Control of a Quadrotor Using Generated ANN With NEAT Algorithm

Montreal, December 5, 2016

By: Maxence Boutet (260757323)
Simon Ho (260479710)
Presented to: Pr. Jeremy R. Cooperstock, PhD

Remark: Without the figures, graphs and appendices, the main body is no longer than five pages as required by the project specification.

I. Introduction

The development of autonomous rotorcraft with the capability of stabilization has generated interest from researchers as the applications are quite desirable in UAV drone inspection, surveillance in military and commercial areas. According to [6], a "quadcopter, also known as quadrotor, is a helicopter with four rotors as illustrated in fig. 1. The rotors are directed upwards and they are placed in a square formation with equal distance from the center of mass of the quadcopter." With 6 degrees of freedom, translational (x,y,z) and rotational (yaw ψ, pitch θ, roll ϕ) and four independent inputs (four rotors) as seen in figs. 1, 2a and 2b, the resulting dynamics are highly nonlinear when accounting for aerodynamics effects. As a result, developing and controlling a quadcopter is a fundamentally difficult and complex task. Many modeling of the movement and control of the quadcopter has been done in the past with linear methods such as PID, LQR and nonlinear methods such as backstepping, SMC and feedback linearization.

Since a solution cannot be solved easily without spending a lot of computational resources and developing complex mathematical expressions using these previous techniques, the aim of this research is control the nonlinear system with a genetic algorithm implementation called NEAT (NeuroEvolution of Augmenting Topologies) by Stanley and Miikkulainen [12]. The reasoning is that in general, due to their random nature, evolutionary algorithms can't find an optimal solution but will often find a good or sufficient solution. This project seeks to observe the relevance of outputs obtained from neural network controllers. It's worth mentioning that it is a highly experimental approach.

Genetic algorithms possesses three basic operators: selection, crossover and mutation. By starting iterations with an initial population with each member of this population evaluated through a fitness function, we hope to find interesting system models to stabilize the aircraft. The purpose of this paper is to correct the behavior of the quadcopter around the 6 degrees of freedom using a NEAT generated ANN (Artificial Neural Network). This ANN will receive the UAV's state as well as the desired value for the 6 degrees of freedom and will output four voltages to control the four rotors. The MATLAB numerical computing environment is used to perform the simulations with the MATLAB NEAT package written by Mayr [7] as a base project. Several modifications with the parameters and functionality of the algorithm are made to optimize and cater to our system specifications.

Figure 1: UAV drone quadcopter commercial model. (image source: [9])

II. System Modelling

In this section, the 6 degrees of freedom (6DOF) nonlinear dynamics of the multicopter is presented.

A. Frames of Reference

Three frames of reference (FOR) are defined. The first is the tangent NED frame (North-East-Down) denoted by $\{\mathsf{t}\}$. This frame has its origin O_t fixed with respect to (wrt) the surface of the Earth [5, p.10]. This FOR is shown in green in fig. 2c. The second FOR is the local NED frame $\{\mathsf{n}\}$ that has the same orientation as $\{\mathsf{t}\}$ excepted its origin O_n is attached to the multicopter's CG. It is shown in blue in fig. 2b. The last FOR is the body frame $\{\mathsf{b}\}$ shown in fig. 2a. Its origin P coincides with the multicopter's CG.

B. Motion Equations

1. Translational Motion

According to [3], the 6DOF translational dynamics is described by:

$$m\left({}^{\mathsf{b}}\dot{\mathbf{v}}^{\mathsf{b}}_{P/\mathsf{t}} + \boldsymbol{\omega}^{\mathsf{b}}_{\mathsf{b}/\mathsf{t}} \times \mathbf{v}^{\mathsf{b}}_{P/\mathsf{t}}\right) = \mathbf{F}^{\mathsf{b}}_{b} \tag{1}$$

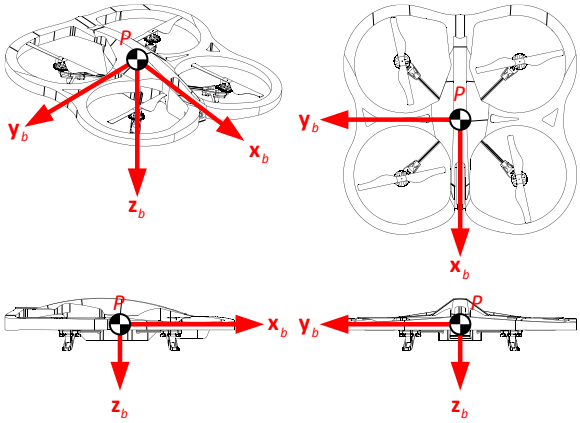
where:

- m is the multicopter's mass.
- $\mathbf{v}^{\mathsf{b}}_{P/\mathsf{t}} = [\,u\ v\ w\,]^\top$ is the velocity of P wrt $\{\mathsf{t}\}$ in $\{\mathsf{b}\}$.
- ${}^{\mathsf{b}}\dot{\mathbf{v}}^{\mathsf{b}}_{P/\mathsf{t}} = [\,\dot{u}\ \dot{v}\ \dot{w}\,]^\top$ is the acceleration of P wrt $\{\mathsf{t}\}$ in $\{\mathsf{b}\}$.
- $\boldsymbol{\omega}^{\mathsf{b}}_{\mathsf{b}/\mathsf{t}} = [\,p\ q\ r\,]^\top$ is the angular velocity of $\{\mathsf{b}\}$ wrt $\{\mathsf{t}\}$ in $\{\mathsf{b}\}$.
- $\mathbf{F}^{\mathsf{b}}_{b} = [\,F^{\mathsf{b}}_{b,x}\ F^{\mathsf{b}}_{b,y}\ F^{\mathsf{b}}_{b,z}\,]^\top$ is the force applied on P in $\{\mathsf{b}\}$.

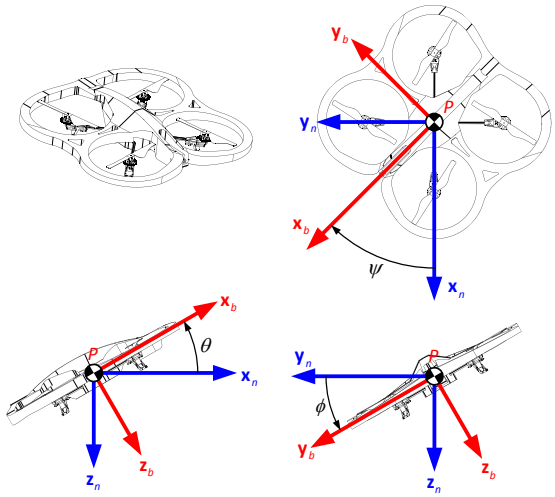
The UAV's velocity expressed in frame $\{\mathsf{n}\}$ is given by:

$$\mathbf{v}^{\mathsf{n}}_{P/\mathsf{t}} = \left(\mathbf{R}^{\mathsf{b}}_{\mathsf{n}}\right)^\top \mathbf{v}^{\mathsf{b}}_{P/\mathsf{t}} = [\,\dot{x}\ \dot{y}\ \dot{z}\,]^\top \tag{2}$$

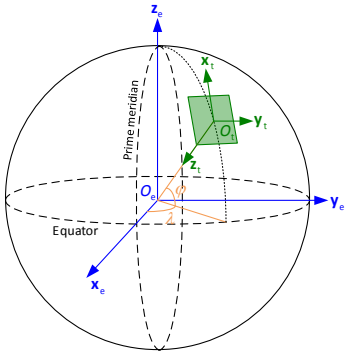
where:



(a) Body frame {b}



(b) Local NED Frame {n} (blue), frame {b} (red) and Euler's angles ψ, θ, ϕ



(c) Tangent NED frame {t} (green). Frame {e} corresponds to the ECEF frame and φ, λ, to the latitude and longitude. However, frame {t} is considered inertial in this document [3] making {e} unnecessary

Figure 2: Body frame, local NED frame and tangent NED frame

- $\mathbf{R}_{\mathsf{n}}^{\mathsf{b}} = \begin{bmatrix} \mathrm{c}_\theta \mathrm{c}_\psi & \mathrm{c}_\theta \mathrm{s}_\psi & -\mathrm{s}_\theta \\ -\mathrm{c}_\phi \mathrm{s}_\psi + \mathrm{s}_\phi \mathrm{s}_\theta \mathrm{c}_\psi & \mathrm{c}_\phi \mathrm{c}_\psi + \mathrm{s}_\phi \mathrm{s}_\theta \mathrm{s}_\psi & \mathrm{s}_\phi \mathrm{c}_\theta \\ \mathrm{s}_\phi \mathrm{s}_\psi + \mathrm{c}_\phi \mathrm{s}_\theta \mathrm{c}_\psi & -\mathrm{s}_\phi \mathrm{c}_\psi + \mathrm{c}_\phi \mathrm{s}_\theta \mathrm{s}_\psi & \mathrm{c}_\phi \mathrm{c}_\theta \end{bmatrix}$ is the rotation matrix from body frame to local NED frame.
- $\dot{x}$, $\dot{y}$, $\dot{z}$ are the UAV's velocities expressed in {n}.

2. Rotational Motion

According to [3], the 6DOF rotational motion is described by:

$$\mathbf{I}^{\mathsf{b}}\dot{\boldsymbol{\omega}}_{\mathsf{b/t}}^{\mathsf{b}} + \boldsymbol{\omega}_{\mathsf{b/t}}^{\mathsf{b}} \times \left(\mathbf{I}\boldsymbol{\omega}_{\mathsf{b/t}}^{\mathsf{b}}\right) = \mathbf{M}_b^{\mathsf{b}} \tag{3a}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \boldsymbol{\omega}_{\mathsf{b/t}}^{\mathsf{b}} \tag{3b}$$

where:

- $\mathbf{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$ is the inertia tensor wrt P.
- ${}^{\mathsf{b}}\dot{\boldsymbol{\omega}}_{\mathsf{b/t}}^{\mathsf{b}} = [\,\dot{p}\ \dot{q}\ \dot{r}\,]^\top$ is the angular acceleration of {b} wrt {t} in {b}.
- ψ, θ, ϕ are the Euler's angles of {b} wrt {n}.
- $\dot{\psi}$, $\dot{\theta}$, $\dot{\phi}$ are the rate of change of Euler's angles.
- $\mathbf{M}_b^{\mathsf{b}} = [\,L\ M\ N\,]^\top$ is the applied moment on P expressed in {b}.

C. DC Motors and Propellers

According to [10, p.48], the DC motor and propeller are modeled as follows (for $i = 1 \dots n_{\text{mot}}$):

$$\frac{d\mathbf{i}_a}{dt} = L_a^{-1}\left(\mathbf{V}_a - k_e\boldsymbol{\omega} - R_a\mathbf{i}_a\right) \tag{4a}$$

$$\dot{\boldsymbol{\omega}} = J_m^{-1}\left(k_m\mathbf{i}_a - B_m\boldsymbol{\omega} - k_D\boldsymbol{\omega}^2\right) \tag{4b}$$

when $L_a \neq 0$ and:

$$\dot{\boldsymbol{\omega}} = J_m^{-1}\left(k_m R_a^{-1}\left(\mathbf{V}_a - k_e\boldsymbol{\omega}\right) - B_m\boldsymbol{\omega} - k_D\boldsymbol{\omega}^2\right) \tag{5}$$

when L_a is neglected. The vectors $\mathbf{i}_a$, $\boldsymbol{\omega}$, $\dot{\boldsymbol{\omega}}$ and $\mathbf{V}_a$ are used for compactness. For instance, $\mathbf{i}_a = [\,i_{a,1}\ \cdots\ i_{a,i}\ \cdots\ i_{a,n_{\text{mot}}}\,]^\top$. The parameters are:

- V_a is the DC motor's input voltage in V (Volt).
- i_a is the armature circuit's current in A (Ampere).
- ω is the angular speed of the rotor in rad/s.
- R_a is the circuit's electric resistance in Ω (Ohms).
- L_a is the circuit's electric inductance in H (Henry).
- k_e is the motor's electromotive force constant in V/rad/s.
- k_m is the motor's torque constant in N.m/A.
- k_D is the propeller's drag parameter in N.m/rad^2/s^2.
- J_m is the rotor and propeller's inertia in kg.m^2.
- B_m is the viscous friction coefficient in N.m/rad/s.

D. Applied Forces and Moments

1. Applied Forces

The total applied force is given by [11, p.30][10, p.50]:

$$\mathbf{F}_b^{\mathsf{b}} = m\mathbf{R}_{\mathsf{n}}^{\mathsf{b}}\mathbf{g}^{\mathsf{n}} + \textstyle\sum_{i=1}^{n_{\text{mot}}}\mathbf{F}_i^{\mathsf{b}} \quad \mathbf{F}_i^{\mathsf{b}} = [\,0\ 0\ -k_T\omega_i^2\,]^\top \tag{6}$$

where:

- $\mathbf{g}^{\mathsf{n}} = [\,0\ 0\ g\,]^\top$ is the gravity vector in {n}.
- k_T is the propeller's thrust coefficient in N/rad^2/s^2.
- n_{mot} is the number of motors.
- ω_i is the angular speed of the i^{th} propeller.

In eq. (6), the term $m\mathbf{R}_{\mathsf{n}}^{\mathsf{b}}\mathbf{g}^{\mathsf{n}}$ corresponds to the multicopter's weight wrt {b} whereas the summation term is the thrust produced by the propellers.

2. Applied Moments

The total applied moment wrt $\{\mathsf{b}\}$ is given by [10, pp.49,51-54]:

$$\mathbf{M}_b^{\mathsf{b}} = \sum_{i=1}^{n_{\text{mot}}} \mathbf{PP}_i^{\mathsf{b}} \times \mathbf{F}_i^{\mathsf{b}} + \sum_{i=1}^{n_{\text{mot}}} \boldsymbol{\tau}_i^{\mathsf{b}} + \sum_{i=1}^{n_{\text{mot}}} \mathbf{G}_i^{\mathsf{b}} \tag{7a}$$

with:

$$\boldsymbol{\tau}_i^{\mathsf{b}} = \begin{bmatrix} 0 & 0 & (-1)^{i-1}\tau_i \end{bmatrix}^\top \qquad \tau_i = k_D\omega_i^2 + J_m\dot{\omega}_i \tag{7b}$$

$$\mathbf{G}_i^{\mathsf{b}} = \boldsymbol{\sigma}_i^{\mathsf{b}} \times \boldsymbol{\omega}_{\mathsf{b/t}}^{\mathsf{b}} \qquad \boldsymbol{\sigma}_i^{\mathsf{b}} = \begin{bmatrix} 0 & 0 & (-1)^{i}\omega_i \end{bmatrix}^\top \tag{7c}$$

where $\mathbf{PP}_i^{\mathsf{b}}$ is the vector from point P to motor i expressed in meter (m). The first summation corresponds to the thrust's generated moments, the second, to the induced moments that are opposite of the propeller's rotation as shown in fig. 3. The last summation are the gyroscopic moments.

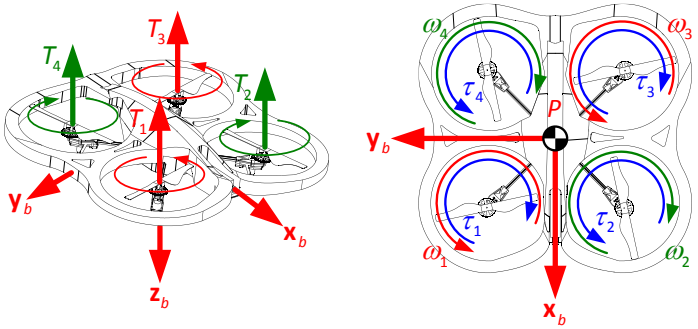
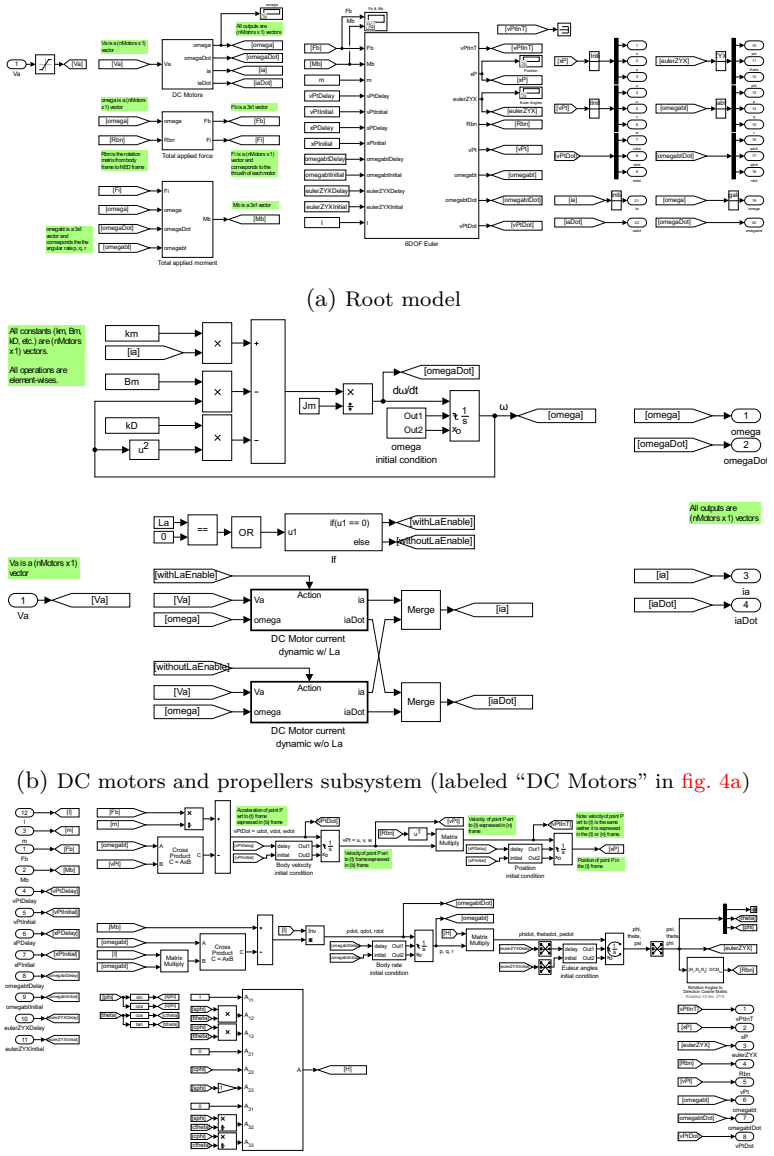


Figure 3: Thrust $T_i = k_T\omega_i^2$ and induced moments τ_i

E. Simulink Implementation

The nonlinear dynamics presented in the preceding sections is implemented in Simulink as shown in fig. 4.



(a) Root model

(b) DC motors and propellers subsystem (labeled "DC Motors" in fig. 4a)

(c) 6DOF motion equations subsystem (labeled "6DOF Euler" in fig. 4a)

Figure 4: Simulink implementation of the nonlinear dynamics

III. NEAT Algorithm

The control of quadrotor is evaluated using ANN. According to [1], ANN are "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." Simply put, the algorithm emulates the brain neuronal structure with neurons connected with each others and forming links. Neurons can be inputs, outputs or hidden. In our case, the quadrotor state (positions, velocities, attitude, desired trajectories, etc.) represents the ANN's input, and the four rotor's voltages are the output.

Neuroevolution (NE), artificial evolution of neural networks uses genetic algorithm to train neural networks. In traditional NE, a fixed topology evolution is chosen and searches through the connection weights and selects the best networks to breed them for the next generation. In this case, both the topology and weights are evolved. One of the main issue of neural networks is building a topology and evaluate parameters such as the number of hidden layers, neurons in each layer and connections between neurons.

Many systems have been developed throughout the years in neuroevolution. The method for encoding networks using an efficient genetic representation is the most important factor. In binary encoding for example, which was developed in 1992, a bit string represents the connection matrix of a network. However, it has limitations since the size of the connection matrix is the square of number of nodes, thus the representation is huge and second, the linear bit of strings which represents a graph of structure don't make any relevant combinations when doing crossovers. In graph encoding, a dual representation is used with a graph structure and a linear genome of node definitions for ingoing, outgoing connections. The main idea is that representations should reflect different kinds of operators. However, this method is also limited by the number of nodes in the network, i.e. the number of nodes in the 2D matrix of the graph structure of the genome. Other researchers have even resorted to give up crossover since they evaluated that the "the prospect of evolving connectionist networks with crossover appears limited in general." [12]

In all these previous encodings, the crossover rarely yielded any useful combinations. There are multiple ways to find a solution to a weight optimization problem with a neural network. Genomes with the same solution have different encodings and crossover is likely to damage future generations.

In 2002, [12] the neuroevolution of augmenting topologies (NEAT) algorithm was developed. NEAT unique feature in its encoding is

to crossover genes with similar features thanks to the innovation number which will be explained in the next section. What it does is search both through the connection space and neural network weight. NEAT changes the neurons weights and network topologies to find a good pool of fitness solutions. This allows networks to be crossed over with each other and apply speciation to keep certain innovations. These concepts will be explored in the next section. NEAT has these main characteristics [12]:

- Have a genetic representation for network topologies to be crossed over in a significative way.
- Protecting topologies innovations which would normally be removed even though additional generations would have resulted in optimized performance.
- Evolving structure such that topologies are minimized and grown incrementally.

What Stanley and Miikkulainen found was a significant performance increase: 5x faster than fixed topology evolution.

NEAT's genetic encoding scheme is formulated with a genome containing node genes and connection genes. The node genes can be represented as inputs, outputs or hidden (for calculations). The connection genes join nodes to form a network. Figure 5 displays multiple information for the connection genes: input & output node, weight, enable bit and an innovation number. This number tracks the history of the nodes, with 2 corresponding innovation numbers representing the same network structure.

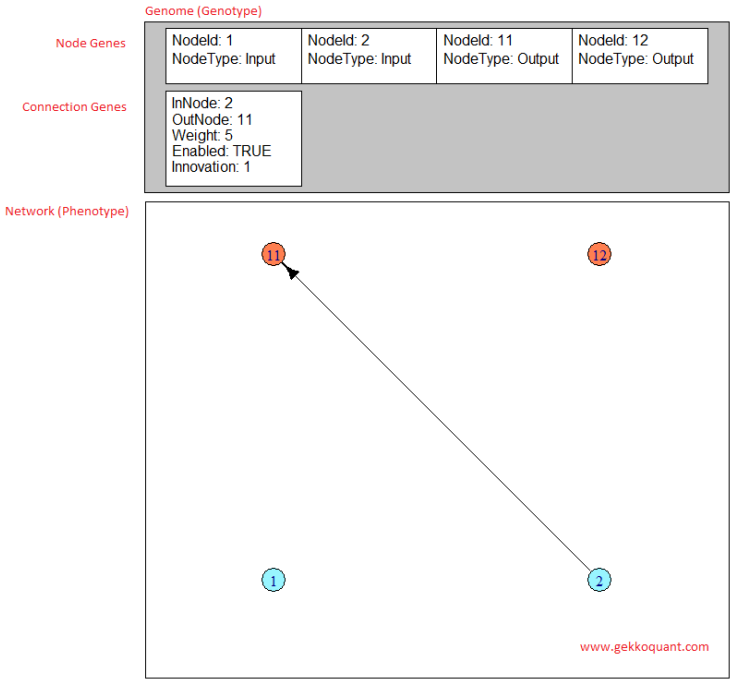


Figure 5: NEAT genome composition with node and connection genes (image source: [2])

When a mutation happens with a certain probability, the connection weights and network topologies can change. Random links and nodes can be added, enabled and disabled in the network. Throughout generations, the genomes get more complex and bigger. These genomes all have different characteristics in size and complexity. As described earlier, one of main goal of NEAT is to cross topologies in a meaningful way. This is done via the innovation number which keeps track of the chronology of genes transformation by passing down this number to offsprings throughout mutations. With this identification method, a genome can crossover effectively to form a new genome.

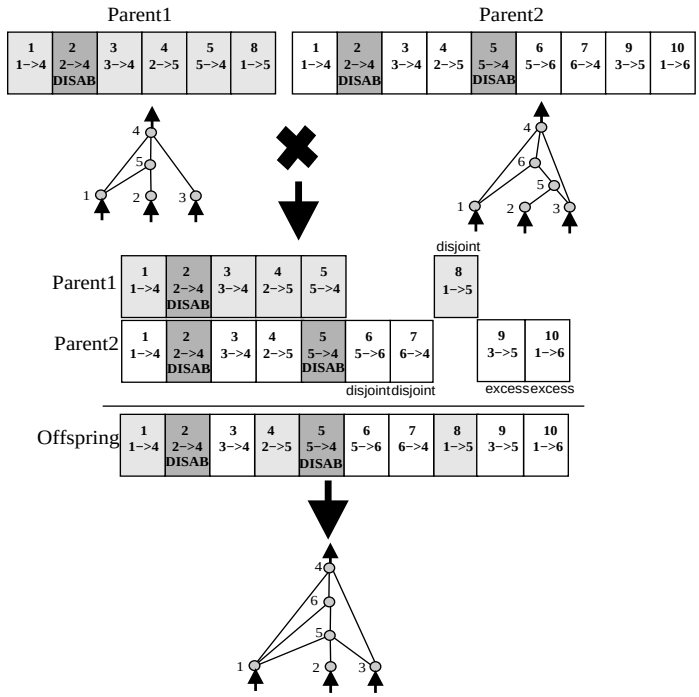


Figure 6: Matching up genomes for different network topologies using innovation numbers (image source: [12, p.109])

As seen in fig. 6, genomes are matched up against each other using innovation number for each gene [12]. For each one of them, the gene with the best fitness is selected for future generation. If they have the same fitness, one of them is selected randomly. Genes without any match which are disjoint or excess are automatically added from the parent with the highest fitness.

Unlike other encoding techniques, NEAT doesn't start with a random population but with a uniform population of networks with no hidden nodes. Structure is evolved, with nodes and links added incrementally. This way, the dimensionality or size of the algorithm is minimized, which reduces the load in calculations and a performance increase. In summary, the historical marking (innovation number), speciation and incremental growth gives the evolving neural network through augmenting topologies a significative advantage.

The last characteristic of NEAT is the protection of topologies innovations to enable them to see further optimizations. Speciation reunites genomes with similar characteristics and compete against each other instead of with everyone else. The way to group them is with the amount of excess and disjoint genes with the following formula:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W} \tag{8}$$

Eq. (8) expresses the compatibility distance δ of different structures in NEAT as a simple linear combination of the number of excess E and disjoint D genes, as well as the average weight differences of matching genes $\bar{W}$.

By creating categories of genomes with similar features, NEAT allows weaker early network structures to grow and perhaps develop

into a topology with a high fitness.

IV. Design Approach

A. Overview

As mentioned earlier, a MATLAB implementation by Mayr [7] is used as a starting point. It is modified to add some features as well as to fit our application. The main modifications are:

- The original unique function has been reorganized into multiple functions.
- Variables renamed to follow the guidelines in [4].
- Visualization (on the command window and as graphics) of the results in real time has been added.
- Possibility of starting NEAT from a previous gene has been implemented.
- Possibility to vary the mutation rates as generations progress.
- Additional commenting in the source code.

The general algorithm architecture is shown in fig. 7. The red blocks enclosed in the region labeled "NEAT" represents the NEAT algorithm. The blue blocks is are part of the evaluation function called by NEAT every generation. This evaluation function is where the simulations take place to assess the individuals' performance (i.e. fitness) to tracking different trajectories.

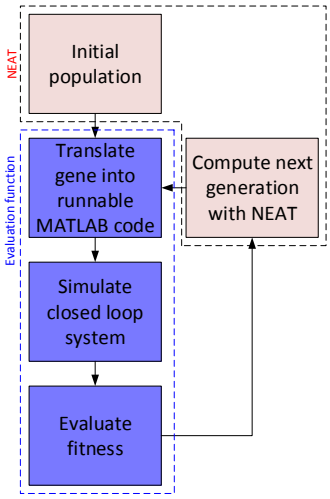


Figure 7: Algorithm overview

B. Evaluation Function

The evaluation function has been the main challenge during this project. Overall, it operates as follows:

1. For each individual in the population, the node and connection genes is translated into a runnable MATLAB function. This is achieved using `genFunction()` provided in the MATLAB Neural Network Toolbox.
2. The generated functions are processed to implement the activation function and define the recurrent connections. This is implemented in `generated_nn_processing()`. This step is necessary since MATLAB automatically uses the standard sigmoid $1/(1+e^{-x})$ whereas we want the modified sigmoid $1/(1+e^{-4.9x})$ as defined by Stanley and Miikkulainen. Recurrent connections are implemented as delays. In other words, a neuron can have an input that is the previous output's value of another neuron (or its own). This gives the network some sort of memory.
3. The neural networks are simulated in Simulink for multiple trajectories and initial conditions. This will be detailed later.
4. The simulation results are then processed to compute each individual's fitness.
5. The individuals' fitness are finally sent back to NEAT in order to compute the next generation using the genetic operators described in chapter III.

The Simulink model used for closed loop simulations is shown in fig. 8. The blocks in the top left blue region are the system's desired inputs for x, y, z, ψ, θ and ϕ. The orange region to the right is where the ANN controller is implemented. In this region, a small purple block with the labels `x` and `y` can be seen. This is where the ANN controller's inputs are normalized between -1 and $+1$. Since the inputs are not known in advance as in batch training where data is known prior to training, the inputs are normalized based on their assumed lower and upper limits. For instance, the positions x, y, z are assumed to be in the range -100 to 100 meters. The top right blue region is where the nonlinear UAV dynamics is implemented (see fig. 4). The pink region at the bottom left is to check for constraints violation and stop the simulation if the controller performs too badly. This way, bad individuals are discarded early on. The concept of constraints violation is explained later. Finally, the green region checks if the variables are within the limits used by the normalization block. If one of the variables violate its range, the simulation is stopped.

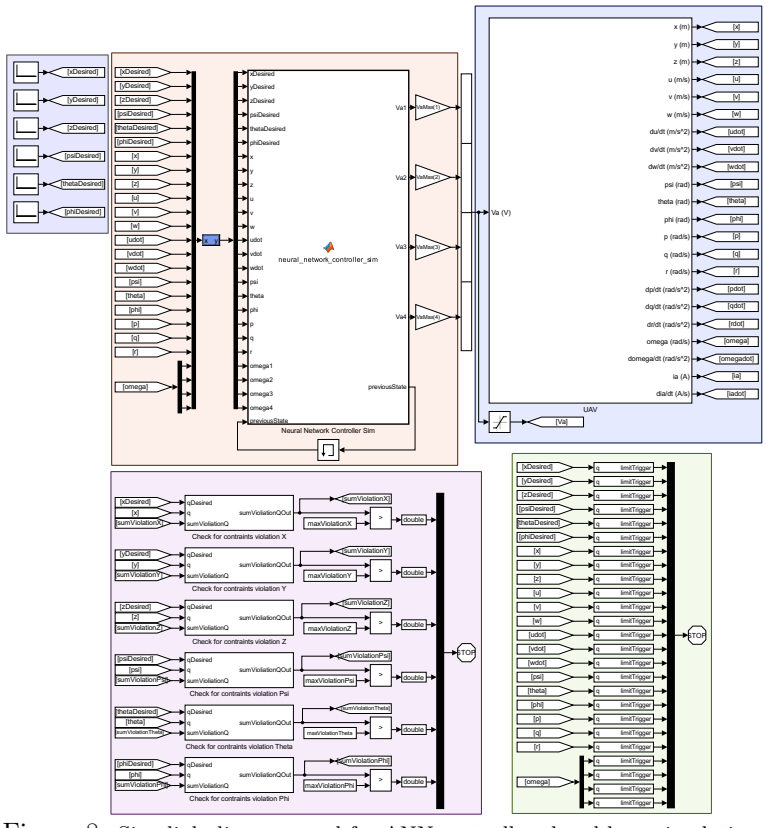


Figure 8: Simulink diagram used for ANN controller closed loop simulations

Each simulation consists of 7 trajectories[1]:

- Maintain an altitude of -90m for 15s with $(x, y, \psi, \theta, \phi) = (0, 0, 0, 0, 0)$.
- Maintain an altitude of -50m for 15s with $(x, y, \psi, \theta, \phi) = (0, 0, 0, 0, 0)$.
- Maintain an altitude of 50m for 15s with $(x, y, \psi, \theta, \phi) = (0, 0, 0, 0, 0)$.
- Maintain an altitude of 0m for 15s with $(x, y, \psi, \theta, \phi) = (50\text{m}, 50\text{m}, 0, 0, 0)$.
- Maintain an altitude of 0m for 15s with $(x, y, \psi, \theta, \phi) = (-50\text{m}, -50\text{m}, 0, 0, 0)$.
- Maintain an altitude of 0m for 15s with $(x, y, \psi, \theta, \phi) = (0, 0, 80°, 0, 0)$.
- Maintain an altitude of 0m for 15s with $(x, y, \psi, \theta, \phi) = (0, 0, -80°, 0, 0)$.

