\xi}$ and attitude $\boldsymbol{\eta}$ relative to the earth frame) and the internal state of the drone are summed up in Figure 2.

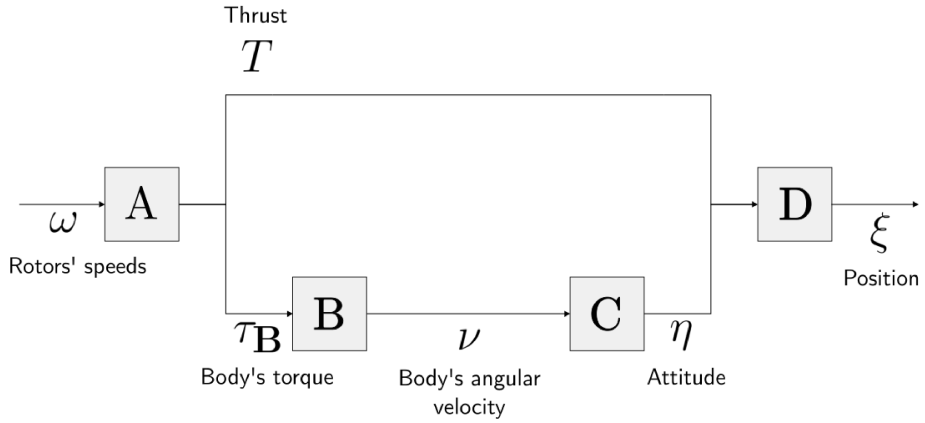



Figure 2: Quadcopter model summary. The model is divided in sub-processes, for the ease of reading, represented by blocks.

A From the current state, or the initial conditions, we can calculate the torque relative to the body frame $\boldsymbol{\tau_B}$ and the thrust $T$ on the z axis relative to the body frame:

$$\boldsymbol{\tau}_B = \begin{bmatrix} lk(-\omega_2^2+\omega_4^2) \\ lk(-\omega_1^2+\omega_3^2) \\ b(-\omega_1^2+\omega_2^2-\omega_3^2+\omega_4^2) \end{bmatrix} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}, \tag{5}$$

$$T = k(\omega_1^2+\omega_2^2+\omega_3^2+\omega_4^2), \tag{6}$$

where $l$ is the length of the drone's arms, $k$ is the lift constant and $b$ is the drag constant of a single rotor.

B With the torque vector $\boldsymbol{\tau}_B$, the angular acceleration relative to the body frame $\dot{\boldsymbol{\nu}}$ can be calculated:

$$\dot{\boldsymbol{\nu}} = \begin{bmatrix} \dot{\nu}_x \\ \dot{\nu}_y \\ \dot{\nu}_z \end{bmatrix} = \begin{bmatrix} \tau_\phi/I_{xx} \\ \tau_\theta/I_{yy} \\ \tau_\psi/I_{zz} \end{bmatrix} + \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}}\nu_y\nu_z \\ \frac{I_{zz}-I_{xx}}{I_{yy}}\nu_x\nu_z \\ \frac{I_{xx}-I_{yy}}{I_{zz}}\nu_x\nu_y \end{bmatrix} - I_r \begin{bmatrix} \nu_y/I_{xx} \\ -\nu_x/I_{yy} \\ 0 \end{bmatrix} (\omega_1-\omega_2+\omega_3-\omega_4) \tag{7}$$

Since the structure is symmetrical, $I_{xx} = I_{yy}$. In the above equation, $I_r$ denotes the rotors' moment of inertia. To compute the angular velocities vector $\boldsymbol{\nu}$ this differential equation can be solved using differential calculus methods or numerical ones.

C Using the transformation matrix $\boldsymbol{W}_\eta$, the attitude-rate of the drone relative to the ground $\dot{\boldsymbol{\eta}}$ can be obtained by a transformation of the angular velocities vector relative to the body frame $\boldsymbol{\nu}$:

$$\dot{\boldsymbol{\eta}} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \boldsymbol{W}_\eta^{-1}\boldsymbol{\nu} = \begin{bmatrix} 1 & S_\phi T_\theta & C_\phi T_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi/C_\theta & C_\phi/C_\theta \end{bmatrix} \begin{bmatrix} \nu_x \\ \nu_y \\ \nu_z \end{bmatrix}, \tag{8}$$

By numerically solving this differential equation, the attitude $\boldsymbol{\eta}$ of the quadcopter can be obtained.

D Finally, the center of mass' acceleration relative to the earth frame $\ddot{\boldsymbol{\xi}}$ can be obtained by applying the gravity acceleration directly and by applying the thrust force, divided by the drone's mass, rotated using the attitude matrix $\boldsymbol{R}$). Note that since the force is only applied to the z axis of the drone, only the third column of $\boldsymbol{R}$ needs to be calculated. The acceleration $\ddot{\boldsymbol{\xi}}$ obeys the law:

$$m\ddot{\boldsymbol{\xi}} = \boldsymbol{G} + T\boldsymbol{R}\boldsymbol{e}_z - \boldsymbol{A}\dot{\boldsymbol{\xi}}, \tag{9}$$

where $m$ is the mass of the quadcopter, $\boldsymbol{G}$ is the gravitational force (weight) directed towards the negative direction of the earth frame's z-axis, namely $\boldsymbol{G} = -mg\boldsymbol{e}_z$, where $g$ denotes the gravitational acceleration and $\boldsymbol{A}$ denotes a viscous drag tensor. The above relationship may be written in plain format as:

$$\ddot{\boldsymbol{\xi}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \frac{T}{m}\begin{bmatrix} C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi S_\theta C_\phi - C_\psi S_\phi \\ C_\theta C_\phi \end{bmatrix} - \frac{1}{m}\begin{bmatrix} A_x & 0 & 0 \\ 0 & A_y & 0 \\ 0 & 0 & A_z \end{bmatrix}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}, \tag{10}$$

where $A_x$, $A_y$ and $A_z$ are the drag force coefficients for the velocities in the $x, y$ and $z$ direction of the earth frame of reference.

## 3.2 Control strategy

The main goal of this project is the design of a controller to attain autonomous flight in a quadcopter system. The first step in the design was the definition of the drone's desired behavior, so the appropriate control scheme and neural network training strategy could be planned.

The main objectives of the autonomous flight controller are to reach a point fixed in space (with the drone's center of mass) and to be stable both in the trajectory and around the target. Defining and measuring the first objective is simple and straight forward, but defining the second one has its complications.

The definition of "stable flight" is in itself very fuzzy and could be interpreted (linguistically and mathematically) in many ways; for example, by defining stable as "the quadcopter body must be as parallel to the ground as possible ($\phi \approx 0, \theta \approx 0$)", results in two conflicting objectives: the quadcopter has to reach the target so it has to tilt to follow a trajectory, but this does not comply with the current definition of stability, thus resulting in two conflicting goals.

This problem can be fixed using multi-objective control and optimization techniques, but instead the requirement of stability was redefined as "the quadcopter has to turn along its axes for as little time and amount as possible, and each rotation in one direction must be compensated and countered by a rotation in the other direction". This has two parts: the first, regarding the spin time and amount, means that the desired final behavior is to have fast rotations (to align itself to the optimal trajectory) and to avoid states in which the drone spins uncontrollably; the second part, about the compensation of a turn in one direction, ensures that the quadcopter will eventually spin in the other direction by an equal amount (compared to the first rotation), thus having a flat attitude once the drone reaches the target.

With this definition, the quadcopter's controller does not have conflicting objectives: to reach the target, it can turn towards the optimal trajectory because there are no constraints to the attitude angles, and to reach the correct flat attitude, the second part of the stabilization requirements (the compensation of the rotation) is used. Also, to further increase stability by avoiding extreme and unwanted angles of rotation, a constraint on the maximum attitude pitch $\phi$ and roll $\theta$ absolute values is enforced. The mathematical definition of all these requirements will be presented in the section about the cost function definition of the genetic algorithm.

This control behavior can be easily extended to follow a path by dividing the path in multiple sequential points and by providing a single target at a time, switching to next one when the quadcopter is close enough to current target.

### 3.2.1 Control diagram and scheme

The observable/measurable state $\boldsymbol{s}$ used to control the quadcopter is represented by:

- $\boldsymbol{\xi}$ - Center of mass position
- $\dot{\boldsymbol{\xi}}$ - Velocity
- $\ddot{\boldsymbol{\xi}}$ - Acceleration
- $\boldsymbol{\eta}$ - Attitude
- $\dot{\boldsymbol{\eta}}$ - Body angular velocities

The reference input (set point) is the desired position of the quadcopter's center of mass (target) $\boldsymbol{\xi}_T$.

The control strategy is a feedback loop, in which the current position is compared with the target's, resulting in the position's error $\boldsymbol{e}_{\text{pos}} = \boldsymbol{\xi} - \boldsymbol{\xi}_T$. The state error $\boldsymbol{\sigma}$ is defined as:

$$\boldsymbol{\sigma} = \begin{bmatrix} \boldsymbol{\xi} \\ \dot{\boldsymbol{\xi}} \\ \ddot{\boldsymbol{\xi}} \\ \boldsymbol{\eta} \\ \dot{\boldsymbol{\eta}} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\xi}_T \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_{\text{pos}} \\ \dot{\boldsymbol{\xi}} \\ \ddot{\boldsymbol{\xi}} \\ \boldsymbol{\eta} \\ \dot{\boldsymbol{\eta}} \end{bmatrix} \tag{11}$$

This state is passed to the neural network controller, which outputs the appropriate motors' angular speeds $\boldsymbol{\omega}$, the input to the quadcopter's mathematical model. There are no reference inputs concerning the attitude of the drone. This allows the quadcopter to follow an optimal trajectory (not defined a priori) based on its training experience. Nevertheless, as a result of the aforementioned objectives' definition and of the learning algorithm, once the quadcopter reaches the set point it will remain hovering in that position having a final state $\boldsymbol{s}_f$:

$$\boldsymbol{s}_f \approx \begin{bmatrix} \boldsymbol{\xi}_T \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{12}$$

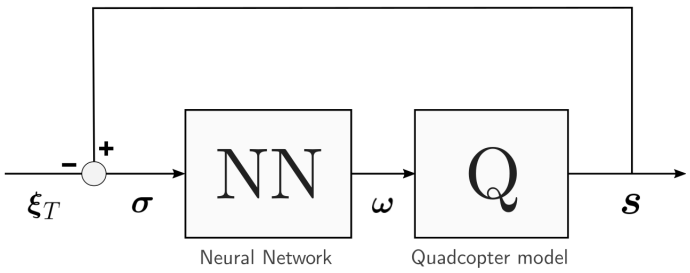



Figure 3: Control diagram of the system. NN: Neuro controller that takes the error state $\boldsymbol{\sigma}$ of the quadcopter. Q: quadcopter model that takes as input the rotors' speeds $\boldsymbol{\omega}$.

### 3.3 Neural network structure

The controller's neural network structure is a multi-layer perceptron illustrated in Figure 4. The network takes in input the state relative to the target $\boldsymbol{\sigma}$ which is composed of 5 $\mathbb{R}^3$ vectors, for a total of 15 input nodes.