This is done in hope of producing a more generalized controller. If we were to train it only to maintain an altitude of 0m with $(x, y, \psi, \theta, \phi) = (0, 0, 0, 0, 0)$, then perhaps the controller would not generalize well to other initial conditions. It can be noted that the controller is only trained for altitude holding, since training it for other trajectories was judged infeasible in term of available time for this project. Fortunately, a trained network for altitude holding can be used as a starting gene with NEAT in order to train it for other types of inputs. The fitness f is computed using $f = f_1 + f_2$. It's worth mentioning that NEAT maximize the fitness. The f_1 term is the duration the UAV is within the tolerance for a given trajectory. This is shown in fig. 9 for a fictitious signal. In this example, the signal is most of the time within the tolerance except at times 15s and 37s. The second fitness f_2 corresponds to the difference between the maximum possible error and the actual error. This way, f_2 is maximal when the signal follows the reference because $f_2 = |\text{error}_{\max} - \text{error}_{\text{actual}}|$ with $\text{error}_{\text{actual}} \approx 0$.

The constraints violation used in the Simulink from fig. 8 works by the same principle except it allows for wider tolerance. For instance, if the tolerance used by f is $\pm 3°$ for ψ, then the constraints violation in Simulink would instead be using $\pm 8°$. Therefore, the instant ψ exceeds a cumulated duration threshold (e.g. 3s) outside these wider tolerances, the simulation is stopped.

In addition to the fitness function described above, penalization has been added later during the project. The penalization are meant to penalize controllers that apply maximum or minimum thrust on all rotors. In fact, this causes the UAV to climb or fall without violating the other trajectories (i.e. x, y, ψ, θ, ϕ) since no moments are generated.

V. Experimental Results and Analysis

For the training, NEAT's parameters are mostly the same as the ones in Stanley and Miikkulainen's paper. However, the following parameters are different:

- Population size of 250. Ideally, a larger population would be used but it would greatly increase the time require to compute each generation.
- The mutation rates for adding nodes and connections are started very high, i.e. 0.90, then decreased in a staircase fashion until they reach 0.10 and 0.30 respectively at the 40$^{\text{th}}$ generation and stay constant after that. This high rate at the beginning is to stimulate a wide range of configurations early on. Then, the standard crossover takes over as generations progress.
- Weights' lower and upper limit are fixed at ± 0.5.
- Weights' mutation value is uniformly distributed between ± 0.125. This low value is to prevent high weight values that would always saturate the neurons' activation functions.
- The initial topology is chosen to be partially connected with the most relevant inputs in order to speed up the evolution process.

[1]The trajectories are defined in `trajectory_definition()`.

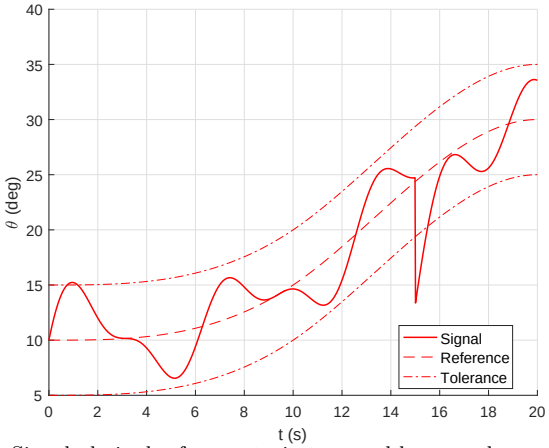


Figure 9: Signal, desired reference trajectory and lower and upper tolerances

Figure 10 shows the evolution of various NEAT's parameters throughout generations. The top left graph presents the maximum fitness. We can see that it start relatively low at 190 then make a small jump to 300 around the 15$^{\text{th}}$ generation. At this time, this is likely due to a mutation. The maximum fitness does not grow much for the next 15 generations and then experiences a sharp increase around the 30$^{\text{th}}$ generation. This jump is purely intentional since the upper voltage limit has been manually decreased from 12V to 10V. This decision has been made after realizing that the best controller would always send maximum voltage on all rotors resulting in symmetrical thrust and higher fitness as explained in B. Evaluation Function. Around the 75$^{\text{th}}$, 90$^{\text{th}}$ and 115$^{\text{th}}$ generations, the maximum fitness undergoes a small decrease. This is caused by the stagnation mechanism implemented in NEAT. When the maximum fitness of a specie does not change for 15 generations, it is eliminated to make room for other more promising species. When looking at the average fitness (1$^{\text{st}}$ row, 2$^{\text{nd}}$ column) and average number of disabled connections (2$^{\text{nd}}$ row, 3$^{\text{rd}}$ column), we observe a large drop and jump respectively. This, again, is intentional. Since the fitness would not increase, the mutation rate were temporarily increased in hope of producing a more fit individual. The reason why the number of disabled connections increases with the mutation rates is because a connection or node mutation can results in disabling as well as creating these elements. On the bottom right two graphs (3$^{\text{rd}}$ row), we see that the number of species suddenly jumps around the 5$^{\text{th}}$ generation. Looking at eq. (8) that expresses at which point two networks are alike, we realize that all the individuals are similar during the few first generations and only after 5$^{\text{th}}$ generation, there have been enough mutations to categorize them in species. Finally, we observe that the average number of hidden nodes (2$^{\text{nd}}$ row, 1$^{\text{st}}$ column), connections (2$^{\text{nd}}$ row, 3$^{\text{rd}}$ column) and recurrent connections (3$^{\text{nd}}$ row, 1$^{\text{st}}$ column) increase almost monotonically. This is expected behavior with NEAT.

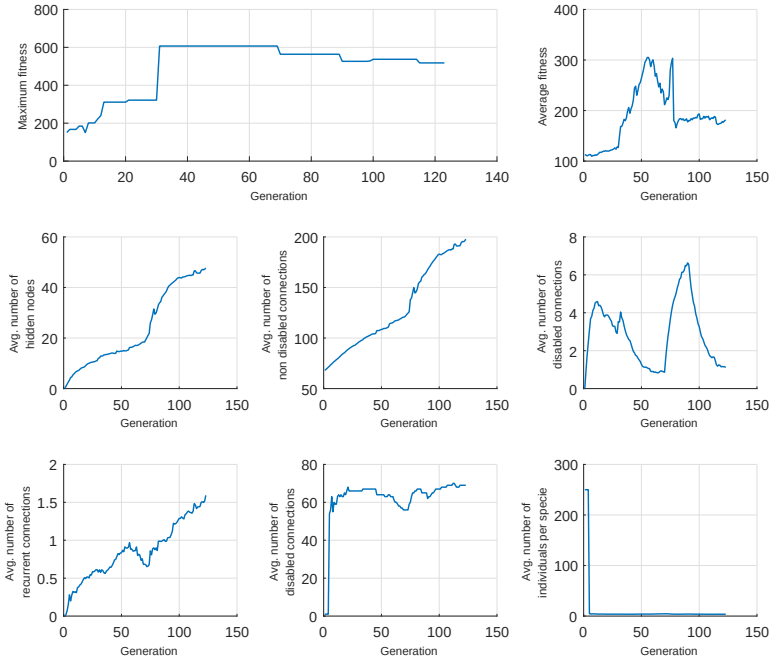
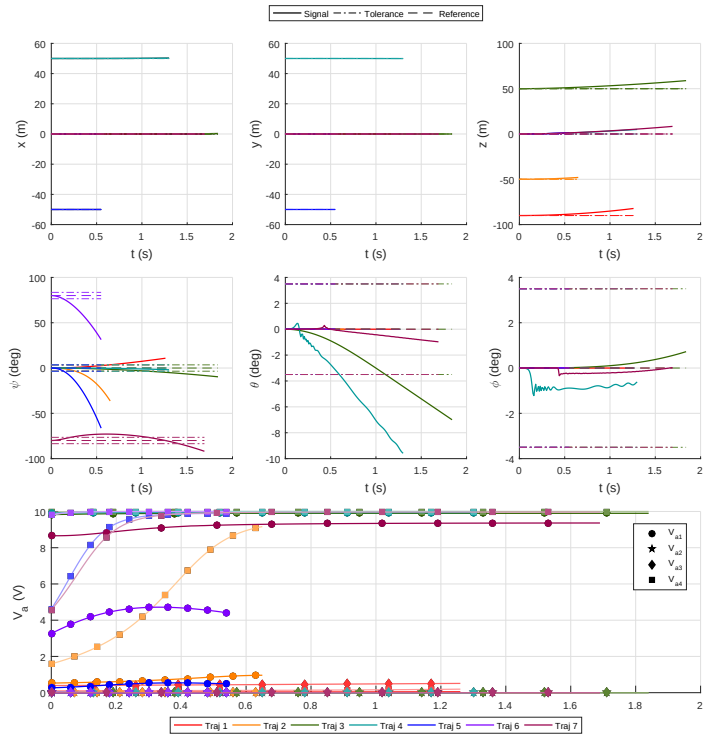


Figure 10: Evolution progress

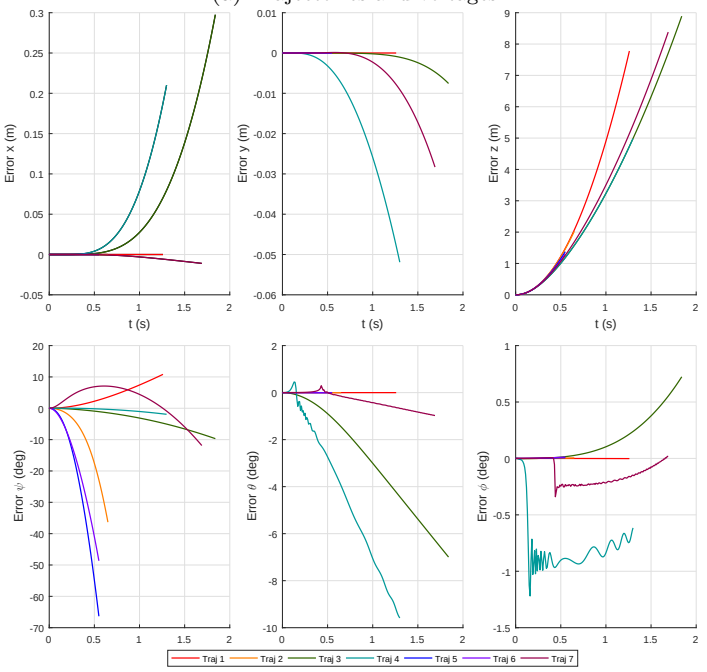
Figure 11 on page 7 presents the best individual's performance at the 1st generation. At this stage, there are no hidden nodes and all the weights are initialized randomly. We observe that the resulting controller performs very badly and for a very short amount of time before the simulation is interrupted for too much constraints violation. At the 9th generation seen in fig. 12 on page 8, the controller now manages to last a little longer for 2 of the 7 trajectories but is still having trouble keeping a constant yaw angle (ψ) and altitude. In fig. 13 on page 8, we note that after 49 generations, the controller has figured out that applying the same thrust on all rotors would keep the UAV from spinning. Unfortunately, it applies maximum thrust resulting in a rapid ascent[2]. For the 91th and 123rd generations in figs. 14 and 15 on page 9, it seems that the controller now produces a more complex output resulting in very small oscillations in pitch and roll angles (θ, ϕ). However, it is hard to tell if it's getting better or worse. Unfortunately, there were not enough time left to let the evolution continue further and we had to stop at generation 123.

The best individual of generation 123 was tested for generalization by simulating for 17 different trajectories and initial conditions. Figure 16 on page 10 shows the results. We see that the four voltages sent by the controller are similar suggesting that different conditions do not drastically affect the resulting output.

The network of generation 123 is depicted in fig. 17 on page 10. The red nodes are inputs, blue nodes are hidden nodes and green nodes are outputs. The connections' line width are proportional to their weights and recurrent connections are colored in magenta. In this case, there is only one recurrent connection from output 83 to output 85. Figure 18 on page 10 shows 12 networks chosen across the 123 generations. We indeed observed that the topologies get more and more complex.



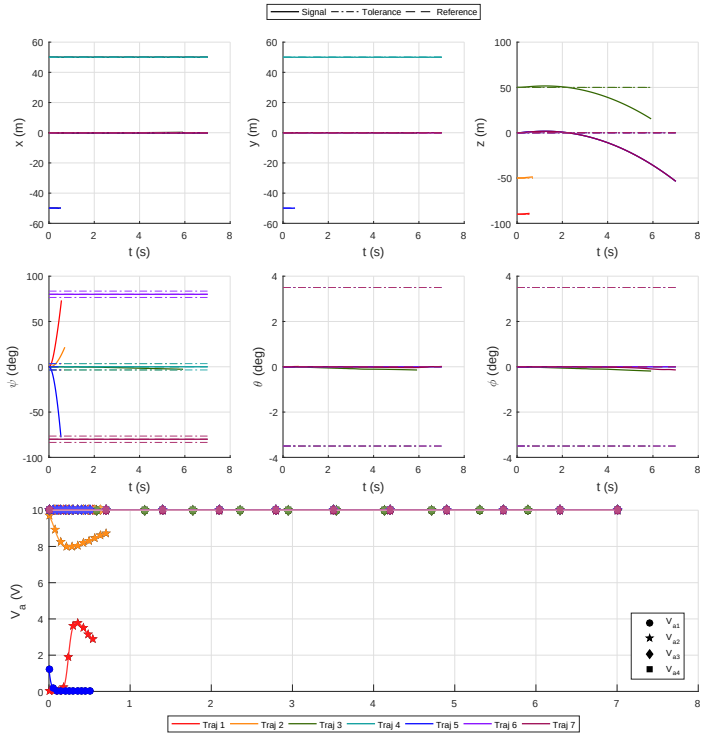
(a) Trajectories and voltages



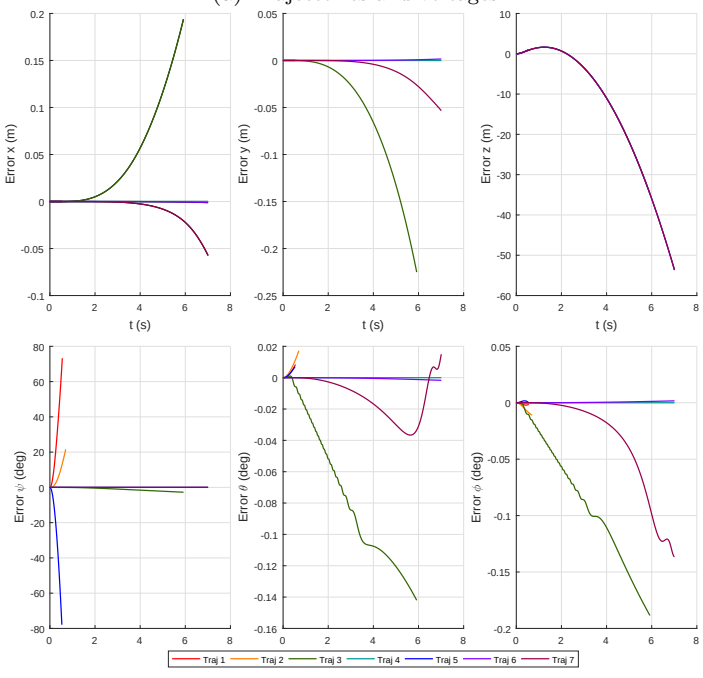
(b) Errors

Figure 11: Best individual's simulation results at generation 1

[2]Note that because of the body axes' definition, an altitude gain is negative since the z axis is pointing down. See fig. 2 on page 2.

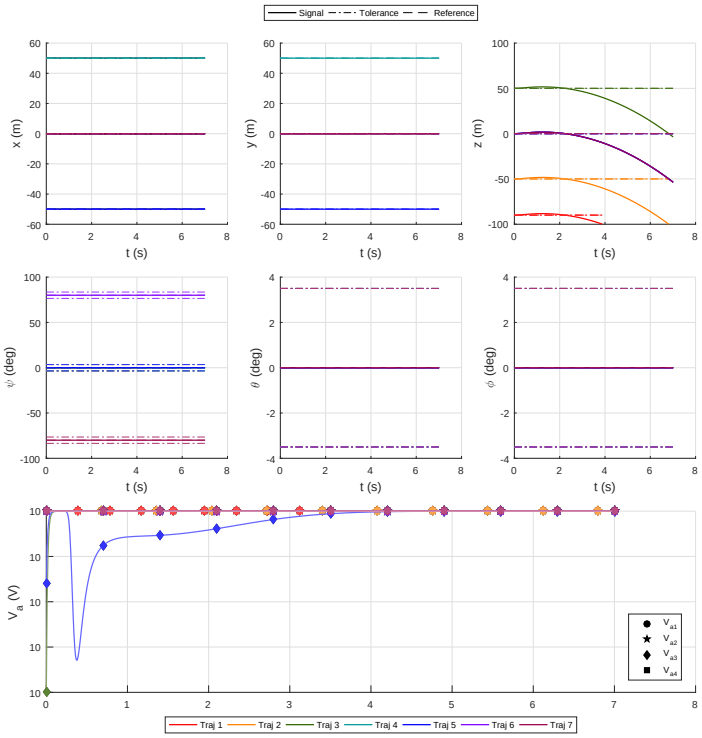


(a) Trajectories and voltages

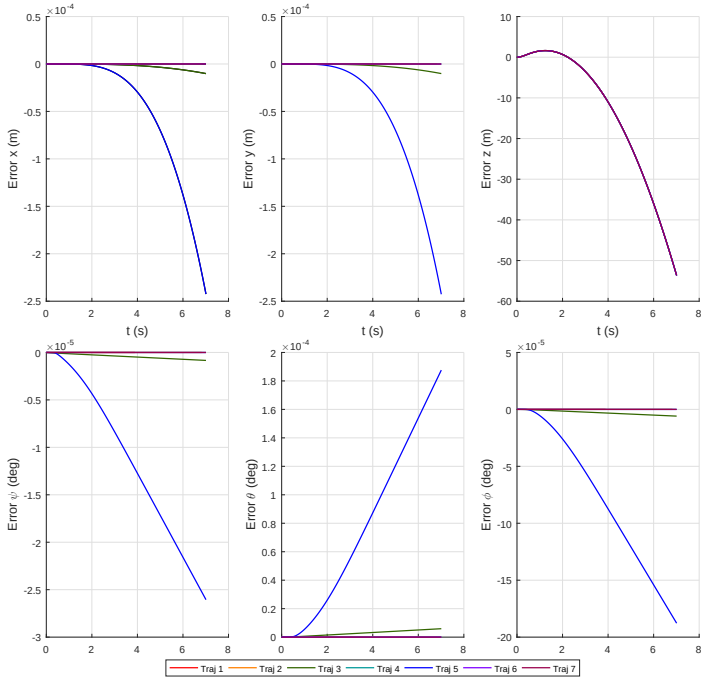


(b) Errors

Figure 12: Best individual's simulation results at generation 9

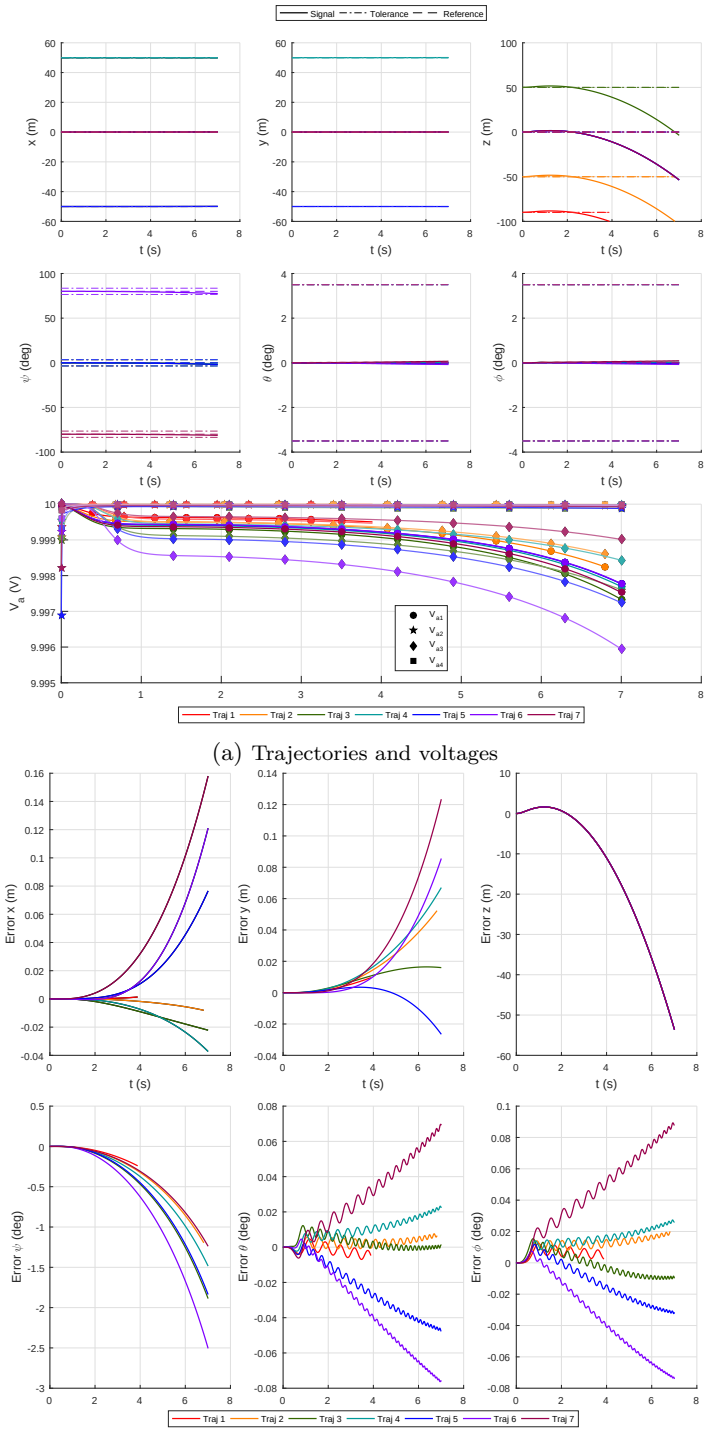


(a) Trajectories and voltages



(b) Errors

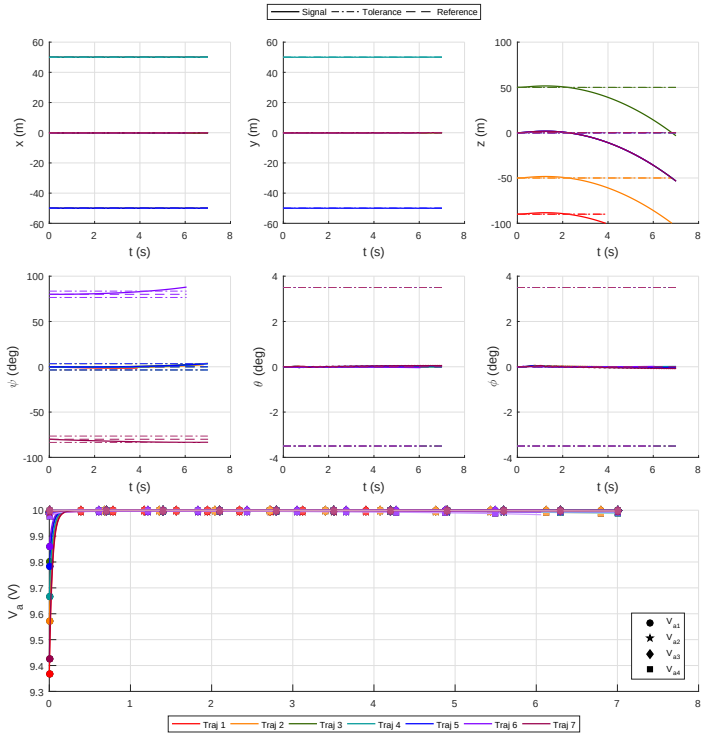
Figure 13: Best individual's simulation results at generation 49



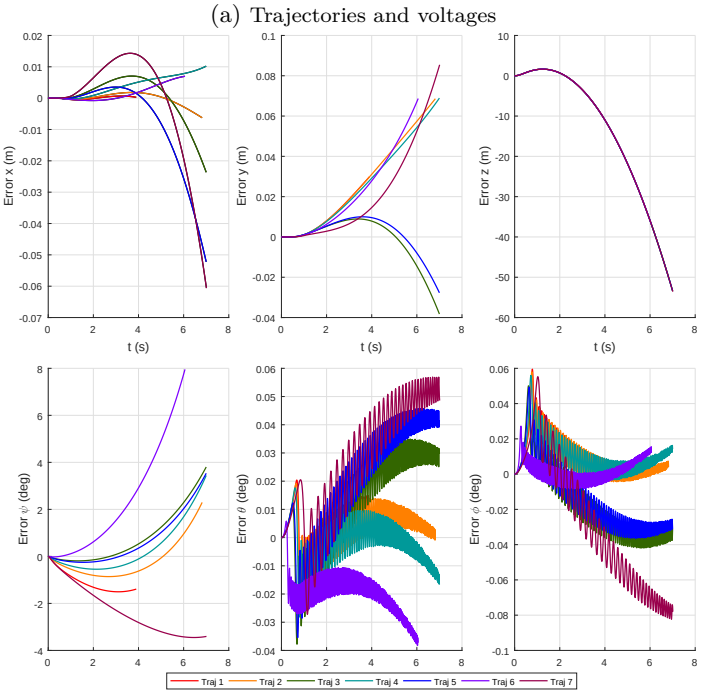
(a) Trajectories and voltages

(b) Errors

Figure 14: Best individual's simulation results at generation 91

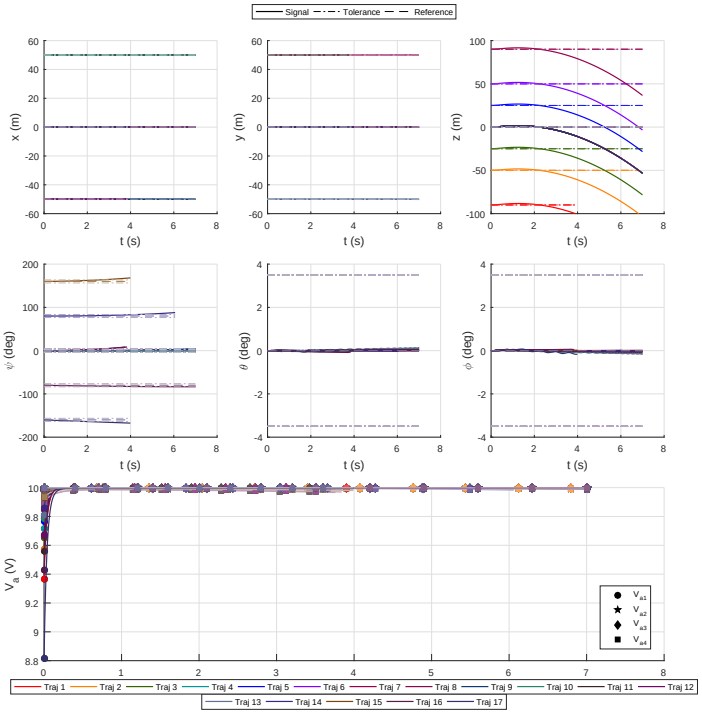


(a) Trajectories and voltages

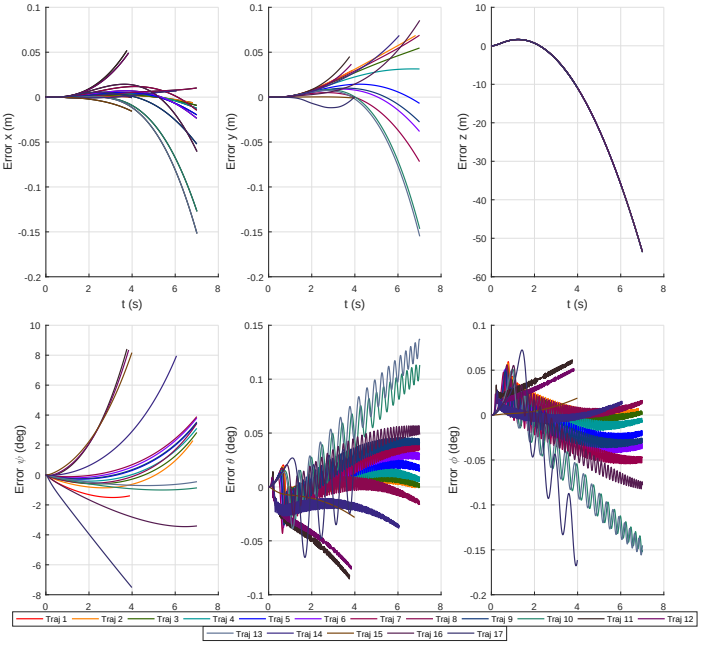


(b) Errors

Figure 15: Best individual's simulation results at generation 123



(a) Trajectories and voltages



(b) Errors

Figure 16: Performance of best individual of generation 123 for 17 different trajectories and initial conditions

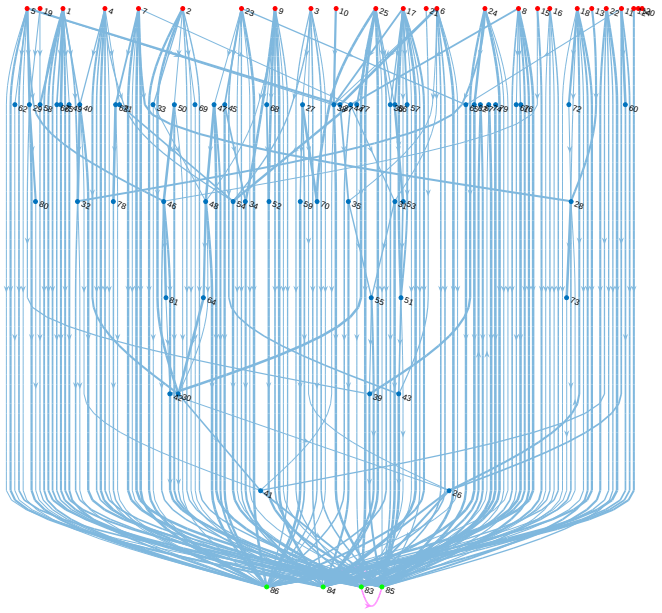


Figure 17: Best neural network topology at last generation

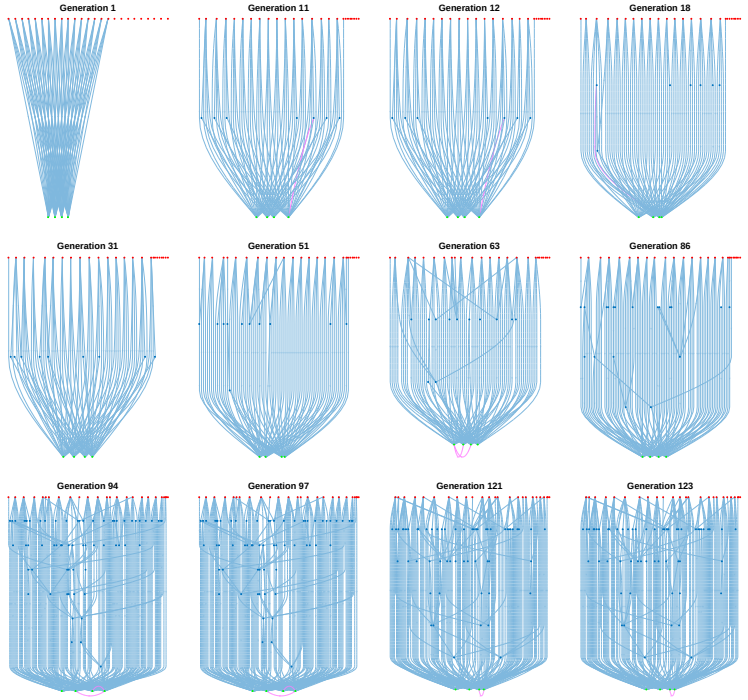


Figure 18: Topology evolution for 12 of the best neural networks throughout generations

VI. Conclusion and Future Work

Due to its highly nonlinear and naturally unstable dynamics, the control of the quadcopter is a very challenging task. This project tackled this control problem from a new perspective by evolving ANN using the NEAT algorithm. As stated at the beginning of this report, this project was highly experimental and somewhat ambitious. We can not say we successfully generated an ANN capable of controlling the UAV. However, a lot has been learned and this knowledge can be used in future work to make the ANN evolving process more efficient and successful. Some proposed changes and enhancements are:

- Using a much larger population with a minimal number of 1000 individuals.
- Speeding up the simulations.
- Parallelizing the evaluation function by distributing to multiple MATLAB workers across multiple computers.
- Tuning the fitness function to penalize unwanted behavior such as oscillations.
- Including multiple trajectories during the training such as translation along x, y, z and rotation about ψ.

Appendix A
Sample Run

This appendix shows a shortened sample command window output during the evolution process. The "....." indicate that lines have been omitted for compactness. Line 1 indicates where the evolution log is stored. The evolution log is what you see here (i.e. the command window output). Line 3 to line 43 show the parameters used for this particular run. The arrays printed at lines 27 and 31 refers to the mutation rates. For instance, between generations 0 and 5, the rate is 0.90, between 6 and 10, it is 0.50, and so on. At the beginning of every generation, the generation number is printed as shown at line 48. From line 51 to line 55, the fitness of each individual is printed as simulations are performed. When all the individuals are evaluated, the maximum fitness is presented at line 56. Line 63 indicates the elapsed time from the beginning of the run. Starting at generation 2, the new mutation rates are printed as shown at line 69. Between lines 75 and 79, the number of recurrent connections is printed for each individual that has at least one. Finally, at the end of every generation, the existing and propagating species array is printed as shown at line line 174. The first row (line 175) is the species' ID, the second (line 176) is the alloted number of offspring for the species and the last row (line 177) indicates the actual number of offspring.

```
1  Diary stored in diary/design_controller_01_Dec_2016_12_15_09.txt
2
3  Using the following parameters:
4  loadFlag                                =    0
5  saveFlag                                =    1
6  nMaxGeneration                          =  200
7  populationSize                          =  250
8  nInputNodes                             =   25
9  nOutputNodes                            =    4
10 initialConnectedInputNodes              = [1 2 3 4 5 6 7 8 9 16 17 18 22 23 24 25]
11 fitnessLimit                            =  Inf
12 Speciation.c1                           = 1.00000
13 Speciation.c2                           = 1.00000
14 Speciation.c3                           = 0.40000
15 Speciation.threshold                    = 12.00000
16 Stagnation.threshold                    = 0.01000
17 Stagnation.nGenerations                 =   15
18 Refocus.threshold                       = 0.01000
19 Refocus.nGenerations                    =   20
20 Initial.killPercentage                  = 0.20000
21 Initial.numberForKill                   = 5.00000
22 Initial.numberCopy                      =    5
23 Selection.pressure                      = 2.00000
24 Crossover.percentage                    = 0.80000
25 Crossover.probabilityInterspecies       = 0.00100
26 Crossover.probabilityMultipoint         = 0.60000
27 Mutation.probabilityAddNode             = [5.00  10.00  15.00  20.00  30.00  40.00  Inf]
28                                           [0.90  0.50   0.40   0.30   0.20   0.10   0.05]
29 Mutation.probabilityAddNodeShrinkRate   = N/A
30 Mutation.probabilityAddNodeMin          = N/A
31 Mutation.probabilityAddConnection       = [5.00  10.00  15.00  20.00  30.00  40.00  Inf]
32                                           [0.90  0.70   0.60   0.45   0.40   0.30   0.20]
33 Mutation.probabilityAddConnectionShrinkRate = N/A
34 Mutation.probabilityAddConnectionMin    = N/A
35 Mutation.probabilityRecurrency          = 0.05000
36 Mutation.probabilityRecurrencyShrinkRate = 0.00000
37 Mutation.probabilityRecurrencyMin       = 0.05000
38 Mutation.probabilityMutateWeight        = 0.90000
39 Mutation.probabilityMutateWeightShrinkRate = 0.00000
40 Mutation.probabilityMutateWeightMin     = 0.90000
41 Mutation.weightCap                      = 8.00000
42 Mutation.weightRange                    = 5.00000
43 Mutation.probabilityGeneReenabled       = 0.10000
44
45 Starting parallel pool (parpool) using the 'local' profile ... connected to 13 workers.
46
47 ------------------------------------------------------------------------
48 |                          Generation    1                             |
49 ------------------------------------------------------------------------
50
51 fitness = 110.57 for individual 13   in generation 1
52 fitness = 111.55 for individual 104  in generation 1
53 ................................................
54 fitness = 108.41 for individual 247  in generation 1
55 fitness = 110.83 for individual 249  in generation 1
56 Max fitness is 150.848335
57
58 Existing and propagating matrix:
59 1
60 250
61 1
62
63 Elapsed time is 00h17m24s
64
65 ------------------------------------------------------------------------
66 |                          Generation    2                             |
67 ------------------------------------------------------------------------
68
69 Mutation probabilities update:
70 Mutation.probabilityAddNode       = 0.90000
71 Mutation.probabilityAddConnection = 0.90000
72 Mutation.probabilityRecurrency    = 0.05000
73 Mutation.probabilityMutateWeight  = 0.90000
74
75 nPreviousStates = 1 for individual 149 in generation 2 (AGL1FMBMI6JJG74ENNFF69M8E99BIA)
76 nPreviousStates = 1 for individual 132 in generation 2 (8C341J2LHOBL7A31HONFMG3JAG779F)
77 nPreviousStates = 1 for individual 103 in generation 2 (D9G34597G34E1B6BNFBLE5E82OMHNA)
78 nPreviousStates = 1 for individual  92 in generation 2 (68GOIGBI5ON8NKK9DFDCFABN9NMOLF)
79 nPreviousStates = 1 for individual 174 in generation 2 (EK59M1IBJM49KLLHPJO9CBK3KGJ7F5)
80 fitness = 104.02 for individual 65  in generation 2
81 fitness = 114.05 for individual 52  in generation 2
82 ................................................
83 fitness = 111.94 for individual 170 in generation 2
84 fitness = 123.37 for individual 92  in generation 2
85 Max fitness is 167.257427
86
87 Existing and propagating matrix:
88 1
89 250
90 1
91
92 Elapsed time is 00h38m01s
93
94 ------------------------------------------------------------------------
95 |                          Generation    3                             |
96 ------------------------------------------------------------------------
97
98 Mutation probabilities update:
99 Mutation.probabilityAddNode       = 0.90000
100 Mutation.probabilityAddConnection = 0.90000
101 Mutation.probabilityRecurrency    = 0.05000
102 Mutation.probabilityMutateWeight  = 0.90000
103
104 nPreviousStates = 1 for individual  74 in generation 3 (699BJJ32F576GLP5BG5I99AA98N9CP)
105 nPreviousStates = 1 for individual  64 in generation 3 (CNO1EMG544482M2991IOM219D5F7HO)
106 ..............................................................................
107 nPreviousStates = 1 for individual 236 in generation 3 (JK9DC5JK3H5A5F5O1CMLFAELI9EONG)
108 nPreviousStates = 1 for individual 225 in generation 3 (7ALJOJ9BO96LO0A2FHO3079NNGFJ7K)
109 fitness = 112.39 for individual 137 in generation 3
110 fitness = 111.34 for individual 13  in generation 3
111 ................................................
112 fitness = 128.63 for individual 164 in generation 3
113 fitness = 103.60 for individual 163 in generation 3
114 Max fitness is 167.257427
115
116 Existing and propagating matrix:
117 1
118 250
119 1
120
121 Elapsed time is 01h05m21s
122
123 ------------------------------------------------------------------------
124 |                          Generation    4                             |
125 ------------------------------------------------------------------------
126
127 Mutation probabilities update:
128 Mutation.probabilityAddNode       = 0.90000
129 Mutation.probabilityAddConnection = 0.90000
130 Mutation.probabilityRecurrency    = 0.05000
131 Mutation.probabilityMutateWeight  = 0.90000
132
133 nPreviousStates = 1 for individual 103 in generation 4 (L636HI0EC048577FBJDL61E0995NFO)
134 nPreviousStates = 1 for individual 101 in generation 4 (16PNDLKNL6OBEP6FO8EB6MLIE33926)
135 ..............................................................................
136 nPreviousStates = 1 for individual 224 in generation 4 (A957BD5I2G9IPP2INL26DKE980BIBN)
137 nPreviousStates = 1 for individual 231 in generation 4 (B16H313C24BB8C57GFO3OOEHAJEODE)
138 fitness = 110.96 for individual 127 in generation 4
139 fitness = 111.27 for individual 39  in generation 4
140 ................................................
141 fitness = 123.38 for individual 231 in generation 4
142 fitness = 111.07 for individual 14  in generation 4
143 Max fitness is 167.257427
144
145 Existing and propagating matrix:
146 1
147 250
148 1
149
150 Elapsed time is 01h41m15s
151
152 ------------------------------------------------------------------------
153 |                          Generation    5                             |
154 ------------------------------------------------------------------------
155
156 Mutation probabilities update:
157 Mutation.probabilityAddNode       = 0.90000
158 Mutation.probabilityAddConnection = 0.90000
```

```
Mutation.probabilityRecurrency    = 0.05000
Mutation.probabilityMutateWeight  = 0.90000

nPreviousStates = 1 for individual  91 in generation 5 (CA9DPK32L0D0C0E4GL138CB84A1OLH)
nPreviousStates = 1 for individual  88 in generation 5 (87DIEAIGAE5G99K8F92PPFKE5HG722)
.....................................................................................
nPreviousStates = 1 for individual 239 in generation 5 (4HD2GI3F1ODPJ8EOH2O8CC24LIN2FM)
nPreviousStates = 1 for individual 238 in generation 5 (C69J6IJ8FOB16CO5P56A3L35G769NK)
fitness = 107.42 for individual 52  in generation 5
fitness = 112.34 for individual 39  in generation 5
.....................................................
fitness = 156.03 for individual 239 in generation 5
fitness = 119.25 for individual 66  in generation 5
Max fitness is 184.580714

Existing and propagating matrix:
1 2 3 4 5 6 7 8 9 10 11 ..... 43 44 45 46 47 48 49 50 51 52 53 54
4 5 5 4 5 5 4 5 5 5  4  ..... 4  5  5  4  5  4  5  4  5  4  5  5
1 1 1 1 1 1 1 1 1 0  0  ..... 0  0  0  0  0  0  0  0  0  0  0  0

Elapsed time is 02h32m39s
```

Appendix B
Source Code

This appendix lists the main scripts and functions. The main script from which the evolution process is started is `design_controller.m` and starts on page 12. The `trajectory_definition.m` function where the trajectories are defined starts on page 14. This function is called once inside `design_controller.m` just before NEAT is launched. The source code for NEAT (`neat.m`) is located on page 18. The evaluation function (`uav_evaluation_function.m`) called inside NEAT every generation begins on page 24. The function (`build_neural_network.m`) that translates the node and connection genes into a Neural Network object is on page 28. This function is called for each individual inside `uav_evaluation_function.m`. Finally, the function to processed (i.e. modify the activation function and implement the recurrent connections) the generated MATLAB code to evaluate the networks (`generated_nn_processing.m`) is listed on page 30.

A. *Design Controller*

```
clear all
close all
clc

% Uncomment for repetability between runs
% rng(6541651)
rng('shuffle');

% Constants
DEG_2_RAD = pi/180;
RAD_2_DEG = 180/pi;
MM_2_M = 1/1000;
M_2_MM = 1000/1;

% Date and time
dateAndTime = datestr(datetime('now','TimeZone','local',...
    'Format','d-MMM-y HH:mm:ss Z'));
dateAndTime = strrep(dateAndTime,':','_');
dateAndTime = strrep(dateAndTime,'-','_');
dateAndTime = strrep(dateAndTime,' ','_');

% Set up diary
if ~exist('diary','dir')
    mkdir('diary');
end
diary(['diary/','design_controller','_',dateAndTime,'.txt']);
fprintf('Diary stored in %s\n\n',...
    ['diary/','design_controller','_',dateAndTime,'.txt']);

% Close all figure handles
figAllHandle = findall(0,'Type','figure');
if ~isempty(figAllHandle)
    for iFigHandle = 1:numel(figAllHandle)
        % figAllHandle(iFigHandle).CloseRequestFcn = @closereq;
        set(0, 'currentfigure', figAllHandle(iFigHandle));
        close('force');
        % close(figAllHandle(iFigHandle));
        delete(figAllHandle(iFigHandle));
    end
    clear figAllHandle;
end

%% Delete paths from MATLAB search path

% Code snippet from https://goo.gl/0OwRMn

% Get a list of all files and folders in this folder.
if exist('temp_uav_nn_controller','dir')
    files = dir('temp_uav_nn_controller');
    files(1:2) = [];
end

% Get a logical vector that tells which is a directory.
if exist('temp_uav_nn_controller','dir')
    dirFlags = [files.isdir];
end

% Extract only those that are directories.
if exist('temp_uav_nn_controller','dir')
    subFolders = files(dirFlags);
end
end

% Remove path
if exist('temp_uav_nn_controller','dir')
    for k = 1:length(subFolders)
        rmpath(['temp_uav_nn_controller/',subFolders(k).name])
    end
end
rehash();

%% Parameters - General

% If set to 1, will load population, generation, innovation_record and
% species_record from neatsave.mat at start of algorithm, if set to 0,
% algorithm will start with initial population, new species record and new
% innovation record, at generation = 1 (default option)
loadFlag = 1;

% If set to 1, will save population, generation, innovation_record and
% species_record to neatsave.mat at every generation (default option)
saveFlag = 1;

% If set to 1, NEAT will start with the gene from geneFile and intialize the
% weight randomnly between [-1,+1]
loadGene = 0;
geneFile = 'neat_data/neatsave_27_Nov_2016_20_27_44_043.mat';

% Maximum number of generations for generational loop
nMaxGeneration = 200; % 200

% Population size
populationSize = 250; % 150, 500, 300

% Input and output nodes numbers
nInputNodes = 25;
nOutputNodes = 4;

% Vector of initially connected input nodes out of complete number of input nodes
% (if you want to start with a subset and let evolution decide which ones are necessary)
% for a fully connected initial population, list all the inputs. For each input
% node listed in initialConnectedInputNodes, it will be connected to every
% output node during population's intialization.
% initialConnectedInputNodes = 1:nInputNodes;
initialConnectedInputNodes = [1,2,3,4,5,6,7,8,9,16,17,18,22,23,24,25];
% initialConnectedInputNodes = 1;

% Fitness limit
fitnessLimit = Inf;

%% Parameters - Speciation

% Speciation parameters as published by Ken Stanley
Speciation.c1 = 1.0;
Speciation.c2 = 1.0;
Speciation.c3 = 0.4; % 0.4
Speciation.threshold = 25.0; % 3 (10.0,12.0,14.0)

%% Parameters - Reproduction - Stagnation + Refocuse

% Threshold to judge if a species is in stagnation (max fitness of species
% varies below threshold) this threshold is of course dependent on your
% fitness function, if you have a fitness function which has a large
% spread, you might want to increase this threshold
Stagnation.threshold = 1e-2;

% If max fitness of species has stayed within Stagnation.threshold in the
% last Stagnation.nGenerations generations, all its fitnesses will be
% reduced to 0, so it will die out
Stagnation.nGenerations = 15; % 15

% Computation is done the following way: the absolute difference between
% the average max fitness of the last Stagnation.nGenerations
% generations and the max fitness of each of these generations is computed
% and compared to Stagnation.threshold.
% if it stays within this threshold for the indicated number of
% generations, the species is eliminated
Refocus.threshold = 1e-2;

% If maximum overall fitness of population doesn't change within threshold
% for this number of generations, only the top two species are allowed to
% reproduce
Refocus.nGenerations = 20; % 20

%% Parameters - Reproduction - Initial

% The percentage of each species which will be eliminated (lowest
% performing individuals). This percentage for eliminating individuals will
% only be used in species which have more individuals than numberForKill
Initial.killPercentage = 0.2;

% Please note that whatever the above settings, the code always ensures
% that at least 2 individuals are kept to be able to cross over, or at
% least the single individual in a species with only one individual
Initial.numberForKill = 5;

% Species which have equal or greater than numberCopy individuals will
% have their best individual copied unchanged into the next generation
Initial.numberCopy = 5;

%% Parameters - Reproduction - Selection

% Selection (ranking and stochastic universal sampling)
% Number between 1.1 and 2.0, determines selective pressure towards most
% fit individual of species
Selection.pressure = 2;

%% Parameters - Reproduction - Crossover

% Percentage governs the way in which new population will be composed from
% old population. exception: species with just one individual can only use
% mutation
Crossover.percentage = 0.8;

% If crossover has been selected, this probability governs the
% intra/interspecies parent composition being used for the
% standard-crossover in which matching connection genes are inherited
% randomly from both parents. In the (1-Crossover.probabilityMultipoint)
% cases, weights of the new connection genes are the mean of the
% corresponding parent genes
Crossover.probabilityInterspecies = 0.001;
Crossover.probabilityMultipoint = 0.6;

%% Parameters - Reproduction - Mutation

% Adding node probability
Mutation.probabilityAddNode                 = 0.60; % 0.03
Mutation.probabilityAddNodeMin              = 0.05; % 0.03
Mutation.probabilityAddNodeShrinkRate       = ...
    (Mutation.probabilityAddNode-Mutation.probabilityAddNodeMin)/60;
if true
    Mutation.probabilityAddNodeScheduling = [...
        0005,  10,  15,  20,  30,  40,  70,  75,  90,  100, Inf;
        0.90,0.50,0.40,0.30,0.20,0.10,0.90,0.85,0.80,0.10,0.05];
else
    Mutation.probabilityAddNodeScheduling = -1;
end

% Adding connection probability
Mutation.probabilityAddConnection           = 0.80; % 0.05
Mutation.probabilityAddConnectionMin        = 0.20;
Mutation.probabilityAddConnectionShrinkRate = ...
    (Mutation.probabilityAddConnection-Mutation.probabilityAddConnectionMin)/60;
```

```
if true
    Mutation.probabilityAddConnectionScheduling = [...
        0005,  10,  15,  20,  30,  40,  70,  75,  90,  Inf;
        0.90,0.70,0.60,0.45,0.40,0.30,0.90,0.85,0.80,0.20];
else
    Mutation.probabilityAddNodeScheduling = -1;
end

% If we are in add_connection_mutation, this governs if a recurrent
% connection is allowed. Note: this will only activate if the random
% connection is a recurrent one, otherwise the connection is simply
% accepted. If no possible non-recurrent connections exist for the
% current node genes, then for e.g. a probability of 0.1, 9 times out
% of 10 no connection is added.
Mutation.probabilityRecurrency              = 0.05; % 0.05
Mutation.probabilityRecurrencyMin           = Mutation.probabilityRecurrency;
Mutation.probabilityRecurrencyShrinkRate    = ...
    (Mutation.probabilityRecurrency-Mutation.probabilityRecurrencyMin)/60;
Mutation.probabilityMutateWeight            = 0.9;  % 0.9
Mutation.probabilityMutateWeightMin         = 0.9;
Mutation.probabilityMutateWeightShrinkRate  = ...
    (Mutation.probabilityMutateWeight-Mutation.probabilityMutateWeightMin)/60;

% Weights will be restricted from -Mutation.weightCap to Mutation.weightCap
Mutation.weightCap = 0.5; % 8

% Random distribution with width Mutation.weightRange, centered on 0.
% mutation range of 5 will give random distribution from -2.5 to 2.5
Mutation.weightRange = 0.25; % 5

% Probability of a connection gene being reenabled in offspring if it was
% inherited disabled
Mutation.probabilityGeneReenabled = 0.10; % 0.20

%% Display parameters

if Mutation.probabilityAddNodeScheduling(1) == -1
    mutationAddNode1Str = 'Mutation.probabilityAddNode                  = %4.5f\n';
    mutationAddNode2Str = 'Mutation.probabilityAddNodeShrinkRate        = %4.5f\n';
    mutationAddNode3Str = 'Mutation.probabilityAddNodeMin               = %4.5f\n';
    mutationAddNode1Val = {Mutation.probabilityAddNode};
    mutationAddNode2Val = Mutation.probabilityAddNodeShrinkRate;
    mutationAddNode3Val = Mutation.probabilityAddNodeMin;
else
    mutationAddNode1Str = [...
        'Mutation.probabilityAddNode                = [%s]\n',...
        '                                             [%s]\n'];
    mutationAddNode2Str = 'Mutation.probabilityAddNodeShrinkRate        = %s\n';
    mutationAddNode3Str = 'Mutation.probabilityAddNodeMin               = %s\n';
    mutationAddNode1Val = {...
        num2str(Mutation.probabilityAddNodeScheduling(1,:),'\t\t%2.2f'), ...
        num2str(Mutation.probabilityAddNodeScheduling(2,:),'\t\t%2.2f')};
    mutationAddNode2Val = 'N/A';
    mutationAddNode3Val = 'N/A';
end

if Mutation.probabilityAddConnectionScheduling(1) == -1
    mutationAddConn1 = 'Mutation.probabilityAddConnection           = %4.5f\n';
    mutationAddConn2 = 'Mutation.probabilityAddConnectionShrinkRate = %4.5f\n';
    mutationAddConn3 = 'Mutation.probabilityAddConnectionMin        = %4.5f\n';
    mutationAddConn1Val = {Mutation.probabilityAddConnection};
    mutationAddConn2Val = Mutation.probabilityAddConnectionShrinkRate;
    mutationAddConn3Val = Mutation.probabilityAddConnectionMin;
else
    mutationAddConn1Str = [...
        'Mutation.probabilityAddConnection          = [%s]\n',...
        '                                             [%s]\n'];
    mutationAddConn2Str = 'Mutation.probabilityAddConnectionShrinkRate = %s\n';
    mutationAddConn3Str = 'Mutation.probabilityAddConnectionMin        = %s\n';
    mutationAddConn1Val = {...
        num2str(Mutation.probabilityAddConnectionScheduling(1,:),'\t\t%2.2f'), ...
        num2str(Mutation.probabilityAddConnectionScheduling(2,:),'\t\t%2.2f')};
    mutationAddConn2Val = 'N/A';
    mutationAddConn3Val = 'N/A';
end

fprintf([...
    'Using the following parameters:\n',...
    'loadFlag                                   = %4.f\n',...
    'saveFlag                                   = %4.f\n',...
    'nMaxGeneration                             = %4.f\n',...
    'populationSize                             = %4.f\n',...
    'nInputNodes                                = %4.f\n',...
    'nOutputNodes                               = %4.f\n',...
    'initialConnectedInputNodes                 = [%s]\n',...
    'fitnessLimit                               = %4.5f\n',...
    'Speciation.c1                              = %4.5f\n',...
    'Speciation.c2                              = %4.5f\n',...
    'Speciation.c3                              = %4.5f\n',...
    'Speciation.threshold                       = %4.5f\n',...
    'Stagnation.threshold                       = %4.5f\n',...
    'Stagnation.nGenerations                    = %4.f\n',...
    'Refocus.threshold                          = %4.5f\n',...
    'Refocus.nGenerations                       = %4.f\n',...
    'Initial.killPercentage                     = %4.5f\n',...
    'Initial.numberForKill                      = %4.5f\n',...
    'Initial.numberCopy                         = %4.f\n',...
    'Selection.pressure                         = %4.5f\n',...
    'Crossover.percentage                       = %4.5f\n',...
    'Crossover.probabilityInterspecies          = %4.5f\n',...
    'Crossover.probabilityMultipoint            = %4.5f\n',...
    mutationAddNode1Str,...
    mutationAddNode2Str,...
    mutationAddNode3Str,...
    mutationAddConn1Str,...
    mutationAddConn2Str,...
    mutationAddConn3Str,...
    'Mutation.probabilityRecurrency             = %4.5f\n',...
    'Mutation.probabilityRecurrencyShrinkRate   = %4.5f\n',...
    'Mutation.probabilityRecurrencyMin          = %4.5f\n',...
    'Mutation.probabilityMutateWeight           = %4.5f\n',...
    'Mutation.probabilityMutateWeightShrinkRate = %4.5f\n',...
    'Mutation.probabilityMutateWeightMin        = %4.5f\n',...
    'Mutation.weightCap                         = %4.5f\n',...
    'Mutation.weightRange                       = %4.5f\n',...
    'Mutation.probabilityGeneReenabled          = %4.5f\n\n',...
    ],...
    loadFlag,...
    saveFlag,...
    nMaxGeneration,...
    populationSize,...
    nInputNodes,...
    nOutputNodes,...
    num2str(initialConnectedInputNodes),...
    fitnessLimit,...
    Speciation.c1,...
    Speciation.c2,...
    Speciation.c3,...
    Speciation.threshold,...
    Stagnation.threshold,...
    Stagnation.nGenerations,...
    Refocus.threshold,...
    Refocus.nGenerations,...
    Initial.killPercentage,...
    Initial.numberForKill,...
    Initial.numberCopy,...
    Selection.pressure,...
    Crossover.percentage,...
    Crossover.probabilityInterspecies,...
    Crossover.probabilityMultipoint,...
    mutationAddNode1Val{:},...
    mutationAddNode2Val,...
    mutationAddNode3Val,...
    mutationAddConn1Val{:},...
    mutationAddConn2Val,...
    mutationAddConn3Val,...
    Mutation.probabilityRecurrency,...
    Mutation.probabilityRecurrencyShrinkRate,...
    Mutation.probabilityRecurrencyMin,...
    Mutation.probabilityMutateWeight,...
    Mutation.probabilityMutateWeightShrinkRate,...
    Mutation.probabilityMutateWeightMin,...
    Mutation.weightCap,...
    Mutation.weightRange,...
    Mutation.probabilityGeneReenabled...
    );

%% Start pool

clusterName = 'local';
nWorkers = 14;
create_parpool(clusterName, nWorkers, false);

%% Evaluation function parameters

% Handle to evaluation function
evalFunHandle = @uav_evaluation_function;

% Use parallel evaluation
EvalFunParameters.useParallel = true;

% Default simulation parameters
EvalFunParameters.SimParam.absTol          = 'auto'; % 'auto'
EvalFunParameters.SimParam.relTol          = 'auto'; % 'auto'
EvalFunParameters.SimParam.stopTime        = '10';
EvalFunParameters.SimParam.solverType      = 'Variable-step'; % 'Variable-step', 'Fixed-step'
EvalFunParameters.SimParam.solver          = 'VariableStepAuto'; % 'VariableStepAuto'
EvalFunParameters.SimParam.minStep         = '0.000001'; % 'auto', '0.000001'
EvalFunParameters.SimParam.maxStep         = 'auto'; % 'auto'
EvalFunParameters.SimParam.maxConsecutiveMinStep = '100'; % '1'
EvalFunParameters.SimParam.fixedStep       = EvalFunParameters.SimParam.maxStep;
EvalFunParameters.SimParam.initialStep     = 'auto'; % 'auto'
EvalFunParameters.SimParam.simulationMode = 'accelerator'; % 'normal', 'accelerator', 'rapid-accelerator'
EvalFunParameters.SimParam.fastRestart     = 'on';

% UAV parameters
EvalFunParameters.UavParam.g        = 9.80665;
EvalFunParameters.UavParam.nMotors  = 4;
EvalFunParameters.UavParam.m        = 0.312;
EvalFunParameters.UavParam.I        = [0.0032373,0,0;0,0.0032373,0;0,0,0.0058673];
EvalFunParameters.UavParam.PP       = 0.1591*[1,1,-1,-1;1,-1,-1,1;0,0,0,0];
EvalFunParameters.UavParam.Ra       = 0.5033*ones(4,1);
EvalFunParameters.UavParam.La       = 0*1e-3*ones(4,1);
EvalFunParameters.UavParam.ke       = 7.8e-3*ones(4,1);
EvalFunParameters.UavParam.km       = 3.7e-3*ones(4,1);
EvalFunParameters.UavParam.kD       = 1.140e-7*ones(4,1);
EvalFunParameters.UavParam.Jm       = 3.357e-5*ones(4,1);
EvalFunParameters.UavParam.Bm       = 1.23e-6*ones(4,1);
EvalFunParameters.UavParam.kT       = 2.980e-6*ones(4,1);
EvalFunParameters.UavParam.VaMin    = zeros(4,1);
EvalFunParameters.UavParam.VaMax    = 10*ones(4,1);

% Input limits
EvalFunParameters.UavParam.xDesiredMin      = -100;
EvalFunParameters.UavParam.xDesiredMax      = +100;
EvalFunParameters.UavParam.yDesiredMin      = -100;
EvalFunParameters.UavParam.yDesiredMax      = +100;
EvalFunParameters.UavParam.zDesiredMin      = -100;
EvalFunParameters.UavParam.zDesiredMax      = +100;
EvalFunParameters.UavParam.psiDesiredMin    = -pi;
EvalFunParameters.UavParam.psiDesiredMax    = +pi;
EvalFunParameters.UavParam.thetaDesiredMin  = -pi/2;
EvalFunParameters.UavParam.thetaDesiredMax  = +pi/2;
EvalFunParameters.UavParam.phiDesiredMin    = -pi;
EvalFunParameters.UavParam.phiDesiredMax    = +pi;
EvalFunParameters.UavParam.xMin             = -100;
EvalFunParameters.UavParam.xMax             = +100;
EvalFunParameters.UavParam.yMin             = -100;
EvalFunParameters.UavParam.yMax             = +100;
EvalFunParameters.UavParam.zMin             = -100;
EvalFunParameters.UavParam.zMax             = +100;
EvalFunParameters.UavParam.uMin             = -30;
EvalFunParameters.UavParam.uMax             = +30;
EvalFunParameters.UavParam.vMin             = -30;
EvalFunParameters.UavParam.vMax             = +30;
EvalFunParameters.UavParam.wMin             = -30;
EvalFunParameters.UavParam.wMax             = +30;
EvalFunParameters.UavParam.udotMin          = -60;
EvalFunParameters.UavParam.udotMax          = +60;
EvalFunParameters.UavParam.vdotMin          = -60;
EvalFunParameters.UavParam.vdotMax          = +60;
EvalFunParameters.UavParam.wdotMin          = -60;
EvalFunParameters.UavParam.wdotMax          = +60;
EvalFunParameters.UavParam.psiMin           = -pi;
EvalFunParameters.UavParam.psiMax           = +pi;
EvalFunParameters.UavParam.thetaMin         = -pi/2;
EvalFunParameters.UavParam.thetaMax         = +pi/2;
EvalFunParameters.UavParam.phiMin           = -pi;
EvalFunParameters.UavParam.phiMax           = +pi;
EvalFunParameters.UavParam.pMin             = -2.4;
EvalFunParameters.UavParam.pMax             = +2.4;
EvalFunParameters.UavParam.qMin             = -2.4;
EvalFunParameters.UavParam.qMax             = +2.4;
EvalFunParameters.UavParam.rMin             = -4.72;
EvalFunParameters.UavParam.rMax             = +4.72;
EvalFunParameters.UavParam.omega1Min        = 0;
EvalFunParameters.UavParam.omega1Max        = +800;
EvalFunParameters.UavParam.omega2Min        = 0;
EvalFunParameters.UavParam.omega2Max        = +800;
EvalFunParameters.UavParam.omega3Min        = 0;
EvalFunParameters.UavParam.omega3Max        = +800;
EvalFunParameters.UavParam.omega4Min        = 0;
EvalFunParameters.UavParam.omega4Max        = +800;

% Load trajectories from trajectory_definition.m
trajectory_definition();

% Save mat file with EvalFunParameters
folderName = 'eval_fun_parameters';
if ~exist(folderName,'dir')
    mkdir(folderName);
end
save([folderName,'/eval_fun_parameters_',dateAndTime,'.mat'],'EvalFunParameters');

%% Run NEAT

% Call NEAT
[Population, bestIndividual] = neat(...
    nMaxGeneration,...
    loadGene,...
    geneFile,...
    loadFlag,...
    saveFlag,...
    populationSize,...
    nInputNodes,...
    nOutputNodes,...
    initialConnectedInputNodes,...
    fitnessLimit,...
    Speciation,...
    Initial,...
    Stagnation,...
    Refocus,...
    Selection,...
    Crossover,...
```

```
    Mutation,...
    evalFunHandle, ...
    EvalFunParameters, ...
    dateAndTime)

% Build best neural network
[neuralNet, nDelays] = build_neural_network(bestIndividual);

% Generate function
genFunctionMod(neuralNet, 'uav_best_nn_controller', ...
    'MatrixOnly', 'yes', 'ShowLinks', 'yes');
pause(0.05);

% Number of previous states
nPreviousStates = nargin(funHandle) - nInputs;

% Post-process generated function
generated_nn_processing(...
    [strrep(pwd,'\','/'),'/uav_best_nn_controller.m'], ...
    'uav_best_nn_controller', nPreviousStates);

% View net
view(neuralNet);

% Terminate diary
diary('off');
```