To improve the learning performance, the input elements are mapped to $(-1, 1)$: each subvector $\boldsymbol{\sigma}_i$ of $\boldsymbol{\sigma}$ is normalized using $\tanh(n_i \boldsymbol{\sigma}_i)$ where $n_i$ is the normalization scaling factor of the corresponding $\boldsymbol{\sigma}_i$. This reduces the search space [30] for the algorithm, thus improving its training performance, by reducing and bounding the dimensionality of the problem from an input space $\mathbb{R}^{15}$ to a space $(-1, 1)^{15}$.

The normalized input $\boldsymbol{\sigma}^*$ then passes through $h$ hidden layers, each of size $s_i$, resulting in the output of the network $\boldsymbol{\omega}^*$. Each node is activated using a weighted and biased sigmoid $P(wx+b) = \frac{1}{1+e^{wx+b}}$:

$$\boldsymbol{L}_{i+1} = P(\boldsymbol{W}_i \boldsymbol{L}_i + \boldsymbol{B}_{i+1}),\ i = 1, \ldots, h+1, \tag{13}$$

where $\boldsymbol{L}_i$ is the nodes' vector of the layer $i$, $\boldsymbol{W}_i$ is the weight matrix from the current layer $i$ to the next, of size $s_{i+1} \times s_i$, $\boldsymbol{B}_i$ is the bias vector of the layer $i$ and $h+2$ is the total number of layers (one input, one output and $h$ hidden layers). The size of $\boldsymbol{L}_1 = \boldsymbol{\sigma}^*$ and $\boldsymbol{L}_{h+2} = \boldsymbol{\omega}^*$ are fixed at $15 \times 1$ and $4 \times 1$ respectively, while the size and number of hidden layers may be varied. The values of $\boldsymbol{W}_i$ and $\boldsymbol{B}_i$ for $i = 1, \ldots, h+1$, which determine the response of the controller to the

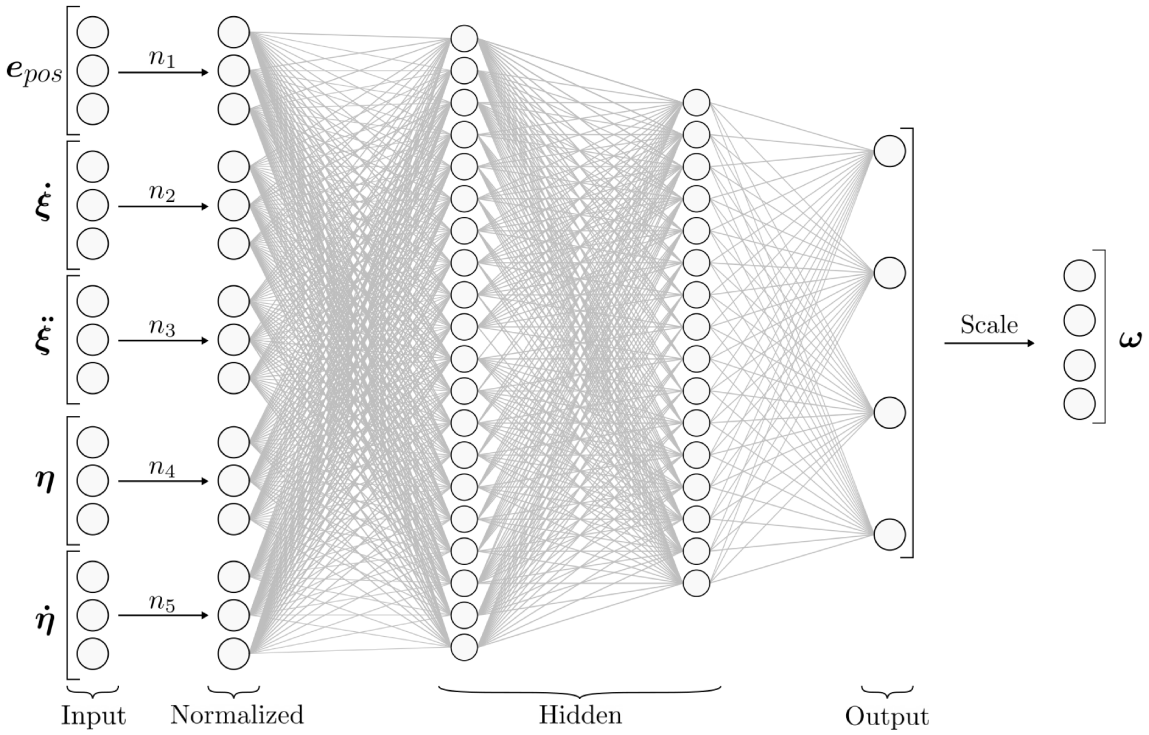



Figure 4: Neural network structure including input and output signals. The topology shown is a {15, 20, 6, 4}.

current state, and the normalization scaling factors $n_j$ for $j = 1, \ldots, 5$ are defined by the learning algorithm.

The output of the network $\boldsymbol{\omega}^*$ is a $(0, 1)^4$ vector, so to obtain the rotors' desired speeds, it's mapped and scaled to the minimum and maximum propellers' rotation speed using:

$$\omega_i = \omega_{\text{MIN}} + (\omega_{\text{MAX}} - \omega_{\text{MIN}})\omega_i^\star, \tag{14}$$

for $i = 1, 2, 3, 4$.

## 3.4 Neuro-evolutionary algorithm

To train the neural-network controller, to comply with the control objectives previously defined, a learning algorithm has to be used together with a numerical simulation. Since the training process cannot be supervised, by using an algorithm such as back-propagation, because of the lack of training set (correct rotors' speeds in response to specific state) and also the difficulty of getting such amounts of data, unsupervised learning or reinforcement learning algorithm was the solution of choice.

The algorithm implemented is a genetic algorithm that, compared to other algorithms in this family like NEAT, does not modify the topology of the neural network, so the topology must be specified heuristically in advance. A general overview of the algorithm is shown in Figure 5.

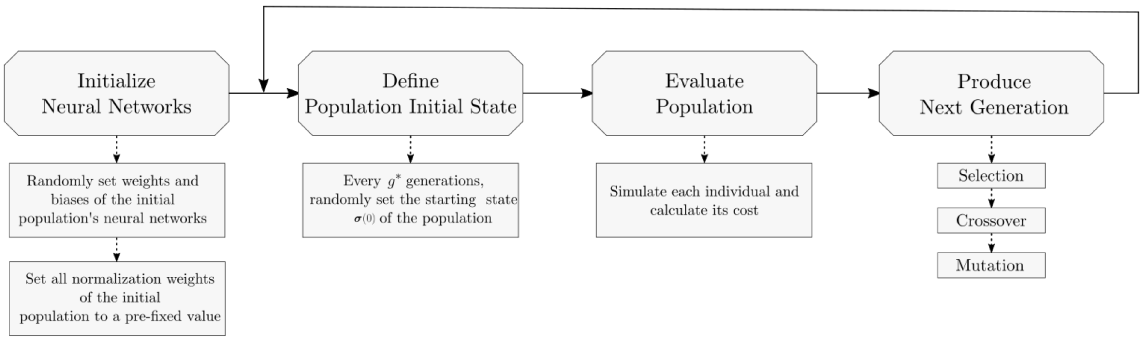



Figure 5: Flow-chart of the neuro-evolutionary algorithm used to train a neural-network-based controller.

The algorithm operates on a population of $p$ individuals, and in every iteration the next population, also called generation, is created from the previous. An individual, or agent, is defined as a single neural-controller configuration (of weights and biases) applied to a quadcopter model. The configuration of the values of the weights and biases matrix is represented by a genome with a direct encoding genotype representation [34], and the normalizer weights $\boldsymbol{n}$ are also added to the genotype. Direct encoding was preferred to other types of encoding because a more complex one was not needed since the algorithm does not influence topology and does not have concepts like speciation that sub-divide the population in different groups.

### 3.4.1 Initialization

At the start of the algorithm, the initial population $\Gamma(0)$, of size $p$, is created by assigning to each individual $a$, a randomly generated genome. The values of the weights are initialized as follows:

$$\boldsymbol{W}_i^a = \begin{bmatrix} r() & \cdots & r() \\ \vdots & \ddots & \vdots \\ r() & \cdots & r() \end{bmatrix}, \ \boldsymbol{B}_i^a = \begin{bmatrix} r() \\ \vdots \\ r() \end{bmatrix} \text{ for } i = 1, \ \ldots, \ h+1 \text{ and } a = 1, \ \ldots, \ p, \tag{15}$$

where $r()$ is a function that returns a real number, bounded by a constant $r_{\text{MAX}}$

$$r() \in [-r_{\text{MAX}}, \ r_{\text{MAX}}].$$

The algorithm also operates on the normalizer weights $\boldsymbol{n}$ so those are also specified at the start:

$$\boldsymbol{n} = \begin{bmatrix} n_1 \\ \vdots \\ n_5 \end{bmatrix} = \begin{bmatrix} \overline{n} \\ \vdots \\ \overline{n} \end{bmatrix}, \tag{16}$$

where $\overline{n}$ is a constant. So in the first generation $\Gamma(0)$ each individual has the same normalizer weights, but since the optimal value of $\boldsymbol{n}$ is not known a priori, in the upcoming generations those weights will vary because of mutations and crossovers. This is important, because it allows the algorithm to also optimize the amount of normalization that the input of the neural network $\boldsymbol{\sigma}$ undergoes, which can be interpreted as the neural network's input sensibility to each sub-vector of the input $\boldsymbol{\sigma}$.

### 3.4.2 Starting state definition

At the start of every generation, all the quadcopters in the population are set to have the same initial condition state $\boldsymbol{\sigma}(0)$. This means that every quadcopter starts in the same conditions as the others, so despite the differences in each individual genome, each one is evaluated in the same way

as the other individuals in the generation. Also, in the first and every $g^*$ generations, the initial state of the quadcopters $\boldsymbol{\sigma}(0)$ is randomly set:

$$\boldsymbol{\sigma}(0) = \begin{bmatrix} \boldsymbol{e}_{pos,0} \\ \dot{\boldsymbol{\xi}}_0 \\ \ddot{\boldsymbol{\xi}}_0 \\ \boldsymbol{\eta}_0 \\ \dot{\boldsymbol{\eta}}_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_e \\ \boldsymbol{r}_{\dot{\xi}} \\ \boldsymbol{r}_{\ddot{\xi}} \\ \boldsymbol{r}_{\eta} \\ \boldsymbol{r}_{\dot{\eta}} \end{bmatrix}, \tag{17}$$

where $\boldsymbol{r}_x$ is a random vector, with the same size as $x$, composed of random values bounded by a constant: $r_{x,i} \in [-\overline{r_x}, \overline{r_x}]$ for $i = 1, \ldots, \text{size}(x)$.

Having an initial state that is not fixed is crucial to the functioning of the algorithm. If in every generation $\Gamma(g)$ every individual were to start in the same state as the previous generation $\Gamma(g-1)$, without variation, the neural networks would just exploit their training by evolving to only have the correct behavior for that fixed starting state, and when presented with a situation different from the one that they were evolved from, the neural network will inevitably fail to control the quadcopter because of its lack of proper training.

Instead, by varying the starting state very frequently, because of the evolutionary nature of the algorithm, only the individuals that can effectively control the quadcopter in a multitude of different states will pass their genes to the next generation, resulting in each generation getting better at handling different states.

To further increase the exploration of possible states, the boundary of the random state generation $\overline{r_x}$ can be set to higher values. For example by increasing $\overline{r_{\dot{\eta}}}$, the boundary for the random value that the elements in the attitude-rate sub-vector $\dot{\boldsymbol{\eta}}$ of the starting state $\boldsymbol{\sigma}(0)$ can have, and $r_{\dot{\xi}}$, the boundary for the initial velocity, the individuals will start the simulation with higher speeds in random directions and higher rotation speeds of their bodies. This increases the harshness of the training and only the best individuals that can withstand those hard conditions will pass their genes and, over many generations, the evolutionary algorithm will produce individuals that can handle such conditions, that correspond to more robust and stabler neural controllers. Note that if the $\overline{r_x}$ parameters are set too high, the starting conditions will be outright impossible to control, therefore the evolution will not produce better individuals because of the impossibility in evaluating and distinguishing the performance of poorly evolved individuals against the good ones.

### 3.4.3 Population evaluation

After the current generation's $\Gamma(g)$ starting state $\boldsymbol{\sigma}^g(0)$ is defined, each individual is numerically simulated in discrete time and its performance evaluated.

Each individual $i$ in the population $\Gamma(g)$ has a lifespan $k_{\text{MAX}}$ in which, from the instant $k = 0$ to $k = k_{\text{MAX}}$, its performance is evaluated: in each instant $k$ from the state $\sigma_i^g(k)$ the individual's neural network outputs the rotor's speeds $\boldsymbol{\omega}_i^g(k)$ which are then used as inputs to the quadcopter mathematical model $Q$ to calculate the next state $\sigma_i^g(k+1)$ of the individual.

Also, if an individual reaches an height $z$ of less then 0 meters from the ground, to the purpose of the simulation, it is considered crashed and its state will not be updated, effectively remaining stuck in the position where it has collided to the ground. This is necessary because otherwise the quadcopters will not evolve to avoid crashing to the ground, and also it speeds up the evaluation process because the crashed individuals do not need to be further simulated.

Evaluating the performance of the current generation's individuals is a fundamental step for the genetic algorithm because, as described in detail in the next section, to produce the next generation, which ideally has to be better than the current, the individuals that performed better than the rest should pass their genes (neural network and normalizer weights) to the next one. To evaluate the performance of the current generation, a cost function [34] $C_i$ that represents the inverse of the performance of an individual $i$ (the higher the cost the worst the individual's performance) in the current generation in its lifespan $t_{\text{MAX}}$ is defined as follows:

$$C_i = C_{i,\xi} + C_{i,\eta} + C_{i,\dot{\eta}} + C_{i,\kappa}, \tag{18}$$

where, relatively to an individual $i$, $C_{i,\xi}$ represents how poorly it reaches the target, $C_{i,\eta}$ represents how much its attitude has exceeded certain tilting thresholds, $C_{i,\dot{\eta}}$ represents its noncompliance with the stability requisite, and $C_{i,\kappa}$ represents if and for how long the quadcopter has crashed.

An individual should reach the target position in the shortest amount of time and stay in that position. This can be evaluated and represented by measuring the sustained displacement of the quadcopter center of mass relatively to the target's position:

$$C_{i,\xi} = \sum_{k=0}^{k_{\text{MAX}}} \left\| \boldsymbol{e}_{i,pos}(k) \right\| \Delta t, \tag{19}$$

where $\Delta t$ represents the time, in seconds, that passes between step $k$ and step $k+1$.

To increase the stability of the desired quadcopter, it is desirable that its attitude angles $\phi$ and $\theta$ do not exceed a certain threshold $\eta_{\text{MAX}}$:

$$C_{i,\eta} = \sum_{k=0}^{k_{\text{MAX}}} c_{i,\eta}(k)\Delta t, \tag{20}$$

$$c_{i,\eta}(k) = \begin{cases} \overline{c_\eta} & \text{if } \left|\phi_i(k)\right| \geq \eta_{\text{MAX}} \ \vee \ \left|\theta_i(k)\right| \geq \eta_{\text{MAX}}, \\ 0 & \text{instead} \end{cases}, \tag{21}$$

in which $\overline{c_\eta}$ is a positive arbitrary constant representing the penalty for exceeding the limit on the attitude angles. The ideal $\eta_{\text{MAX}}$ is a value between $\frac{\pi}{6}$ rad and $\frac{\pi}{4}$ rad.

The aforementioned requirement of stability can be represented mathematically by defining:

$$C_{i,\dot{\eta}} = \left\| \sum_{k=0}^{k_{\text{MAX}}} \boldsymbol{f}(\dot{\boldsymbol{\eta}}_i(k))\Delta t \right\|, \tag{22}$$

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \text{sgn}(x_1)\log(1+|x_1|) \\ \vdots \\ \text{sgn}(x_n)\log(1+|x_n|) \end{bmatrix}, \text{ with } n = \text{size}(\boldsymbol{x}). \tag{23}$$

In the quantity $C_{i,\dot{\eta}}$, the attitude rates $\dot{\boldsymbol{\eta}}$ ($\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$) are compressed using a non linear function $\boldsymbol{f}(\boldsymbol{x})$, and then these values are summed in the life span of each individual. The compression of the attitude rate is an important step: since the function is non linear, small changes in attitude rate maintained for longer periods of time result in a higher cost $C_{i,\dot{\eta}}$ compared to fast changes in attitude maintained for shorter periods of time. Also, since the terms inside $\boldsymbol{f}(\boldsymbol{x})$ have the same sign as the components of $\boldsymbol{x}$, the sum of the values of $\boldsymbol{f}(\dot{\boldsymbol{\eta}}_i(k))$ over the lifespan of the individual is zero (the lowest cost possible) only if the individual $i$ rotates in such a manner as to align itself to the correct trajectory for reaching the target and then, once it is about to reach the set point, starts to rotate in the same exact way as the first rotation. So by having $C_{i,\dot{\eta}} = 0$, ideally the quadcopter has performed two opposite sequences of rotations, thus returning to the attitude $\boldsymbol{\eta}(0)$ defined in $\boldsymbol{\sigma}(0)$. Since it is ideal that once an individual $i$ reaches the target its attitude is $\boldsymbol{\eta}_i(k_{\text{MAX}}) = 0$ and if $\boldsymbol{\eta}(0) \neq 0$, the cost of the attitude change must necessarily be $C_{i,\dot{\eta}} > 0$; in this case, the cost relative to the individual's $i$ displacement $C_{i,\xi}$ is predominant (this means that to stay as close as possible to the target, the individual must necessarily have $\boldsymbol{\eta} = 0$ in the last period of its life span) and because of the fact that all individuals start with the same $\boldsymbol{\eta}(0)$, thus resulting in a lower bound for $C_{i,\dot{\eta}}$ that is equal for all the individuals in the generation, the individuals will still evolve to have a zero attitude once they reach the target. In summary, this attitude rate cost function promotes individuals that exhibit impulsive rotational behaviors and that compensate each rotation in one direction with one in the opposite.

Since it is desirable that the final controller avoids crashing the quadcopter to the ground, a

cost relative to this behavior $C_{i,\kappa}$ is introduced:

$$C_{i,\kappa} = \sum_{k=0}^{k_{\text{MAX}}} c_{i,\kappa}(k)\Delta t, \tag{24}$$

$$c_{i,\kappa}(t) = \begin{cases} \overline{\kappa} & \text{if } z_i(k) \leq 0 \text{ (crashed)}, \\ 0 & \text{instead}, \end{cases} \tag{25}$$

where $\overline{\kappa}$ is a constant taking a large value, since the avoidance of crashing is a very important requisite.

#### 3.4.4 Next generation

The core of the genetic algorithm is how to make the next generation perform better than the previous and its performance, the amount of time it takes to reach an optimal solution, is mainly determined by this phase. To achieve this, for each individual $j$ of the next generation $\Gamma(g+1)$, three steps are applied in sequence: two individuals (parents) $p_1$ and $p_2$ are selected [8] from the generation $g$, then their genes are combined (crossover) to create the offspring $j$ and then the offspring undergoes a mutation process in its genome.

Also a number $n_P$ of the best individuals of the current generation will be kept in the new generation. This is to avoid genetic drift, that can occur in two cases: in the event that all the offspring of the new generation get heavily mutated resulting in a worst performance than the previous one; and if the random starting state $\boldsymbol{\sigma}^g(0)$ is too difficult to control (for example if all the individuals start very close to the ground and upside down, thus hitting the ground immediately), then the costs of the mutated population would be all the same and not representative of the quality of an individual's genome.

**Selection.** For each individual $j$ of the next generation $\Gamma(g+1)$ two parents $p_1$ and $p_2$ are selected from the population $\Gamma(g)$ by using custom algorithm inspired by "tournament selection" [18]: $n_T$ individuals from the population $\Gamma(g)$ are randomly chosen to be candidates parents and then only the two best individuals (the ones with the lowest costs $C^g_{p_1}, C^g_{p_2}$ of the pool) are selected to be the parents $p_1$, $p_2$ of the individual $j$ of the next generation $\Gamma(g+1)$. The problem in selection algorithms is finding a balance between exploration and exploitation [33]. If the algorithm is focused on exploitation, it will get stuck in a local minimum and if it only explores it will never settle on a minimum. By increasing the pool size, the algorithm favors exploitation (if the maximum tournament size is the size of the population, the best two individuals will always be chosen) and by lowering the size, exploration is favored (if maximum tournament size is two, it is basically a completely random selection).

**Crossover.** Once two parents $p_1, p_2$ are selected, to generate their offspring, a crossover [23] between the genomes of $p_1$ and $p_2$ needs to occur. Since the genomes (weights, biases and normalizer weights) are directly encoded, the chosen method for generating the genome of the offspring was choosing each of its genes to be equal to one of its parents' genes, with an equal 50% chance.

**Mutation.** After an offspring $j$ is produced, it passes through a mutation [2] function that randomly chooses if the individual has to mutate based on a mutation rate $m_r$ and if so one of its genes gets mutated by adding a random number $r \in [-r_{\text{MAX}}, r_{\text{MAX}}]$. Then, since the mutation algorithm modifies only one gene at a time, if the $j$ gets mutated, the function is called again to further mutate the individual. Because of this, the mutation must be set relatively high ($\geq 50\%$) to increase the chance of mutating more than one gene. The mutation step in genetic algorithms is a fundamental step because without it, the algorithm will only generate subsequent generations with genes that are only a combination of the first generation's ($\Gamma(0)$) genes, resulting in a very limited exploration of the solution space. By adding a random number $r$, instead of directly assigning a random value, it ensures that the genes are not strictly bounded by an interval. Also better results in learning performance are given by setting the random number bound $r_{\text{MAX}}$ small, because it

makes the exploration strategy more gradual. In fact, if the interval is too small exploration, albeit more precise, is slower and more prone to get stuck in local minima.

The next-population generation procedure is outlined in Algorithm 1.

**Algorithm 1** Next population generation algorithm

h!