B. Trajectory Definition

```
% This file contains the trajectories used for training of the neural network

if ~exist('iTrajSelection','var')
    iTrajSelection = [1,2,6,8,9,14,16];
end

%% Trajectory 1 - zDesired - zInitial = -90 m

iTraj = 1;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = -90; % Start position (m)
tf      = 15;  % Finish time (s)
qf      = -90; % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;-90];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5; % 300
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5; % 300
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2; % 300
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3; % 150
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3; % 150
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3; % 150
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 2 - zDesired - zInitial = -50 m

iTraj = 2;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = -50; % Start position (m)
tf      = 15;  % Finish time (s)
qf      = -50; % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;-50];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 3 - zDesired - zInitial = -25 m

iTraj = 3;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = -25; % Start position (m)
tf      = 15;  % Finish time (s)
qf      = -25; % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 4 - zDesired - zInitial = 0 m

iTraj = 4;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;
```

```

%% Trajectory 5 - zDesired - zInitial = 25 m

iTraj = 5;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;    % Start time (s)
qs      = 25;   % Start position (m)
tf      = 15;   % Finish time (s)
qf      = 25;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;25];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 6 - zDesired - zInitial = 50 m

iTraj = 6;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 50;  % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 50;  % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;50];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 7 - zDesired - zInitial = 90 m

iTraj = 7;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 90;  % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 90;  % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;90];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 8 - zDesired - zInitial = 0 m - xInitial = 50 m - yInitial = 50 m

iTraj = 8;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = 50*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = 50*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [50;50;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 9 - zDesired - zInitial = 0 m - xInitial = -50 m - yInitial = -50 m

iTraj = 9;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = -50*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = -50*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [-50;-50;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
```

```
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 10 - zDesired - zInitial = 0 m - xInitial = 50 m - yInitial = 0 m

iTraj = 10;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = 50*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = 0*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [50;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 11 - zDesired - zInitial = 0 m - xInitial = 0 m - yInitial = 50 m

iTraj = 11;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = 0*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = 50*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;50;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 12 - zDesired - zInitial = 0 m - xInitial = -50 m - yInitial = 0 m

iTraj = 12;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = -50*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = 0*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [-50;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 13 - zDesired - zInitial = 0 m - xInitial = 0 m - yInitial = -50 m

iTraj = 13;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = 0*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = -50*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;-50;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 14 - zDesired - zInitial = 0 m - xInitial = 0 m - yInitial = 0 m - psiInitial = 80

iTraj = 14;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = 80*DEG_2_RAD*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
```

```
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = [80*DEG_2_RAD;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 15 - zDesired - zInitial = 0 m - xInitial = 0 m - yInitial = 0 m - psiInitial = 160

iTraj = 15;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = 160*DEG_2_RAD*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = [160*DEG_2_RAD;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 16 - zDesired - zInitial = 0 m - xInitial = 0 m - yInitial = 0 m - psiInitial = -80

iTraj = 16;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = -80*DEG_2_RAD*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = [-80*DEG_2_RAD;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 3;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 1.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.25;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.25;

%% Trajectory 17 - zDesired - zInitial = 0 m - xInitial = 0 m - yInitial = 0 m - psiInitial = -160

iTraj = 17;
nPoints = 1000;
trajectoryType = 'Linear';
ts      = 0;   % Start time (s)
qs      = 0;   % Start position (m)
tf      = 15;  % Finish time (s)
qf      = 0;   % Finish position (m)
[zDesiredTime, zDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, tf, qf);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zDesired;
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = -160*DEG_2_RAD*ones(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = zDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-3.5,+3.5]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = [-160*DEG_2_RAD;0;0];
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 3.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 3.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.5;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.5;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.5;

%% Trajectory 18 - xDesired

iTraj = 18;
nPoints = 1000;
trajectoryType = 'Poly3';
ts      = 0;  % Start time (s)
qs      = 0;  % Start position (m)
qsDot   = 0;  % Start position (m)
tf      = 5;  % Finish time (s)
qf      = 1;  % Finish position (m)
qfDot   = 0;  % Finish position (m)
[xDesiredTime, xDesired]  = trajectory_generation(nPoints, trajectoryType, ...
    ts, qs, qsDot, tf, qf, qfDot);
EvalFunParameters.Trajectory(iTraj).XDesired.t          = xDesiredTime;
EvalFunParameters.Trajectory(iTraj).XDesired.val        = xDesired;
EvalFunParameters.Trajectory(iTraj).XDesired.tol        = [-80,+80]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).YDesired.t          = xDesiredTime;
EvalFunParameters.Trajectory(iTraj).YDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).YDesired.tol        = [-30,+30]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).ZDesired.t          = xDesiredTime;
EvalFunParameters.Trajectory(iTraj).ZDesired.val        = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ZDesired.tol        = [-60,+60]*MM_2_M;
EvalFunParameters.Trajectory(iTraj).PsiDesired.t        = xDesiredTime;
EvalFunParameters.Trajectory(iTraj).PsiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PsiDesired.tol      = [-5.0,+5.0]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.t      = xDesiredTime;
EvalFunParameters.Trajectory(iTraj).ThetaDesired.val    = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).ThetaDesired.tol    = [-7.0,+7.0]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).PhiDesired.t        = xDesiredTime;
EvalFunParameters.Trajectory(iTraj).PhiDesired.val      = zeros(1,nPoints);
EvalFunParameters.Trajectory(iTraj).PhiDesired.tol      = [-5.0,+5.0]*DEG_2_RAD;
EvalFunParameters.Trajectory(iTraj).IniCond.omegaInitial    = 0*ones(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegaDelay      = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaInitial       = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.iaDelay         = zeros(4,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtInitial      = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.vPtDelay        = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.xPInitial       = [0;0;90];
EvalFunParameters.Trajectory(iTraj).IniCond.xPDelay         = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtInitial  = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.omegabtDelay    = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXInitial = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).IniCond.eulerZYXDelay   = zeros(3,1);
EvalFunParameters.Trajectory(iTraj).maxViolationX           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationY           = 1.5;
EvalFunParameters.Trajectory(iTraj).maxViolationZ           = 2;
EvalFunParameters.Trajectory(iTraj).maxViolationPsi         = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationTheta       = 0.3;
EvalFunParameters.Trajectory(iTraj).maxViolationPhi         = 0.3;
EvalFunParameters.Trajectory(iTraj).violationFactorX        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorY        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorZ        = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorPsi      = 5;
EvalFunParameters.Trajectory(iTraj).violationFactorTheta    = 2;
EvalFunParameters.Trajectory(iTraj).violationFactorPhi      = 2;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdX = 3.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdY = 3.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdZ = 5.0;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPsi   = 0.5;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdTheta = 0.5;
EvalFunParameters.Trajectory(iTraj).violationTimeThresholdPhi   = 0.5;

%% Trajectory selection

EvalFunParameters.Trajectory = EvalFunParameters.Trajectory(iTrajSelection);
```