Set the first $n_P$ individuals of $\Gamma(g+1)$ as the best $b$ individuals of $\Gamma(g)$
**for all** remaining individuals $i$ **in** $\Gamma(g+1)$ **do**
  **Selection:**
  Randomly choose $n_T$ individuals from $\Gamma(g)$, generating a pool $P$
  In $P$ select the two distinct individuals $p_1$ and $p_2$ that exhibit the lowest cost $C^g_{p_1}, C^g_{p_2}$
  **Crossover:**
  **for all** genes $\gamma_a$ defined in the genome **do**
    $\gamma^i_a \leftarrow \gamma^{p_1}_a$ **or** $\gamma^i_a \leftarrow \gamma^{p_2}_a$ (randomly)
  **end for**
  **Mutation:**
  Randomly determine if $i$ has to mutate
  **while** $i$ has to mutate **do**
    Randomly choose a gene $\gamma^i_a$
    Randomly determine a value to mutate $r \in [-r_{\text{MAX}}, r_{\text{MAX}}]$
    $\gamma^i_a \leftarrow \gamma^i_a + r$
    Randomly determine if $i$ has to mutate again
  **end while**
**end for**

# 4 Implementation

In this section, programming techniques used to implement the learning algorithm, as well as the neural control algorithm, are summarized.

## 4.1 Numerical implementation

The quadcopter model equations, that are in the continuous time domain, are converted to the discrete time domain by using the forward Euler method. Since the step size of the simulation $\Delta t$ has to be necessarily small ($\Delta t \approx 0.01$) to give the neural network controller more instants in which to control the quadcopter, the forward Euler method, albeit not the most precise of all the numerical procedures for calculating differential equations, has shown very good results. Also in this project performance was more important than absolute numerical precision, and the forward Euler method has proven to be a good compromise. Nevertheless, the quadcopter model discrete equations can be extended and solved using different methods, without compromising the functioning of the program.

## 4.2 Programming implementation

The project is implemented in JAVA. Since the calculations involve matrices, the Efficient Java Matrix Library (EJML) is used, which has a very good performance, especially for dense small matrices like the ones present in this project.

To view the algorithm in action, a simple 3D environment is implemented using Processing 3, a library that uses OpenGL as its engine and can make both 3D and 2D interactive visualizations. Also, to get more functionalities in the program, described below, a simple GUI using Swing is used. To decrease the training times, the genetic algorithm uses multi-threading to simulate each individual in its life span.

The parameters of the quadcopter model, the neural network topology and the parameters of the genetic algorithm can be configured directly in a JSON file, without the need to modify the source code.

### 4.2.1 Functionalities

The main functionalities of the program are:

- Define the simulation parameters in a configuration file;
- Apply the genetic algorithm, with the specified configuration, with a 3D environment showing the progressive evolution of the generations;
- Apply the genetic algorithm without the 3D environment, to increase training speed;
- Load and save the neural networks of a population produced by the algorithm in a JSON file;
- From a neural network file, simulate and show in 3D the best individual in that population following a randomly generated path;
- Run multiple genetic algorithms or path following simulations, each with its configurations of parameters specified as files in a folder, while saving the results in multiple files.

### 4.2.2 Screenshots

In the following figures, some screenshots of the 3D environments are shown. In Figure 6 a static view of the 3D environment for the training is shown and in Figure 7 is shown the 3D simulation of a single trained quadcopter following a path. The quadcopters shown in Figure 6 are the result of 20,000 generations using the "harsh configuration", as described in detail in Section 5 to highlight the differences in the individuals performance, while the neural controller applied to the quadcopter in Figure 7 is trained for 250,000 generations with the same configuration.

On the basis of Figure 6 a brief summary of the genetic algorithm for a generation can be presented: initially all the individuals are given the same starting state $\boldsymbol{\sigma}(0)$, then they start controlling the quadcopter to reach the target point (the sphere in the images' center) and their performance is evaluated until the lifespan of the population $k_{\text{MAX}}$ is reached. As it can be seen, a portion of the simulation becomes unstable and starts falling to the ground. This is a normal behavior since the individuals have only been trained for 20,000 generations and is also a result of the random mutation process, which can improve or diminish the performance of certain individuals. Nevertheless, the better the performance of an individual the higher the probability of passing its genes to the next generation.

After training the neural networks for a desired amount of generations or time, the resulting controller's performance can be qualitatively evaluated in the path following environment shown in Figure 7. The program generates a path for the quadcopter to follow, displayed by a series of spheres connected by lines, and the quadcopter has to visit each point in sequence. A quantitative evaluation of the quadcopter's performance can be assessed by running the path following tests programmatically, without video, outputting useful and meaningful data results to a file. This feature was used in Section 5 to collect the values shown in figures and tables.

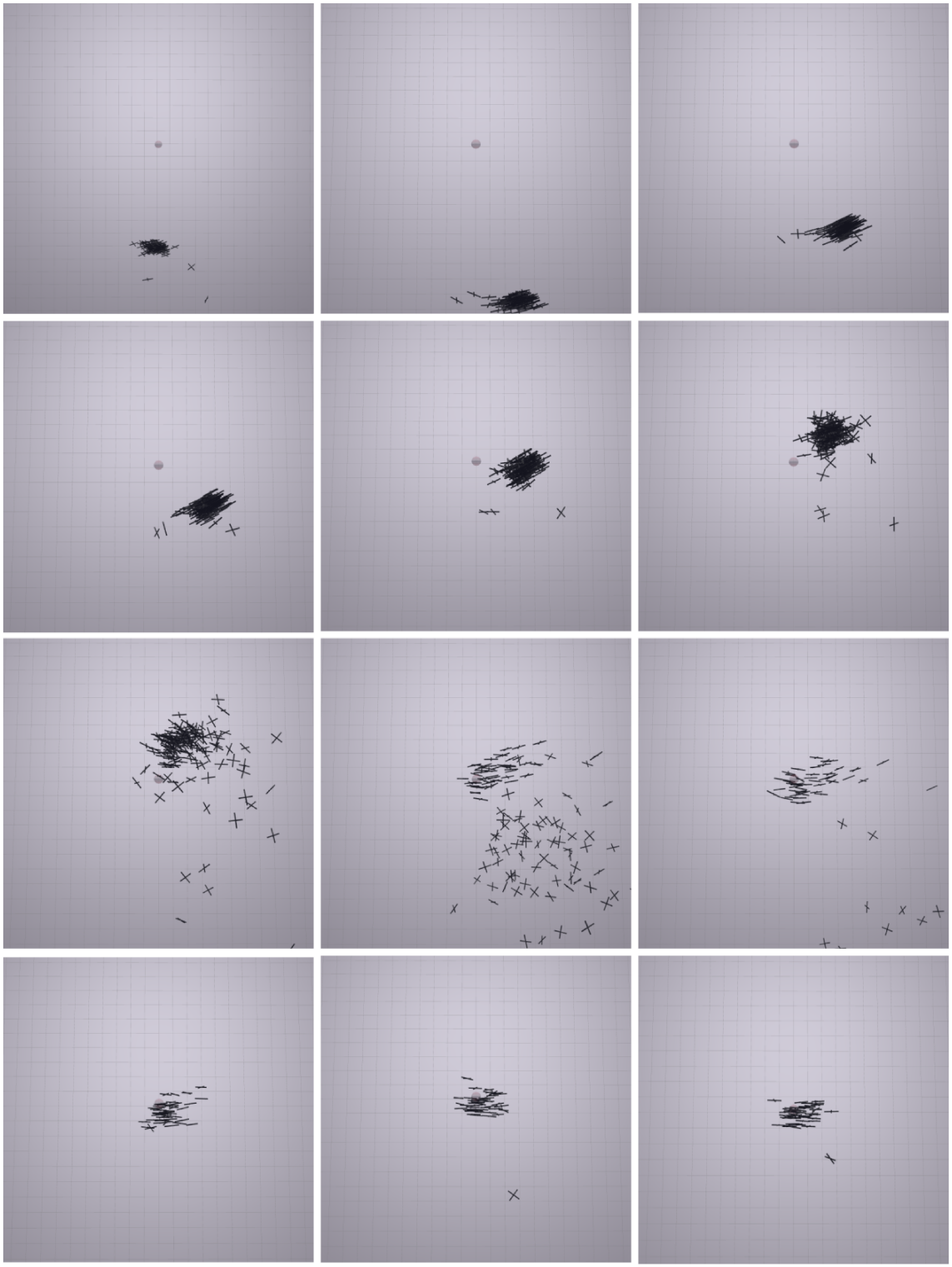


Figure 6: 3D training environment of the genetic algorithm, for a trained generation of 150 individuals. In each frame the point in the center represents the target (set point) and the crosses represent each individual. From left to right and top to bottom, screenshots of the current generation's quadcopters behaviors in time, each 0.1 seconds apart.

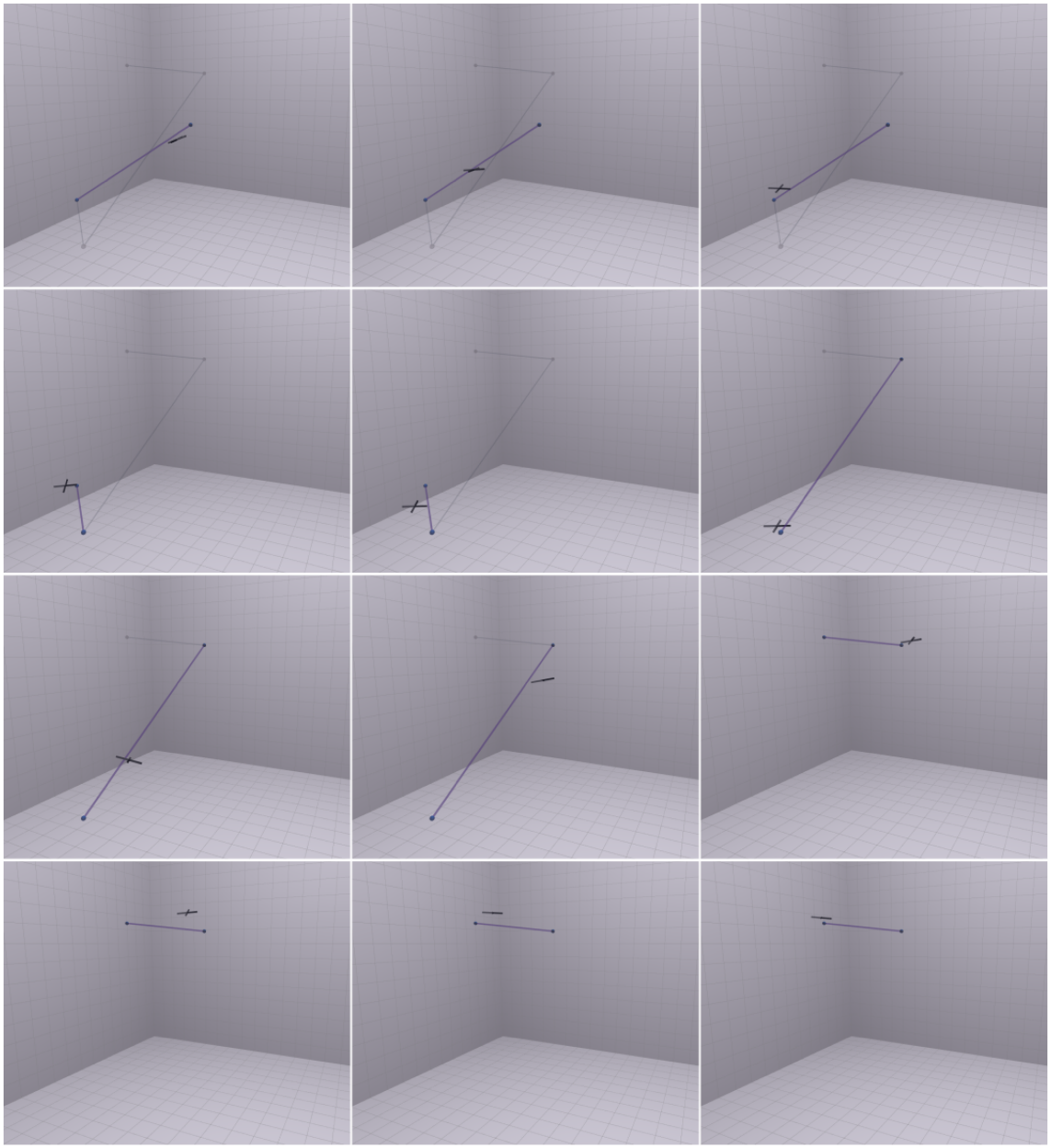


Figure 7: 3D path following environment. In the figures, from left to right and top to bottom, screenshots of a trained quadcopter's behavior in time (each taken 0.6 seconds apart from the previous). In the images, the quadcopter is represented by a cross and the path, composed by points connected by segments, is represented by spheres and lines. Segments yet to be completed are drawn in light gray color, while the segment joining the previous target to the current is highlighted in blue/violet color.

# 5 Numerical experiments

This section presents two types of experiments intended to prove the performance of the neural controller produced by the algorithm and also discuss the effect of neural network topology in both training effectiveness and outcome.

## 5.1 General configuration

The parameters used in the following numerical simulations are presented in Table 1.

| Parameter | Value | Parameter | Value | Parameter | Value |
|---|---|---|---|---|---|
| $\Delta t$ | $= 0.02$ | $b$ | $= 1.14 \times 10^{-7}$ | $k$ | $= 2.98 \times 10^{-6}$ |
| $A_x = A_y = A_z$ | $= 0.25$ | $m$ | $= 0.469$ | $l$ | $= 0.225$ |
| $I_{xx} = I_{yy}$ | $= 0.005$ | $I_{zz}$ | $= 0.006$ | $I_r$ | $= 0.006$ |
| $\omega_{\text{MAX}}$ | $= 1000$ | $\omega_{\text{MIN}}$ | $= 0$ | $r_{\text{MAX}}$ | $= 0.25$ |
| $\overline{n}$ | $= 0.25$ | $p$ | $= 150$ | $k_{\text{MAX}}$ | $= 10/\Delta t$ |
| $\boldsymbol{r}_e$ | $= 10$ | $g^*$ | $= 3$ | $n_T$ | $= 5$ |
| $n_P$ | $= 10$ | $m_r$ | $= 0.85$ | $\eta_{\text{MAX}}$ | $= \pi/3$ |

Table 1: Table of the parameters used in all of the following simulations. Upper rows: parameters regarding the quadcopter model and simulation; Lower rows: Genetic algorithm's parameters.

## 5.2 General performance and harshness configuration testing

The purpose of the tests presented in this section is to assess whether harsher training conditions generate more robust (less prone to crashing) and better performing controllers for autonomous flight. Also, the following tests demonstrate the performance of the controller in the task of following a trajectory, proving that – with further refining – neural controllers are a viable alternative to other methods of control.

The harshness of the training is determined by the maximum starting state $\boldsymbol{\sigma}(0)$ parameters $\overline{r_x}$ where $x$ is a sub-vector of the state $\boldsymbol{\sigma}$. Higher values in those parameters mean that the quadcopter will explore harder states to control, therefore evolution should favor individuals that can adapt to those situations. Every configuration refers to the same parameters values described in Table 1, and the starting state parameters $\overline{r_x}$ are presented in Table 2.

| Parameter | Configuration | | |
|---|---|---|---|
| | Harsh | Medium | Soft |
| $\overline{r_\eta}$ | 1.2 | 0.6 | 0.2 |
| $\overline{r_{\dot{\eta}}}$ | 1.5 | 0.75 | 0.1 |
| $\overline{r_{\dot{\xi}}}$ | 4 | 2 | 0.1 |
| $\overline{r_{\ddot{\xi}}}$ | 1 | 0.5 | 0.0 |

Table 2: Maximum random starting parameters of the harshness tests.

The structure of the neural network is the same for every configuration and the chosen topology is $\{15, 30, 16, 8, 4\}$.

The test begins by running the genetic algorithm for each configuration, for $g_{\text{MAX}} = 250,000$ generations, producing the trained population $\Gamma(g_{\text{MAX}})$. As a benchmark for the algorithm performance, to run 250,000 generations three times (one for each configuration), with the parameters in Table 1, it takes 8 hours on a dedicated server machine [Intel core i7 9xx (Nehalem class core i7) 2.30 GHz 8 core, 16 GB RAM], hence less than 3 hours to reach optimal results for a single configuration.

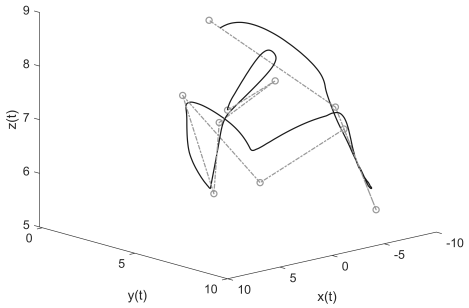
From the last generation of each configuration, the best individual (the one with the lowest cost) is chosen to participate in the path following tests. In these tests, the three individuals are given the same series of 1,000 randomly determined paths. Each path is composed of 10 random

points in a sequence and each point has a maximum distance of 10 meters from the previous point, to form a random spatial path.

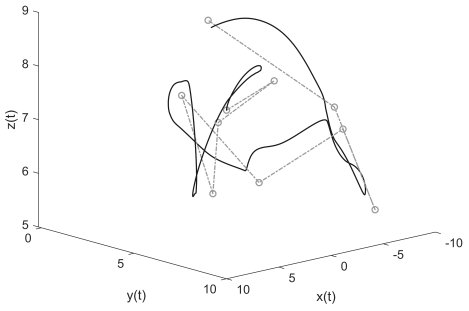
Since the controllers are trained to follow only one point, once the quadcopter's center of mass enters a spherical neighborhood of the current target point, with radius $\rho$, it switches to the next point in the sequence, thus approximately covering the whole path. As it will be shown in the following results, the choice of $\rho$ influences significantly the performances in the three configurations, and determines the precision of quadcopter's trajectory.

### 5.2.1 Approximate path following

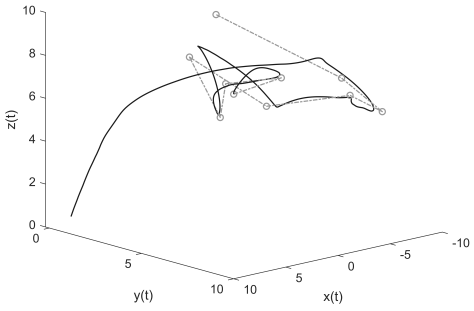
In these tests, the radius $\rho$ equals 1 meter. This means that when a quadcopter gets closer than one meter to the current target it will start to chase the next one in the sequence. With a radius $\rho = 1$ meter, the trajectory will be quite approximated hence the name of this test, so the precision in reaching the targets will be less favored compared to average speeds of flight.




Harsh




Medium




Soft

Figure 8: 3D trajectories obtained in an approximate ($\rho = 1$ meter) path following tests. The dashed line denotes the reference path, the continuous line denotes a drone trajectory.

Figure 8 shows a trajectory of the quadcopter over one of the randomly generated paths. A first qualitative analysis can be made: The harsh configuration shows no problem in following the path and its trajectory is more steady than the rest; the medium configuration also completes the path, but more jerky movements can be seen; the soft configuration initially starts following the path but eventually crashes to the ground.

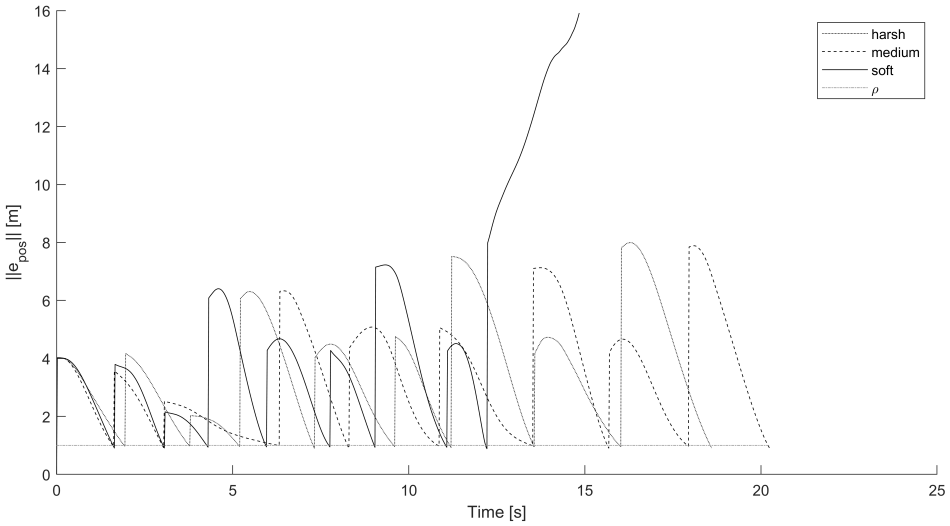



Figure 9: Norm of the error of position $\left\|\boldsymbol{e}_{\text{pos}}(t)\right\|$ of the three configurations, in the approximate ($\rho = 1$ meter) path following test, relative to the path points, over time.