C. NEAT

```
%% neat() (main)
function [Population, bestIndividual, varargout] = neat(...
    nMaxGeneration,...
    loadGene,...
    geneFile,...
    loadFlag,...
    saveFlag,...
    populationSize,...
    nInputNodes,...
    nOutputNodes,...
    initialConnectedInputNodes,...
    fitnessLimit,...
    Speciation,...
    Initial,...
    Stagnation, ...
    Refocus, ...
    Selection, ...
    Crossover, ...
    Mutation, ...
    evalFunHandle, ...
    EvalFunParameters, ...
    dateAndTime, ...
    varargin)

%% Initialization

% Timing
tic;

% Initialize vectors
nAverageNonDisabledConnections = [];
nAverageHiddenNodes = [];
maxOverallFitness = [];
nAverageDisabledConnections = [];
nSpeciesArray = [];
meanOverallFitness = [];
meanRecurrentConnection = [];
nIndPerSpecie = [];

% Initialize Speciation structure. It will contain various information on
% single species. This data will be used for fitness sharing, reproduction,
% and for visualisation purposes.
SpeciesRecord(1).id = 0; % Consecutive species ID's
SpeciesRecord(1).nIndividuals = 0; % Number of individuals in species
% generationRecord matrix will be 4 rows by (number of generations
% existent) columns, will contain (from top to bottom):
% - Number of generation
% - Mean raw fitness
% - Max raw fitness
% - Index of individual in population which has produced max raw fitness
SpeciesRecord(1).generationRecord = [];

if loadFlag == 0 && loadGene == 0

    % Call function to create Initial population
    % for information about the make-up of the population structure and the
    % innovationRecord, look at initialize_population()
    [Population, innovationRecord] = initialize_population(...
        populationSize, nInputNodes, nOutputNodes, initialConnectedInputNodes);

    % Initial speciation
    [Population, SpeciesRecord] = ...
        initial_speciation(initialConnectedInputNodes, Population, nOutputNodes, ...
        SpeciesRecord, Speciation, loadGene);

    % Generation counter
    iGeneration = 1;

elseif loadFlag == 0 && loadGene == 1

    % Load gene
    TmpStruct = load(geneFile,'Population');
    PopulationTmp = TmpStruct.Population;
    if numel([PopulationTmp(:).fitness]) == 1
        startNodeGenes       = PopulationTmp.nodeGenes;
        startConnectionGenes = PopulationTmp.connectionGenes;
    else
        tmpFitness           = [PopulationTmp(:).fitness];
        [~, iTopIndividual]  = max(tmpFitness);
        bestIndividual       = PopulationTmp(iTopIndividual);
        startNodeGenes       = bestIndividual.nodeGenes;
        startConnectionGenes = bestIndividual.connectionGenes;
    end

    % Initialize population with gene
    [Population,innovationRecord] = initialize_population_custom_gene(...
        populationSize,...
        startNodeGenes,...
        startConnectionGenes);

    % Initial speciation
    [Population, SpeciesRecord] = ...
        initial_speciation(initialConnectedInputNodes, Population, nOutputNodes, ...
        SpeciesRecord, Speciation, loadGene);

    % Generation counter
    iGeneration = 1;

else

    % Backup mutation schedulings
    probabilityAddNodeSchedulingTmp       = Mutation.probabilityAddNodeScheduling;
    probabilityAddConnectionSchedulingTmp = Mutation.probabilityAddConnectionScheduling;
    if true
        weightCapTmp   = Mutation.weightCap;
        weightRangeTmp = Mutation.weightRange;
    end

    % Start with saved version of evolution
    load('neatsave.mat');

    % Restore mutation schedulings
    Mutation.probabilityAddNodeScheduling       = probabilityAddNodeSchedulingTmp;
    Mutation.probabilityAddConnectionScheduling = probabilityAddConnectionSchedulingTmp;
    if true
        Mutation.weightCap   = weightCapTmp;
        Mutation.weightRange = weightRangeTmp;
    end

end

if Mutation.probabilityAddNodeScheduling(1) ~= -1
    Mutation.probabilityAddNode = Mutation.probabilityAddNodeScheduling(2,1);
end

if Mutation.probabilityAddConnectionScheduling(1) ~= -1
    Mutation.probabilityAddConnection = Mutation.probabilityAddConnectionScheduling(2,1);
end

%% Generational Loop

print_generation(iGeneration);
flagSolution = 0;
while (iGeneration < nMaxGeneration) && (flagSolution == 0)

    % Backup copies of current generation
    if saveFlag == 1
        save('neatsave.mat','Population','iGeneration','innovationRecord',...
            'SpeciesRecord','nAverageNonDisabledConnections','nAverageHiddenNodes',...
            'maxOverallFitness','dateAndTime','Mutation','nAverageDisabledConnections',...
            'nSpeciesArray','meanOverallFitness','meanRecurrentConnection','nIndPerSpecie');
        if exist('figHandle','var')
            if ishandle(figHandle)
                savefig(figHandle,'neatfig.fig');
            end
        end
    end

    % Call evaluation function (in this case XOR), fitnesses of individuals
    % will be stored in Population(:).fitness.
    % IMPORTANT: reproduction assumes an (all positive!) evaluation function
    % where a higher value means better fitness (in other words, the
    % algorithm is geared towards maximizing a fitness function which can
    % only assume values between 0 and +Inf)
    % Population = xor_experiment(Population);
    % Population = fulladder_experiment(Population);
    if isempty(EvalFunParameters)
        Population = evalFunHandle(Population, fitnessLimit);
    else
        nSimPoints = 5000;
        SimData = initialize_sim_data_structure(EvalFunParameters, populationSize, ...
            nSimPoints);
        EvalFunParameters.iGeneration = iGeneration;
        [Population, SimData] = evalFunHandle(Population, EvalFunParameters, ...
            fitnessLimit, SimData);
    end

    % Check for stagnation
    [maxFitnessesCurrentGeneration, SpeciesRecord] = ...
        stagnation(SpeciesRecord, Population, Stagnation, iGeneration);

    % Check for refocus
    [SpeciesRecord] = ...
        refocus(maxFitnessesCurrentGeneration, SpeciesRecord, Refocus);

    % Save current generation
    folderName = 'neat_data';
    if ~exist(folderName,'dir')
        mkdir(folderName);
    end
    save([folderName,'/neatsave_',dateAndTime,'_',num2str(iGeneration,'%03.0f'),'.mat'],...
        'Population','iGeneration','innovationRecord','SpeciesRecord');

    % Compute stats
    [nAverageNonDisabledConnections, nAverageHiddenNodes, maxFitness,...
        maxOverallFitness, nAverageDisabledConnections, ...
        nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecie] = ...
        compute_stats(Population, populationSize, iGeneration, ...
        SpeciesRecord, nAverageNonDisabledConnections, nAverageHiddenNodes, ...
        maxOverallFitness, nAverageDisabledConnections, ...
        nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecie);

    % Create figure handle and axes if don't exist or have been deleted
    if exist('figHandle','var')
        [figHandle, AxHandle] = neat_plot_helper(loadFlag, figHandle, AxHandle);
        figHandle.CloseRequestFcn = @closereqmod;
        figHandle.Resize = 'off';
    else
        [figHandle, AxHandle] = neat_plot_helper(loadFlag);
        figHandle.CloseRequestFcn = @closereqmod;
        figHandle.Resize = 'off';
    end

    % Visualisation fitness & species
    visualization(figHandle, AxHandle, nAverageNonDisabledConnections, ...
        nAverageHiddenNodes, maxOverallFitness, nAverageDisabledConnections, ...
        nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecie);

    % Find best individual of this generation
    tmpFitness = [Population(:).fitness];
    [~, iTopIndividual] = max(tmpFitness);
    bestIndividual = Population(iTopIndividual);

    % Visualisation of simulation results of the best individual
    if exist('SimData','var')
        if ~exist('figHandleSim','var'); figHandleSim = []; end
        if ~exist('figHandleSimError','var'); figHandleSimError = []; end
        [figHandleSim, figHandleSimError] = ...
            sim_visualization(figHandleSim, figHandleSimError, SimData, ...
            iTopIndividual, EvalFunParameters);
        figHandleSim.CloseRequestFcn = @closereqmod;
        figHandleSimError.CloseRequestFcn = @closereqmod;
        figHandleSim.Resize = 'off';
        figHandleSimError.Resize = 'off';
    end

    % Visualisation of neural network of the best individual
    if exist('figHandleNeuralNet','var')
        figHandleNeuralNet = nn_visualization(Population, figHandleNeuralNet);
        figHandleNeuralNet.CloseRequestFcn = @closereqmod;
        figHandleNeuralNet.Resize = 'off';
    else
        figHandleNeuralNet = nn_visualization(Population);
        figHandleNeuralNet.CloseRequestFcn = @closereqmod;
        figHandleNeuralNet.Resize = 'off';
    end

    % Check for solution
    flagSolution = solution_check(maxFitness, fitnessLimit, flagSolution);

    % We call reproduce if there's not solution yet
    if flagSolution == 0
        % Call reproduction function with parameters, current population
        % and species record, returns new population, new species record
        % and new innovation record
        [Population, SpeciesRecord, innovationRecord] = ...
            reproduce(Population, SpeciesRecord, innovationRecord, Initial,...
            Selection, Crossover, Mutation, Speciation, iGeneration, populationSize);
        % Display time
        display_elapsed_time(toc());
    end

    % Increment generational counter
    iGeneration = iGeneration + 1;

    % Print generation to command window
    print_generation(iGeneration);

    % Reduce mutation probabilities (if shrink rate > 0)
    Mutation = shrink_mutation(Mutation, iGeneration);

end

end

%% initialize_sim_data_structure()
function SimData = ...
    initialize_sim_data_structure(EvalFunParameters, populationSize, nSimPoints)

cellArrayTmp = cell(1,numel(EvalFunParameters.Trajectory));
[cellArrayTmp{:}]                   = deal(zeros(nSimPoints,1));
[SimData(1:populationSize).t]       = deal(cellArrayTmp);
[SimData(1:populationSize).xDesired]    = deal(cellArrayTmp);
[SimData(1:populationSize).yDesired]    = deal(cellArrayTmp);
[SimData(1:populationSize).zDesired]    = deal(cellArrayTmp);
[SimData(1:populationSize).psiDesired]  = deal(cellArrayTmp);
[SimData(1:populationSize).thetaDesired]    = deal(cellArrayTmp);
```

```
[SimData(1:populationSize).phiDesired]      = deal(cellArrayTmp);
[SimData(1:populationSize).x]               = deal(cellArrayTmp);
[SimData(1:populationSize).y]               = deal(cellArrayTmp);
[SimData(1:populationSize).z]               = deal(cellArrayTmp);
[SimData(1:populationSize).psi]             = deal(cellArrayTmp);
[SimData(1:populationSize).theta]           = deal(cellArrayTmp);
[SimData(1:populationSize).phi]             = deal(cellArrayTmp);
[SimData(1:populationSize).Va1]             = deal(cellArrayTmp);
[SimData(1:populationSize).Va2]             = deal(cellArrayTmp);
[SimData(1:populationSize).Va3]             = deal(cellArrayTmp);
[SimData(1:populationSize).Va4]             = deal(cellArrayTmp);

end

%% display_elapsed_time()
function display_elapsed_time(elapsedTime)

% elapsedTime = toc;
elapsedTimeHours = floor(elapsedTime/3600);
elapsedTimeMins  = floor((elapsedTime - 3600*elapsedTimeHours)/60);
elapsedTimeSecs  = floor((elapsedTime - 3600*elapsedTimeHours - 60*elapsedTimeMins));
fprintf('Elapsed time is %02.0fh%02.0fm%02.0fs\n\n',...
    elapsedTimeHours, elapsedTimeMins, elapsedTimeSecs);

end

%% sim_visualization()
function [figHandleSim, figHandleSimError] = ...
    sim_visualization(figHandleSim, figHandleSimError, SimData, iTopIndividual, ...
    EvalFunParameters)

if exist('SimData','var')

    if ~isempty(figHandleSim) && ~isempty(figHandleSimError)
        if ishandle(figHandleSim) && ishandle(figHandleSimError)
            clf(figHandleSim);
            clf(figHandleSimError);
        elseif ~ishandle(figHandleSim) || ~ishandle(figHandleSimError)
            if ishandle(figHandleSim); close(figHandleSim); end;
            if ishandle(figHandleSimError); close(figHandleSimError); end;
            delete figHandleSim;
            delete figHandleSimError;
        end
    end

    nTraj = numel(EvalFunParameters.Trajectory);
    [Data(1:nTraj).t]               = deal(SimData(iTopIndividual).t{:});
    [Data(1:nTraj).xDesired]        = deal(SimData(iTopIndividual).xDesired{:});
    [Data(1:nTraj).yDesired]        = deal(SimData(iTopIndividual).yDesired{:});
    [Data(1:nTraj).zDesired]        = deal(SimData(iTopIndividual).zDesired{:});
    [Data(1:nTraj).psiDesired]      = deal(SimData(iTopIndividual).psiDesired{:});
    [Data(1:nTraj).thetaDesired]    = deal(SimData(iTopIndividual).thetaDesired{:});
    [Data(1:nTraj).phiDesired]      = deal(SimData(iTopIndividual).phiDesired{:});
    [Data(1:nTraj).x]               = deal(SimData(iTopIndividual).x{:});
    [Data(1:nTraj).y]               = deal(SimData(iTopIndividual).y{:});
    [Data(1:nTraj).z]               = deal(SimData(iTopIndividual).z{:});
    [Data(1:nTraj).psi]             = deal(SimData(iTopIndividual).psi{:});
    [Data(1:nTraj).theta]           = deal(SimData(iTopIndividual).theta{:});
    [Data(1:nTraj).phi]             = deal(SimData(iTopIndividual).phi{:});
    [Data(1:nTraj).Va1]             = deal(SimData(iTopIndividual).Va1{:});
    [Data(1:nTraj).Va2]             = deal(SimData(iTopIndividual).Va2{:});
    [Data(1:nTraj).Va3]             = deal(SimData(iTopIndividual).Va3{:});
    [Data(1:nTraj).Va4]             = deal(SimData(iTopIndividual).Va4{:});

    customLegend = cell(1,nTraj);
    for jTraj = 1:nTraj; customLegend{jTraj} = ['Traj ',num2str(jTraj)]; end;

    if ~isempty(figHandleSim) && ~isempty(figHandleSimError) && ...
            ishandle(figHandleSim) && ishandle(figHandleSimError)
        [figHandleSim, figHandleSimError] = ...
            plot_uav_sim(Data, true, false, false, true, false, '', '', true, ...
            false, customLegend, true, figHandleSim, figHandleSimError);
        drawnow;
    else
        [figHandleSim, figHandleSimError] = ...
            plot_uav_sim(Data, true, false, false, true, false, '', '', true, ...
            false, customLegend, true);
        drawnow;
    end

end

end

%% nn_visualization()
function figHandleNeuralNet = nn_visualization(Population, varargin)

if nargin == 2
    figHandleNeuralNet = varargin{1};
end

if exist('figHandleNeuralNet','var') && ishandle(figHandleNeuralNet)
    clf(figHandleNeuralNet);
end

if exist('figHandleNeuralNet','var') && ishandle(figHandleNeuralNet)
    figHandleNeuralNet = plot_neural_network([], ...
        [], [], [], 1, true, false, Population, figHandleNeuralNet);
    drawnow;
else
    figHandleNeuralNet = plot_neural_network([], ...
        [], [], [], 1, true, false, Population);
    drawnow;
end

end

%% neat_plot_helper()
function [figHandle, AxHandle] = neat_plot_helper(loadFlag, varargin)

% This function simply manages the figure and axes of neat plot.

if nargin == 3
    figHandle = varargin{1};
    AxHandle = varargin{2};
end

marginSubPlot = 0.10;
subplotSettings = {marginSubPlot, 'NextPlot', 'replacechildren', 'Tag'};

if ~exist('figHandle','var') && loadFlag ~= 1

    figHandle = create_figure(800, 700, 'on');
    AxHandle.subPlot1 = subplot_tight(3,3,[1,2], subplotSettings{:}, 'subPlot1');
    AxHandle.subPlot2 = subplot_tight(3,3,3, subplotSettings{:}, 'subPlot2');
    AxHandle.subPlot3 = subplot_tight(3,3,4, subplotSettings{:}, 'subPlot3');
    AxHandle.subPlot4 = subplot_tight(3,3,5, subplotSettings{:}, 'subPlot4');
    AxHandle.subPlot5 = subplot_tight(3,3,6, subplotSettings{:}, 'subPlot5');
    AxHandle.subPlot6 = subplot_tight(3,3,7, subplotSettings{:}, 'subPlot6');
    AxHandle.subPlot7 = subplot_tight(3,3,8, subplotSettings{:}, 'subPlot7');
    AxHandle.subPlot8 = subplot_tight(3,3,9, subplotSettings{:}, 'subPlot8');

elseif ~exist('figHandle','var') && loadFlag == 1

    figHandle = openfig('neatfig.fig');
    AxHandle.subPlot1 = findobj(figHandle, 'Type', 'axes', 'Tag', 'subPlot1');
    AxHandle.subPlot2 = findobj(figHandle, 'Type', 'axes', 'Tag', 'subPlot2');
    AxHandle.subPlot3 = findobj(figHandle, 'Type', 'axes', 'Tag', 'subPlot3');
    AxHandle.subPlot4 = findobj(figHandle, 'Type', 'axes', 'Tag', 'subPlot4');
    AxHandle.subPlot5 = findobj(figHandle, 'Type', 'axes', 'Tag', 'subPlot5');
    AxHandle.subPlot6 = findobj(figHandle, 'Type', 'axes', 'Tag', 'subPlot6');
    AxHandle.subPlot7 = findobj(figHandle, 'Type', 'axes', 'Tag', 'subPlot7');
    AxHandle.subPlot8 = findobj(figHandle, 'Type', 'axes', 'Tag', 'subPlot8');

end

% In case the figure has been closed
if exist('figHandle','var') && ishandle(figHandle)

    fprintf('\nNEAT plot window has been closed. It will reopened the next generation.\n');
    delete AxHandle; clear AxHandle;
    delete figHandle; clear figHandle;
    pause(0.1);
    figHandle = create_figure(800, 700, 'on');
    drawnow;
    set(0, 'currentfigure', figHandle);
    AxHandle.subPlot1 = subplot_tight(3,3,[1,2], subplotSettings{:}, 'subPlot1');
    set(0, 'currentfigure', figHandle);
    AxHandle.subPlot2 = subplot_tight(3,3,3, subplotSettings{:}, 'subPlot2');
    set(0, 'currentfigure', figHandle);
    AxHandle.subPlot3 = subplot_tight(3,3,4, subplotSettings{:}, 'subPlot3');
    set(0, 'currentfigure', figHandle);
    AxHandle.subPlot4 = subplot_tight(3,3,5, subplotSettings{:}, 'subPlot4');
    set(0, 'currentfigure', figHandle);
    AxHandle.subPlot5 = subplot_tight(3,3,6, subplotSettings{:}, 'subPlot5');
    set(0, 'currentfigure', figHandle);
    AxHandle.subPlot6 = subplot_tight(3,3,7, subplotSettings{:}, 'subPlot6');
    set(0, 'currentfigure', figHandle);
    AxHandle.subPlot7 = subplot_tight(3,3,8, subplotSettings{:}, 'subPlot7');
    set(0, 'currentfigure', figHandle);
    AxHandle.subPlot8 = subplot_tight(3,3,9, subplotSettings{:}, 'subPlot8');

end

end

%% initialize_population()
function [Population,innovationRecord] = initialize_population(...
    populationSize,...
    nInputNodes,...
    nOutputNodes,...
    initialConnectedInputNodes)

% nodeGenes is array 4 rows * (nInputNodes + nOutputNodes +
% hidden-nodes (not existent in initial population) + 1 (bias-node)) columns
% nodeGenes contains:
% - Consecutive node ID's (upper row)
% - Node type (lower row) 1 = input 2 = output 3 = hidden 4 = bias
% - Node input state
% - Node output state (used for evaluation, all input states zero
%   initially, except bias node, which is always 1)
%
% connectionGenes is array 5 rows * nConnections columns
% from top to bottom, those five rows contain:
% - Innovation number
% - Connection from
% - Connection to
% - Weight
% - Enable bit
% The rest of the elements in the structure for an individual should be
% self-explanatory
%
% innovationRecord tracks innovations in a 5 rows by (number of
% innovations) columns matrix, contains:
% - innovation number
% - connectFromNode for this innovation
% - connectToNode for this innovation
% - The new node (if it is a new node mutation, then this node will appear in
%   the 4th row when it is first connected. There will always be two
%   innovations with one node mutation, since there is a connection to and
%   from the new node. In the initial population, this will be abbreviated to
%   the node with the highest number appearing in the last column of the record,
%   since only this is needed as starting point for the rest of the algorithm)
% - 5th row is generation this innovation occured (generation is assumed
%   to be zero for the innovations in the initial population)

% Compute number and matrix of initial connections (all connections between
% output nodes and the nodes listed in initialConnectedInputNodes). In
% nConnections, we add 1 since the bias is connected. vectorConnectionFrom is a
% row vector (i.e. one row, multiple columns) consists of a nOutputNodes times
% repeated row vector containing the ID of the initial connected input plus the
% bias node ID. For example, if there are 3 inputs (all connected) and 3
% outputs, the vector will be:
% vectorConnectionFrom = [[1,2,3,4],[1,2,3,4],[1,2,3,4]]
% where 4 is the bias' node ID.
nConnections         = (length(initialConnectedInputNodes)+1)*nOutputNodes;
vectorConnectionFrom = repmat([initialConnectedInputNodes,nInputNodes+1],[1,nOutputNodes]);
% vectorConnectionTo contains the node ID to which are connected every node
% listed in vectorConnectionFrom. For instance, if we take the example above
% with 3 inputs (all connected) and 3 outputs, we'd have:
% vectorConnectionTo = [[5,5,5,5],[6,6,6,6],[7,7,7,7]]
vectorConnectionTo   = [];
% Outputs' node IDs begin at the ID after the bias' ID which is why we take
% nInputNodes + 2. For instance, if we have 3 inputs + 1 bias, the IDs 1,2,3,4
% are taken, so we start at 3 + 2 = 5 for the first output ID.
iOutputNodeStart = nInputNodes + 2;
iOutputNodeEnd   = nInputNodes + 1 + nOutputNodes;
for iOutputNode  = iOutputNodeStart:iOutputNodeEnd
    % There is a +1 because the bias is connected.
    vectorConnectionTo = ...
        [vectorConnectionTo, ...
        iOutputNode*ones(1,length(initialConnectedInputNodes)+1)];
end
% Connection matrix is a matrix of two rows by (number of connections) columns
connectionMatrix = [...
    vectorConnectionFrom;
    vectorConnectionTo];

% We build the initial population nodeGenes and codeGenes
for iIndividual = 1:populationSize
    Population(iIndividual).nodeGenes = [...
        1:(nInputNodes+1+nOutputNodes);
        ones(1,nInputNodes),4,2*ones(1,nOutputNodes);
        zeros(1,nInputNodes),1,zeros(1,nOutputNodes);
        zeros(1,nInputNodes+1+nOutputNodes)];
    % All weights uniformly distributed in [-1,+1], all connections enabled
    Population(iIndividual).connectionGenes = [...
        1:nConnections;
        connectionMatrix;
        rand(1,nConnections)*2-1;
        ones(1,nConnections)];
    Population(iIndividual).fitness = 0;
    Population(iIndividual).species = 0;
end
innovationRecord = [...
    Population(populationSize).connectionGenes(1:3,:);
    zeros(size(Population(populationSize).connectionGenes(1:2,:)))];
% Highest node ID for initial population
innovationRecord(4,size(innovationRecord,2)) = max(Population(1).nodeGenes(1,:));

end

%% initialize_population_custom_gene()
function [Population,innovationRecord] = initialize_population_custom_gene(...
    populationSize,...
    startNodeGenes,...
    startConnectionGenes)

% See initialize_population() for documentation

% Number of connections
nConnections      = numel(startConnectionGenes(1,:));

% We build the initial population nodeGenes and codeGenes
```

```
for iIndividual = 1:populationSize
    Population(iIndividual).nodeGenes = startNodeGenes;
    % All weights uniformly distributed in [-1,+1]
    Population(iIndividual).connectionGenes = [...
        startConnectionGenes(1:3,:);
        rand(1,nConnections)*2-1;
        startConnectionGenes(5,:);];
    Population(iIndividual).fitness = 0;
    Population(iIndividual).species = 0;
end
innovationRecord = [...
    Population(populationSize).connectionGenes(1:3,:);
    zeros(size(Population(populationSize).connectionGenes(1:2,:)))];
% Highest node ID for initial population
innovationRecord(4,size(innovationRecord,2)) = max(Population(1).nodeGenes(1,:));

end

%% initial_speciation()
function [Population, SpeciesRecord] = ...
    initial_speciation(initialConnectedInputNodes, Population, nOutputNodes, ...
    SpeciesRecord, Speciation, loadGene)

% This function performs initial speciation. That is, it assigns each
% individual in a specie according to their compatibility distance.

% Number of connections
if loadGene == 0
    nConnections = (length(initialConnectedInputNodes)+1)*nOutputNodes;
else
    nConnections = numel(Population(1).connectionGenes(1,:));
end
% Put first individual in species one and update SpeciesRecord
Population(1).species = 1;
% Species reference matrix (abbreviated, only weights, since there are
% no topology differences in initial population)
matrixReferenceIndividuals = Population(1).connectionGenes(4,:);
SpeciesRecord(1).id = 1;
SpeciesRecord(1).nIndividuals = 1;

% Loop through rest of individuals and either assign to existing species
% or create new species and use first individual of new species as
% reference
for iIndividual = 2:size(Population,2);
    assignedExistingSpeciesFlag = 0;
    newSpeciesFlag = 0;
    iSpecie = 1;
    % Loops through the existing species, terminates when either the
    % individual is assigned to existing species or there are no more
    % species to test it against, which means it is a new species
    while (assignedExistingSpeciesFlag == 0) && (newSpeciesFlag == 0)
        % Computes compatibility distance, abbreviated, only average
        % weight distance considered
        averageWeightDiffMatchingGenes = ...
            sum(...
            abs(Population(iIndividual).connectionGenes(4,:) - ...
            matrixReferenceIndividuals(iSpecie,:)) ...
            );
        distance = Speciation.c3*averageWeightDiffMatchingGenes/nConnections;
        % If within threshold, assign to the existing species
        if distance < Speciation.threshold
            Population(iIndividual).species = iSpecie;
            assignedExistingSpeciesFlag = 1;
            SpeciesRecord(iSpecie).nIndividuals = SpeciesRecord(iSpecie).nIndividuals+1;
        end
        iSpecie = iSpecie + 1;
        % Outside of species references, must be new species
        if (iSpecie > size(matrixReferenceIndividuals,1)) && (assignedExistingSpeciesFlag == 0)
            newSpeciesFlag = 1;
        end
    end
    % Check for new species, if it is, update the SpeciesRecord and
    % use individual as reference for new species
    if newSpeciesFlag == 1
        Population(iIndividual).species = iSpecie;
        matrixReferenceIndividuals = [...
            matrixReferenceIndividuals;
            Population(iIndividual).connectionGenes(4,:)];
        SpeciesRecord(iSpecie).id = iSpecie;
        % If number individuals in a species is zero, that species is extinct
        SpeciesRecord(iSpecie).nIndividuals = 1;
    end
end

end

%% stagnation()
function [maxFitnessesCurrentGeneration, SpeciesRecord] = ...
    stagnation(SpeciesRecord, Population, Stagnation, iGeneration)

% This function checks for stagnation. We also compute mean and max raw
% fitnesses in each species and store in SpeciesRecord.generationRecord

% Number of species in current generation
nSpecies = size(SpeciesRecord,2);

% Initialize maxFitnessesCurrentGeneration vector
maxFitnessesCurrentGeneration = zeros(1,nSpecies);

% The vector species contains the species index of each individual in the
% population
popSpecies = [Population(:).species];

% The vector fitness contains the fitness of each individual in the
% population
popFitness = [Population(:).fitness];

% Cycle through each specie
for iSpecie = 1:nSpecies
    if SpeciesRecord(iSpecie).nIndividuals > 0
        % Max fitness of current specie
        [maxFitness, iMaxFitness] = max((popSpecies == iSpecie).*popFitness);
        % Mean fitness of current specie
        meanFitness = ...
            sum((popSpecies == iSpecie).*popFitness)/SpeciesRecord(iSpecie).nIndividuals;
        % Number of generation in current specie
        nGenerations = size(SpeciesRecord(iSpecie).generationRecord,2);
        % Compute stagnation vector (last Stagnation.nGenerations-1 max
        % fitnesses plus current fitness)
        if nGenerations > Stagnation.nGenerations-2
            iGenerationStart = nGenerations - Stagnation.nGenerations + 2;
            lastMaxFitnesses = ...
                SpeciesRecord(iSpecie).generationRecord(3,iGenerationStart:nGenerations);
            stagnationVector = [lastMaxFitnesses,maxFitness];
            % Average max fitnesses change for the last Stagnation.nGenerations generations
            averageMaxFitnessesChange = abs(stagnationVector-mean(stagnationVector));
            % Number of stagned generation in current specie
            nStagnedGenerations = sum(averageMaxFitnessesChange < Stagnation.threshold);
            % Check for stagnation
            if nStagnedGenerations == Stagnation.nGenerations
                % Set mean fitness to small value to eliminate species
                % (cannot be set to 0, if only one species is present,
                % we would have divide by zero in fitness sharing.
                % anyways, with only one species present, we have to keep it)
                meanFitness = 0.01;
            end
        end
        % We add new generation to current specie
        SpeciesRecord(iSpecie).generationRecord = [...
            SpeciesRecord(iSpecie).generationRecord,...
            [iGeneration;meanFitness;maxFitness;iMaxFitness]];
        maxFitnessesCurrentGeneration(1,iSpecie) = maxFitness;
    end
end

end

%% refocus()
function [SpeciesRecord] = ...
    refocus(maxFitnessesCurrentGeneration, SpeciesRecord, Refocus)

% In rare cases when the fitness of the entire population does not
% improve for more than Refocus.nGenerations generations, only the top
% two species are allowed to reproduce, refocusing the search into the
% most promising spaces

% Index of the top specie
[~, iTopSpecie] = max(maxFitnessesCurrentGeneration);

% Number of generations in the top specie
nGenerationsTopSpecie = size(SpeciesRecord(iTopSpecie).generationRecord,2);

% Check for refocus
if nGenerationsTopSpecie > Refocus.nGenerations
    index1 = nGenerationsTopSpecie-Refocus.nGenerations;
    index2 = nGenerationsTopSpecie;
    % Max fitnesses from (nGenerationsTopSpecie-Refocus.nGenerations) to
    % nGenerationsTopSpecie of top specie
    maxFitnessesTopSpecie = SpeciesRecord(iTopSpecie).generationRecord(3,index1:index2);
    % Average max fitnesses change for the top specie over the last
    % nGenerationsTopSpecie generations
    averageMaxFitnessesChangeTopSpecie = abs(maxFitnessesTopSpecie - mean(maxFitnessesTopSpecie));
    % Number of stagned generation in current specie
    nStagnedGenerationsTopSpecie = sum(averageMaxFitnessesChangeTopSpecie < Refocus.threshold);
    % Check if number of stagned generations in the top specie is equal
    % to the threshold refocus generations number
    if nStagnedGenerationsTopSpecie == Refocus.nGenerations
        % Sort max species' fitnesses from high fitnesses to low fitnesses
        [~, vectorCull] = sort(-maxFitnessesCurrentGeneration);
        % Index of the species to discard
        vectorCull = vectorCull(1,3:sum(maxFitnessesCurrentGeneration > 0));
        for iSpecie = 1:size(vectorCull,2)
            % Index of specie to discard
            iCull = vectorCull(1,iSpecie);
            % Index of the the last generation of the current specie to
            % discard
            iGenerationCull = size(SpeciesRecord(iCull).generationRecord,2);
            % We assign a mean raw fitness of 0.01 to the last
            % generation of the current specie to discard
            SpeciesRecord(iCull).generationRecord(2,iGenerationCull) = 0.01;
        end
    end
end

end

%% reproduce()
function [NewPopulation, updatedSpeciesRecord, updatedInnovationRecord] = ...
    reproduce(Population, SpeciesRecord, innovationRecord, Initial, Selection,...
    Crossover, Mutation, Speciation, iGeneration, populationSize)

%% Initial selection

% Initial selection
[matExistingAndPropagSpecies, NewPopulation, Population, iIndividual] = ...
    intial_selection(SpeciesRecord, Initial, Population, populationSize);

% Generate reference population
populationRef = initialize_ref_population(Population, SpeciesRecord);

fprintf('\nExisting and propagating matrix:\n');
tmpMatExistingAndPropagSpecies = num2str(matExistingAndPropagSpecies,' %4.0f');
fprintf([tmpMatExistingAndPropagSpecies(1,:),'\n']);
fprintf([tmpMatExistingAndPropagSpecies(2,:),'\n']);
fprintf([tmpMatExistingAndPropagSpecies(3,:),'\n\n']);

%% Reproduction

% Cycle through all existing species
for iSpecie = 1:size(matExistingAndPropagSpecies,2)

    % Number of individuals in specie
    countIndividualsSpecies = 0;

    % This is the ID of species which will be reproduced this cycle.
    % IMPORTANT: iSpecie only has relevance to
    % matExistingAndPropagSpecies, all other mechanisms using
    % species in some way must use speciesId
    speciesId = matExistingAndPropagSpecies(1,iSpecie);

    % Linear Ranking and stochastic universal sampling Ranking with
    % Selection.pressure
    [NewChrIx, numberCrossover, numberMutate] = ...
        ranking_and_sampling(Population, Selection, speciesId, iSpecie, ...
        matExistingAndPropagSpecies, Crossover);

    % Cycle until actual number of offspring has reached allotted number of offspring
    while matExistingAndPropagSpecies(3,iSpecie) < matExistingAndPropagSpecies(2,iSpecie)

        % Increment number of individuals in population
        iIndividual = iIndividual + 1;

        % Increment number of individuals in current specie
        countIndividualsSpecies = countIndividualsSpecies + 1;

        % Crossover
        if countIndividualsSpecies <= numberCrossover
            % OK we are doing crossover
            NewIndividual = crossover(...
                Population, NewChrIx, countIndividualsSpecies, Crossover, ...
                matExistingAndPropagSpecies);
        else
            % No crossover, just copy a individual of the species and mutate in
            % subsequent steps
            NewIndividual = Population(NewChrIx(numberCrossover+countIndividualsSpecies));
        end

        % Hidden nodes culling
        NewIndividual = node_culling(NewIndividual);

        % Disabled Genes Mutation
        NewIndividual = enable_gene_mutation(NewIndividual, Mutation);

        % Weight Mutation
        NewIndividual = weight_mutation(NewIndividual, Mutation);

        % IMPORTANT: The checks for duplicate innovations in the following
        % two types of mutation can only check in the current generation

        flag1 = rand() < Mutation.probabilityAddNode;
        if flag1 == 0
            % Add Connection Mutation
            [NewIndividual, innovationRecord] = ...
                add_connection_mutation(NewIndividual, innovationRecord,...
                Mutation, iGeneration, flag1);
        elseif flag1 == 1
            % Add (insert) Node Mutation
            [NewIndividual, innovationRecord] = ...
                add_node_mutation(innovationRecord, iGeneration, NewIndividual);
        end

        % Speciation
        [NewIndividual, SpeciesRecord, populationRef] = ...
            speciation(NewIndividual, SpeciesRecord, populationRef, Speciation);
```

```
846
847         % Add NewIndividual to NewPopulation
848         NewPopulation(iIndividual) = NewIndividual;
849
850         % Increment species
851         matExistingAndPropagSpecies(3,iSpecie) = ...
852             matExistingAndPropagSpecies(3,iSpecie) + 1;
853
854     end
855 end
856
857 % Final update of SpeciesRecord (can only be done now since old population
858 % sizes were needed during reproduction cycle)
859 for iSpecie = 1:size(SpeciesRecord,2)
860     SpeciesRecord(iSpecie).nIndividuals = sum([NewPopulation(:).species] == iSpecie);
861 end
862
863 % Assign updated SpeciesRecord to output
864 updatedSpeciesRecord = SpeciesRecord;
865
866 % Assign updated innovationRecord to output
867 updatedInnovationRecord = innovationRecord;
868
869 end
870
871 %% compute_sum_average_fitnesses()
872 function sumAverageFitnesses = compute_sum_average_fitnesses(SpeciesRecord)
873
874 % Compute the sum of average fitenesses
875 sumAverageFitnesses = 0;
876 for iSpecie = 1:size(SpeciesRecord,2)
877     % Number of generation in current specie
878     nGenerations = size(SpeciesRecord(iSpecie).generationRecord,2);
879     % Mean raw fitness of the last generation of current specie "iSpecie"
880     meanRawFitness = SpeciesRecord(iSpecie).generationRecord(2,nGenerations);
881     % Sum average fitnesses of specie if number of individuals in this
882     % specie is > 0
883     sumAverageFitnesses = sumAverageFitnesses + ...
884         meanRawFitness*(SpeciesRecord(iSpecie).nIndividuals > 0);
885 end
886
887 end
888
889 %% initial_selection()
890 function [matExistingAndPropagSpecies, NewPopulation, Population, iIndividual] = ...
891     intial_selection(SpeciesRecord, Initial, Population, populationSize)
892
893 % This function implements this part of the paper:
894 % Every species is assigned a potentially different number of offspring in
895 % proportion to the sum of adjusted fitnesses of its member organisms.
896 % Species then reproduce by first eliminating the lowest performing members
897 % from the population.
898 %
899 % The following 'for' loop has these three objectives:
900 %
901 % 1. Compute matrix of existing and propagating species from SpeciesRecord
902 % (first row), assign their alloted number of offspring from the shared
903 % fitness (second row), and set their actual number of offspring to zero
904 % (third row) (will be incremented as new individuals are created from this
905 % species)
906 %
907 % 2. Copy most fit individual in every species with more than
908 % Initial.numberCopy individuals unchanged into new generation (elitism)
909 % (But only if species is not dying out, i.e. has at least one individual
910 % allotted to itself in the new generation) utilizes data stored in
911 % SpeciesRecord.generationRecord (index of individual in population
912 % having the highest fitness)
913 %
914 % 3. Erase lowest percentage (Initial.killPercentage) in species with more
915 % than Initial.numberForKill individuals to keep them from taking part in
916 % reproduction
917
918 % Compute sum of average fitenesses
919 sumAverageFitnesses = compute_sum_average_fitnesses(SpeciesRecord);
920
921 % The following two lines only initialize the NewPopulation structure.
922 % Since its species is set to 0, the rest of the algorithm will not bother
923 % with it. It gets overwritten as soon as the first really new individual
924 % is created
925 NewPopulation(1) = Population(1);
926 NewPopulation(1).species = 0;
927
928 % Cycle through all existing species
929 overflow = 0;
930 iIndividual = 0;
931 matExistingAndPropagSpecies = [];
932 for iSpecie = 1:size(SpeciesRecord,2)
933     % Test if species existed in old generation
934     if SpeciesRecord(iSpecie).nIndividuals > 0
935
936         % Number of generation in current specie
937         nGenerations = size(SpeciesRecord(iSpecie).generationRecord,2);
938         % Mean raw fiteness of current generation
939         meanRawFitness = SpeciesRecord(iSpecie).generationRecord(2,nGenerations);
940         % Compute number of offspring in new generation
941         nOffsprings = (meanRawFitness/sumAverageFitnesses)*populationSize;
942         overflow = overflow + nOffsprings - floor(nOffsprings);
943         % Since new species sizes are fractions, overflow sums up the
944         % difference between size and floor(size), and everytime this
945         % overflow is above 1, the species gets one additional individual
946         % alotted to it
947         if overflow >= 1
948             nOffsprings = ceil(nOffsprings);
949             overflow = overflow - 1;
950         else
951             nOffsprings = floor(nOffsprings);
952         end
953
954         % Check to see if species is dying out, only add those species to
955         % matExistingAndPropagSpecies which have offspring in the
956         % new generation
957         if nOffsprings > 0
958             % Matrix (objective 1)
959             matExistingAndPropagSpecies = ...
960                 [matExistingAndPropagSpecies,...
961                 [SpeciesRecord(iSpecie).id;
962                 nOffsprings;
963                 0]];
964             % Check for condition for objective 2
965             if SpeciesRecord(iSpecie).nIndividuals >= Initial.numberCopy
966                 iIndividual = iIndividual + 1;
967                 % Objective 2
968                 NewPopulation(iIndividual) = ...
969                     Population(SpeciesRecord(iSpecie).generationRecord(4,nGenerations));
970                 % Update matExistingAndPropagSpecies
971                 matExistingAndPropagSpecies(3,size(matExistingAndPropagSpecies,2)) = 1;
972             end
973         end
974
975         % condition1 checks if specie has more individuals than numberForKill
976         condition1 = SpeciesRecord(iSpecie).nIndividuals > Initial.killPercentage;
977         % condition2 checks if after killing killPercentage, there's at least 2
978         % individuals to be able to cross over or at least the single
979         % individual in a species with only one individual
980         condition2 = ceil(SpeciesRecord(iSpecie).nIndividuals*(1-Initial.killPercentage)) > 2;
981         % Check condition for objective 3
982         if condition1 && condition2
983             % The vector popSpecies contains the species index of each
984             % individual in the population
985             popSpecies = [Population(:).species];
986             % Index in population of individuals in iSpecie
987             iIndividualsToKill = find(popSpecies == iSpecie);
988             % Matrix containing on first row the index of individual in
989             % current specie to kill and their fitness on the second row
990             matrixIndividualsSpecies = [...
991                 iIndividualsToKill;
992                 [Population(iIndividualsToKill).fitness]];
993             % We sort the individual in the specie according to their fitness
994             [-, sortingVector] = sort(matrixIndividualsSpecies(2,:));
995             % Sorted matrixIndividualsSpecies
996             matrixIndividualsSpecies = matrixIndividualsSpecies(:,sortingVector);
997             % Index of sorted individual in specie
998             sortingVector = matrixIndividualsSpecies(1,:);
999             % MATLAB actually doesn't offer a facility for redirecting the
1000            % pointers which link one element in a structure with the next,
1001            % so essentially this individual entry in the population
1002            % structure cannot be erased. Rather, it will have its species'
1003            % ID set to zero, which has the same effect of removing it from
1004            % the rest of the reproduction cycle, since all reproduction
1005            % functions access the population structure through the
1006            % species' ID and no species has an ID of zero.
1007            % We kill the Initial.killPercentage first individual in the specie
1008            nKills = floor(SpeciesRecord(iSpecie).nIndividuals*Initial.killPercentage);
1009            for ikill = 1:nKills
1010                % Objective 3
1011                Population(sortingVector(ikill)).species = 0;
1012            end
1013        end
1014
1015    end
1016 end
1017
1018 end
1019
1020 %% initialize_ref_population()
1021 function populationRef = initialize_ref_population(Population, SpeciesRecord)
1022
1023 % Generate reference population of random individuals from every species from
1024 % old population. Cycle through species ID's, and add reference individuals from
1025 % old population. New species of new population will be added during reproduction.
1026
1027 % The vector popSpecies contains the species' index of each individual in
1028 % the population (the index of the specie in which each individual belongs to)
1029 popSpecies = [Population(:).species];
1030
1031 % Number of species in current generation
1032 nSpecies = size(SpeciesRecord,2);
1033
1034 % Create reference population
1035 iRef = 0;
1036 for iSpecieRef = 1:nSpecies
1037     % Index of individuals in current specie (iSpecieRef) that exists in
1038     % old population. By exist, we mean the individuals that haven't been
1039     % killed during intial_selection()
1040     iExistingIndividual = popSpecies == iSpecieRef;
1041     % Check if species exists in old population
1042     if sum(iExistingIndividual) > 0
1043         iRef = iRef + 1;
1044         % Number of individuals in population
1045         nIndividuals = size(Population,2);
1046         % iRefOld is the index of the individual in the old population that
1047         % is assigned to the new population as the representative of the
1048         % specie
1049         [-, iRefOld] = max(iExistingIndividual.*rand(1,nIndividuals));
1050         % We assign the chosen individual of the old population in the
1051         % reference population
1052         populationRef(iRef) = Population(iRefOld);
1053     end
1054 end
1055
1056 end
1057
1058 %% ranking_and_sampling()
1059 function [NewChrIx, numberCrossover, numberMutate] = ...
1060     ranking_and_sampling(Population, Selection, speciesId, iSpecie, ...
1061     matExistingAndPropagSpecies, Crossover)
1062
1063 % This function compute select the individuals within the current specie
1064 % for reproduction (NewChrIx). It also compute the number of crossovers and
1065 % mutations to perform within the selected individuals in the specie
1066 %
1067 % See http://www.geatbx.com/ver_3_7/selsus.html and
1068 % http://www.geatbx.com/docu/algindex-02.html for explaination
1069
1070 %%  Linear Ranking with Selection.pressure
1071
1072 % Index of individual with specie's ID equal to speciesId
1073 iFitnessesSpecies      = find([Population(:).species] == speciesId);
1074 % Fitnesses of individual with specie's ID equal to speciesId
1075 fitnessesSpecies       = [Population(iFitnessesSpecies).fitness];
1076 % Sorting fitnesses in ascending order
1077 [-, sortedFitnesses]   = sort(fitnessesSpecies);
1078 % Number of individuals in speciesId
1079 nIndividualsInSpecie     = size(fitnessesSpecies,2);
1080 % Ranking according to their fitnesses
1081 ranking                = zeros(1,nIndividualsInSpecie);
1082 ranking(sortedFitnesses) = 1:nIndividualsInSpecie;
1083 if nIndividualsInSpecie > 1
1084     FitnV = 2 - Selection.pressure + ...
1085         2*(ranking-1)*(Selection.pressure-1)/(nIndividualsInSpecie - 1);
1086     FitnV = FitnV';
1087 else
1088     FitnV = 2;
1089 end
1090
1091 % Compute number of individuals to be selected (two parents
1092 % required for every offspring through crossover, one for mutation)
1093 numberOverall   = matExistingAndPropagSpecies(2,iSpecie) - ...
1094     matExistingAndPropagSpecies(3,iSpecie);
1095 numberCrossover = round(Crossover.percentage*numberOverall);
1096 numberMutate    = numberOverall - numberCrossover;
1097 nInd            = nIndividualsInSpecie;
1098 nSel            = 2*numberCrossover + numberMutate;
1099
1100 % Rare case, in which a species with at least Initial.numberCopy
1101 % individuals gets individual copied, but compares poorly to new
1102 % generation, which results in this copied individual being the only
1103 % individual of this species in new generation, so no crossover or
1104 % mutation takes place. setting nSel to 1 will prevent division by zero error,
1105 % but will have no other effect since the propagation loop is governed by
1106 % matExistingAndPropagSpecies, not by nSel
1107 if nSel == 0
1108     nSel = 1;
1109 end
1110
1111 %% Perform stochastic universal sampling
1112
1113 % (Code Snippet from Genetic Algorithm toolbox 1.2 by Chipperfield et al)
1114
1115 cumFit = cumsum(FitnV);
1116 trials = cumFit(nInd) / nSel * (rand() + (0:nSel-1)');
1117 mF = cumFit(:, ones(1, nSel));
1118 mT = trials(:, ones(1, nInd))';
1119 [NewChrIx, -] = find(mT < mF & [zeros(1, nSel); mF(1:nInd-1, :)] <= mT);
1120 % Shuffle selected Individuals
1121 [-, shuf] = sort(rand(nSel, 1));
1122 % NewChrIx is a column vector containing the indexes of the selected
1123 % individuals relative to the original population, shuffeld
1124 NewChrIx = NewChrIx(shuf);
1125 % Relate to indexes of individuals in population
1126 NewChrIx = iFitnessesSpecies(NewChrIx);
1127
1128 end
1129
```

```
%% crossover()
function NewIndividual = crossover(...
    Population, NewChrIx, countIndividualsSpecies, Crossover, ...
    matExistingAndPropagSpecies)

% Select Parent1
Parent1 = Population(NewChrIx(2*countIndividualsSpecies-1));

% Select Parent2
foundParent2 = 0;
% Select Parent2 from other species (can only be done if there
% is more than one species in old population)
condition1 = rand() < Crossover.probabilityInterspecies;
condition2 = size(matExistingAndPropagSpecies,2) > 1;
if condition1 && condition2
    while foundParent2 == 0
        [~, iParent2] = max(rand(1,size(Population,2)));
        Parent2 = Population(iParent2);
        % Check if Parent2 species is not species 0 (deleted
        % individual) or species of Parent1
        foundParent2 = ...
            ((Parent2.species ~= 0) & (Parent2.species ~= Parent1.species));
    end
    % Set fitnesses to same to ensure that disjoint and excess
    % genes are inherited fully from both parents (tip from ken)
    Parent2.fitness = Parent1.fitness;
else
    % OK we take Parent2 from same species as Parent1
    Parent2 = Population(NewChrIx(2*countIndividualsSpecies));
end

% Inherit nodes from both parents
NewIndividual.nodeGenes = [];
nGenesP1 = size(Parent1.nodeGenes,2);
nGenesP2 = size(Parent2.nodeGenes,2);
matrixNodeLineup    = [...
    [Parent1.nodeGenes(1,:);1:nGenesP1;zeros(1,nGenesP1)],...
    [Parent2.nodeGenes(1,:);zeros(1,nGenesP2);1:nGenesP2]];
[~, sortNodeVec] = sort(matrixNodeLineup(1,:));
matrixNodeLineup = matrixNodeLineup(:,sortNodeVec);
nodeNumber       = 0;
for iNodeSort = 1:size(matrixNodeLineup,2)
    if nodeNumber ~= matrixNodeLineup(1,iNodeSort)
        if matrixNodeLineup(2,iNodeSort) > 0
            NewIndividual.nodeGenes = ...
                [NewIndividual.nodeGenes,...
                Parent1.nodeGenes(:,matrixNodeLineup(2,iNodeSort))];
        else
            NewIndividual.nodeGenes = ...
                [NewIndividual.nodeGenes,...
                Parent2.nodeGenes(:,matrixNodeLineup(3,iNodeSort))];
        end
        nodeNumber = matrixNodeLineup(1,iNodeSort);
    end
end

% Crossover connection genes
% First do lineup of connection genes
nConnGenesP1 = size(Parent1.connectionGenes,2);
nConnGenesP2 = size(Parent2.connectionGenes,2);
matrixLineup = [...
    [Parent1.connectionGenes(1,:);1:nConnGenesP1;zeros(1,nConnGenesP1)],...
    [Parent2.connectionGenes(1,:);zeros(1,nConnGenesP2);1:nConnGenesP2]];
[~, sortVec] = sort(matrixLineup(1,:));
matrixLineup = matrixLineup(:,sortVec);
finalMatrixLineup = [];
innovationNumber = 0;
for iSort = 1:size(matrixLineup,2)
    if innovationNumber ~= matrixLineup(1,iSort)
        finalMatrixLineup = [finalMatrixLineup, matrixLineup(:,iSort)];
        innovationNumber = matrixLineup(1,iSort);
    else
        finalMatrixLineup(2:3,size(finalMatrixLineup,2)) = ...
            finalMatrixLineup(2:3,size(finalMatrixLineup,2)) + ...
            matrixLineup(2:3,iSort);
    end
end

% OK Connection Genes are lined up, start with crossover
NewIndividual.connectionGenes = [];
for iLineup = 1:size(finalMatrixLineup,2)
    % Check for matching genes, do crossover
    if (finalMatrixLineup(2,iLineup) > 0) && (finalMatrixLineup(3,iLineup) > 0)
        % Random crossover for matching genes
        if rand() < 0.5
            NewIndividual.connectionGenes = ...
                [NewIndividual.connectionGenes,...
                Parent1.connectionGenes(:,finalMatrixLineup(2,iLineup))];
        else
            NewIndividual.connectionGenes = ...
                [NewIndividual.connectionGenes,...
                Parent2.connectionGenes(:,finalMatrixLineup(3,iLineup))];
        end
        % Weight averaging for offspring, otherwise the randomly inherited
        % weights are left undisturbed
        if rand() > Crossover.probabilityMultipoint
            NewIndividual.connectionGenes(4,size(NewIndividual.connectionGenes,2)) = ...
                (...
                Parent1.connectionGenes(4,finalMatrixLineup(2,iLineup)) + ...
                Parent2.connectionGenes(4,finalMatrixLineup(3,iLineup))...
                )/2;
        end
    end
    % Test if there exist further connection genes from
    % iLineup+1 to end of finalMatrixLineup for Parent1 (to
    % detect excess)
    parent1Flag = sum(finalMatrixLineup(2,iLineup+1:size(finalMatrixLineup,2)));
    % Test if there exist further connection genes from
    % iLineup+1 to end of finalMatrixLineup for Parent1 (to
    % detect excess)
    parent2Flag = sum(finalMatrixLineup(3,iLineup+1:size(finalMatrixLineup,2)));
    % Two cases to check (excess is taken care of in the disjoint gene checks)
    % Disjoint Parent1
    if (finalMatrixLineup(2,iLineup) > 0) && (finalMatrixLineup(3,iLineup) == 0)
        if Parent1.fitness >= Parent2.fitness
            NewIndividual.connectionGenes = ...
                [NewIndividual.connectionGenes,...
                Parent1.connectionGenes(:,finalMatrixLineup(2,iLineup))];
        end
    end
    % Disjoint Parent2
    if (finalMatrixLineup(2,iLineup) == 0) && (finalMatrixLineup(3,iLineup) > 0)
        if Parent2.fitness >= Parent1.fitness
            NewIndividual.connectionGenes = ...
                [NewIndividual.connectionGenes,...
                Parent2.connectionGenes(:,finalMatrixLineup(3,iLineup))];
        end
    end
end

% Has no impact on algorithm, only required for assignment to new population
NewIndividual.fitness = 0;

% Will be species hint for speciation
NewIndividual.species = Parent1.species;

end

%% node_culling()
function NewIndividual = node_culling(NewIndividual)

% Hidden nodes culling (remove any hidden nodes where there is no
% corresponding connection gene in the new individual)
connectedNodes = [];
for iNodeCulling = 1:size(NewIndividual.nodeGenes,2)
    nodeConnectedFlag = ...
        sum(NewIndividual.connectionGenes(2,:) == NewIndividual.nodeGenes(1,iNodeCulling)) + ...
        sum(NewIndividual.connectionGenes(3,:) == NewIndividual.nodeGenes(1,iNodeCulling));
    if (nodeConnectedFlag > 0) || (NewIndividual.nodeGenes(2,iNodeCulling) ~= 3)
        connectedNodes = [connectedNodes,NewIndividual.nodeGenes(:,iNodeCulling)];
    end
end
NewIndividual.nodeGenes = connectedNodes;

end

%% enable_gene_mutation()
function NewIndividual = enable_gene_mutation(NewIndividual, Mutation)

% Disabled Genes Mutation
% Run through all connection genes in a NewIndividual, find disabled
% connection genes, enable again with Crossover.probabilityGeneReenabled
% probability
for iConnectionGene = 1:size(NewIndividual.connectionGenes,2)
    condition1 = NewIndividual.connectionGenes(5,iConnectionGene) == 0;
    condition2 = rand() < Mutation.probabilityGeneReenabled;
    if condition1 && condition2
        NewIndividual.connectionGenes(5,iConnectionGene) = 1;
    end
end

end

%% weight_mutation()
function NewIndividual = weight_mutation(NewIndividual, Mutation)

% Weight Mutation
% Run through all connection genes in a NewIndividual, decide on
% mutating or not
for iConnectionGene = 1:size(NewIndividual.connectionGenes,2)
    % *iConnectionGene/size(NewIndividual.connectionGenes,2)
    % linearly biased towards higher probability of mutation at end of
    % connection genes
    if rand() < Mutation.probabilityMutateWeight
        NewIndividual.connectionGenes(4,iConnectionGene) = ...
            NewIndividual.connectionGenes(4,iConnectionGene) + ...
            Mutation.weightRange*(rand()-0.5);
    end
    % Weight of connection genes of NewIndividual
    weightConnGenes = NewIndividual.connectionGenes(4,iConnectionGene);
    % Weight capping
    NewIndividual.connectionGenes(4,iConnectionGene) = ...
        weightConnGenes*(abs(weightConnGenes) <= Mutation.weightCap) + ...
        (sign(weightConnGenes)*Mutation.weightCap)*(abs(weightConnGenes) > Mutation.weightCap);
end

end

%% add_connection_mutation()
function [NewIndividual, innovationRecord] = ...
    add_connection_mutation(NewIndividual, innovationRecord, Mutation,...
    iGeneration, flag1)

% Add Connection Mutation
flagRecurrencyEnabled = rand() < Mutation.probabilityRecurrency;

% Connections can run from every node
vectorPossibleConnectFromNodes = NewIndividual.nodeGenes(1,:);

% Connections can only run into hidden and output nodes
vectorPossibleConnectToNodes = ...
    NewIndividual.nodeGenes(...
    1,find((NewIndividual.nodeGenes(2,:) == 2) + (NewIndividual.nodeGenes(2,:) == 3))...
    );
numberPossibleConnection = ...
    length(vectorPossibleConnectFromNodes)*length(vectorPossibleConnectToNodes) - ...
    size(NewIndividual.connectionGenes,2);

% Check if new connections can be added to genes (if there are any
% possible connections which are not already existing in genes of
% new individual)
condition1 = rand() < Mutation.probabilityAddConnection;
condition2 = numberPossibleConnection > 0;
condition3 = flag1 == 0;
if condition1 && condition2 && condition3
    % First build matrix containing all possible new connection for
    % nodegene of new individual
    newConnectionMatrix = [];
    for iConnectFrom = 1:length(vectorPossibleConnectFromNodes)
        for iConnectTo = 1:length(vectorPossibleConnectToNodes)
            possibleConnection = ...
                [...
                vectorPossibleConnectFromNodes(iConnectFrom);
                vectorPossibleConnectToNodes(iConnectTo)];
            % Check if proposed connection is not already contained in gene
            condition4 = sum(...
                (NewIndividual.connectionGenes(2,:) == possibleConnection(1)).* ...
                (NewIndividual.connectionGenes(3,:) == possibleConnection(2)) ...
                ) == 0;
            if condition4
                newConnectionMatrix = [newConnectionMatrix, possibleConnection];
            end
        end
    end
    % Shuffle possible new connections randomly
    [~, shuffle] = sort(rand(1,size(newConnectionMatrix,2)));
    newConnectionMatrix = newConnectionMatrix(:,shuffle);

    iNewConnection = 0;
    flagConnectionOk = 0;
    % Check if connection is ok. (meaning either non-recurrent or
    % recurrent and flagRecurrencyEnabled set to 1) if no
    % connection is found which is ok, no connection will be added
    % to connection genes of new individual
    while (flagConnectionOk == 0) && (iNewConnection < size(newConnectionMatrix,2))
        iNewConnection = iNewConnection + 1;
        newConnection = newConnectionMatrix(:,iNewConnection);
        % Test new connection if it is recurrent (i.e. at least one
        % of the possibles path starting from connect_to node in
        % the network leads back to the connect_from node
        flagRecurrent = 0;
        % Trivial recurrency
        if newConnection(1) == newConnection(2)
            flagRecurrent = 1;
        end
        nodesCurrentLevel = newConnection(2);
        depth = 0;
        while (flagRecurrent == 0) && ...
                (depth < size(NewIndividual.connectionGenes,2)) && ...
                ~isempty(nodesCurrentLevel)
            depth = depth + 1;
            nodesNextLevel = [];
            for indexCheck = 1:size(nodesCurrentLevel);
                indexTemp = find(...
                    NewIndividual.connectionGenes(2,:) == ...
                    nodesCurrentLevel(indexCheck) ...
                    );
                nodesNextLevel = ...
                    [nodesNextLevel,...
                    NewIndividual.connectionGenes(3,indexTemp)];
            end
            if sum(nodesNextLevel(:) == newConnection(1)) > 0
                flagRecurrent = 1;
            end
```

```
            nodesCurrentLevel = nodesNextLevel;
        end
        if flagRecurrent == 0
            flagConnectionOk = 1;
        elseif flagRecurrencyEnabled
            flagConnectionOk = 1;
        end
    end

    % Now we test if it is a true innovation (i.e. hasn't already
    % happened in this generation) we can only do this if a valid
    % new connection has been found
    if flagConnectionOk
        % Set flag signifying new innovation (connection not
        % contained in innovationRecord of this generation)
        iAlreadyHappened = find(...
            (innovationRecord(5,:) == iGeneration).* ...
            (innovationRecord(2,:) == newConnection(1)).* ...
            (innovationRecord(3,:) == newConnection(2)) ...
            );
        newInnovation = not(sum(iAlreadyHappened));
        if newInnovation == 1 % OK is new innovation
            % Update the new connection with its innovation number
            newConnection = [max(innovationRecord(1,:))+1;newConnection];
            % Update connection_genes. Weight is uniformly distributed between
            % [-1,+1]
            NewIndividual.connectionGenes = [...
                NewIndividual.connectionGenes,...
                [newConnection;rand()*2-1;1]];
            % Update innovationRecord
            innovationRecord = [innovationRecord,[newConnection;0;iGeneration]];
        else % Connection gene already exists in innovationRecord of this generation
            % Update connectionGenes. Weight is uniformly distributed between
            % [-1,+1]
            NewIndividual.connectionGenes = [...
                NewIndividual.connectionGenes,...
                [innovationRecord(1:3,iAlreadyHappened);rand()*2-1;1]];
        end
    end

end

end

%% add_node_mutation()
function [NewIndividual, innovationRecord] = ...
    add_node_mutation(innovationRecord, iGeneration, NewIndividual)

% Add (Insert) Node Mutation
newInnovation = 0;
% Highest innovation number from last generation (to ensure
% that only connections from from last generation or older are
% chosen for add node mutation, otherwise a new connection
% added in the last mutation might instantly be disabled)
maxOldInnovationNumber = max(...
    (innovationRecord(5,:) < iGeneration).*innovationRecord(1,:) ...
    );
% Compute vector of connections into which a new node could be
% inserted and their positions in the connectionGene matrix.
% This vector is composed of all nondisabled connections which
% stem at least from the last generation or older
indexTemp = find(...
    (NewIndividual.connectionGenes(5,:) == 1) & ...
    (NewIndividual.connectionGenes(1,:) <= maxOldInnovationNumber) ...
    );
vectorPossibleConnections = [...
    NewIndividual.connectionGenes(2:3,indexTemp);
    indexTemp];
insertNodeConnection = ...
    vectorPossibleConnections(:,round(rand()*size(vectorPossibleConnections,2) + 0.5));
% Set provisionally to 1, will be checked
newInnovation = 1;
% Beginning of check innovation record to test for real
% innovation. existInnovation contains vector of index of
% elements in innovation record which fulfil three things:
% current generation, add node mutation and same connect from
% as current innovation
existInnovation = find((innovationRecord(5,:) == ...
    iGeneration).*...
    (innovationRecord(4,:) > 0).*...
    (innovationRecord(2,:) == insertNodeConnection(1)));
% If these are fulfilled, we have to test for connect_to node
% to see if innovation really is the same
if sum(existInnovation) > 0
    for indexCheck = 1:length(existInnovation)
        if innovationRecord(3,existInnovation(indexCheck)+1) == insertNodeConnection(2)
            newInnovation = 0;
            iAlreadyExistentThisGeneration = existInnovation(indexCheck);
        end
    end
end
if newInnovation == 1 % OK is true innovation for current generation
    % Update node genes. By [newNodeNumber;3;0;0], we see that only hidden nodes
    % can be added by add_node_mutation.
    newNodeNumber = max(innovationRecord(4,:)) + 1;
    NewIndividual.nodeGenes = [NewIndividual.nodeGenes,[newNodeNumber;3;0;0]];
    % Disable old connection gene
    NewIndividual.connectionGenes(5,insertNodeConnection(3)) = 0;
    % Update connectionGenes. newConnections is a 5 rows by two columns matrix
    % where the first column is the new connection TO the new node whereas the
    % second column is the connection FROM the new node
    newConnections = [...
        [max(innovationRecord(1,:)) + 1;
        insertNodeConnection(1);
        newNodeNumber;
        1;
        1],...
        [max(innovationRecord(1,:)) + 2;
        newNodeNumber;
        insertNodeConnection(2);
        NewIndividual.connectionGenes(4,insertNodeConnection(3));
        1]];
    % Extend connection genes by the two new connections
    NewIndividual.connectionGenes = [NewIndividual.connectionGenes, newConnections];
    % Update innovationRecord
    innovationRecord = [...
        innovationRecord,...
        [newConnections(1:3,:);newNodeNumber,0;iGeneration,iGeneration]];
else % No new innovation, has already happened at least once in this generation
    % Update node genes
    nodeNumber = innovationRecord(4,iAlreadyExistentThisGeneration);
    NewIndividual.nodeGenes = [NewIndividual.nodeGenes,[nodeNumber;3;0;0]];
    % Update connectionGenes
    % Disable old connection gene
    NewIndividual.connectionGenes(5,insertNodeConnection(3)) = 0;
    newConnections = [...
        innovationRecord(1:3,iAlreadyExistentThisGeneration:iAlreadyExistentThisGeneration+1);
        1,NewIndividual.connectionGenes(4,insertNodeConnection(3));1,1];
    % Length of the connection genes of current NewIndividual
    lengthConGen = size(NewIndividual.connectionGenes,2);
    % Check if there was an add_connection_mutation to current
    % NewIndividual which has a higher innovation number than
    % current add_node_mutation
    if NewIndividual.connectionGenes(1,lengthConGen) > newConnections(1,2)
        NewIndividual.connectionGenes = [...
            NewIndividual.connectionGenes(:,1:lengthConGen-1), ...
            newConnections, ...
            NewIndividual.connectionGenes(:,lengthConGen)];
    else
        NewIndividual.connectionGenes = [...
            NewIndividual.connectionGenes,...
            newConnections];
    end
end

end

%% speciation()
function [NewIndividual, SpeciesRecord, populationRef] = ...
    speciation(NewIndividual, SpeciesRecord, populationRef, Speciation)

% This function performs speciation to assign the new individual into an
% existing or new specie

% Loop through comparison vector
speciesAssigned = 0;
iPopulationRef = 0;
while (speciesAssigned == 0) && (iPopulationRef < size(populationRef,2))
    % Extract referenceIndividual from reference population
    iPopulationRef = iPopulationRef + 1;
    referenceIndividual = populationRef(iPopulationRef);
    % Run through both connection genes, compute disjoint, excess,
    % and average weight difference.
    % Maximum number of genes between new individual and refenrence individual
    maxNumGenes = max(...
        [size(NewIndividual.connectionGenes,2),...
        size(referenceIndividual.connectionGenes,2)]...
        );
    % Maximum innovation number between new individual and refenrence individual
    maxNumInnovation = max(...
        [NewIndividual.connectionGenes(1,:),...
        referenceIndividual.connectionGenes(1,:)]...
        );
    % New innovation vector
    vectorInnovationNew = [...
        zeros(1,max(NewIndividual.connectionGenes(1,:))),...
        ones(1,maxNumInnovation - max(NewIndividual.connectionGenes(1,:)))];
    vectorInnovationNew(NewIndividual.connectionGenes(1,:)) = 2;
    % New weight vector
    vectorWeightNew = zeros(1,maxNumInnovation);
    vectorWeightNew(NewIndividual.connectionGenes(1,:)) = ...
        NewIndividual.connectionGenes(4,:);
    % Innovation vector for refenrence individual
    vectorInnovationRef = [...
        4*ones(1,max(referenceIndividual.connectionGenes(1,:))),...
        8*ones(1,maxNumInnovation - max(referenceIndividual.connectionGenes(1,:)))];
    vectorInnovationRef(referenceIndividual.connectionGenes(1,:)) = 16;
    % Weight vector for refenrence individual
    vectorWeightRef = zeros(1,maxNumInnovation);
    vectorWeightRef(referenceIndividual.connectionGenes(1,:)) = ...
        referenceIndividual.connectionGenes(4,:);
    % Lineup vector between new individual and reference individual
    vectorLineup = vectorInnovationNew + vectorInnovationRef;
    % Number of excess genes between new and refrence individuals
    excess = sum(vectorLineup == 10) + sum(vectorLineup == 17);
    % Number of disjoint genes between new and refrence individuals
    disjoint = sum(vectorLineup == 6) + sum(vectorLineup == 16);
    % Matching genes
    vectorMatching = find(vectorLineup == 18);
    % Average weight difference
    averageWeightDifference = ...
        sum(...
        abs(vectorWeightNew(vectorMatching) - vectorWeightRef(vectorMatching))...
        )/length(vectorMatching);
    % Compute compatbility distance between new and refenrence individuals
    maxNumGenes = 1;
    distance = Speciation.c1*excess/maxNumGenes + ...
        Speciation.c2*disjoint/maxNumGenes + ...
        Speciation.c3*averageWeightDifference;
    if distance < Speciation.threshold
        % Assign individual to same species as current reference individual
        NewIndividual.species = referenceIndividual.species;
        % Set flag indicating NewIndividual has been assigned to species
        speciesAssigned = 1;
    end
end

% Not compatible with any? well, then create new species
if speciesAssigned == 0
    newSpeciesId = size(SpeciesRecord,2) + 1;
    % Assign individual to new species
    NewIndividual.species = newSpeciesId;
    % Update SpeciesRecord
    SpeciesRecord(newSpeciesId).id = newSpeciesId;
    SpeciesRecord(newSpeciesId).nIndividuals = 1;
    SpeciesRecord(newSpeciesId).generationRecord = [];
    % Update population reference
    populationRef(size(populationRef,2)+1) = NewIndividual;
end

end

%% compute_stats()
function [nAveragеNonDisabledConnections, nAveragеHiddenNodes, maxFitness,...
    maxOverallFitness, nAveragеDisabledConnections, ...
    nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecie] = ...
    compute_stats(Population, populationSize, iGeneration, ...
    SpeciesRecord, nAveragеNonDisabledConnections, nAveragеHiddenNodes, ...
    maxOverallFitness, nAveragеDisabledConnections, ...
    nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecie)

% Number of individuals
nIndividuals = size(Population,2);

% Compute the average number of enabled connections and hidden nodes in current
% population
a = 0; % Sum of enabled connection in current population
b = 0; % Sum of hidden nodes in current population
c = 0; % Sum of disabled connection in current population
for iIndividual = 1:nIndividuals
    a = a + sum(Population(iIndividual).connectionGenes(5,:) == 1);
    b = b + sum(Population(iIndividual).nodeGenes(2,:) == 3);
    c = c + sum(Population(iIndividual).connectionGenes(5,:) == 0);
end
nAveragеNonDisabledConnections = [...
    nAveragеNonDisabledConnections,...
    [a/populationSize;iGeneration]];
nAveragеDisabledConnections = [...
    nAveragеDisabledConnections,...
    [c/populationSize;iGeneration]];
nAveragеHiddenNodes = [...
    nAveragеHiddenNodes,...
    [b/populationSize;iGeneration]];

% Number of species in current generation
nSpecies = size(SpeciesRecord,2);
% nSpeciesArray = [nSpeciesArray,nSpecies];
nSpeciesArray = [nSpeciesArray,numel(unique([Population(:).species]))];

% Average number of individuals per specie
uniqueSpecieId = unique([Population(:).species]);
nIndPerSpecieTmp = zeros(1,numel(uniqueSpecieId));
for k = 1:numel(uniqueSpecieId)
    nIndPerSpecieTmp(k) = sum([Population(:).species] == uniqueSpecieId(k));
end
nIndPerSpecie = [nIndPerSpecie,mean(nIndPerSpecieTmp)];

% We build the fitCurrGenSpecie matrix with 3 rows and nSpecies columns. The
% first row is the index of the current generation of the specie. The second row
% is mean raw fitness of the current generation for the specie and the third row
% is the max raw fitness of the current generation for the specie.
fitCurrGenSpecie = [];
for iSpecie = 1:nSpecies
    % Number of generations in current specie
```

```
    nGenerations = size(SpeciesRecord(iSpecie).generationRecord,2);
    fitCurrGenSpecie = [...
        fitCurrGenSpecie,...
        SpeciesRecord(iSpecie).generationRecord(1:3,nGenerations)];
end

% Maximal fitness for current generation
maxFitness = max(fitCurrGenSpecie(3,:).*(fitCurrGenSpecie(1,:) == iGeneration));
fprintf('Max fitness is %8.6f\n', maxFitness);

% Maximum overall fitness. It is a matrix with 2 rows and nGenerations columns.
% The first row is the maximum overall fitness of the current generation and the
% second row is the generation number.
maxOverallFitness = [...
    maxOverallFitness,...
    [maxFitness;iGeneration]];

% Mean fitness for current generation
meanFitness = mean([Population(:).fitness]);
meanOverallFitness = [...
    meanOverallFitness,...
    [meanFitness;iGeneration]];

% Number of recurrent connections
nRecurrentConnections = ...
    find_recurrent_connection(Population);
meanRecurrentConnection = [meanRecurrentConnection,mean(nRecurrentConnections)];

end

%% solution_check()
function flagSolution = solution_check(maxFitness, fitnessLimit, flagSolution)

% Check for solution
if maxFitness > fitnessLimit
    flagSolution = 1;
end

end

%% visualization()
function visualization(figHandle, AxHandle, nAverageNonDisabledConnections, ...
    nAveragеHiddenNodes, maxOverallFitness, nAverageDisabledConnections, ...
    nSpeciesArray, meanOverallFitness, meanRecurrentConnection, nIndPerSpecie)

nGenerations = numel(nSpeciesArray);

lineSpec = {'LineWidth',1};
fontSize = 8;
nYTick = 9;
if 1 <= nGenerations && nGenerations < 6
    XTick1 = unique(round(linspace(1,5,5)));
    XTick2 = unique(round(linspace(1,5,5)));
elseif 6 <= nGenerations && nGenerations < 10
    XTick1 = unique(round(linspace(1,nGenerations,9)));
    XTick2 = unique(round(linspace(1,nGenerations,6)));
elseif 10 <= nGenerations && nGenerations < Inf
    XTick1 = unique(round(linspace(1,nGenerations,15)));
    XTick2 = unique(round(linspace(1,nGenerations,6)));
end
XTickLabel1 = cellfun(@num2str,num2cell(XTick1(:)),'uniformoutput',false);
XTickLabel2 = cellfun(@num2str,num2cell(XTick2(:)),'uniformoutput',false);

% Maximum fitness
set(figHandle, 'CurrentAxes', AxHandle.subPlot1);
hold(AxHandle.subPlot1,'on');
plot(AxHandle.subPlot1, maxOverallFitness(2,:), maxOverallFitness(1,:), lineSpec{:});
xlabel(AxHandle.subPlot1,'Generation','FontSize', fontSize);
ylabel(AxHandle.subPlot1,'Maximum fitness','FontSize', fontSize);
grid(AxHandle.subPlot1,'on');
hold(AxHandle.subPlot1,'off');

% Average fitness
set(figHandle, 'CurrentAxes', AxHandle.subPlot2);
hold(AxHandle.subPlot2,'on');
plot(AxHandle.subPlot2, meanOverallFitness(2,:), meanOverallFitness(1,:), lineSpec{:});
xlabel(AxHandle.subPlot2,'Generation','FontSize', fontSize);
ylabel(AxHandle.subPlot2,'Average fitness','FontSize', fontSize);
grid(AxHandle.subPlot2,'on');
hold(AxHandle.subPlot2,'off');

% Average number of hidden nodes
set(figHandle, 'CurrentAxes', AxHandle.subPlot3);
hold(AxHandle.subPlot3,'on');
plot(AxHandle.subPlot3, nAveragеHiddenNodes(2,:), nAveragеHiddenNodes(1,:), lineSpec{:});
xlabel(AxHandle.subPlot3,'Generation','FontSize', fontSize);
ylabel(AxHandle.subPlot3,{'Avg. number of';'hidden nodes'},'FontSize', fontSize);
grid(AxHandle.subPlot3,'on');
hold(AxHandle.subPlot3,'off');

% Average number of non disabled connections
set(figHandle, 'CurrentAxes', AxHandle.subPlot4);
hold(AxHandle.subPlot4,'on');
plot(AxHandle.subPlot4, nAverageNonDisabledConnections(2,:),...
    nAverageNonDisabledConnections(1,:), lineSpec{:});
xlabel(AxHandle.subPlot4,'Generation','FontSize', fontSize);
ylabel(AxHandle.subPlot4,{'Avg. number of';'non disabled connections'}, ...
    'FontSize', fontSize);
grid(AxHandle.subPlot4,'on');
hold(AxHandle.subPlot4,'off');

% Average number of disabled connections
set(figHandle, 'CurrentAxes', AxHandle.subPlot5);
hold(AxHandle.subPlot5,'on');
plot(AxHandle.subPlot5, nAverageDisabledConnections(2,:), ...
    nAverageDisabledConnections(1,:), lineSpec{:});
xlabel(AxHandle.subPlot5,'Generation','FontSize', fontSize);
ylabel(AxHandle.subPlot5,{'Avg. number of';'disabled connections'}, ...
    'FontSize', fontSize);
grid(AxHandle.subPlot5,'on');
hold(AxHandle.subPlot5,'off');

% Average number of recurrent connections
set(figHandle, 'CurrentAxes', AxHandle.subPlot6);
hold(AxHandle.subPlot6,'on');
plot(AxHandle.subPlot6, nAverageDisabledConnections(2,:), ...
    meanRecurrentConnection, lineSpec{:});
xlabel(AxHandle.subPlot6,'Generation','FontSize', fontSize);
ylabel(AxHandle.subPlot6,{'Avg. number of';'recurrent connections'}, ...
    'FontSize', fontSize);
grid(AxHandle.subPlot6,'on');
hold(AxHandle.subPlot6,'off');

% Number of species
set(figHandle, 'CurrentAxes', AxHandle.subPlot7);
hold(AxHandle.subPlot7,'on');
plot(AxHandle.subPlot7, nAverageDisabledConnections(2,:), nSpeciesArray, lineSpec{:});
xlabel(AxHandle.subPlot7,'Generation','FontSize', fontSize);
ylabel(AxHandle.subPlot7,{'Number of';'species'}, ...
    'FontSize', fontSize);
grid(AxHandle.subPlot7,'on');
hold(AxHandle.subPlot7,'off');

% Average number of individuals per specie
set(figHandle, 'CurrentAxes', AxHandle.subPlot8);
hold(AxHandle.subPlot8,'on');
plot(AxHandle.subPlot8, nAverageDisabledConnections(2,:), nIndPerSpecie, lineSpec{:});
xlabel(AxHandle.subPlot8,'Generation','FontSize', fontSize);
ylabel(AxHandle.subPlot8,{'Avg. number of';'individuals per specie'}, ...
    'FontSize', fontSize);
grid(AxHandle.subPlot8,'on');
hold(AxHandle.subPlot8,'off');

drawnow;

hold off

end

%% print_generation()
function print_generation(iGeneration)

fprintf([...
    '\n\n-------------------------------------------------------------------------------\n',...
    '|                              Generation %4.f                                |\n',...
    '-------------------------------------------------------------------------------\n\n'...
    ],iGeneration);

end

%% shrink_mutation()
function Mutation = shrink_mutation(Mutation, iGeneration)

% Add node
if Mutation.probabilityAddNodeScheduling(1) ~= -1
    for iGen = 1:numel(Mutation.probabilityAddNodeScheduling(1,:))
        if iGeneration <= Mutation.probabilityAddNodeScheduling(1,iGen)
            Mutation.probabilityAddNode = Mutation.probabilityAddNodeScheduling(2,iGen);
            break;
        end
    end
else
    Mutation.probabilityAddNode = max([...
        Mutation.probabilityAddNode - Mutation.probabilityAddNodeShrinkRate,...
        Mutation.probabilityAddNodeMin]);
end

% Add connection
if Mutation.probabilityAddConnectionScheduling(1) ~= -1
    for iGen = 1:numel(Mutation.probabilityAddConnectionScheduling(1,:))
        if iGeneration <= Mutation.probabilityAddConnectionScheduling(1,iGen)
            Mutation.probabilityAddConnection = Mutation.probabilityAddConnectionScheduling(2,iGen);
            break;
        end
    end
else
    Mutation.probabilityAddConnection = max([...
        Mutation.probabilityAddConnection - Mutation.probabilityAddConnectionShrinkRate,...
        Mutation.probabilityAddConnectionMin]);
end

% Recurrent connection
Mutation.probabilityRecurrency = max([...
    Mutation.probabilityRecurrency - Mutation.probabilityRecurrencyShrinkRate,...
    Mutation.probabilityRecurrencyMin]);

% Weight mutation
Mutation.probabilityMutateWeight = max([...
    Mutation.probabilityMutateWeight - Mutation.probabilityMutateWeightShrinkRate,...
    Mutation.probabilityMutateWeightMin]);

% Display new mutation probabilities
fprintf([...
    'Mutation probabilities update:\n',...
    'Mutation.probabilityAddNode       = %4.5f\n',...
    'Mutation.probabilityAddConnection = %4.5f\n',...
    'Mutation.probabilityRecurrency    = %4.5f\n',...
    'Mutation.probabilityMutateWeight  = %4.5f\n\n',...
    ],...
    Mutation.probabilityAddNode,...
    Mutation.probabilityAddConnection,...
    Mutation.probabilityRecurrency,...
    Mutation.probabilityMutateWeight...
    );

end
```