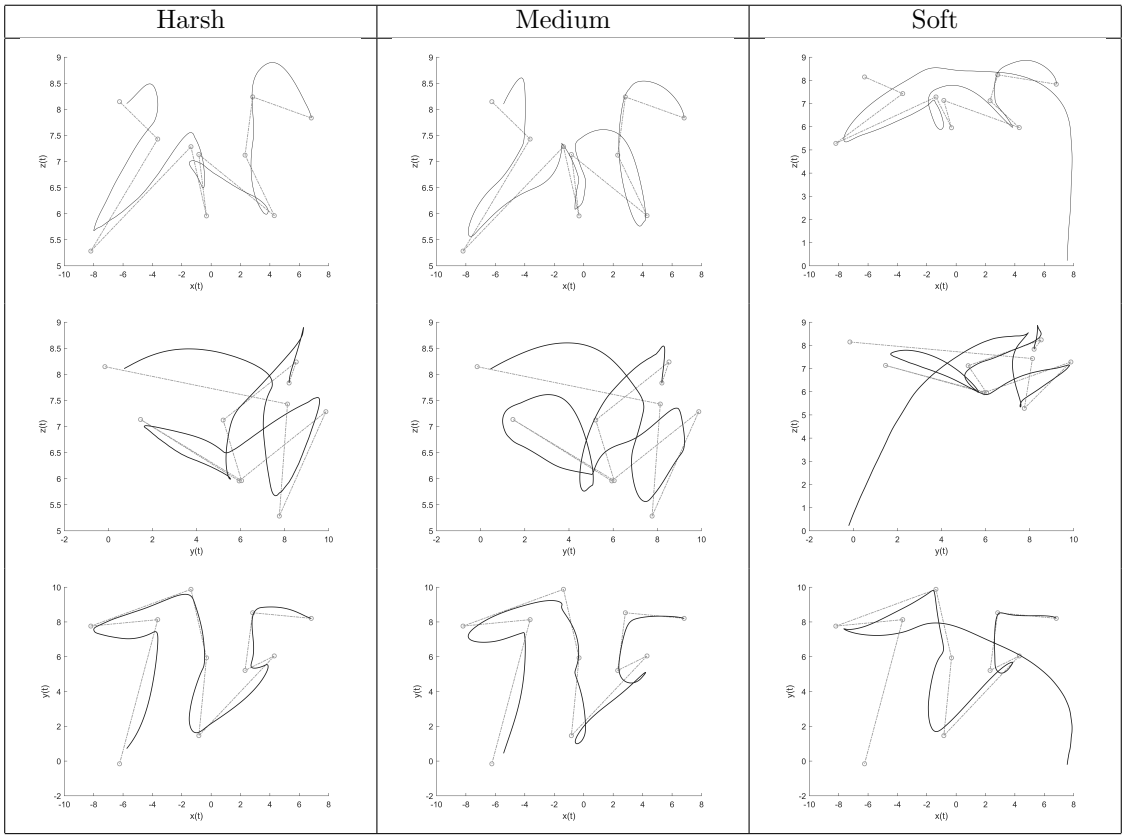



Figure 10: Projections of the 3D path in Figure 8. Upper row: x-z plane; Middle row: y-z plane; Lower row: x-y plane.

The trajectories taken by the three quadcopters can be seen in more detail in Figure 10, where the 3D path is projected on the 3 planes of the axes. Here it can be seen that the center of mass never quite reaches the trajectory's points, because of $\rho$ set to 1 meter. Also, especially in the x-y plane (third row), the difference in the 3 trajectories is more marked.

Note that in Figure 8 and Figure 10 the dashed trajectories are just the connection between successive target points and do not constitute a reference input for the quadcopter. Also, the dashed trajectory is not always guaranteed to be the optimal trajectory for a quadcopter to follow.

The harsh configuration exhibits smooth movements with less pronounced bumps, and its trajectory follows more closely the fastest path between the points. The medium configuration follows a more turbulent trajectory compared to the harsh configuration, but nevertheless reaches the end of the path. The soft configuration initially follows closely the path. Surprisingly, on the x-y plane, it even follows the path better the harsh configuration, that was expected to perform better than the other two configurations. Then, once it reaches a condition that it has not evolved to endure, the quadcopter crashes into the ground.

Since every controller of each configuration has to traverse the same 1,000 paths as the other, the time each quadcopter takes to traverse a path can be used as a reference of its performance; by averaging these 1,000 times, the average travel time $t_{\text{AVG}}$ can be determined. Also, the minimum $t_{\text{MIN}}$ and maximum $t_{\text{MAX}}$ travel times can be obtained. Note that not all trajectories reach the end of the path because either the quadcopter crashes into the ground or the controller is subject to a steady-state positional error $e_{\text{pos}}$, hence it hovers around the target without ever reaching the required neighborhood to switch to the next target (this situation is referred to as stall).

In these tests, the incomplete trajectories are only caused by crashes, since the spherical neighborhood is relatively wide ($\rho = 1$ meter) and the steady-state positional error should have been larger than 1 meter to get in a stall. This behavior can be seen in Figure 9.

| Configuration | $t_{\text{MIN}}$ | $t_{\text{AVG}}$ | $t_{\text{MAX}}$ | No. of crashes | No. of stalls |
|---|---|---|---|---|---|
| Harsh | 14.64 | 19.98 | 25.06 | 0 | 0 |
| Medium | 17.54 | 24.28 | 32.64 | 749 | 0 |
| Soft | 12.24 | 17.59 | 24.00 | 101 | 0 |

Table 3: Results' summary of 1,000 approximate ($\rho = 1$ meter) path-following tests.

The implications of the values shown in Table 3 may be evaluated as follows. The harsh quadcopter is the most stable of the three: with 1000 paths traversed it has never crashed or stalled once but, albeit stabler, is slower than the quadcopter evolved with softer starting states. Oddly, the soft configuration performs the fastest, with an average time of 17.59 seconds to perform a course of 100 meters (10 targets $\times$ 10 meters), means an average speed of 20.47 km/h. Also, the medium configuration values do not stand in between the harsh and the soft, with much higher crashes and slower times compared to the other configurations.

An interpretation of these results is:

- The harsh configuration produces stabler and more careful individuals that are not prone to crashing, at the cost of some travel speed in the path.
- The soft configuration produces faster and more reckless individuals, due to the training conditions that allow so, at the cost of occasional crashes where the quadcopter has visited a "harsh" state that was not evolved to endure.
- The medium configuration performs, both in speed and reliability, worse than the harsh and the soft configurations. This can be due to the fact that the training conditions are not harsh enough to produce reliable individuals, yet they are harsh enough to hinder the training process in evolving fast individuals.

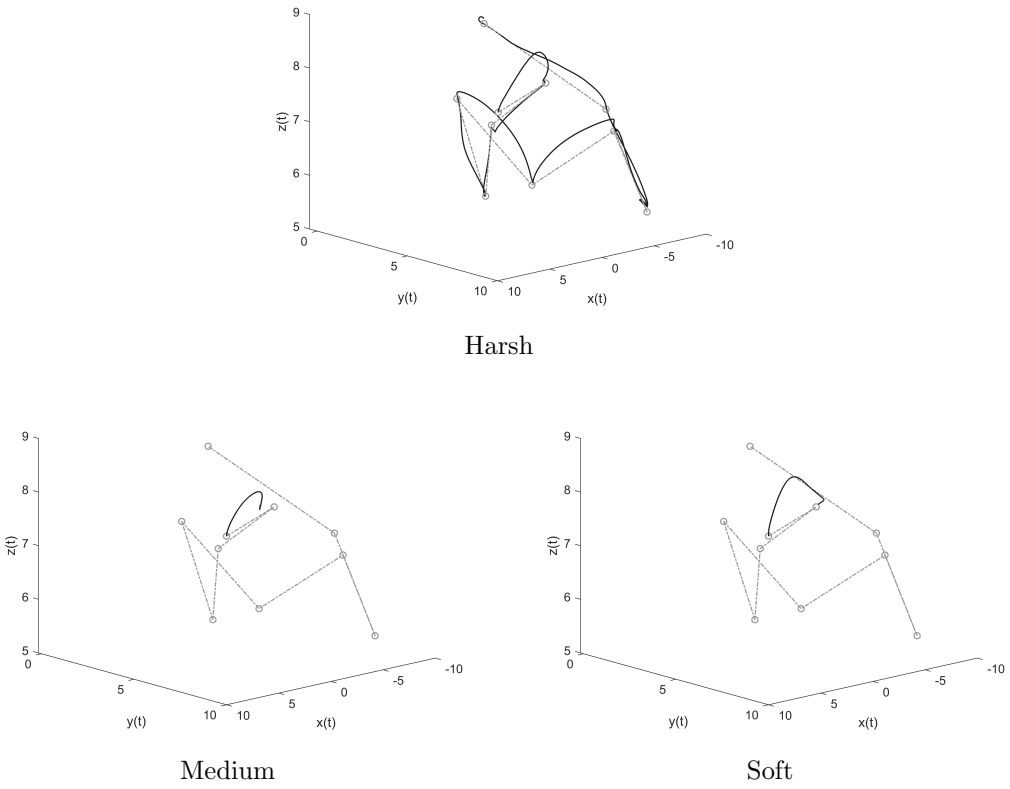
Since these tests have been run only on one training session of 250000 generations per configuration, to draw more precise conclusions more tests need to be run, thus gaining more statistical significance. Nevertheless, the data shows that with harsh training conditions, the genetic algorithm produces neural controllers that are both reliable and well performing in the task of approximate autonomous flight.

### 5.2.2 Precise path following

In order to gain more insight about the features of the considered learning configurations, the tests, with the same individuals of the previous tests and on the same exact paths, are run again with a

target-switch threshold $\rho$ equal to 0.1 meters. This means that the controllers will have to reach the current target (set point) more closely to start chasing the next one. To do so, and reach the end of the paths, stability in control is required to avoid crashing and a low steady state error in $\boldsymbol{e}_{\text{pos}}$ is mandatory to avoid stalling. Since the neural networks were already trained, the genetic algorithm does not need to be run again.

In contrast with the previous tests, precision in reaching the target points is more important than travel speed, so greater travel times are to be expected.




Harsh

Medium

Soft

Figure 11: 3D trajectories for the precise ($\rho = 0.1$ meter) path following tests. The dashed line refers to the reference path, the continuous line refers to the trajectory of a drone.

As it can be seen in Figure 11, the quadcopter trained with harsh conditions is more than capable of following the trajectory with precision (relatively to the target points), where the medium and soft training configurations fail to reach the end of the path because they stall, hovering around one point, without reaching a distance less than $\rho$ to the current target and never advancing towards the end of the sequence. These trajectories can be also seen projected on the three axes in Figure 12.

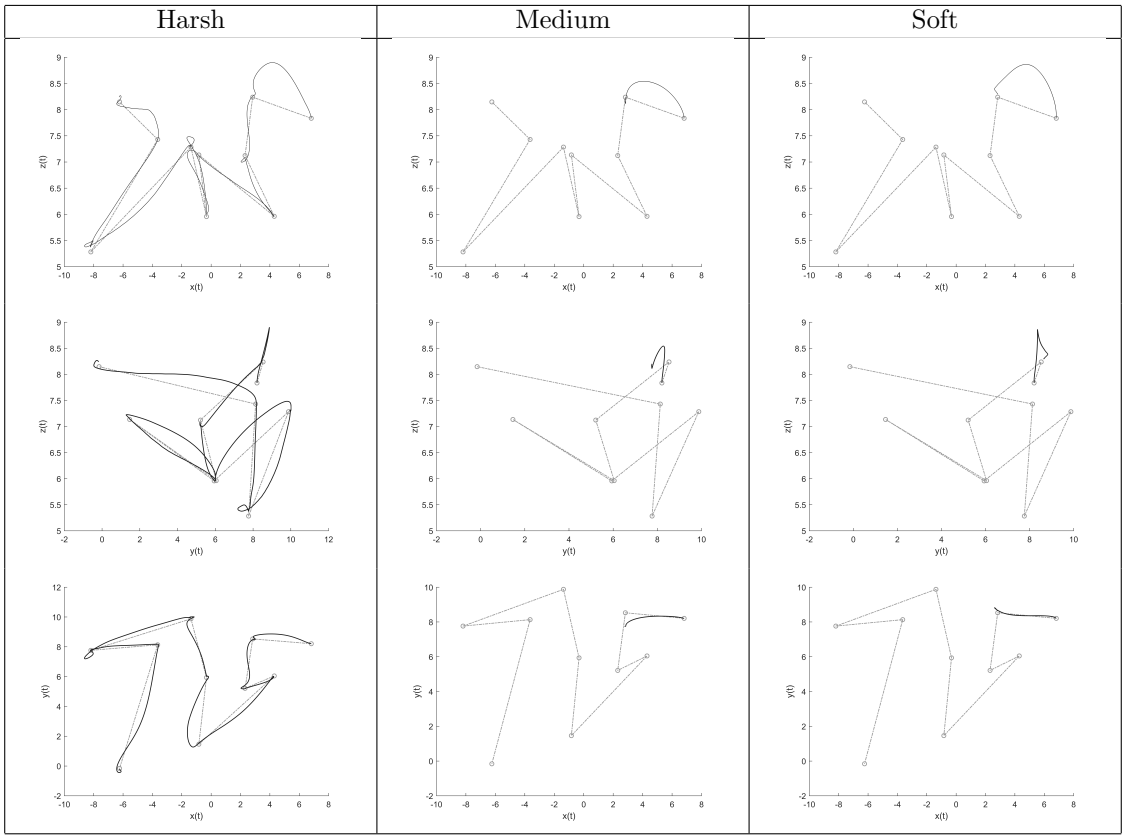



Figure 12: Projections of the 3D path in Figure 11. Upper row: x-z plane; Middle row: y-z plane; Lower row: x-y plane.

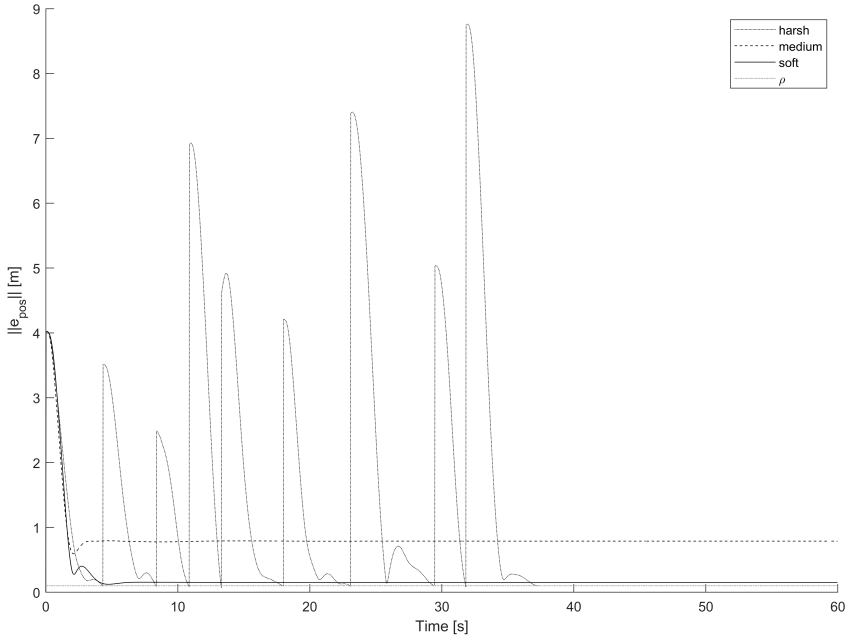



Figure 13: Norm of the error of position $\left\|\boldsymbol{e}_{\text{pos}}(t)\right\|$ of the three configurations, in the precise ($\rho = 0.1$ meters) path following test, relative to the path points, over time.

In Figure 13, the cause of stalling can be inspected more closely. A decrease in the positional error $\|\boldsymbol{e}_{\text{pos}}\|$ means that the individual gets closer to the current target, where a spike in the error means that the individual reached the spherical neighborhood of radius $\rho = 0.1$ meters and starts following the next point in the sequence. The error of the harsh configuration exhibits 9 spikes, which indicate that all of the 10 points in the path have been visited. The medium and soft individuals start chasing the first target point, but never reach an error $\|\boldsymbol{e}_{\text{pos}}\|$ less than $\rho$, thus these drones keep hovering around the first point without visiting any other points in the sequence. The medium individual exhibits a steady-state error of about 0.8 meters, while the soft individual shows a steady-state error of about 0.12 meters.

The travel times and number of incomplete paths (due to stalling and crashes) are presented in Table 4. By comparing these results with the ones in Table 3, the difference in the three configurations appears more evident. The harsh configuration shows no issues in completing both the approximated path with $\rho = 1$ meter and the more precise path with $\rho = 0.1$ meters, with an increase in travel times for the more precise path. The neural controllers produced by the medium and soft training configuration do not achieve a small enough steady-state error to even complete one path out of 1,000 paths.

| Configuration | $t_{\text{MIN}}$ | $t_{\text{AVG}}$ | $t_{\text{MAX}}$ | No. of crashes | No. of stalls |
|---|---|---|---|---|---|
| Harsh | 29.18 | 42.49 | 54.74 | 0 | 0 |
| Medium | - | - | - | 31 | 969 |
| Soft | - | - | - | 5 | 995 |

Table 4: Results' summary of 1,000 precise ($\rho = 0.1$ meters) path following tests.

The results of this test clearly show that harsh training conditions are required to produce controllers that are stable enough to avoid crashing, even at high speeds (shown in Table 3) and with steady-state errors low enough to follow paths more precisely (as shown in Table 4).

### 5.2.3 Further precision tests

Since the medium and soft training configuration results have been proven inadequate, to further test the precision of the neural controller produced by the harsh configuration, ulterior tests can be conducted. To evaluate the performance of the harsh configuration, a series of tests, each composed by 10,000 randomly generated paths with different settings, are realized. The obtained results are presented in Table 5.

| Path parameters | | | Travel times | | | Paths not completed | |
|---|---|---|---|---|---|---|---|
| $l$ | $\Delta l$ | $\rho$ | $t_{\text{MIN}}$ | $t_{\text{AVG}}$ | $t_{\text{MAX}}$ | No. of crashes | No. of stalls |
| 200 | 0.3 | 0.15 | 177.94 | 202.24 | 223.5 | 0 | 0 |
| 200 | 0.2 | 0.1 | 212.92 | 247.09 | 277.06 | 0 | 0 |
| 150 | 0.18 | 0.09 | 179.06 | 208.92 | 240.52 | 0 | 0 |
| 150 | 0.18 | 0.085 | 194.32 | 236.13 | 268.4 | 0 | 0 |
| 200 | 0.2 | 0.08 | - | - | - | 0 | 10,000 |
| 150 | 0.18 | 0.075 | - | - | - | 0 | 10,000 |
| 200 | 0.15 | 0.075 | - | - | - | 0 | 10,000 |

Table 5: Results obtained by an harsh configuration over 7 experiments, each composed of 10,000 paths with different randomly generated parameters $l$ and $\Delta l$.

Each row of Table 5 corresponds to the results of 10,000 paths randomly generated with specific parameters: $l$ represents the number of points in a path and $\Delta l$ is the maximum distance between consecutive points of the paths (in meters).

The results summarized in Table 5 indicate that for values of $\rho$ less than 0.08 meters the controller steady-state positional error is larger than the required threshold $\rho$ to advance the sequence of points in the path. This gives an approximate indication that the steady-state error $\|e_{\text{pos}}\|$ is not greater that 0.085 meters.

To further refine the precision of the controller, the definition of the cost function (18) guiding the genetic algorithm can be improved by increasing the weight of the cost term $C^g_{i,\xi}$ (that represents positional error) and, to follow a more precise path, a new cost term $C^g_{i,p}$ that would represent the discrepancy between the individual's trajectory and the segment conjoining the starting point to the target can be added. Also the genetic algorithm could be run for more generations, since the algorithm has been run for only 250,000 generations (3 hours of run time on the aforementioned computing platform).

Nevertheless, the neural controller produced by the genetic algorithm with the harsh configuration has reached adequate levels of precision, with a steady state error of 8 cm, and stability, with 0 crashes over the course of all the previous tests.

## 5.3 Topology experiments

The purpose of the following tests is to compare the effect that a network's topology has on both the performance of the training algorithm (costs compared to number of generations) and the resulting controller performance. Three configurations, differing only by topology, have been chosen:

- **Complex**: $\{15, 45, 60, 32, 8, 4\}$
- **Medium**: $\{15, 30, 16, 8, 4\}$
- **Simple**: $\{15, 20, 8, 4\}$

The chosen parameters to train the network were 200,000 total generations each composed by 100 individuals. These parameters produce less trained individuals than the parameters chosen in the previous tests (namely, 50,000 generations and 50 individuals less than the previous harshness tests, for a total of $50,000 \times 50$ fewer total individuals in the training), but were chosen to highlight the effects of neural network topology on training times instead of producing fully trained individuals. Also, the primary objective of this section is to qualitatively evaluate the effect that topology has on the training algorithm, but some path following tests have also been performed to give an approximate evaluation on the performance of the evolved individuals, albeit not fully trained.

To gather information on the training process, in each configuration's generation the minimum cost of the entire population is recorded. Since the starting state resets every $g^* = 3$ generations, the costs exhibit large fluctuations. Notice that the starting states $\boldsymbol{\sigma}(0)$ are randomly determined, and the starting states that each configuration's training session endures are independent from the starting states of the other two configurations. To increase the readability of Figure 14, the values reported in the plots are the moving average (MA) of each generation's minimum costs, within a window of 2,000 generations.

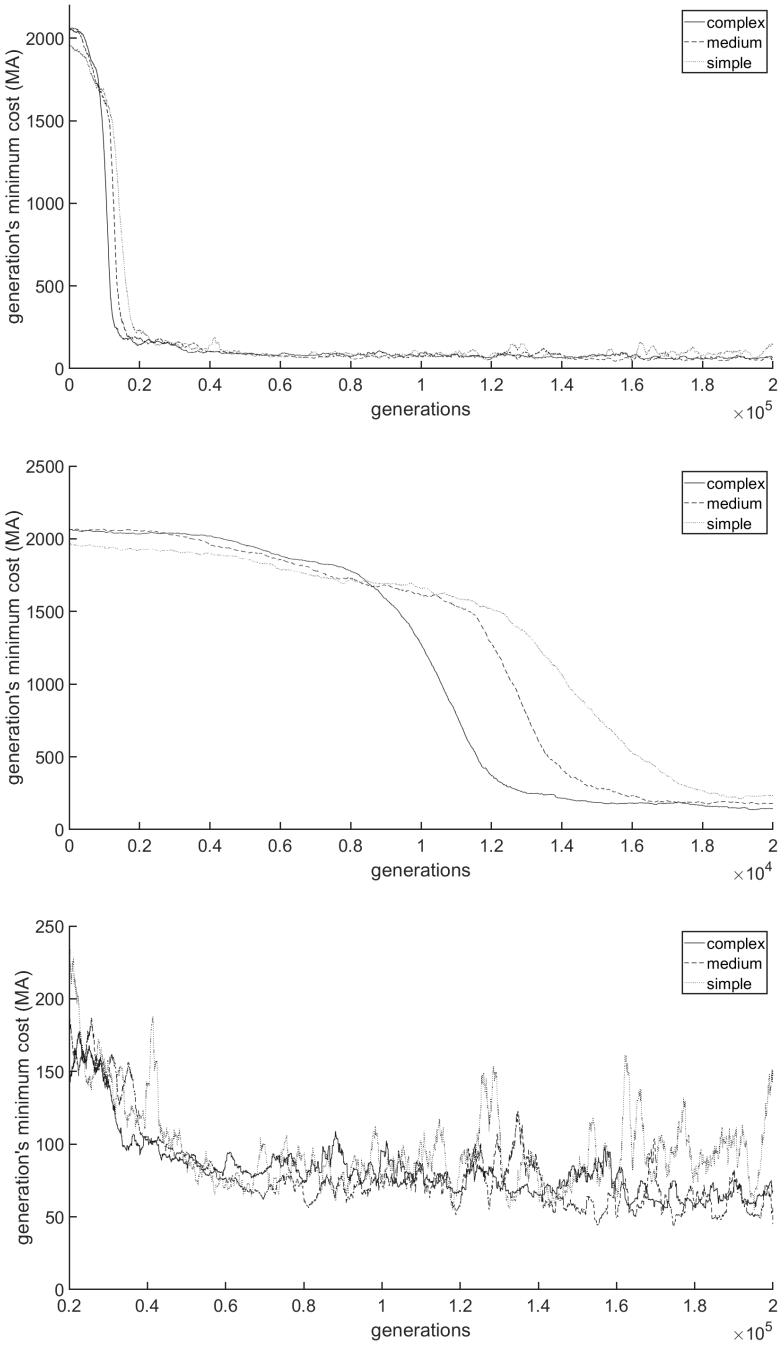



Figure 14: Moving average over each generation's minimum costs of each topology configuration, within a window of 2,000 generations. The curves represent, from top to bottom, the values for the entire training (200,000 generations), the values at the start of the training (from generation 0 to 20,000) and in the remaining generations (from 20,000 to 200,000).