D. Evaluation Function

```
function [PopulationWithFitnesses, SimDataOut] = ...
    uav_evaluation_function(Population, EvalFunParameters, fitnessLimit, SimData)

%% Init

% It seems that initializing the SimData structure inside
% uav_evaluation_function() causes problem in parfor simulation (i.e. fitness
% are all the same)
SimDataOut = SimData;

% We store Population in PopulationWithFitnesses
PopulationWithFitnesses = Population;

% Number of individuals
nIndividuals = size(Population, 2);

% Number of inputs/outputs excluding previous layers' states
nInputs = 25;
nOutputs = 4;

% Create temporary directory to put the generated neural networks if it doesn't
% already exist
folderName = 'temp_uav_nn_controller';
if ~exist(folderName,'dir')
    mkdir(folderName);
end

% Number of trajectories
nTrajectories = numel(EvalFunParameters.Trajectory);

% Current generation number
iGeneration = EvalFunParameters.iGeneration;

%% Generate neural network functions for each individual

% Function name string
funNameString = 'uav_nn_controller';

% Folders name initialization
subFolderNameArray                = cell(1,nIndividuals);
folderWithSubFolderName           = cell(1,nIndividuals);
folderWithSubFolderAndFileWOExtName = cell(1,nIndividuals);
folderWithSubFolderAndFileWExtName  = cell(1,nIndividuals);
workerDir                         = cell(1,nIndividuals);

% Main directory
mainDir = strrep(pwd,'\','/');

% Temporary subfolder. This helps avoid fuck up when using parfor
for iIndividual = 1:nIndividuals
    subFolderNameArray{iIndividual} = gen_temp_string(30);
end

% We create an array to log which individual failed to have their network built.
% This is to make the code more robust to errors. A 1 means the individual is ok
```

```
% and a zero means it is not.
statusIndividualRecord = ones(1,nIndividuals);

% We create the temporary directories and generate the neural network functions
parfor iIndividual = 1:nIndividuals

    % Suffle random number generator seed
    rng('shuffle');

    % Worker directory
    workerDir{iIndividual} = strrep(pwd,'\','/');

    % cd to main directory
    cd(mainDir);

    % Build neural network of current individual
    try
        [neuralNet, nDelays] = build_neural_network(Population(iIndividual));
    catch
        statusIndividualRecord(iIndividual) = 0;
        fprintf('Neural network build failed for individual % 4.f in generation % 4.f\n', ...
            iIndividual, iGeneration);
    end

    % Folder/subfolder path
    folderWithSubFolderName{iIndividual} = ...
        [folderName,'/',subFolderNameArray{iIndividual}];

    % Folder/subfolder/file without extension path
    folderWithSubFolderAndFileWOExtName{iIndividual} = ...
        [folderWithSubFolderName{iIndividual},'/',funNameString];

    % Folder/subfolder/file with extension path
    folderWithSubFolderAndFileWExtName{iIndividual} = ...
        [folderWithSubFolderAndFileWOExtName{iIndividual},'.m'];

    % Add temporary folder to path
    cd([mainDir,'/',folderName]);
    mkdir(subFolderNameArray{iIndividual});
    cd(mainDir);
    addpath(folderWithSubFolderName{iIndividual});

    % Generate function
    if statusIndividualRecord(iIndividual)
        genFunctionMod(neuralNet, ...
            folderWithSubFolderAndFileWOExtName{iIndividual}, ...
            'MatrixOnly', 'yes', 'ShowLinks', 'no');
    end

    % Rehash (https://goo.gl/IDtpzk)
    exist([funNameString,'.m'],'file');
    rehash();

    % Remove path
    rmpath(folderWithSubFolderName{iIndividual});
    rehash();

    % cd back to worker directory
    cd(workerDir{iIndividual});

end

% We add a little pause in order for the filesystem to record to new created files
pause(1);

% We then modify the generated function. Initially, this was in the same parfor
% loop above but it was not reliable (don't know why). Moving this step in a
% seperate parfor loop seems to have helped avoid problem.
nPreviousStatesArray = zeros(1,nIndividuals);
parfor iIndividual = 1:nIndividuals

    if statusIndividualRecord(iIndividual)
        % Suffle random number generator seed
        rng('shuffle');

        % Worker directory
        % workerDir{iIndividual} = strrep(pwd,'\','/');

        % cd to main directory
        cd(mainDir);

        % Add temporary folder to path
        addpath(folderWithSubFolderName{iIndividual});

        % Create function handle
        funHandle = str2func(funNameString);

        % https://goo.gl/IDtpzk
        exist([funNameString,'.m'],'file');
        rehash();

        % Number of previous states
        nPreviousStates = nargin(funHandle) - nInputs;
        if nPreviousStates > 0
            fprintf('nPreviousStates = % 4.f for individual % 4.f in generation % 4.f (%s)\n', ...
                nPreviousStates, iIndividual, iGeneration, subFolderNameArray{iIndividual});
        end
        nPreviousStatesArray(iIndividual) = nPreviousStates;

        % Post-process generated function
        generated_nn_processing(...
            folderWithSubFolderAndFileWExtName{iIndividual}, ...
            funNameString, nPreviousStates);

        % Remove path
        rmpath(folderWithSubFolderName{iIndividual});
        rehash();

        % cd back to worker directory
        cd(workerDir{iIndividual});
    end

end

pause(1);

%% Evaluate networks

% To avoid broadcasting the entire structure into the parfor loop
Trajectory = EvalFunParameters.Trajectory;
SimParam = EvalFunParameters.SimParam;
UavParam = EvalFunParameters.UavParam;
xMin     = UavParam.xMin;
xMax     = UavParam.xMax;
yMin     = UavParam.yMin;
yMax     = UavParam.yMax;
zMin     = UavParam.zMin;
zMax     = UavParam.zMax;
psiMin   = UavParam.psiMin;
psiMax   = UavParam.psiMax;
thetaMin = UavParam.thetaMin;
thetaMax = UavParam.thetaMax;
phiMin   = UavParam.phiMin;
phiMax   = UavParam.phiMax;

% Compute maximum time fitness
timeCellArray = {Trajectory(:).XDesired};
timeVector = zeros(1,nTrajectories);
for iTraj = 1:nTrajectories
    timeVector(iTraj) = timeCellArray{iTraj}.t(end);
end
maxTimeFitness = sum(timeVector);

% Model name
modelName = 'uav_closed_loop_v5';

% https://goo.gl/uNCs79 (In parfor, how to create a temp folder for every worker)
if EvalFunParameters.useParallel

    %% Parallel evaluation

    % Load model
    load_system(modelName);

    % Cycle through population
    parfor iIndividual = 1:nIndividuals

        % We only simulate if the individual is ok
        if statusIndividualRecord(iIndividual)

            % Suffle random number generator seed
            rng('shuffle');

            % Current work directory
            cwd = strrep(pwd,'\','/');

            % Add current work directory in current session
            addpath(cwd);

            % Create temporary folder and change current directory to this folder
            tmpdir = strrep(tempname,'\','/');
            mkdir(tmpdir);
            cd(tmpdir);

            % Add path to uav_nn_controller
            addpath([mainDir,'/',folderWithSubFolderName{iIndividual}]);
            rehash();

            % Load system
            load_system(modelName);
            % load_system(modelName);

            % Create function handle
            funHandle = str2func(funNameString);

            % Initial state of recurrent layer
            if nPreviousStatesArray(iIndividual) == 0
                initialState = 0;
            else
                initialState = zeros(1,nPreviousStatesArray(iIndividual));
            end

            % Neural network Id
            neuralNetworkId = iIndividual;

            % Initialize cell array that will contain simulation results
            t            = cell(1,nTrajectories);
            xDesired     = cell(1,nTrajectories);
            yDesired     = cell(1,nTrajectories);
            zDesired     = cell(1,nTrajectories);
            psiDesired   = cell(1,nTrajectories);
            thetaDesired = cell(1,nTrajectories);
            phiDesired   = cell(1,nTrajectories);
            x            = cell(1,nTrajectories);
            y            = cell(1,nTrajectories);
            z            = cell(1,nTrajectories);
            psi          = cell(1,nTrajectories);
            theta        = cell(1,nTrajectories);
            phi          = cell(1,nTrajectories);
            Va1          = cell(1,nTrajectories);
            Va2          = cell(1,nTrajectories);
            Va3          = cell(1,nTrajectories);
            Va4          = cell(1,nTrajectories);

            % Simulate trajectories
            rapidAccEn = false;
            if strcmp(SimParam.solverType,'Variable-step')
                set_param(modelName,...
                    'SolverType',SimParam.solverType,...
                    'Solver',SimParam.solver,...
                    'SimulationMode',SimParam.simulationMode,...
                    'InitialStep',SimParam.initialStep,...
                    'SaveFormat','StructureWithTime',...
                    'RelTol',SimParam.relTol,...
                    'AbsTol',SimParam.absTol,...
                    'MinStep',SimParam.minStep,...
                    'MaxStep',SimParam.maxStep,...
                    'MaxConsecutiveMinStep',SimParam.maxConsecutiveMinStep,...
                    'SFSimEcho','off',...
                    'SimIntegrity','off');
            else
                set_param(modelName,...
                    'SolverType',SimParam.solverType,...
                    'Solver',SimParam.solver,...
                    'SimulationMode',SimParam.simulationMode,...
                    'FixedStep',SimParam.fixedStep,...
                    'SaveFormat','StructureWithTime',...
                    'RelTol',SimParam.relTol,...
                    'AbsTol',SimParam.absTol,...
                    'SFSimEcho','off',...
                    'SimIntegrity','off');
            end
            if nPreviousStatesArray(iIndividual) > 0 && rapidAccEn
                set_param(modelName, ...
                    'SimulationMode','rapid-accelerator');
            end
            % set_param(modelName,'SimCtrlC','on');
            for iTraj = 1:nTrajectories

                %% Assign simulation variables to base workspace
                assignin('base', 'g', UavParam.g); %#ok
                assignin('base', 'nMotors', UavParam.nMotors); %#ok
                assignin('base', 'm', UavParam.m); %#ok
                assignin('base', 'I', UavParam.I); %#ok
                assignin('base', 'PP', UavParam.PP); %#ok
                assignin('base', 'Ra', UavParam.Ra); %#ok
                assignin('base', 'La', UavParam.La); %#ok
                assignin('base', 'ke', UavParam.ke); %#ok
                assignin('base', 'km', UavParam.km); %#ok
                assignin('base', 'kD', UavParam.kD); %#ok
                assignin('base', 'Jm', UavParam.Jm); %#ok
                assignin('base', 'Bm', UavParam.Bm); %#ok
                assignin('base', 'kT', UavParam.kT); %#ok
                assignin('base', 'VaMin', UavParam.VaMin); %#ok
                assignin('base', 'VaMax', UavParam.VaMax); %#ok
                assignin('base', 'omegaInitial', Trajectory(iTraj).IniCond.omegaInitial); %#ok
                assignin('base', 'omegaDelay', Trajectory(iTraj).IniCond.omegaDelay); %#ok
                assignin('base', 'iaInitial', Trajectory(iTraj).IniCond.iaInitial); %#ok
                assignin('base', 'iaDelay', Trajectory(iTraj).IniCond.iaDelay); %#ok
                assignin('base', 'vPtInitial', Trajectory(iTraj).IniCond.vPtInitial); %#ok
                assignin('base', 'vPtDelay', Trajectory(iTraj).IniCond.vPtDelay); %#ok
                assignin('base', 'xPInitial', Trajectory(iTraj).IniCond.xPInitial); %#ok
                assignin('base', 'xPDelay', Trajectory(iTraj).IniCond.xPDelay); %#ok
                assignin('base', 'omegabtInitial', Trajectory(iTraj).IniCond.omegabtInitial); %#ok
                assignin('base', 'omegabtDelay', Trajectory(iTraj).IniCond.omegabtDelay); %#ok
                assignin('base', 'eulerZYXInitial', Trajectory(iTraj).IniCond.eulerZYXInitial); %#ok
                assignin('base', 'eulerZYXDelay', Trajectory(iTraj).IniCond.eulerZYXDelay); %#ok
                assignin('base', 'XDesired', Trajectory(iTraj).XDesired); %#ok
                assignin('base', 'YDesired', Trajectory(iTraj).YDesired); %#ok
                assignin('base', 'ZDesired', Trajectory(iTraj).ZDesired); %#ok
                assignin('base', 'PsiDesired', Trajectory(iTraj).PsiDesired); %#ok
                assignin('base', 'ThetaDesired', Trajectory(iTraj).ThetaDesired); %#ok
                assignin('base', 'PhiDesired', Trajectory(iTraj).PhiDesired); %#ok
                assignin('base', 'neuralNetworkId', neuralNetworkId); %#ok
```

```
            assignin('base', 'nPreviousStates', nPreviousStatesArray(iIndividual)); %#ok
            assignin('base', 'initialState', initialState); %#ok
            assignin('base', 'funNameString', double(funNameString)); %#ok
            assignin('base', 'lowerXTol', Trajectory(iTraj).XDesired.tol(1)); %#ok
            assignin('base', 'upperXTol', Trajectory(iTraj).XDesired.tol(2)); %#ok
            assignin('base', 'lowerYTol', Trajectory(iTraj).YDesired.tol(1)); %#ok
            assignin('base', 'upperYTol', Trajectory(iTraj).YDesired.tol(2)); %#ok
            assignin('base', 'lowerZTol', Trajectory(iTraj).ZDesired.tol(1)); %#ok
            assignin('base', 'upperZTol', Trajectory(iTraj).ZDesired.tol(2)); %#ok
            assignin('base', 'lowerPsiTol', Trajectory(iTraj).PsiDesired.tol(1)); %#ok
            assignin('base', 'upperPsiTol', Trajectory(iTraj).PsiDesired.tol(2)); %#ok
            assignin('base', 'lowerThetaTol', Trajectory(iTraj).ThetaDesired.tol(1)); %#ok
            assignin('base', 'upperThetaTol', Trajectory(iTraj).ThetaDesired.tol(2)); %#ok
            assignin('base', 'lowerPhiTol', Trajectory(iTraj).PhiDesired.tol(1)); %#ok
            assignin('base', 'upperPhiTol', Trajectory(iTraj).PhiDesired.tol(2)); %#ok
            assignin('base', 'maxViolationX', Trajectory(iTraj).maxViolationX); %#ok
            assignin('base', 'maxViolationY', Trajectory(iTraj).maxViolationY); %#ok
            assignin('base', 'maxViolationZ', Trajectory(iTraj).maxViolationZ); %#ok
            assignin('base', 'maxViolationPsi', Trajectory(iTraj).maxViolationPsi); %#ok
            assignin('base', 'maxViolationTheta', Trajectory(iTraj).maxViolationTheta); %#ok
            assignin('base', 'maxViolationPhi', Trajectory(iTraj).maxViolationPhi); %#ok
            assignin('base', 'violationFactorX', Trajectory(iTraj).violationFactorX); %#ok
            assignin('base', 'violationFactorY', Trajectory(iTraj).violationFactorY); %#ok
            assignin('base', 'violationFactorZ', Trajectory(iTraj).violationFactorZ); %#ok
            assignin('base', 'violationFactorPsi', Trajectory(iTraj).violationFactorPsi); %#ok
            assignin('base', 'violationFactorTheta', Trajectory(iTraj).violationFactorTheta); %#ok
            assignin('base', 'violationFactorPhi', Trajectory(iTraj).violationFactorPhi); %#ok
            assignin('base', 'currentTimeThresholdX', Trajectory(iTraj).violationTimeThresholdX); %#ok
            assignin('base', 'currentTimeThresholdY', Trajectory(iTraj).violationTimeThresholdY); %#ok
            assignin('base', 'currentTimeThresholdZ', Trajectory(iTraj).violationTimeThresholdZ); %#ok
            assignin('base', 'currentTimeThresholdPsi', Trajectory(iTraj).violationTimeThresholdPsi); %#ok
            assignin('base', 'currentTimeThresholdTheta', Trajectory(iTraj).violationTimeThresholdTheta); %#ok
            assignin('base', 'currentTimeThresholdPhi', Trajectory(iTraj).violationTimeThresholdPhi); %#ok
            assignin('base', 'fullLogging', -1); %#ok
            assignin('base', 'loggingSampleTime', 0.01); %#ok
            assignin('base', 'xDesiredMin', UavParam.xDesiredMin); %#ok
            assignin('base', 'xDesiredMax', UavParam.xDesiredMax); %#ok
            assignin('base', 'yDesiredMin', UavParam.yDesiredMin); %#ok
            assignin('base', 'yDesiredMax', UavParam.yDesiredMax); %#ok
            assignin('base', 'zDesiredMin', UavParam.zDesiredMin); %#ok
            assignin('base', 'zDesiredMax', UavParam.zDesiredMax); %#ok
            assignin('base', 'psiDesiredMin', UavParam.psiDesiredMin); %#ok
            assignin('base', 'psiDesiredMax', UavParam.psiDesiredMax); %#ok
            assignin('base', 'thetaDesiredMin', UavParam.thetaDesiredMin); %#ok
            assignin('base', 'thetaDesiredMax', UavParam.thetaDesiredMax); %#ok
            assignin('base', 'phiDesiredMin', UavParam.phiDesiredMin); %#ok
            assignin('base', 'phiDesiredMax', UavParam.phiDesiredMax); %#ok
            assignin('base', 'xMin', UavParam.xMin); %#ok
            assignin('base', 'xMax', UavParam.xMax); %#ok
            assignin('base', 'yMin', UavParam.yMin); %#ok
            assignin('base', 'yMax', UavParam.yMax); %#ok
            assignin('base', 'zMin', UavParam.zMin); %#ok
            assignin('base', 'zMax', UavParam.zMax); %#ok
            assignin('base', 'uMin', UavParam.uMin); %#ok
            assignin('base', 'uMax', UavParam.uMax); %#ok
            assignin('base', 'vMin', UavParam.vMin); %#ok
            assignin('base', 'vMax', UavParam.vMax); %#ok
            assignin('base', 'wMin', UavParam.wMin); %#ok
            assignin('base', 'wMax', UavParam.wMax); %#ok
            assignin('base', 'udotMin', UavParam.udotMin); %#ok
            assignin('base', 'udotMax', UavParam.udotMax); %#ok
            assignin('base', 'vdotMin', UavParam.vdotMin); %#ok
            assignin('base', 'vdotMax', UavParam.vdotMax); %#ok
            assignin('base', 'wdotMin', UavParam.wdotMin); %#ok
            assignin('base', 'wdotMax', UavParam.wdotMax); %#ok
            assignin('base', 'psiMin', UavParam.psiMin); %#ok
            assignin('base', 'psiMax', UavParam.psiMax); %#ok
            assignin('base', 'thetaMin', UavParam.thetaMin); %#ok
            assignin('base', 'thetaMax', UavParam.thetaMax); %#ok
            assignin('base', 'phiMin', UavParam.phiMin); %#ok
            assignin('base', 'phiMax', UavParam.phiMax); %#ok
            assignin('base', 'pMin', UavParam.pMin); %#ok
            assignin('base', 'pMax', UavParam.pMax); %#ok
            assignin('base', 'qMin', UavParam.qMin); %#ok
            assignin('base', 'qMax', UavParam.qMax); %#ok
            assignin('base', 'rMin', UavParam.rMin); %#ok
            assignin('base', 'rMax', UavParam.rMax); %#ok
            assignin('base', 'omega1Min', UavParam.omega1Min); %#ok
            assignin('base', 'omega1Max', UavParam.omega1Max); %#ok
            assignin('base', 'omega2Min', UavParam.omega2Min); %#ok
            assignin('base', 'omega2Max', UavParam.omega2Max); %#ok
            assignin('base', 'omega3Min', UavParam.omega3Min); %#ok
            assignin('base', 'omega3Max', UavParam.omega3Max); %#ok
            assignin('base', 'omega4Min', UavParam.omega4Min); %#ok
            assignin('base', 'omega4Max', UavParam.omega4Max); %#ok

            %% Launch simulation
            try
                if nPreviousStatesArray(iIndividual) == 0 || ~rapidAccEn
                    simout = sim(modelName,...
                        'StopTime',num2str(max(Trajectory(iTraj).XDesired.t)),...
                        'FastRestart',SimParam.fastRestart);
                else
                    simout = sim(modelName,...
                        'StopTime',num2str(max(Trajectory(iTraj).XDesired.t)));
                end
            catch ME
                statusIndividualRecord(iIndividual) = 0;
                fprintf('Simulation failed for individual % 8.f     in generation % 8.f. Error is:\n', ...
                    iIndividual, iGeneration);
                fprintf('identifier: %s    message: %s\n', ME.identifier, ME.message);
                break;
            end

            %% Extract simulation results
            simout1              = simout.get('simout1');
            t{iTraj}             = simout1.time;
            xDesired{iTraj}      = simout1.signals.values(:,1);
            yDesired{iTraj}      = simout1.signals.values(:,2);
            zDesired{iTraj}      = simout1.signals.values(:,3);
            psiDesired{iTraj}    = simout1.signals.values(:,4);
            thetaDesired{iTraj}  = simout1.signals.values(:,5);
            phiDesired{iTraj}    = simout1.signals.values(:,6);
            x{iTraj}             = simout1.signals.values(:,7);
            y{iTraj}             = simout1.signals.values(:,8);
            z{iTraj}             = simout1.signals.values(:,9);
            psi{iTraj}           = simout1.signals.values(:,16);
            theta{iTraj}         = simout1.signals.values(:,17);
            phi{iTraj}           = simout1.signals.values(:,18);
            Va1{iTraj}           = simout1.signals.values(:,41);
            Va2{iTraj}           = simout1.signals.values(:,42);
            Va3{iTraj}           = simout1.signals.values(:,43);
            Va4{iTraj}           = simout1.signals.values(:,44);

        end
        if nPreviousStatesArray(iIndividual) == 0 || ~rapidAccEn
            set_param(modelName,'FastRestart','off');
        end

        % Close system
        close_system(modelName,0);

        % Remove path and folder to uav_nn_controller
        rmpath([mainDir,'/',folderWithSubFolderName{iIndividual}]);
        try
            rmdir([mainDir,'/',folderWithSubFolderName{iIndividual}], 's');
        catch
            try
                rmdir([mainDir,'/',folderWithSubFolderName{iIndividual}]);
            catch
                fprintf('Unable to remove %s for individual % 8.f     in generation % 8.f\n', ...
                    [mainDir,'/',folderWithSubFolderName{iIndividual}], iIndividual, iGeneration);
            end
        end
        rehash();

        % Change back to work directory and remove temporary directory
        cd(cwd);
        try
            rmdir(tmpdir, 's');
        catch
            try
                rmdir(tmpdir);
            catch
                fprintf('Unable to remove %s for individual % 8.f     in generation % 8.f\n', ...
                    tmpdir, iIndividual, iGeneration);
            end
        end
        rmpath(cwd);

    end

    % Compute fitness for this individual
    if statusIndividualRecord(iIndividual)
        [fitnessSignal,rewardTime] = compute_signal_error_fitness(nTrajectories, maxTimeFitness, ...
            xMin, xMax, yMin, yMax, zMin, zMax, psiMin, psiMax, thetaMin, thetaMax, phiMin, phiMax, ...
            t, xDesired, yDesired, zDesired, psiDesired, thetaDesired, phiDesired, ...
            x, y, z, psi, theta, phi, Va1, Va2, Va3, Va4);
        fitness = compute_fitness(nTrajectories, Trajectory, ...
            t, xDesired, yDesired, zDesired, psiDesired, thetaDesired, phiDesired, ...
            x, y, z, psi, theta, phi, rewardTime);
        PopulationWithFitnesses(iIndividual).fitness = fitness + fitnessSignal;
    else
        PopulationWithFitnesses(iIndividual).fitness = 0.0;
    end

    fprintf('fitness = % 8.2f     for individual % 8.f     in generation % 8.f\n', ...
        PopulationWithFitnesses(iIndividual).fitness, iIndividual, iGeneration);

    % Store simulation results
    if statusIndividualRecord(iIndividual)
        for iTraj = 1:nTrajectories
            nPtsTmp = numel(t{iTraj});
            SimDataOut(iIndividual).t{iTraj}(1:nPtsTmp)           = t{iTraj};
            SimDataOut(iIndividual).xDesired{iTraj}(1:nPtsTmp)    = xDesired{iTraj};
            SimDataOut(iIndividual).yDesired{iTraj}(1:nPtsTmp)    = yDesired{iTraj};
            SimDataOut(iIndividual).zDesired{iTraj}(1:nPtsTmp)    = zDesired{iTraj};
            SimDataOut(iIndividual).psiDesired{iTraj}(1:nPtsTmp)  = psiDesired{iTraj};
            SimDataOut(iIndividual).thetaDesired{iTraj}(1:nPtsTmp) = thetaDesired{iTraj};
            SimDataOut(iIndividual).phiDesired{iTraj}(1:nPtsTmp)  = phiDesired{iTraj};
            SimDataOut(iIndividual).x{iTraj}(1:nPtsTmp)           = x{iTraj};
            SimDataOut(iIndividual).y{iTraj}(1:nPtsTmp)           = y{iTraj};
            SimDataOut(iIndividual).z{iTraj}(1:nPtsTmp)           = z{iTraj};
            SimDataOut(iIndividual).psi{iTraj}(1:nPtsTmp)         = psi{iTraj};
            SimDataOut(iIndividual).theta{iTraj}(1:nPtsTmp)       = theta{iTraj};
            SimDataOut(iIndividual).phi{iTraj}(1:nPtsTmp)         = phi{iTraj};
            SimDataOut(iIndividual).Va1{iTraj}(1:nPtsTmp)         = Va1{iTraj};
            SimDataOut(iIndividual).Va2{iTraj}(1:nPtsTmp)         = Va2{iTraj};
            SimDataOut(iIndividual).Va3{iTraj}(1:nPtsTmp)         = Va3{iTraj};
            SimDataOut(iIndividual).Va4{iTraj}(1:nPtsTmp)         = Va4{iTraj};

            SimDataOut(iIndividual).t{iTraj}           = SimDataOut(iIndividual).t{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).xDesired{iTraj}    = SimDataOut(iIndividual).xDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).yDesired{iTraj}    = SimDataOut(iIndividual).yDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).zDesired{iTraj}    = SimDataOut(iIndividual).zDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).psiDesired{iTraj}  = SimDataOut(iIndividual).psiDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).thetaDesired{iTraj} = SimDataOut(iIndividual).thetaDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).phiDesired{iTraj}  = SimDataOut(iIndividual).phiDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).x{iTraj}           = SimDataOut(iIndividual).x{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).y{iTraj}           = SimDataOut(iIndividual).y{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).z{iTraj}           = SimDataOut(iIndividual).z{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).psi{iTraj}         = SimDataOut(iIndividual).psi{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).theta{iTraj}       = SimDataOut(iIndividual).theta{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).phi{iTraj}         = SimDataOut(iIndividual).phi{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).Va1{iTraj}         = SimDataOut(iIndividual).Va1{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).Va2{iTraj}         = SimDataOut(iIndividual).Va2{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).Va3{iTraj}         = SimDataOut(iIndividual).Va3{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).Va4{iTraj}         = SimDataOut(iIndividual).Va4{iTraj}(1:nPtsTmp);
        end
    end

end

else % Don't use parallel

    %% Serial evaluation

    % Load system (just once)
    load_system(modelName);

    % Cycle through population
    for iIndividual = 1:nIndividuals

        % We only simulate if the individual is ok
        if statusIndividualRecord(iIndividual)

            % Add temporary folder in path
            addpath(folderWithSubFolderName{iIndividual});
            rehash();

            % Create function handle
            funHandle = str2func(funNameString);

            % Initial state of recurrent layer
            if nPreviousStatesArray(iIndividual) == 0
                initialState = 0;
            else
                initialState = zeros(1,nPreviousStatesArray(iIndividual));
            end

            % Neural network Id
            neuralNetworkId = iIndividual;

            % Initialize cell array that will contain simulation results
            t            = cell(1,nTrajectories);
            xDesired     = cell(1,nTrajectories);
            yDesired     = cell(1,nTrajectories);
            zDesired     = cell(1,nTrajectories);
            psiDesired   = cell(1,nTrajectories);
            thetaDesired = cell(1,nTrajectories);
            phiDesired   = cell(1,nTrajectories);
            x            = cell(1,nTrajectories);
            y            = cell(1,nTrajectories);
            z            = cell(1,nTrajectories);
            psi          = cell(1,nTrajectories);
            theta        = cell(1,nTrajectories);
            phi          = cell(1,nTrajectories);
            Va1          = cell(1,nTrajectories);
            Va2          = cell(1,nTrajectories);
            Va3          = cell(1,nTrajectories);
            Va4          = cell(1,nTrajectories);

            % Simulate trajectories
            rapidAccEn = true;
            set_param(modelName,'FastRestart','off');
            if strcmp(SimParam.solverType,'Variable-step')
                set_param(modelName,...
                    'SolverType',SimParam.solverType,...
                    'Solver',SimParam.solver,...
                    'SimulationMode',SimParam.simulationMode,...
                    'InitialStep',SimParam.initialStep,...
                    'SaveFormat','StructureWithTime',...
                    'RelTol',SimParam.relTol,...
                    'AbsTol',SimParam.absTol,...
```

```
                'MinStep',SimParam.minStep,...
                'MaxStep',SimParam.maxStep,...
                'MaxConsecutiveMinStep',SimParam.maxConsecutiveMinStep,...
                'SFSimEcho','off',...
                'SimIntegrity','off');
        else
            set_param(modelName,...
                'SolverType',SimParam.solverType,...
                'Solver',SimParam.solver,...
                'SimulationMode',SimParam.simulationMode,...
                'FixedStep',SimParam.fixedStep,...
                'SaveFormat','StructureWithTime',...
                'RelTol',SimParam.relTol,...
                'AbsTol',SimParam.absTol,...
                'SFSimEcho','off',...
                'SimIntegrity','off');
        end
        if nPreviousStatesArray(iIndividual) > 0 && rapidAccEn
            set_param(modelName,...
                'SimulationMode','rapid-accelerator');
        end
        if nPreviousStatesArray(iIndividual) == 0 || ~rapidAccEn
            set_param(modelName,'FastRestart',SimParam.fastRestart);
        end
        for iTraj = 1:nTrajectories

            %% Assign simulation variables to base workspace
            assignin('base', 'g', UavParam.g);
            assignin('base', 'nMotors', UavParam.nMotors);
            assignin('base', 'm', UavParam.m);
            assignin('base', 'I', UavParam.I);
            assignin('base', 'PP', UavParam.PP);
            assignin('base', 'Ra', UavParam.Ra);
            assignin('base', 'La', UavParam.La);
            assignin('base', 'ke', UavParam.ke);
            assignin('base', 'km', UavParam.km);
            assignin('base', 'kD', UavParam.kD);
            assignin('base', 'Jm', UavParam.Jm);
            assignin('base', 'Bm', UavParam.Bm);
            assignin('base', 'kT', UavParam.kT);
            assignin('base', 'VaMin', UavParam.VaMin);
            assignin('base', 'VaMax', UavParam.VaMax);
            assignin('base', 'omegaInitial', Trajectory(iTraj).IniCond.omegaInitial);
            assignin('base', 'omegaDelay', Trajectory(iTraj).IniCond.omegaDelay);
            assignin('base', 'iaInitial', Trajectory(iTraj).IniCond.iaInitial);
            assignin('base', 'iaDelay', Trajectory(iTraj).IniCond.iaDelay);
            assignin('base', 'vPtInitial', Trajectory(iTraj).IniCond.vPtInitial);
            assignin('base', 'vPtDelay', Trajectory(iTraj).IniCond.vPtDelay)
            assignin('base', 'xPInitial', Trajectory(iTraj).IniCond.xPInitial);
            assignin('base', 'xPDelay', Trajectory(iTraj).IniCond.xPDelay);
            assignin('base', 'omegabtInitial', Trajectory(iTraj).IniCond.omegabtInitial);
            assignin('base', 'omegabtDelay', Trajectory(iTraj).IniCond.omegabtDelay);
            assignin('base', 'eulerZYXInitial', Trajectory(iTraj).IniCond.eulerZYXInitial);
            assignin('base', 'eulerZYXDelay', Trajectory(iTraj).IniCond.eulerZYXDelay);
            assignin('base', 'XDesired', Trajectory(iTraj).XDesired);
            assignin('base', 'YDesired', Trajectory(iTraj).YDesired);
            assignin('base', 'ZDesired', Trajectory(iTraj).ZDesired);
            assignin('base', 'PsiDesired', Trajectory(iTraj).PsiDesired);
            assignin('base', 'ThetaDesired', Trajectory(iTraj).ThetaDesired);
            assignin('base', 'PhiDesired', Trajectory(iTraj).PhiDesired);
            assignin('base', 'neuralNetworkId', neuralNetworkId);
            assignin('base', 'nPreviousStates', nPreviousStatesArray(iIndividual));
            assignin('base', 'initialState', initialState);
            assignin('base', 'funNameString', double(funNameString));
            assignin('base', 'lowerXTol', Trajectory(iTraj).XDesired.tol(1));
            assignin('base', 'upperXTol', Trajectory(iTraj).XDesired.tol(2));
            assignin('base', 'lowerYTol', Trajectory(iTraj).YDesired.tol(1));
            assignin('base', 'upperYTol', Trajectory(iTraj).YDesired.tol(2));
            assignin('base', 'lowerZTol', Trajectory(iTraj).ZDesired.tol(1));
            assignin('base', 'upperZTol', Trajectory(iTraj).ZDesired.tol(2));
            assignin('base', 'lowerPsiTol', Trajectory(iTraj).PsiDesired.tol(1));
            assignin('base', 'upperPsiTol', Trajectory(iTraj).PsiDesired.tol(2));
            assignin('base', 'lowerThetaTol', Trajectory(iTraj).ThetaDesired.tol(1));
            assignin('base', 'upperThetaTol', Trajectory(iTraj).ThetaDesired.tol(2));
            assignin('base', 'lowerPhiTol', Trajectory(iTraj).PhiDesired.tol(1));
            assignin('base', 'upperPhiTol', Trajectory(iTraj).PhiDesired.tol(2));
            assignin('base', 'maxViolationX', Trajectory(iTraj).maxViolationX);
            assignin('base', 'maxViolationY', Trajectory(iTraj).maxViolationY);
            assignin('base', 'maxViolationZ', Trajectory(iTraj).maxViolationZ);
            assignin('base', 'maxViolationPsi', Trajectory(iTraj).maxViolationPsi);
            assignin('base', 'maxViolationTheta', Trajectory(iTraj).maxViolationTheta);
            assignin('base', 'maxViolationPhi', Trajectory(iTraj).maxViolationPhi);
            assignin('base', 'violationFactorX', Trajectory(iTraj).violationFactorX);
            assignin('base', 'violationFactorY', Trajectory(iTraj).violationFactorY);
            assignin('base', 'violationFactorZ', Trajectory(iTraj).violationFactorZ);
            assignin('base', 'violationFactorPsi', Trajectory(iTraj).violationFactorPsi);
            assignin('base', 'violationFactorTheta', Trajectory(iTraj).violationFactorTheta);
            assignin('base', 'violationFactorPhi', Trajectory(iTraj).violationFactorPhi);
            assignin('base', 'currentTimeThresholdX', Trajectory(iTraj).violationTimeThresholdX);
            assignin('base', 'currentTimeThresholdY', Trajectory(iTraj).violationTimeThresholdY);
            assignin('base', 'currentTimeThresholdZ', Trajectory(iTraj).violationTimeThresholdZ);
            assignin('base', 'currentTimeThresholdPsi', Trajectory(iTraj).violationTimeThresholdPsi);
            assignin('base', 'currentTimeThresholdTheta', Trajectory(iTraj).violationTimeThresholdTheta);
            assignin('base', 'currentTimeThresholdPhi', Trajectory(iTraj).violationTimeThresholdPhi);
            assignin('base', 'fullLogging', ~1);
            assignin('base', 'loggingSampleTime', 0.01);
            assignin('base', 'xDesiredMin', UavParam.xDesiredMin);
            assignin('base', 'xDesiredMax', UavParam.xDesiredMax);
            assignin('base', 'yDesiredMin', UavParam.yDesiredMin);
            assignin('base', 'yDesiredMax', UavParam.yDesiredMax);
            assignin('base', 'zDesiredMin', UavParam.zDesiredMin);
            assignin('base', 'zDesiredMax', UavParam.zDesiredMax);
            assignin('base', 'psiDesiredMin', UavParam.psiDesiredMin);
            assignin('base', 'psiDesiredMax', UavParam.psiDesiredMax);
            assignin('base', 'thetaDesiredMin', UavParam.thetaDesiredMin);
            assignin('base', 'thetaDesiredMax', UavParam.thetaDesiredMax);
            assignin('base', 'phiDesiredMin', UavParam.phiDesiredMin);
            assignin('base', 'phiDesiredMax', UavParam.phiDesiredMax);
            assignin('base', 'xMin', UavParam.xMin);
            assignin('base', 'xMax', UavParam.xMax);
            assignin('base', 'yMin', UavParam.yMin);
            assignin('base', 'yMax', UavParam.yMax);
            assignin('base', 'zMin', UavParam.zMin);
            assignin('base', 'zMax', UavParam.zMax);
            assignin('base', 'uMin', UavParam.uMin);
            assignin('base', 'uMax', UavParam.uMax);
            assignin('base', 'vMin', UavParam.vMin);
            assignin('base', 'vMax', UavParam.vMax);
            assignin('base', 'wMin', UavParam.wMin);
            assignin('base', 'wMax', UavParam.wMax);
            assignin('base', 'udotMin', UavParam.udotMin);
            assignin('base', 'udotMax', UavParam.udotMax);
            assignin('base', 'vdotMin', UavParam.vdotMin);
            assignin('base', 'vdotMax', UavParam.vdotMax);
            assignin('base', 'wdotMin', UavParam.wdotMin);
            assignin('base', 'wdotMax', UavParam.wdotMax);
            assignin('base', 'psiMin', UavParam.psiMin);
            assignin('base', 'psiMax', UavParam.psiMax);
            assignin('base', 'thetaMin', UavParam.thetaMin);
            assignin('base', 'thetaMax', UavParam.thetaMax);
            assignin('base', 'phiMin', UavParam.phiMin);
            assignin('base', 'phiMax', UavParam.phiMax);
            assignin('base', 'pMin', UavParam.pMin);
            assignin('base', 'pMax', UavParam.pMax);
            assignin('base', 'qMin', UavParam.qMin);
            assignin('base', 'qMax', UavParam.qMax);
            assignin('base', 'rMin', UavParam.rMin);
            assignin('base', 'rMax', UavParam.rMax);
            assignin('base', 'omega1Min', UavParam.omega1Min);
            assignin('base', 'omega1Max', UavParam.omega1Max);
            assignin('base', 'omega2Min', UavParam.omega2Min);
            assignin('base', 'omega2Max', UavParam.omega2Max);
            assignin('base', 'omega3Min', UavParam.omega3Min);
            assignin('base', 'omega3Max', UavParam.omega3Max);
            assignin('base', 'omega4Min', UavParam.omega4Min);
            assignin('base', 'omega4Max', UavParam.omega4Max);

            %% Launch simulation
            try
                simout = sim(modelName,...
                    'StopTime',num2str(max(Trajectory(iTraj).XDesired.t)));
            catch ME
                statusIndividualRecord(iIndividual) = 0;
                fprintf('Simulation failed for individual % 8.f     in generation % 8.f. Error is:\n', ...
                    iIndividual, iGeneration);
                fprintf('identifier: %s    message: %s\n', ME.identifier, ME.message);
                break;
            end

            %% Extract simulation results
            simout1                 = simout.get('simout1');
            t{iTraj}                = simout1.time;
            xDesired{iTraj}         = simout1.signals.values(:,1);
            yDesired{iTraj}         = simout1.signals.values(:,2);
            zDesired{iTraj}         = simout1.signals.values(:,3);
            psiDesired{iTraj}       = simout1.signals.values(:,4);
            thetaDesired{iTraj}     = simout1.signals.values(:,5);
            phiDesired{iTraj}       = simout1.signals.values(:,6);
            x{iTraj}                = simout1.signals.values(:,7);
            y{iTraj}                = simout1.signals.values(:,8);
            z{iTraj}                = simout1.signals.values(:,9);
            psi{iTraj}              = simout1.signals.values(:,16);
            theta{iTraj}            = simout1.signals.values(:,17);
            phi{iTraj}              = simout1.signals.values(:,18);
            Va1{iTraj}              = simout1.signals.values(:,41);
            Va2{iTraj}              = simout1.signals.values(:,42);
            Va3{iTraj}              = simout1.signals.values(:,43);
            Va4{iTraj}              = simout1.signals.values(:,44);

        end
        if nPreviousStatesArray(iIndividual) == 0 || ~rapidAccEn
            set_param(modelName,'FastRestart','off');
        end

        % Remove temporary folder in path
        rmpath(folderWithSubFolderName{iIndividual});
        try
            rmdir(folderWithSubFolderName{iIndividual}, 's');
        catch
            try
                rmdir(folderWithSubFolderName{iIndividual});
            catch
                fprintf('Unable to remove %s for individual % 8.f     in generation % 8.f\n', ...
                    folderWithSubFolderName{iIndividual}, iIndividual, iGeneration);
            end
        end
        rehash();

    end

    % Compute fitness for this individual
    if statusIndividualRecord(iIndividual)
        [fitnessSignal,rewardTime] = compute_signal_error_fitness(nTrajectories, maxTimeFitness, ...
            xMin, xMax, yMin, yMax, zMin, zMax, psiMin, psiMax, thetaMin, thetaMax, phiMin, phiMax, ...
            t, xDesired, yDesired, zDesired, psiDesired, thetaDesired, phiDesired, ...
            x, y, z, psi, theta, phi, Va1, Va2, Va3, Va4);
        fitness = compute_fitness(nTrajectories, Trajectory, ...
            t, xDesired, yDesired, zDesired, psiDesired, thetaDesired, phiDesired, ...
            x, y, z, psi, theta, phi, rewardTime);
        PopulationWithFitnesses(iIndividual).fitness = fitness + fitnessSignal;
    else
        PopulationWithFitnesses(iIndividual).fitness = 0.0;
    end

    fprintf('fitness = % 8.2f     for individual % 8.f     in generation % 8.f\n', ...
        PopulationWithFitnesses(iIndividual).fitness, iIndividual, iGeneration);

    % Store simulation results
    if statusIndividualRecord(iIndividual)
        for iTraj = 1:nTrajectories
            nPtsTmp = numel(t{iTraj});
            SimDataOut(iIndividual).t{iTraj}(1:nPtsTmp)           = t{iTraj};
            SimDataOut(iIndividual).xDesired{iTraj}(1:nPtsTmp)    = xDesired{iTraj};
            SimDataOut(iIndividual).yDesired{iTraj}(1:nPtsTmp)    = yDesired{iTraj};
            SimDataOut(iIndividual).zDesired{iTraj}(1:nPtsTmp)    = zDesired{iTraj};
            SimDataOut(iIndividual).psiDesired{iTraj}(1:nPtsTmp)  = psiDesired{iTraj};
            SimDataOut(iIndividual).thetaDesired{iTraj}(1:nPtsTmp) = thetaDesired{iTraj};
            SimDataOut(iIndividual).phiDesired{iTraj}(1:nPtsTmp)  = phiDesired{iTraj};
            SimDataOut(iIndividual).x{iTraj}(1:nPtsTmp)           = x{iTraj};
            SimDataOut(iIndividual).y{iTraj}(1:nPtsTmp)           = y{iTraj};
            SimDataOut(iIndividual).z{iTraj}(1:nPtsTmp)           = z{iTraj};
            SimDataOut(iIndividual).psi{iTraj}(1:nPtsTmp)         = psi{iTraj};
            SimDataOut(iIndividual).theta{iTraj}(1:nPtsTmp)       = theta{iTraj};
            SimDataOut(iIndividual).phi{iTraj}(1:nPtsTmp)         = phi{iTraj};
            SimDataOut(iIndividual).Va1{iTraj}(1:nPtsTmp)         = Va1{iTraj};
            SimDataOut(iIndividual).Va2{iTraj}(1:nPtsTmp)         = Va2{iTraj};
            SimDataOut(iIndividual).Va3{iTraj}(1:nPtsTmp)         = Va3{iTraj};
            SimDataOut(iIndividual).Va4{iTraj}(1:nPtsTmp)         = Va4{iTraj};

            SimDataOut(iIndividual).t{iTraj}           = SimDataOut(iIndividual).t{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).xDesired{iTraj}    = SimDataOut(iIndividual).xDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).yDesired{iTraj}    = SimDataOut(iIndividual).yDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).zDesired{iTraj}    = SimDataOut(iIndividual).zDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).psiDesired{iTraj}  = SimDataOut(iIndividual).psiDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).thetaDesired{iTraj} = SimDataOut(iIndividual).thetaDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).phiDesired{iTraj}  = SimDataOut(iIndividual).phiDesired{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).x{iTraj}           = SimDataOut(iIndividual).x{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).y{iTraj}           = SimDataOut(iIndividual).y{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).z{iTraj}           = SimDataOut(iIndividual).z{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).psi{iTraj}         = SimDataOut(iIndividual).psi{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).theta{iTraj}       = SimDataOut(iIndividual).theta{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).phi{iTraj}         = SimDataOut(iIndividual).phi{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).Va1{iTraj}         = SimDataOut(iIndividual).Va1{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).Va2{iTraj}         = SimDataOut(iIndividual).Va2{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).Va3{iTraj}         = SimDataOut(iIndividual).Va3{iTraj}(1:nPtsTmp);
            SimDataOut(iIndividual).Va4{iTraj}         = SimDataOut(iIndividual).Va4{iTraj}(1:nPtsTmp);
        end
    end

end

% Close system
close_system(modelName,0);

end

end

%% Function to compute the fitness of an individual
function f = compute_fitness(nTrajectories, Trajectory, ...
    t, xDesired, yDesired, zDesired, psiDesired, thetaDesired, phiDesired, ...
    x, y, z, psi, theta, phi, rewardTime)

% Initialize total fitness to zero
f = 0;

% Compute fitness for each trajectory
for iTraj = 1:nTrajectories

    % Compute the sample time vector (i.e. t_i - t_i-1)
    deltaT = [0,diff(t{iTraj})'];

    % Compute the parallel trajectory curves according to their tolerance
    [xLower, xUpper] = compute_lower_upper_bound(Trajectory(iTraj).XDesired.tol(1), ...
```

```
        Trajectory(iTraj).XDesired.tol(2), t{iTraj}, xDesired{iTraj});

    [yLower, yUpper] = compute_lower_upper_bound(Trajectory(iTraj).YDesired.tol(1), ...
        Trajectory(iTraj).YDesired.tol(2), t{iTraj}, yDesired{iTraj});

    [zLower, zUpper] = compute_lower_upper_bound(Trajectory(iTraj).ZDesired.tol(1), ...
        Trajectory(iTraj).ZDesired.tol(2), t{iTraj}, zDesired{iTraj});

    [psiLower, psiUpper] = compute_lower_upper_bound(Trajectory(iTraj).PsiDesired.tol(1), ...
        Trajectory(iTraj).PsiDesired.tol(2), t{iTraj}, psiDesired{iTraj});

    [thetaLower, thetaUpper] = compute_lower_upper_bound(Trajectory(iTraj).ThetaDesired.tol(1), ...
        Trajectory(iTraj).ThetaDesired.tol(2), t{iTraj}, thetaDesired{iTraj});

    [phiLower, phiUpper] = compute_lower_upper_bound(Trajectory(iTraj).PhiDesired.tol(1), ...
        Trajectory(iTraj).PhiDesired.tol(2), t{iTraj}, phiDesired{iTraj});

    % Compute the total duration that each trajectory is within the tolerance
    durationX     = sum(deltaT((xLower     <= x{iTraj})     &   (x{iTraj}     <= xUpper)));
    durationY     = sum(deltaT((yLower     <= y{iTraj})     &   (y{iTraj}     <= yUpper)));
    durationZ     = sum(deltaT((zLower     <= z{iTraj})     &   (z{iTraj}     <= zUpper)));
    durationPsi   = sum(deltaT((psiLower   <= psi{iTraj})   &   (psi{iTraj}   <= psiUpper)));
    durationTheta = sum(deltaT((thetaLower <= theta{iTraj}) &   (theta{iTraj} <= thetaUpper)));
    durationPhi   = sum(deltaT((phiLower   <= phi{iTraj})   &   (phi{iTraj}   <= phiUpper)));

    % Compute total fitness
    f = f + rewardTime(iTraj)*(durationX + durationY + durationZ + durationPsi + durationTheta + durationPhi);

end

end

%% Function to compute the fitness associated to the trajectory error
function [fitnessSignal,rewardTime] = compute_signal_error_fitness(nTrajectories, maxTimeFitness, ...
    xMin, xMax, yMin, yMax, zMin, zMax, psiMin, psiMax, thetaMin, thetaMax, phiMin, phiMax, ...
    t, xDesired, yDesired, zDesired, psiDesired, thetaDesired, phiDesired, ...
    x, y, z, psi, theta, phi, Va1, Va2, Va3, Va4)

% Initialize total fitness to zero
fitnessSignal = 0;
fitnessSignalMax = 0;
rewardTime = zeros(1,nTrajectories);

% Compute fitness for each trajectory
for iTraj = 1:nTrajectories

    % Compute fitnesses
    [fitnessSignalX,fitnessMaxX] = sub_fun_error_fitness(xMin, xMax, xDesired{iTraj}, x{iTraj});
    [fitnessSignalY,fitnessMaxY] = sub_fun_error_fitness(yMin, yMax, yDesired{iTraj}, y{iTraj});
    [fitnessSignalZ,fitnessMaxZ,signalErrorSignedZ] = ...
        sub_fun_error_fitness(zMin, zMax, zDesired{iTraj}, z{iTraj});
    [fitnessSignalPsi,fitnessMaxPsi,signalErrorSignedPsi] = ...
        sub_fun_error_fitness(psiMin, psiMax, psiDesired{iTraj}, psi{iTraj});
    [fitnessSignalTheta,fitnessMaxTheta] = sub_fun_error_fitness(thetaMin, thetaMax, thetaDesired{iTraj}, theta{iTraj});
    [fitnessSignalPhi,fitnessMaxPhi]     = sub_fun_error_fitness(phiMin, phiMax, phiDesired{iTraj}, phi{iTraj});

    % Conditions to penalize controllers that send very low or very high voltages
    VaMinThreshold     = 1;
    VaMinRatioThreshold = 0.80;
    VaMinTimeThreshold  = 2;
    VaMaxThreshold      = 8.8;
    VaMaxRatioThreshold = 0.80;
    VaMaxTimeThreshold  = 2;
    [condVaMin,condVaMax] = check_cond_Va(t{iTraj},Va1{iTraj},Va2{iTraj},Va3{iTraj},Va4{iTraj}, ...
        VaMinThreshold,VaMinRatioThreshold,VaMinTimeThreshold, ...
        VaMaxThreshold,VaMaxRatioThreshold,VaMaxTimeThreshold);

    % Conditions to penalize controllers that let the altitude deacrease or increase
    % sharply in Z and/or psi. Compute gradient of signalErrorZ and signalErrorPsi:
    minErrRatioThrshld = 0.80;
    errZThrshld        = 0.5;
    maxErrZThrshld     = 2;
    errPsiThrshld      = 4*pi/180;
    maxErrPsiThrshld   = 8*pi/180;
    gradErrZ     = gradient(signalErrorSignedZ);
    gradErrPsi   = gradient(signalErrorSignedPsi);
    [condGradPosZ,condGradNegZ] = check_cond_grad(gradErrZ, signalErrorSignedZ, ...
        errZThrshld, maxErrZThrshld, minErrRatioThrshld);
    [condGradPosPsi,condGradNegPsi] = check_cond_grad(gradErrPsi, signalErrorSignedPsi, ...
        errPsiThrshld, maxErrPsiThrshld, minErrRatioThrshld);

    % Compute rewards
    overallCondGrad = condGradPosZ || condGradNegZ || condGradPosPsi || condGradNegPsi;
    [rewardTime(iTraj),rewardZ,rewardPsi] = compute_reward(overallCondGrad,...
        condGradPosZ,condGradNegZ,condGradPosPsi,condGradNegPsi,t{iTraj});

    % Compute total fitness
    if overallCondGrad && ~condVaMin && ~condVaMax
        fitnessSignal = fitnessSignal + fitnessSignalX + fitnessSignalY + ...
            rewardZ*fitnessSignalZ + rewardPsi*fitnessSignalPsi + ...
            fitnessSignalTheta + fitnessSignalPhi;
        fitnessSignalMax = fitnessSignalMax + fitnessMaxX + fitnessMaxY + ...
            fitnessMaxZ + fitnessMaxPsi + fitnessMaxTheta + fitnessMaxPhi;
    elseif condVaMin || condVaMax
        fitnessSignalMax = fitnessSignalMax + fitnessMaxX + fitnessMaxY + ...
            fitnessMaxZ + fitnessMaxPsi + fitnessMaxTheta + fitnessMaxPhi;
    else
        fitnessSignal = fitnessSignal + fitnessSignalX + fitnessSignalY + ...
            rewardZ*fitnessSignalZ + rewardPsi*fitnessSignalPsi + ...
            fitnessSignalTheta + fitnessSignalPhi;
        fitnessSignalMax = fitnessSignalMax + fitnessMaxX + fitnessMaxY + ...
            fitnessMaxZ + fitnessMaxPsi + fitnessMaxTheta + fitnessMaxPhi;
    end

end

% Fitness scaling
fitnessSignal = maxTimeFitness*fitnessSignal/fitnessSignalMax;

    function [fitnessSignal,fitnessMax,signalErrorSigned] = sub_fun_error_fitness(qMin, qMax, qDesired, q)
        % Compute max possible error
        sumSignalError1    = sum(abs(qDesired - qMin*ones(size(qDesired))));
        sumSignalError2    = sum(abs(qDesired - qMax*ones(size(qDesired))));
        sumSignalErrorMax  = max([sumSignalError1,sumSignalError2]);
        % Compute actual error
        signalErrorSigned = q - qDesired;
        signalError     = abs(signalErrorSigned);
        sumSignalError  = sum(signalError);
        % Fitness
        fitnessSignal   = abs(sumSignalErrorMax - sumSignalError);
        fitnessMax      = abs(sumSignalErrorMax);
    end

    function [condVaMin,condVaMax] = check_cond_Va(t,Va1,Va2,Va3,Va4, ...
            VaMinThreshold,VaMinRatioThreshold,VaMinTimeThreshold, ...
            VaMaxThreshold,VaMaxRatioThreshold,VaMaxTimeThreshold)
        % Conditions to penalize controllers that send very low voltages
        condVaMin1 = sum(Va1 < VaMinThreshold) >= VaMinRatioThreshold*numel(Va1);
        condVaMin2 = sum(Va2 < VaMinThreshold) >= VaMinRatioThreshold*numel(Va2);
        condVaMin3 = sum(Va3 < VaMinThreshold) >= VaMinRatioThreshold*numel(Va3);
        condVaMin4 = sum(Va4 < VaMinThreshold) >= VaMinRatioThreshold*numel(Va4);
        % Conditions to penalize controllers that send very high voltages
        condVaMax1 = sum(Va1 > VaMaxThreshold) >= VaMaxRatioThreshold*numel(Va1);
        condVaMax2 = sum(Va2 > VaMaxThreshold) >= VaMaxRatioThreshold*numel(Va2);
        condVaMax3 = sum(Va3 > VaMaxThreshold) >= VaMaxRatioThreshold*numel(Va3);
        condVaMax4 = sum(Va4 > VaMaxThreshold) >= VaMaxRatioThreshold*numel(Va4);
        % Compute final condition
        if t(end) >= VaMinTimeThreshold
            condVaMin = condVaMin1 || condVaMin2 || condVaMin3 || condVaMin4;
        else
            condVaMin = false;
        end
        if t(end) >= VaMaxTimeThreshold
            condVaMax = condVaMax1 || condVaMax2 || condVaMax3 || condVaMax4;
        else
            condVaMax = false;
        end
    end

    function [condGradPos,condGradNeg] = check_cond_grad(gradErr, signalErrorSigned, ...
            errThrshld, maxErrThrshld, minErrRatioThrshld)
        % Positive gradient
        condGradPosa = sum(gradErr > 0 & abs(signalErrorSigned) > errThrshld) >= minErrRatioThrshld*numel(gradErr);
        condGradPosb = max(signalErrorSigned) > maxErrThrshld;
        condGradPos  = condGradPosa && condGradPosb;
        % Negative gradient
        condGradNega = sum(gradErr < 0 & abs(signalErrorSigned) > errThrshld) >= minErrRatioThrshld*numel(gradErr);
        condGradNegb = min(signalErrorSigned) < -maxErrThrshld;
        condGradNeg  = condGradNega && condGradNegb;
    end

    function [rewardTime,rewardZ,rewardPsi] = compute_reward(overallCondGrad,...
            condGradPosZ,condGradNegZ,condGradPosPsi,condGradNegPsi,t)
        if overallCondGrad
            if t(end) >= 0 && t(end) < 3
                rewardTime = 0.5;
            elseif t(end) >= 3 && t(end) < 6
                rewardTime = 0.25;
            elseif t(end) >= 6
                rewardTime = 0.125;
            end
            if t(end) >= 1 && t(end) < 3
                rewardZ    = 1;
                rewardPsi  = 1;
            elseif t(end) >= 3 && t(end) < 6
                rewardZ    = 1.25;
                rewardPsi  = 1.25;
            elseif t(end) >= 6 && t(end) < 8
                rewardZ    = 1.5;
                rewardPsi  = 1.5;
            elseif t(end) >= 8
                rewardZ    = 1.75;
                rewardPsi  = 1.75;
            else
                rewardZ    = 0.5;
                rewardPsi  = 0.5;
            end
            if condGradPosZ || condGradNegZ
                rewardZ = 0;
            end
            if condGradPosPsi || condGradNegPsi
                rewardPsi = 0;
            end
        else
            if t(end) >= 1 && t(end) < 3
                rewardTime = 1.5;
                rewardZ    = 1.5;
                rewardPsi  = 1.5;
            elseif t(end) >= 3 && t(end) < 6
                rewardTime = 2;
                rewardZ    = 2;
                rewardPsi  = 2;
            elseif t(end) >= 6 && t(end) < 8
                rewardTime = 2.5;
                rewardZ    = 2.5;
                rewardPsi  = 2.5;
            elseif t(end) >= 8
                rewardTime = 3;
                rewardZ    = 3;
                rewardPsi  = 3;
            else
                rewardTime = 1;
                rewardZ    = 1;
                rewardPsi  = 1;
            end
        end
    end

end
```