The curves in Figure 14 show that initially the simple topology learns the quickest (lowest decreasing cost over each generation), the complex is the slowest and the medium learning performance is in between the two. After the initial learning phase, the simple topology exhibits, on average, the highest costs, while the other configurations show lower costs. Between the medium and complex topology, the best topology cannot be determined.

To evaluate the performance of the trained controllers, each controller is evaluated on a series of path-following tests, each composed by 500 paths with different parameters, as shown in Table 6.

| Path parameters | | | Topology | Travel times | | | Paths not completed | |
|---|---|---|---|---|---|---|---|---|
| $l$ | $\Delta l$ | $\rho$ | | $t_{\text{MIN}}$ | $t_{\text{AVG}}$ | $t_{\text{MAX}}$ | No. of crashes | No. of stalls |
| 20 | 10 | 1 | Complex | 56.80 | 71.93 | 86.72 | 27 | 0 |
| | | | Medium | 41.08 | 52.41 | 60.64 | 8 | 0 |
| | | | Simple | - | - | - | 500 | 0 |
| 20 | 10 | 0.5 | Complex | - | - | - | 6 | 494 |
| | | | Medium | 54.26 | 67.98 | 82.26 | 3 | 0 |
| | | | Simple | - | - | - | 500 | 0 |
| 100 | 0.2 | 0.1 | Complex | - | - | - | 0 | 500 |
| | | | Medium | - | - | - | 0 | 500 |
| | | | Simple | - | - | - | 0 | 500 |

Table 6: Results of a series of tests, each composed by 500 randomly generated paths with different configurations, run by the final best individuals produced by the genetic algorithm, for each of the three topology.

These values show that in approximately following a path ($\rho = 1$ meter) the neural network with the medium-complexity topology performs better, both in terms of travel time and stability, than the other two instances of topology. The neural network with the simple topology cannot complete a single test without crashing or stalling, over the course of 1,500 ($500 \times 3$) path-following tests. The most complex of the three topologies in the approximate tests ($\rho = 1$ meter) performs adequately, but in the tests where the quadcopter must get closer to the path's target-points ($\rho = 0.5$ meters and $\rho = 0.1$ meters), the controller is not precise enough to complete the paths reliably. This suggests that the topology of a neural controller must be chosen carefully: A complex topology (both in depth and width) requires more training time and does not entail necessarily an optimal performance; in contrast, a simpler topology requires less training time but does not possess enough capabilities for learning more complex tasks nor for learning the correct behavior for the states visited in training. So a balance between training speed and learning potential must be established by specifying a correct neural network topology.

# 6 Conclusions

This thesis proves that neural networks are a feasible approach in controlling quadcopters systems. Also, the potentialities of genetic algorithms as a method for neural network's unsupervised learning (neuro evolutionary algorithms) have been shown: by defining an appropriate cost function for the task and by tweaking the algorithm's parameters, neuro evolutionary techniques are able to train well-performing neural networks in a relatively short amount of time. This learning algorithm, applied to quadcopter control, yields impressive results: the neural controllers performed the task of autonomous flight with outstanding stability and excellent flight times in a simulated environment with no obstacles.

To further extend this work and make this form of control more viable in real world applications, the quadcopter can be equipped with an array of sensors for collision avoidance and its structure and learning processes modified for accounting the supplementary inputs. Also the training environment can be ameliorated by adding disturbances such as wind or sensor noise to further improve the robustness and stability of the controllers. The applications of this form of controller and training can also be extended to a vast amount of necessities. For example, to adapt the controller for logistic purposes such as autonomous package delivery, the simulation can be expanded by adding a variable payload to the quadcopter so that the neuro evolutionary algorithm produces neural networks capable of adapting to deliver any type of reasonable cargo.

# References


[1] L. M. Argentim, W. C. Rezende, P. E. Santos, and R. A. Aguiar. PID, LQR and LQR-PID on a quadcopter platform. In *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 1–6, 2013.

[2] T. Bäck. Optimal mutation rates in genetic search. In *Proceedings of the fifth international conference on genetic algorithms*, 1993.

[3] Y. Bai and S. Gururajan. Evaluation of a baseline controller for autonomous "figure-8" flights of a morphing geometry quadcopter: Flight performance. *Drones*, 3(3), 2019.

[4] J.G.A. Barbedo. A review on the use of unmanned aerial vehicles and imaging sensors for monitoring and assessing plant stresses. *Drones*, 3(2), Apr 2019.

[5] L. Besnard, Y.B. Shtessel, and B. Landrum. Quadrotor vehicle control via sliding mode controller driven by sliding mode disturbance observer. *Journal of the Franklin Institute*, 349(2):658 – 684, 2012.

[6] H. Bolandi, M. Rezaei, R. Mohsenipour, H. Nemati, and S.M. Smailzadeh. Attitude control of a quadrotor with optimized PID controller. *Intelligent Control and Automation*, 4(3):335 – 342, 2013.

[7] W. Budiharto, A. Chowanda, A. A. S. Gunawan, E. Irwansyah, and J. S. Suroso. A review and progress of research on autonomous drone in agriculture, delivering items and geographical information systems (gis). In *2019 2nd World Symposium on Communication Engineering (WSCE)*, pages 205–209, 2019.

[8] T. Bäck and F. Hoffmeister. Extended selection mechanisms in genetic algorithms. In *Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991*, pages 92–99. Morgan Kaufmann, 1991.

[9] T. Dierks and S. Jagannathan. Neural network control of quadrotor uav formations. In *2009 American Control Conference*, pages 2990–2996, 2009.

[10] B. Erginer and E. Altug. Modeling and PD control of a quadrotor VTOL vehicle. In *2007 IEEE Intelligent Vehicles Symposium*, pages 894 – 899. IEEE, 2007.

[11] S. Grzonka, G. Grisetti, and W. Burgard. A fully autonomous indoor quadrotor. *IEEE Transactions on Robotics*, 28(1):90 – 100, 2011.

[12] M. Hassanalian, D. Rice, and A. Abdelkefi. Evolution of space drones for planetary exploration: A review. *Progress in Aerospace Sciences*, 97:61 – 105, 2018.

[13] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, volume 87 of *Proceedings of Machine Learning Research*, pages 133 – 145. PMLR, 2018.

[14] N.O. Lambert, D.S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K.S.J. Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224 – 4230, 2019.

[15] S. Lee and Y. Choi. Reviews of unmanned aerial vehicle (drone) technology trends and its applications in the mining industry. *Geosystem Engineering*, 19(4):197 – 204, 2016.

[16] J. Li and Y. Li. Dynamic analysis and PID control for a quadrotor. In *2011 IEEE International Conference on Mechatronics and Automation*, pages 573 – 578. IEEE, 2011.

[17] T. Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22, 2011.

[18] B.L. Miller and D.E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3):193 – 212, 1995.

[19] K. C. Morris, C. Schlenoff, and V. Srinivasan. Guest editorial: A remarkable resurgence of artificial intelligence and its impact on automation and autonomy. *IEEE Transactions on Automation Science and Engineering*, 14(2):407 – 409, 2017.

[20] R.E. Perez, J. Arnal, and P.W. Jansen. Neuro-evolutionary control for optimal dynamic soaring. In *Proceedings of the AIAA Scitech 2020 Forum, Orlando, FL (USA)*, 2020.

[21] H.A. Pham, T. Soriano, V.H. Ngo, and V. Gies. Distributed adaptive neural network control applied to a formation tracking of a group of low-cost underwater drones in hazardous environments. *Applied Sciences*, 10(5), 2020.

[22] T.H. Pham, D. Ichalal, and S. Mammar. LPV and nonlinear-based control of an autonomous quadcopter under variations of mass and moment of inertia. *IFAC-PapersOnLine*, 52(28):176 – 183, 2019. 3rd IFAC Workshop on Linear Parameter Varying Systems LPVS 2019.

[23] P.W. Poon and J.N. Carter. Genetic algorithm crossover operators for ordering applications. *Computers & Operations Research*, 22(1):135 – 147, 1995.

[24] U.M. Rao Mogili and B.B.V.L. Deepak. Review on application of drone systems in precision agriculture. *Procedia Computer Science*, 133:502 – 509, 2018. International Conference on Robotics and Smart Manufacturing (RoSMa2018).

[25] A. Rodić, G. Mester, and I. Stojković. Qualitative evaluation of flight controller performances for autonomous quadrotors. In *Intelligent Systems: Models and Applications*, pages 115 – 134. Springer, 2013.

[26] A.-W.A. Saif, M. Dhaifullah, M. Al-Malki, and M. El Shafie. Modified backstepping control of quadrotor. In *International Multi-Conference on Systems, Signals & Devices*, pages 1 – 6. IEEE, 2012.

[27] A. L. Salih, M. Moghavvemi, H. A. F. Mohamed, and K. S. Gaeid. Modelling and PID controller design for a quadrotor unmanned air vehicle. In *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, volume 1, pages 1–5, 2010.

[28] J.F. Shepherd and K. Tumer. Robust neuro-control for a micro quadrotor. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, page 1131 – 1138, New York, NY, USA, 2010. Association for Computing Machinery.

[29] I. Siti, M. Mjahed, H. Ayad, and A. El Kari. New trajectory tracking approach for a quadcopter using genetic algorithm and reference model methods. *Applied Sciences*, 9(9), 2019.

[30] M. Srinivas and L.M. Patnaik. Learning neural network weights using genetic algorithms-improving performance by search-space reduction. In *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, volume 3, pages 2331 – 2336, 1991.

[31] K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99 – 127, 2002.

[32] G. Szafranski and R. Czyba. Different approaches of PID control UAV type quadrotor. In *Proceedings of the International Micro Air Vehicles conference 2011 summer edition*, pages 70 – 75, 2011.

[33] M. Črepinšek, S.-H. Liu, and M. Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.*, 45(3), 2013.

[34] D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65 – 85, 1994.