E. Build Neural Network

```
%% build_neural_network() (Main)
function [neuralNet, nDelays, varargout] = build_neural_network(Individual)
% This function generate a neural network from each individual's codeGenes and
% connectionGenes
% References:   https://goo.gl/FcHG0z
%               https://goo.gl/rdtEzo
%               https://goo.gl/odfpyB (genFunction)
%
% Note: Neural network toolbox uses layers for nodes since a layer can contains
% multiple neurons. However, NEAT generates only nodes with single neuron. Hence,
% in this function, layers correspond to nodes.
%
% Also, NEAT's input nodes (nodeType = 1) are input ports in Neural network
% toolbox's terminology. The Neural network toolbox's input layers are the
% layers directly connected to the input ports. To simplify things, we'll define
% NEAT's input nodes as input layers with weight = 1 and linear activation
% function. NEAT's output nodes corresponds to the toolbox's last layer before
% connection to output.

%% Preprocessing

% Extract node ID and node type
nodeId   = Individual.nodeGenes(1,:);
nodeType = Individual.nodeGenes(2,:);

% Sort nodeId in ascending order
[nodeId, sortIndex] = sort(nodeId);

% Sort nodeType according to sortIndex
nodeType = nodeType(sortIndex);

% We put output layer at the end for visualisation purposes (when using the view
% function)
nodeId   = [nodeId(nodeType ~= 2),   nodeId(nodeType == 2)];
nodeType = [nodeType(nodeType ~= 2), nodeType(nodeType == 2)];

% Extract connections info
connId      = Individual.connectionGenes(1,:);
connFrom    = Individual.connectionGenes(2,:);
connTo      = Individual.connectionGenes(3,:);
connWeight  = Individual.connectionGenes(4,:);
connEnabled = Individual.connectionGenes(5,:);

% We remove disabled connections
indexEnabled = connEnabled == 1;
connId       = connId(indexEnabled);
connFrom     = connFrom(indexEnabled);
connTo       = connTo(indexEnabled);
connWeight   = connWeight(indexEnabled);
connEnabled  = connEnabled(indexEnabled);

% Sort connId in ascending order
```

```
[connId, sortIndex] = sort(connId);

% Sort connFrom, connTo, connWeight and connEnabled according to sortIndex
connFrom    = connFrom(sortIndex);
connTo      = connTo(sortIndex);
connWeight  = connWeight(sortIndex);
connEnabled = connEnabled(sortIndex);

% Inputs' node ID
inputId = nodeId(nodeType == 1);

% Outputs' node ID
outputId = nodeId(nodeType == 2);

% Hidden nodes' node ID
hiddenId = nodeId(nodeType == 3);

% Biases' node ID. NEAT uses only one bias for all the networks and choose
% whether to connect it to other nodes or not. By assigning a different weight
% to every connection from the bias to a node, it is as if these nodes had their
% own bias.
biasId = nodeId(nodeType == 4);

%% Network architecture

% Initialize network object
neuralNet = network();

% Number of nodes
nNodes = numel(nodeId);

% Number of connections (including biases)
nConns = numel(connId);

% Number of biases. NEAT uses only one bias for all the networks and choose
% whether to connect it to other nodes or not. By assigning a different weight
% to every connection from the bias to a node, it is as if these nodes had their
% own bias.
nBiases = sum(nodeType == 4);

% Number of inputs
neuralNet.numInputs = sum(nodeType == 1);

% Number of layers
% Neural network toolbox does not consider biases as input nodes
neuralNet.numLayers = nNodes - nBiases;

% Note: Neural network toolbox uses layers for nodes since a layer can contains
% multiple neurons. However, NEAT generates only nodes with single neuron. So,
% layers correspond to nodes here.

% Array with first column containing the layer ID and second column its
% corresponding ID. We need to do this since neural network toolbox does not
% consider biases as input nodes
layerList = [[1:neuralNet.numLayers]',[nodeId(nodeType -= 4)]'];

% Inputs node's layer ID according to layerList
inputLayerId = layerList(ismember(layerList(:,2),inputId),1);

% Outputs node's layer ID according to layerList
outputLayerId = layerList(ismember(layerList(:,2),outputId),1);

% Hidden node's layer ID according to layerList
hiddenLayerId = layerList(ismember(layerList(:,2),hiddenId),1);

%% Bias connections

% We connect the biases to their layers
for jj = biasId
    % Find nodes connected to this bias. connFrom == jj gives a logical indexing
    % vector of the node that receive a connection from the bias.
    % connTo(connFrom == jj) gives the actual node IDs connected to the bias.
    iNodeConnectedToBias = connTo(connFrom == jj);
    % Find layers connected to this bias
    iLayerConnectedToBias = [];
    for kk = 1:numel(iNodeConnectedToBias)
        iLayerConnectedToBias = ...
            [iLayerConnectedToBias, ...
            layerList(layerList(:,2) == iNodeConnectedToBias(kk),1)];
    end
    % Set bias for the above layers (if any)
    if ~isempty(iLayerConnectedToBias)
        for ii = iLayerConnectedToBias
            % neuralNet.biasConnect(TO ith layer)
            neuralNet.biasConnect(ii) = 1;
        end
    end
end

%% Connect inputs, outputs and hidden layers

% Input counter for the following loop
inputCounter = 1;

% Cycle through layers (nodes)
for iNode = 1:nNodes

    %% Input connections

    % We specify which layers are input layers. Note that this loop assumes that
    % the input nodes' IDs come before the hidden and output layers.
    if nodeType(iNode) == 1 % If is input node
        % Find layer connected to this input
        iLayerConnectedToInput = layerList(layerList(:,2) == nodeId(iNode),1);
        % Set input for the above layers (if any)
        if ~isempty(iLayerConnectedToInput)
            % neuralNet.inputConnect(TO ith layer, FROM jth input)
            neuralNet.inputConnect(iLayerConnectedToInput,inputCounter) = 1;
        end
        inputCounter = inputCounter + 1;
    end

    %% Layer connection from input layer and hidden layers

    % We connect input layers to hidden layers as well as hidden layers to other
    % hidden layers
    if nodeType(iNode) == 1 || nodeType(iNode) == 2 || nodeType(iNode) == 3 % If is input or hidden node
        % Find nodes that receive connection FROM this node
        iNodeConnectedFromCurrLayer = connTo(connFrom == nodeId(iNode));
        % Find layers that receive connection FROM this node
        iLayerConnectedFromCurrLayer = [];
        for kk = 1:numel(iNodeConnectedFromCurrLayer)
            iLayerConnectedFromCurrLayer = ...
                [iLayerConnectedFromCurrLayer, ...
                layerList(layerList(:,2) == iNodeConnectedFromCurrLayer(kk),1)];
        end
        % Set FROM current layer for the above layers (if any)
        if ~isempty(iLayerConnectedFromCurrLayer)
            for ii = iLayerConnectedFromCurrLayer
                % neuralNet.layerConnect(TO ith layer, FROM jth layer)
                iTmp = layerList(layerList(:,2) == nodeId(iNode),1);
                neuralNet.layerConnect(ii,iTmp) = 1;
            end
        end
    end

    %% Output connections

    % We specify which layers are output layers
    if nodeType(iNode) == 2 % If is output node
        % Find layer ID corresponding to current node ID
        iLayerConnectedToOutput = layerList(layerList(:,2) == nodeId(iNode),1);
        % Set FROM for this output
        if ~isempty(iLayerConnectedToOutput)
            % neuralNet.outputConnect(FROM ith layer)
            neuralNet.outputConnect(iLayerConnectedToOutput) = 1;
        end
    end

end

%% Set layer connection and delays for recurrent connections
% We use MATLAB's graph theory toolbox (https://goo.gl/D0cbX5) to find
% recurrences in the network

% First, remove "from connections" of biases from connFrom
connFromNoBias = connFrom;
connToNoBias   = connTo;
for iBias = 1:nBiases
    indexOfBias      = connFromNoBias -= biasId(iBias);
    connFromNoBias  = connFromNoBias(indexOfBias);
    connToNoBias    = connToNoBias(indexOfBias);
end

% Next, create start and target layers arrays
% Important: This will fail if no input are connected!
startLayers  = [];
targetLayers = [];
for ii = 1:numel(connFromNoBias)
    startLayers  = [startLayers,layerList(layerList(:,2) == connFromNoBias(ii),1)];
    targetLayers = [targetLayers,layerList(layerList(:,2) == connToNoBias(ii),1)];
end
tmpArray = [...
    startLayers;
    targetLayers]';
tmpArray = unique(tmpArray,'rows');
startLayers = tmpArray(:,1)';
targetLayers = tmpArray(:,2)';

% Build directed graph object
directedGraph = digraph(startLayers, targetLayers);

% Find recurrent connections
recurrentConnection = dfsearch(directedGraph, 1, 'edgetodiscovered', 'Restart', true);
% plot(directedGraph, 'Layout', 'force');

% Find connections that go from one output to another output
% excluding output that goes into itself
for ii = 1:numel(connFromNoBias)
    condition1 = nodeType(nodeId == connFromNoBias(ii)) == 2;
    condition2 = nodeType(nodeId == connToNoBias(ii)) == 2;
    condition3 = nodeId(nodeId == connFromNoBias(ii)) -= nodeId(nodeId == connToNoBias(ii));
    if condition1 && condition2 && condition3
        recurrentConnection = ...
            [recurrentConnection;
            [layerList(layerList(:,2) == connFromNoBias(ii),1),...
            layerList(layerList(:,2) == connToNoBias(ii),1)]...
            ];
    end
end

% Finally, assign delays to recurrent connections (layers)
for ii = 1:numel(recurrentConnection(:,1))
    % neuralNet.layerWeights{TO ith layer, FROM jth layer}.delays
    iLayer = recurrentConnection(ii,2);
    jLayer = recurrentConnection(ii,1);
    neuralNet.layerConnect(iLayer,jLayer) = 1; % Is this necessary ?
    neuralNet.layerWeights{iLayer,jLayer}.delays = 1;
end

%% Inputs and hidden layers size and activation function

% Set input sizes (all inputs are size 1)
neuralNet.inputs{:}.size = 1;

% Set layers size, activation function and weight initialization function
% We don't use the weight initialization function but the toolbox requires it
neuralNet.layers{:}.size = 1;
neuralNet.layers{hiddenLayerId}.transferFcn = 'logsig';
neuralNet.layers{outputLayerId}.transferFcn = 'logsig';
neuralNet.layers{inputLayerId}.transferFcn  = 'purelin';
neuralNet.layers{:}.initFcn                 = ''; % initnw

%% Set network functions

% We don't need those but Neural network toolbox requires them in order to
% initialize the network
neuralNet.initFcn = ''; % initlay

neuralNet.performFcn = ''; % mse
neuralNet.trainFcn = ''; % trainlm

neuralNet.divideFcn = ''; % dividerand

% neuralNet.plotFcns = {'plotperform','plottrainstate'};
neuralNet.plotFcns = {};

%% Initialize network

neuralNet = init(neuralNet);

%% Set biases weights

for jj = biasId
    % Find nodes connected to this bias (i.e. nodes that receive connection from
    % the bias)
    iNodeConnectedToBias = connTo(connFrom == jj);
    % Find layers connected to this bias
    iLayerConnectedToBias = [];
    for kk = 1:numel(iNodeConnectedToBias)
        iLayerConnectedToBias = ...
            [iLayerConnectedToBias, ...
            layerList(layerList(:,2) == iNodeConnectedToBias(kk),1)];
    end
    % Set bias
    if ~isempty(iLayerConnectedToBias)
        for kk = 1:numel(iLayerConnectedToBias)
            ii = iLayerConnectedToBias(kk);
            biasWeight = connWeight(connFrom == jj & connTo == iNodeConnectedToBias(kk));
            neuralNet.b{ii} = biasWeight;
        end
    end
end

%% Set layers weights

% Input counter for the following loop
inputCounter = 1;

% Cycle through layers (nodes)
for iNode = 1:nNodes

    %% Weight input layer

    if nodeType(iNode) == 1 % If is input node
        % Find input's layer Id connected to this input
        iLayerConnectedToInput = layerList(layerList(:,2) == nodeId(iNode),1);
        % Set weight
        if ~isempty(iLayerConnectedToInput)
            % neuralNet.IW{TO ith layer, FROM jth input}
            neuralNet.IW{iLayerConnectedToInput,inputCounter} = 1;
        end
        inputCounter = inputCounter + 1;
    end
```

```
337
338     %% Connections to hidden and output layers
339
340     if nodeType(iNode) == 2 || nodeType(iNode) == 3 % If output or hidden node
341         % Find nodes that are connected FROM this node
342         iNodeConnectedFromCurrLayer = connFrom(connTo == nodeId(iNode));
343         % Find layers connected from this node
344         iLayerConnectedFromCurrLayer = [];
345         for kk = 1:numel(iNodeConnectedFromCurrLayer)
346             iLayerConnectedFromCurrLayer = ...
347                 [iLayerConnectedFromCurrLayer, ...
348                 layerList(layerList(:,2) == iNodeConnectedFromCurrLayer(kk),1)];
349         end
350         % Set FROM current layer for the above layers (if any)
351         if ~isempty(iLayerConnectedFromCurrLayer)
352             for jj = iLayerConnectedFromCurrLayer
353                 % neuralNet.LW{TO ith layer, FROM jth layer}
354                 iConnId = ...
355                     (connFrom == layerList(jj,2)) & (connTo == nodeId(iNode));
356                 ii = layerList(layerList(:,2) == nodeId(iNode),1);
357
358                 % Sometimes, NEAT will create two same connections but with
359                 % different weights. When this happens, neuralNet.LW{ii,jj}
360                 % fails because there are more than one weight. A solution could
361                 % be to add the two weights.
362                 neuralNet.LW{ii,jj} = sum(connWeight(iConnId));
363             end
364         end
365     end
366
367 end
368
369 % Weights for reccurent connections
370 nDelays = numel(recurrentConnection(:,1));
371 for ii = 1:nDelays
372     % neuralNet.LW{TO ith layer, FROM jth layer}
373     iLayer = recurrentConnection(ii,2);
374     jLayer = recurrentConnection(ii,1);
375     iConnId = (connFrom == layerList(jLayer,2)) & (connTo == layerList(iLayer,2));
376     % We sum in case NEAT decides to duplicate the recurrent connection
377     neuralNet.LW{iLayer,jLayer} = sum(connWeight(iConnId));
378 end
379
380 end
```

F. *Neural Network Processing*

```
1  function generated_nn_processing(funPath, funName, nPreviousStates)
2  % This function add lines of code at the beginning and end of the generated
3  % neural network functions in order to store into matrices the previous
4  % recurrent layer states in place of havin a seperate input for every state.
5
6  % Read in the file as binary and convert to chars.
7  if nPreviousStates >= 1
8      fid = fopen(funPath);
9      text = fread(fid, inf, '*char')';
10     fclose(fid);
11 end
12
13 % Find recurrent layer ID
14 if nPreviousStates >= 1
15     if nPreviousStates > 1
16         out1 = regexp(text,'ai(\d*),','tokens');
17         out2 = regexp(text,',ai(\d*)\)','tokens');
18     else
19         out1 = {{}};
20         out2 = regexp(text,',ai(\d*)\)','tokens');
21     end
22     out1 = [out1{:}];
23     out2 = [out2{:}];
24     out = [out1,out2];
25     out = unique(out);
26     recurrLayerId = sort(str2double(out));
27 end
28
29 if nPreviousStates >= 1
30     newCodeStart = [];
31     % newCodeEnd = ['afMat = zeros\(1,',num2str(numel(recurrLayerId)),'\);\n'];
32     newCodeEnd = ['afMat = zeros\(',num2str(numel(recurrLayerId)),',1\);\n'];
33     % Form code to add in function
34     for iLayer = 1:numel(recurrLayerId)
35         % Code to add at the beginning of the file
36         newCodeStart = [...
37             newCodeStart,...
38             ['ai',num2str(recurrLayerId(iLayer)),' = aiMat\(',num2str(iLayer),'\);\n']];
39         % Code to add at the end of the file
40         newCodeEnd = [...
41             newCodeEnd,...
42             ['afMat\(',num2str(iLayer),'\) = af',num2str(recurrLayerId(iLayer)),';\n']];
43     end
44 else
45     newCodeStart = 'afMat = aiMat;\n';
46 end
47
48 % Replace ai1,ai2,etc. by aiMat
49 if nPreviousStates >= 1
50     find_and_replace(funPath,',ai(.*?)\)', ',aiMat\)');
51 else
52     find_and_replace(funPath,[funName,'\((.*?)\)'],[funName,'\($1,aiMat\)']);
53 end
54
55 % Replace af1,af2,etc. by afMat
56 if nPreviousStates >= 1
57     find_and_replace(funPath,',af(.*?)\]', ',afMat\]');
58 else
59     find_and_replace(funPath,['\[(.*?)\] = ',funName],['\[$1,afMat\] = ',funName]);
60 end
61
62 % Add code at the beginning
63 find_and_replace(funPath,...
64     '% ===== NEURAL NETWORK CONSTANTS =====\n',...
65     ['% ===== NEURAL NETWORK CONSTANTS =====\n\n',newCodeStart]);
66
67 % Add code at the end
68 if nPreviousStates >= 1
69     find_and_replace(funPath,...
70         'end\n\n% ===',...
71         [newCodeEnd,'end\n\n% ===']);
72 end
73
74 % Replace activation function for Kenneth Stanley version
75 find_and_replace(funPath, 'exp\(-n\)', 'exp(-4.9*n)');
76
77 end
```

References

[1] Maureen Caudill. "Neural networks primer, part I". In: *AI expert* 2.12 (1987), pp. 46–52 (cit. on p. 3).

[2] GekkoQuant. *Evolving Neural Networks through Augmenting Topologies – Part 1 of 4*. Mar. 13, 2016. URL: http://gekkoquant.com/2016/03/13/evolving-neural-networks-through-augmenting-topologies-part-1-of-4/ (visited on 12/03/2016) (cit. on p. 4).

[3] The MathWorks Inc. *6DOF (Euler Angles)*. MATLAB version R2016a. URL: http://www.mathworks.com/help/aeroblks/6dofeulerangles.html (visited on 09/08/2016) (cit. on pp. 1, 2).

[4] Richard Johnson. *MATLAB Programming Style Guidelines*. Version 1.5. Oct. 1, 2002. URL: http://www.datatool.com/downloads/matlab_style_guidelines.pdfl (visited on 12/03/2016) (cit. on p. 5).

[5] Jerome Le Ny. *Integrated Navigation Systems*. Version May 1, 2016. Course notes from ELE6209A – Navigation Systems. École Polytechnique Montréal (cit. on p. 1).

[6] Teppo Luukkonen. "Modelling and control of quadcopter". In: *Independent research project in applied mathematics, Espoo* (2011) (cit. on p. 1).

[7] Christian Mayr. *NEAT Matlab*. Version 1.0. MATLAB NEAT implementation. Aug. 31, 2003. URL: http://nn.cs.utexas.edu/?neatmatlab (visited on 12/03/2016) (cit. on pp. 1, 5).

[8] Tucker McClure. *Find and Replace in Files*. Version 1.1.0.1. Published on MATLAB file exchange. Sept. 1, 2016. URL: https://www.mathworks.com/matlabcentral/fileexchange/42877-find-and-replace-in-files (visited on 12/03/2016).

[9] Robotshop. *550mm RTF Quadcopter UAV*. Image file. URL: http://www.robotshop.com/en/550mm-rtf-quadcopter-uav.html (visited on 12/03/2016) (cit. on p. 1).

[10] David Saussié. *1. Modélisation des systèmes aéronautiques. 1.3 Forces et moments*. Version 15 février 2016. Lecture notes from ELE6208A – Commande des systèmes aéronautiques et spatiaux. École Polytechnique de Montréal, 2016 (cit. on pp. 2, 3).

[11] David Saussié. *1. Modélisation des systèmes aéronautiques. 1.1 Repères*. Version 1 février 2016. Lecture notes from ELE6208A – Commande des systèmes aéronautiques et spatiaux. École Polytechnique de Montréal, 2016 (cit. on p. 2).

[12] Kenneth O Stanley and Risto Miikkulainen. "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2 (2002), pp. 99–127 (cit. on pp. 1, 3–6).